SdsDroneSdk

Version 1.5.3

Sixdof Space

Chapter 1

SdsDroneSdk Documentation

1.1 Introduction

Sixdof Space has created an optical tracking solution offering precision landing with cm level accuracy both indoors and outdoors in direct sunlight. Our patented technology leverages infrared lighting to serve as location beacons. Our sensor unit, connected to your drone can see the beacons on the ground at up to 40 meters. Our algorithms will report the 6dof positional data to the drone, to direct an autonomous landing.

SdsDroneSdk is a shared library that enables users to interface with the Sixdof tracking technology. Drone guidance is the main application of this library, however other applications that require a programmatic interface to the Sixdof technology can use this library as well.

The SdsDroneSdk shared library is available on the following platforms:

- Windows
- Linux Ubuntu
- Linux Raspberry Pi OS (32 bit)
- Linux Raspberry Pi OS (64 bit)

If you want to use the SdsDroneSdk on another platform, please contact the Sixdof team.

1.2 Coordinate Systems

Before using the Sixdof tracking system it is important to understand the coordinate systems involved. The sensor coordinate system is defined according to the image below:

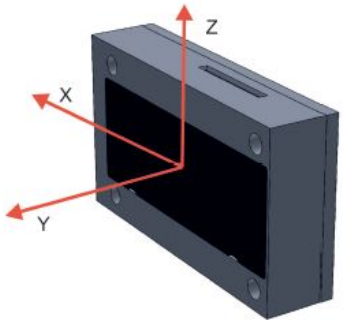


Figure 1.1 Sensor coordinate system

Where the positive x-axis points to the right of the sensor, the positive y-axis point out from the sensor and the positive z-axis completes the right handed system. The field of view can be approximated by a 60 degrees vision cone around the y-axis.

In many applications the user is interested in the location of the drone, and not just the location of the sensor. In these applications, it is the users responsibility to do the rigid body transformation from the sensor to the drone, based on how the sensor is mounted on the drone.

1.3 Tracking Modes

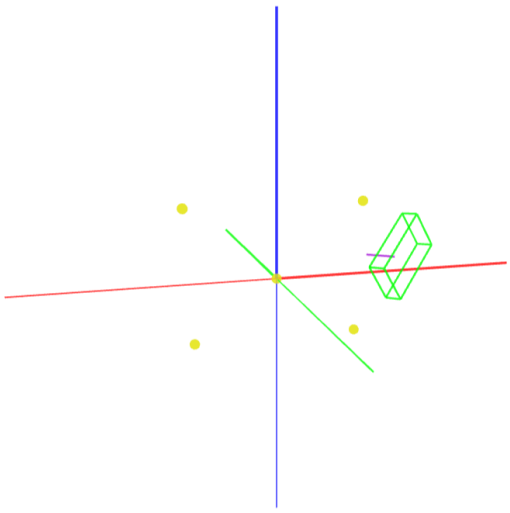
Three tracking modes are available:

- Sixdof: Track the full 6 degrees of freedom relative to the map. This is intended for applications where the drone is within 6m of the map.
- Relative Beacon: Track the position of a beacon relative to the sensor. This is intended to guide a drone to a landing pad with a high powered beacon from 40m down to the final 2m.
- Relative Angle: Track the angle of a beacon relative to the sensor. The intent of this mode is similar to Relative Beacon, however it is designed for tracking from very far ranges down to the final descent.

The SdsDroneSdk provides an easy way to run these two tracking modes, and seamlessly transition between them.

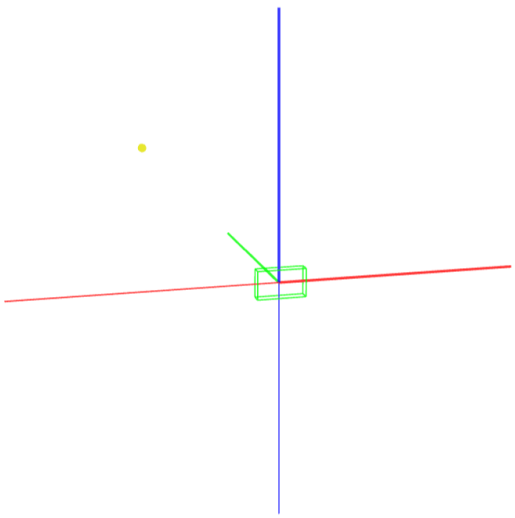
In addition to tracking, a method is provided to enable the user to do gyroscope calibration. During gyroscope calibration, the sensor must be still. In many applications, it is desirable to initialize the system while already in motion. Therefore, gyroscope calibration can be done in advance to the mission and the calibrated values can be fed in at run time.

1.4 Tracking mode Sixdof



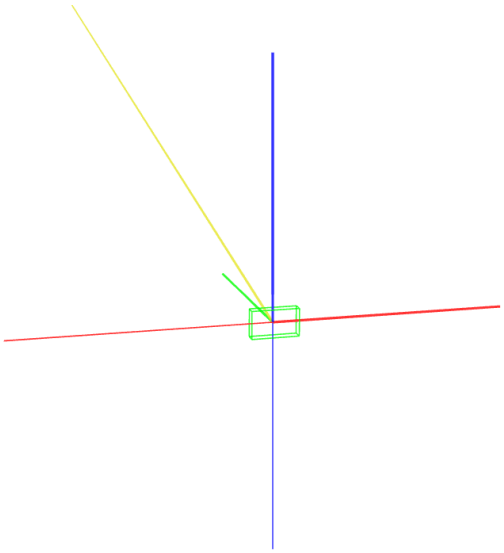
In this mode, the postion of the sensor is tracked relative to the map. The map is made up of multiple IR beacons that flash unique ids, multiple beacons are driven by a single Basalt board. The user must place these beacons in a rigid formation and provide the X, Y, Z location of each beacon. The unique id of each beacon is provided on each LED individually. The user can define the coordinate system of the map in any way they choose, however it must use a right handed coordinate system. When designing a map, it is important to ensure the lights are not co-linear. Keep in mind the desired area of operation, try to design the map so that there will always be at least 3 beacons in the field of view of the sensor.

1.5 Tracking mode Relative Beacon



In this mode, the position of a high-powered beacon is tracked relative to the coordinate system of the sensor. The goal of this mode is to be able to guide a drone from a high elevation (40m) down to a landing pad. This mode does not provide any orientation information. It is important to note that the distance estimate (y-direction) of the beacon is not accurate when far from the beacon. However, this mode does provide the ability to derive the direction of the beacon in order to align for the descent. When the beacon is closer ($\sim$2m) to the sensor, the distance estimate will be more accurate. The reason for this inaccuracy at large distances is due to the very small baseline on the sensor.

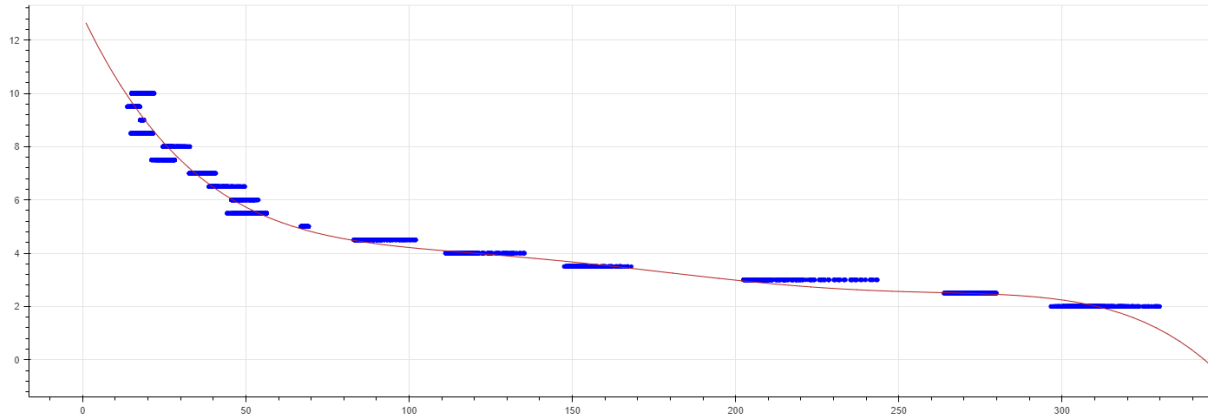
1.6 Tracking mode Relative Angle



In this mode the angle of a beacon is tracked relative to the coordinate system of the sensor. The goal of this mode is to be able to guide a drone from a high elevation ($>$ 40m) all the way down to a landing pad. This mode does not provide any orientation information of the senor. Additionally, due to the far distance of the beacon the baseline on the sensor is insufficent to calculate the distance directly. In this mode the intensity of the beacon and the peak width in pixels are reported in order to get a rough estimate of the distance of the beacon. In a configuration where full Sixdof landing is desired, the rough distance estimate is can be used as a signal that the sensor is close enough to transition to Sixdof mode.

1.7 Switching between Tracking Modes

Switching between tracking modes can be done almost instantly via the setTrackingMode function. In many drone applications there is a need for the drone to be guided to the landing pad from a high elevation, as well as do a precision landing. In these cases a single high-powered LED can be used along side a map of multiple small LEDs. When the drone is at a high elevation it is guided to the landing target by using ¨Relative Angles¨ mode, then when it gets sufficiently close ($\sim$6m) it switches to ¨Sixdof¨ mode for the final high-precision landing. The combination of ¨Relative Angles¨ and ¨Sixdof¨ modes is a highly effective way of achieving a targeted landing from a high elevation, however the key component to this solution is the ability to decided when to switch between the tracking modes. In the ¨Relative Angles¨ tracking modes the distance to the LED is unknown, however the intensity of the LED and the width in pixel space are provided. From this information a rough estimate of the distance to the landing target can be calculated. Shown below are the results of an experiment where the intensity of the LED was measured at known distances, y-axis is distance in meters and x-axis is the intensity, a polynomial trend line is shown in red:



A rough estimate of the distance between the sensor and the landing target can be obtained from the polynomial fit shown above. Additionally, the width in pixels of the LED can help to enhance the distance estimate when the sensor is close to the light. Note that this relationship is only valid for the beacon that was used in the experiment, in this case a 100 watt beacon, and the intensity to distance relationship can change slightly with lighting conditions. Many drones are equipped with other sensors such as range finders and barometers, these sensors can assist with estimating the distance to the landing target.

1.8 Change Log

Version	Date	Notes
1.5.3	29/02/2024	Added NetworkAdvancedConfig, getFirmwareVersion, skip_gyro_calibration is now defaulted to true
1.5.2	20/02/2024	Added expected_beacon_ids to ConfigRelativeBeacon and ConfigRelativeAngle, also changed shutter backed, deprecated setShutter function
1.5.1	01/02/2024	Added skip_gyro_calibration flag to Config6Dof
1.5.0	20/12/2023	Added setShutter feature, current shutter callback, put shutter_settings into each of the config objects
1.4.1	27/11/2023	Added swapMap feature
1.4.0	05/11/2023	Added Relative Angle feature
1.3.0	17/10/2023	Added health monitoring feature in the form of Heartbeat callbacks
1.2.1	06/09/2023	Exposed ¨No Gyro¨ mode as a parameter in Config6Dof. Made changes to support multiple sensors each with its own DroneTrackingManager
1.2.0	22/08/2023	Changed API so that PoseRelativeBeacons that happen at the same time are bundled together
1.1.2	14/08/2023	Updated RelativeBeacon logic, parameter rel_lights_cooldown_ms is no longer needed, instead we have the parameter field_of_view_cutoff_deg
1.1.1	14/08/2023	Statically link GCC libraries, add MSVC build to the release
1.1.0-beta	24/07/2023	Changed header, included gyro calibration, field of view report features and data logging for replay
1.0.0-beta	01/05/2023	inital beta version

1.9 Trouble Shooting Guide

1.9.1 Missing Libraries

Make sure to always copy the two shared libraries (.dll on Windows and .so on Linux) that are provided in the release folder for your platform into the the directory with your program. Alternatively, you can statically link the

shared libraries to your program. Also on Windows you have the option of adding the shared libraries to your system PATH.

1.9.2 Ping Failed

Errors relating to ping failing mean that the host computer was unable to connect to the Sixdof sensor. Try the following steps:

1. Ensure the Sixdof sensor is connected to power and the RJ45 ethernet cable is connected to your computer
2. Look at the red indicator lights that are exposed on the top of the senor casing, the lights should be flashing, if the lights are constant please contact the Sixdof team
3. Ensure your NetworkConfig parameters are correct, this is the sensor board number and the network name that the sensor is connected to
4. Your computers ARP table is used to connect to the sensor, try clearing the ARP table and trying again. The ARP table can be cleared vai a command console with administrator privileges, use the command ¨arp -d¨

1.9.3 Firewall

If SdsDroneSdk is reporting errors related to communication issues, it is recommended to turn off your firewall.

1.9.4 Underprocessing

If you are getting the ¨Warning¨ status message that says ¨Not receiving enough buffers from board¨ that means that your program is not getting enough CPU time. On Windows systems this is usually due to a spontaneous virus scan, you can temporarily turn off virus scans from in your ¨Windows Security¨ settings.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Sds::DroneSDK::Beacon Struct Reference

A structure representing a beacon with 3D coordinates and an id.

`#include <SdsDroneSdk.h>`

Public Attributes

- double **x**
  x-coordinate in meters.
- double **y**
  y-coordinate in meters.
- double **z**
  z-coordinate in meters.
- uint16_t **id**
  unique id.

4.1.1 Detailed Description

A structure representing a beacon with 3D coordinates and an id.

This structure contains the 3D coordinates (x, y, z) of a beacon in meters and its unique id. Both the small beacons used for 6dof tracking, and the large beacon used for Relative Light tracking are represented with the Beacon struct. Each beacon provided is labeled with its unique id.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.2 Sds::DroneSDK::Config6Dof Struct Reference

A structure representing configuration settings for the Sixdof tracking mode.

`#include <SdsDroneSdk.h>`

Public Attributes

- std::vector< Beacon > **map**

  vector of beacons for Sixdof tracking. (Required)
- std::vector< GyroOffset > **gyroCalibration**

  vector of previously recorded gyro calibration data. (Optional)
- bool **skip_gyro_calibration** { true }

  set to true to skip gyro calibration, this will speed up the initialization process.
- ShutterSettings **shutter_settings** { getShutterSettingsAuto() }
- bool **no_gyro_mode** { false }

  flag to disable gyro sensor, this mode should only be used in select cases, please contact the Sixdof team for guidance. (Optional)
- std::string **sixdof_dump_path** { "" }

  full path to file to log data to, default value of empty string will not log data. (Optional)

4.2.1 Detailed Description

A structure representing configuration settings for the Sixdof tracking mode.

This structure contains configuration settings for the Sixdof tracking mode. The only required data is the map, other fields can be left to defaults.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.3 Sds::DroneSDK::ConfigGyroCalibration Struct Reference

A structure representing configuration settings for gyro calibration.

`#include <SdsDroneSdk.h>`

Public Attributes

- int **duration_seconds** { 30 }

  duration of gyro calibration capture in seconds.

4.3.1 Detailed Description

A structure representing configuration settings for gyro calibration.

This structure contains configuration settings for gyro calibration, all members can be left to defaults.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.4 Sds::DroneSDK::ConfigRelativeAngle Struct Reference

A structure representing configuration settings for RelativeAngle tracking mode.

`#include <SdsDroneSdk.h>`

Public Attributes

- ShutterSettings **shutter_settings** { getShutterSettingsAuto() }
- std::vector< uint16_t > **expected_beacon_ids**

  expected beacon ids for this mode, typically a large beacon. This information will be used to optimize the shutter for the beacons you are looking for. (Optional)
- bool **matching_mode_single_beacon** { false }

  set the matching mode to match based on the assumption that there is only one beacon in the feild of view. It is possible to increase the tracking range with this mode.
- float **field_of_view_cutoff_deg** { 45.0 }

  field of view cutoff for RelativeAngle mode, in degrees. (Optional)
- std::string **rel_angles_dump_path** { "" }

  full path to file to log data to, default value of empty string will not log data. (Optional)

4.4.1 Detailed Description

A structure representing configuration settings for RelativeAngle tracking mode.

This structure contains configuration settings for RelativeAngle tracking mode. All members can be left to default values.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.5 Sds::DroneSDK::ConfigRelativeBeacon Struct Reference

A structure representing configuration settings for RelativeBeacon tracking mode.

`#include <SdsDroneSdk.h>`

Public Attributes

- ShutterSettings **shutter_settings** { getShutterSettingsAuto() }
- std::vector< uint16_t > **expected_beacon_ids**

  expected beacon ids for this mode, typically a large beacon. This information will be used to optimize the shutter for the beacons you are looking for. (Optional)
- float **field_of_view_cutoff_deg** { 45.0 }

  field of view cutoff for RelativeBeacon mode, in degrees. (Optional)
- std::string **rel_lights_dump_path** { "" }

  full path to file to log data to, default value of empty string will not log data. (Optional)

4.5.1 Detailed Description

A structure representing configuration settings for RelativeBeacon tracking mode.

This structure contains configuration settings for RelativeBeacon tracking mode. All members can be left to default values.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.6 Sds::DroneSDK::CurrentShutter Struct Reference

A structure representing the current shutter value.

`#include <SdsDroneSdk.h>`

Public Attributes

- uint8_t **shutter_value**
  current shutter value

4.6.1 Detailed Description

A structure representing the current shutter value.

This structure is output via a callback, and indicates the current shutter value. Shutter values are between 0 and 16, where 0 is the shortest shutter and 16 is the longest.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.7 Sds::DroneSDK::DroneTrackingManager Class Reference

Class for managing the Sixdof tracking system.

`#include <SdsDroneSdk.h>`

Public Member Functions

- virtual bool initializeNetwork (const NetworkConfig &)=0

  Initialize the network.
- virtual bool initializeNetwork (const NetworkAdvancedConfig &)=0

  Initialize the network.
- virtual bool initialize6Dof (const Config6Dof &)=0

  Initialize Sixdof tracking mode.
- virtual bool initializeRelativeBeacon (const ConfigRelativeBeacon &)=0

  Initialize RelativeBeacon tracking mode.
- virtual bool initializeRelativeAngle (const ConfigRelativeAngle &)=0

  Initialize RelativeAngle tracking mode.
- virtual void setTrackingMode (const TrackingMode &)=0

  Set the tracking mode.
- virtual Version **getFirmwareVersion** ()=0

  Gets the firmware version of the sensor.
- virtual std::vector< GyroOffset > getGyroCalibration (const ConfigGyroCalibration &)=0

  Do gyro calibration and retrieve the results.
- virtual Pose6Dof **getPose6Dof** ()=0

  get most recent Pose6Dof.
- virtual PoseRelativeBeaconCollection **getPoseRelativeBeaconCollection** ()=0

  get most recent PoseRelativeBeaconCollection.
- virtual RelativeAngleCollection **getRelativeAngleCollection** ()=0

  get most recent RelativeAngle.
- virtual StatusMessage **getStatusMessage** ()=0

  get first StatusMessage that was called but not received yet.
- virtual FieldOfViewReport **getFieldOfViewReport** ()=0

  get most recent field of view report.
- virtual CurrentShutter **getCurrentShutter** ()=0

  get most recent shutter value.
- virtual CallbackHandle **registerPose6DofCallback** (const std::function< void(Pose6Dof)> &)=0

  register a function callback that takes Pose6Dof as an input.
- virtual CallbackHandle **registerPoseRelativeBeaconCallback** (const std::function< void(PoseRelativeBeaconCollection)> &)=0

  register a function callback that takes PoseRelativeBeaconCollection as an input.
- virtual CallbackHandle **registerRelativeAngleCallback** (const std::function< void(RelativeAngleCollection)> &)=0

  register a function callback that takes RelativeAngleCollection as an input.
- virtual CallbackHandle **registerMessageCallback** (const std::function< void(StatusMessage)> &)=0

  register a function callback that takes StatusMessage as an input.
- virtual CallbackHandle **registerFieldOfViewReportCallback** (const std::function< void(FieldOfViewReport)> &)=0

  register a function callback that takes FieldOfViewReport as an input.
- virtual CallbackHandle **registerHeartbeatCallback** (const std::function< void(Heartbeat)> &)=0

  register a function callback that takes Heartbeat as an input.
- virtual CallbackHandle **registerCurrentShutterCallback** (const std::function< void(CurrentShutter)> &)=0

  register a function callback that takes a CurrentShutter as an input.
- virtual void **removePose6DofCallback** (const CallbackHandle &)=0

  < remove a Pose6Dof function callback.
- virtual void **removePoseRelativeBeaconCallback** (const CallbackHandle &)=0

  < remove a PoseRelativeBeacon function callback.

- virtual void **removeRelativeAngleCallback** (const CallbackHandle &)=0

  < *remove a RelativeAngle function callback.*
- virtual void **removeMessageCallback** (const CallbackHandle &)=0

  < *remove a StatusMessage function callback.*
- virtual void **removeFieldOfViewCallback** (const CallbackHandle &)=0

  < *remove a FieldOfViewReport function callback.*
- virtual void **removeHeartbeatCallback** (const CallbackHandle &)=0

  < *remove a Heartbeat function callback.*
- virtual void **removeCurrentShutterCallback** (const CallbackHandle &)=0

  < *remove a Current Shutter function callback.*
- virtual void swapMap (const std::vector< Beacon > &map)=0

  Dynamically swap the map.

4.7.1 Detailed Description

Class for managing the Sixdof tracking system.

This class provides an interface for managing drone tracking. The correct order of function calls is first initializeNetwork, then initialize6dof, initializeRelativeBeacon, initializeRelativeAngle or multiple.

Finally you can start the tracking with the setTrackingMode function.

There are two ways to get tracking data and status messages out of the DroneTrackingManager:
1) Get functions. these will return the most recent data point. However the getStatusMessage function will return all messages in the order they appeared.
2) Function callbacks. The callbacks provided will get called each time a new data point is provided.

It is recommended to use function callbacks for status messages to ensure all messages are received immediately. It is also recommended to register for status messages before initializing the network, this way we can get all messages from the initialization stage. Each function callback is run on its own thread and it only gets updated once it has finished processing its current data point. This means that long running function callbacks may miss data points that got overshadowed by new data, this is by design.

The DroneTrackingManager also allows for the client to do gyro calibration before flight, see the getGyroCalibration() function.

4.7.2 Member Function Documentation

4.7.2.1 getGyroCalibration()

```
virtual std::vector< GyroOffset > Sds::DroneSDK::DroneTrackingManager::getGyroCalibration (
            const ConfigGyroCalibration &  )  [pure virtual]
```

Do gyro calibration and retrieve the results.

This function retrieves gyro calibration data for the sensor. This data should be saved and used at a later time by passing it into the gyroCalibration member for Config6Dof

Parameters

ConfigGyroCalibration	The configuration for the gyro calibration procedure.

Returns

A vector of GyroOffset objects representing the gyro calibration data.

4.7.2.2 initialize6Dof()

```
virtual bool Sds::DroneSDK::DroneTrackingManager::initialize6Dof (
            const Config6Dof &  )  [pure virtual]
```

Initialize Sixdof tracking mode.

This function initializes the Sixdof tracking mode. If you are not using the Sixdof tracking mode this is unnecessary.

Parameters

Config6Dof	The configuration for the Sixdof tracking mode.

Returns

True if Sixdof tracking is successfully initialized, false otherwise.

4.7.2.3 initializeNetwork() [1/2]

```
virtual bool Sds::DroneSDK::DroneTrackingManager::initializeNetwork (
            const NetworkAdvancedConfig &  )  [pure virtual]
```

Initialize the network.

This function initializes the network configuration for drone tracking.

Parameters

NetworkAdvancedConfig	The network configuration.

Returns

True if the network is successfully initialized, false otherwise.

4.7.2.4 initializeNetwork() [2/2]

```
virtual bool Sds::DroneSDK::DroneTrackingManager::initializeNetwork (
            const NetworkConfig &  )  [pure virtual]
```

Initialize the network.

This function initializes the network configuration for drone tracking.

Parameters

NetworkConfig	The network configuration.

Returns

True if the network is successfully initialized, false otherwise.

4.7.2.5 initializeRelativeAngle()

```
virtual bool Sds::DroneSDK::DroneTrackingManager::initializeRelativeAngle (
            const ConfigRelativeAngle &  )  [pure virtual]
```

Initialize RelativeAngle tracking mode.

This function initializes the RelativeAngle tracking mode. If you are not using the RelativeAngle tracking mode this is unnecessary.

Parameters

ConfigRelativeAngle	The configuration for RelativeAngle tracking mode.

Returns

True if RelativeAngle tracking is successfully initialized, false otherwise.

4.7.2.6 initializeRelativeBeacon()

```
virtual bool Sds::DroneSDK::DroneTrackingManager::initializeRelativeBeacon (
            const ConfigRelativeBeacon &  )  [pure virtual]
```

Initialize RelativeBeacon tracking mode.

This function initializes the RelativeBeacon tracking mode. If you are not using the RelativeBeacon tracking mode this is unnecessary.

Parameters

ConfigRelativeBeacon	The configuration for RelativeBeacon tracking mode.

Returns

True if RelativeBeacon tracking is successfully initialized, false otherwise.

4.7.2.7 setTrackingMode()

```
virtual void Sds::DroneSDK::DroneTrackingManager::setTrackingMode (
            const TrackingMode &  )  [pure virtual]
```

Set the tracking mode.

Parameters

TrackingMode	The desired TrackingMode.

4.7.2.8 swapMap()

```
virtual void Sds::DroneSDK::DroneTrackingManager::swapMap (
            const std::vector< Beacon > & map )  [pure virtual]
```

Dynamically swap the map.

This enables the user to swap the map during execution. It can only be called after initialize6Dof has already been called.

Parameters

map	The new beacon positions.

The documentation for this class was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.8 Sds::DroneSDK::FieldOfViewReport Struct Reference

A report of the beacons in the field of view.

```
#include <SdsDroneSdk.h>
```

Public Attributes

- std::map< uint16_t, int > **seenBeacons**

  Map of beacon ids to the number of optical sensors that the beacon was seen on.

4.8.1 Detailed Description

A report of the beacons in the field of view.

This structure contains information about which beacons are in the field of view of the sensor, and if they are seen on 1, 2 or 3 optical sensors.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.9 Sds::DroneSDK::GyroOffset Struct Reference

A structure representing a single gyro calibration offset for a specific sensor.

`#include <SdsDroneSdk.h>`

Public Attributes

- uint16_t **sensorId** { 0 }
  sensor id.
- std::string **datetime** { "" }
  date time in Y-m-d H:M:S format.
- int8_t **temperature** { 0 }
  temperature in degrees. This is the internal temperature of board, not the temperature of the external environment.
- int16_t **gyro_x** { 0 }
  bias in the x-coordinate of the IMU. Units are in gyro units.
- int16_t **gyro_y** { 0 }
  bias in the y-coordinate of the IMU. Units are in gyro units.
- int16_t **gyro_z** { 0 }
  bias in the z-coordinate of the IMU. Units are in gyro units.
- uint16_t **standard_dev** { 0 }
  standard deviation of IMU noise. Units are in gyro units.

4.9.1 Detailed Description

A structure representing a single gyro calibration offset for a specific sensor.

This structure contains information about gyro calibration offset for a specific sensor.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.10 Sds::DroneSDK::Heartbeat Struct Reference

A structure representing health data of the SDK.

`#include <SdsDroneSdk.h>`

Public Attributes

- bool **sensor_communication_ok** { false }
  status of communication with the sensor.
- TrackingMode **current_tracking_mode** { TrackingMode::TrackingOFF }
  current tracking mode.
- TrackingModeState **current_tracking_mode_state** { TrackingModeState::Initializing }
  state of the current tracking mode.

4.10.1 Detailed Description

A structure representing health data of the SDK.

This structure is output via a callback every second. This enables the user to quickly identify that the SDK is still running. Additionally, the Heartbeat structure contains information regarding the state of communications with the sensor and the tracking algorithm.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.11 Sds::DroneSDK::NetworkAdvancedConfig Struct Reference

A structure representing advanced network configuration for the Sixdof sensor.

```
#include <SdsDroneSdk.h>
```

Public Attributes

- uint16_t **sensor_id**

  sensor id.
- std::string **sensor_ip**

  desired sensor ip address.
- uint16_t **udp_read_port** { 0 }

  udp port to read from the sensor, to be auto-assigned a port leave as 0. (Optional)

4.11.1 Detailed Description

A structure representing advanced network configuration for the Sixdof sensor.

This structure contains the necessary information in order to connect to a Sixdof sensor. This is different to the NetworkConfig object that only needs the sensor id and the network name. NetworkAdvancedConfig is for cases where the user will add the entry to the ARP table themselves. Typically this is used on system where the basic network configuration will fail, for example in situations where the computer operating system is not in English.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.12 Sds::DroneSDK::NetworkConfig Struct Reference

A structure representing network configuration for the Sixdof sensor.

```
#include <SdsDroneSdk.h>
```

Public Attributes

- uint16_t **sensor_id**

  sensor id.
- std::string **sensor_network_name**

  name of the network the sensor is connected to.
- uint16_t **udp_read_port** { 0 }

  udp port to read from the sensor, to be auto-assigned a port leave as 0. (Optional)

4.12.1 Detailed Description

A structure representing network configuration for the Sixdof sensor.

This structure contains the necessary information in order to connect to a Sixdof sensor.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.13 Sds::DroneSDK::Pose6Dof Struct Reference

A structure representing a 6dof pose.

`#include <SdsDroneSdk.h>`

Public Attributes

- bool **valid** { false }

  flag indicating if the pose is valid. This can be False when calling the getPose6Dof function before a valid pose is obtained.
- double **x**

  x-coordinate in meters.
- double **y**

  y-coordinate in meters.
- double **z**

  z-coordinate in meters.
- double **qw**

  quaternion w component.
- double **qx**

  quaternion x component.
- double **qy**

  quaternion y component.
- double **qz**

  quaternion x component.
- float **var_x**
- float **var_y**
- float **var_z**
- float **var_h**
- float **var_p**
- float **var_r**

4.13.1 Detailed Description

A structure representing a 6dof pose.

This structure contains 6dof (6 degree of freedom) pose information, specifically the pose of the Sixdof sensor with respect to the map. The pose is represented by position (x, y, z) in meters, and orientation as a quaternion (qx, qw, qy, qz). Pose accuracy is represented as variance of the position (var_x, var_y, var_z) in meters squared, and the variance of the orientation (var_h, var_p, var_r) in radians squared.

This is the return type when in Sixdof tracking mode.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.14 Sds::DroneSDK::PoseRelativeBeacon Struct Reference

A structure representing the relative pose of a beacon.

`#include <SdsDroneSdk.h>`

Public Attributes

- double **x**

  x-coordinate in meters.
- double **y**

  y-coordinate in meters.
- double **z**

  z-coordinate in meters.
- uint16_t **id**

  unique id of the beacon.

4.14.1 Detailed Description

A structure representing the relative pose of a beacon.

This structure contains the 3D pose of a single beacon with respect to the sensor. Pose is represented as 3D position (x, y, z) in meters.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.15 Sds::DroneSDK::RelativeAngle Struct Reference

A structure representing the relative angle of a beacon.

`#include <SdsDroneSdk.h>`

Public Attributes

- double **x_angle**

  angular offset in the x-axis in radians.
- double **z_angle**

  angular offset in the z-axis in radians.
- uint16_t **id**

  unique id of the beacon.
- double **intensity**

  light intensity of the beacon, this can be used to roughly indicate distance.
- double **width**

  width of the detected peak in pixels, also useful for indicating distance.

4.15.1 Detailed Description

A structure representing the relative angle of a beacon.

This structure contains the angle of a single beacon with respect to the sensor. Where x_angle is the angular offset along the x-axis, and z_angle is the angular offset along the z-axis.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.16 Sds::DroneSDK::ShutterSettings Struct Reference

Specifies the shutter settings.

`#include <SdsDroneSdk.h>`

Public Attributes

- uint8_t **min_shutter**
- uint8_t **max_shutter**

4.16.1 Detailed Description

Specifies the shutter settings.

This struct is used to control the sensors shutter. There are three shutter modes:

- Auto shutter - the shutter setting is detected automatically by the sensor.
- Fixed shutter - the shutter is fixed at a specific value and will not be changed by the sensor.
- Range shutter - the shutter is set automatically by the sensor but it will be fixed to a specific range.

For typical usage Auto shutter is recommended, Fixed and Range shutter modes are used only in cases where there are abnormal optical conditions. For example when the sun is directly in the field of view of the sensor. The ShutterSettings struct should be instantiated by one of the following functions: getShutterSettingsAuto, getShutter↩SettingsFixed, or getShutterSettingsRange.

Shutter values are between 0 and 16. Additionally the value 63 is used to set the sensor into Auto shutter mode.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.17 Sds::DroneSDK::StatusMessage Struct Reference

A structure representing a status message.

`#include <SdsDroneSdk.h>`

Public Attributes

- Severity **severity**
  severity level of the status message.
- CallingLayer **layer**
  calling layer for the status message.
- uint8_t **event_code**
  event code of the status message.
- std::string **message**
  verbose status message.

4.17.1 Detailed Description

A structure representing a status message.

This structure contains information about a status message, including its severity, calling layer, event code, and the content of the message.

The documentation for this struct was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

4.18 Sds::DroneSDK::Version Class Reference

A class representing the version of either the shared library, or the sensor firmware.

`#include <SdsDroneSdk.h>`

Public Member Functions

- **Version** (uint8_t major, uint8_t minor, uint8_t patch)
- uint8_t getMajor () const
  Get the major component of the version.
- uint8_t getMinor () const
  Get the minor component of the version.
- uint8_t getPatch () const
  Get the patch component of the version.
- bool isEqual (uint8_t major, uint8_t minor, uint8_t patch) const
  Check if the version is equal to the specified components.
- bool isAtLeast (uint8_t major, uint8_t minor, uint8_t patch) const
  Check if the version is at least the specified version.
- std::string getString () const
  Get a string representation of the version.

4.18.1 Detailed Description

A class representing the version of either the shared library, or the sensor firmware.

This class provides utility functions around a system version (major, minor, patch), either the version of the shared library, or the sensor firmware version.

4.18.2 Member Function Documentation

4.18.2.1 getMajor()

```
uint8_t Sds::DroneSDK::Version::getMajor ( ) const
```

Get the major component of the version.

Returns

The major component of the version as a uint8_t.

4.18.2.2 getMinor()

```
uint8_t Sds::DroneSDK::Version::getMinor ( ) const
```

Get the minor component of the version.

Returns

The minor component of the version as a uint8_t.

4.18.2.3 getPatch()

```
uint8_t Sds::DroneSDK::Version::getPatch ( ) const
```

Get the patch component of the version.

Returns

The patch component of the version as a uint8_t.

4.18.2.4 getString()

```
std::string Sds::DroneSDK::Version::getString ( ) const
```

Get a string representation of the version.

Get a string representation of the version.

Returns

A string representing the version.

4.18.2.5 isAtLeast()

```
bool Sds::DroneSDK::Version::isAtLeast (
            uint8_t major,
            uint8_t minor,
            uint8_t patch ) const
```

Check if the version is at least the specified version.

Parameters

major	The major component to compare.
minor	The minor component to compare.
patch	The patch component to compare.

Returns

True if the version is at least the specified version, false otherwise.

4.18.2.6 isEqual()

```
bool Sds::DroneSDK::Version::isEqual (
            uint8_t major,
            uint8_t minor,
            uint8_t patch ) const
```

Check if the version is equal to the specified components.

Parameters

major	The major component to compare.
minor	The minor component to compare.
patch	The patch component to compare.

Returns

True if the version is equal to the specified components, false otherwise.

The documentation for this class was generated from the following file:

- C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h

Chapter 5

File Documentation

5.1 C:/sixdof/kiwi/src/drone_sdk/SdsDroneSdk.h File Reference

Header file containing declarations for the SDS DroneSDK library.

```
#include <stdint.h>
#include <map>
#include <vector>
#include <string>
#include <memory>
#include <functional>
```

Classes

- class Sds::DroneSDK::Version

  A class representing the version of either the shared library, or the sensor firmware.
- struct Sds::DroneSDK::NetworkConfig

  A structure representing network configuration for the Sixdof sensor.
- struct Sds::DroneSDK::NetworkAdvancedConfig

  A structure representing advanced network configuration for the Sixdof sensor.
- struct Sds::DroneSDK::Beacon

  A structure representing a beacon with 3D coordinates and an id.
- struct Sds::DroneSDK::Pose6Dof

  A structure representing a 6dof pose.
- struct Sds::DroneSDK::PoseRelativeBeacon

  A structure representing the relative pose of a beacon.
- struct Sds::DroneSDK::RelativeAngle

  A structure representing the relative angle of a beacon.
- struct Sds::DroneSDK::StatusMessage

  A structure representing a status message.
- struct Sds::DroneSDK::Heartbeat

  A structure representing health data of the SDK.
- struct Sds::DroneSDK::CurrentShutter

  A structure representing the current shutter value.
- struct Sds::DroneSDK::GyroOffset

A structure representing a single gyro calibration offset for a specific sensor.

- struct Sds::DroneSDK::FieldOfViewReport

  A report of the beacons in the field of view.
- struct Sds::DroneSDK::ShutterSettings

  Specifies the shutter settings.
- struct Sds::DroneSDK::Config6Dof

  A structure representing configuration settings for the Sixdof tracking mode.
- struct Sds::DroneSDK::ConfigRelativeBeacon

  A structure representing configuration settings for RelativeBeacon tracking mode.
- struct Sds::DroneSDK::ConfigRelativeAngle

  A structure representing configuration settings for RelativeAngle tracking mode.
- struct Sds::DroneSDK::ConfigGyroCalibration

  A structure representing configuration settings for gyro calibration.
- class Sds::DroneSDK::DroneTrackingManager

  Class for managing the Sixdof tracking system.

Macros

- #define **SDS_API** __attribute__((visibility("default")))
- #define **DEPRECATED**(msg)

Typedefs

- typedef std::vector< PoseRelativeBeacon > Sds::DroneSDK::PoseRelativeBeaconCollection

  A structure representing a collection of PoseRelativeBeacon.
- typedef std::vector< RelativeAngle > Sds::DroneSDK::RelativeAngleCollection

  A structure representing a collection of RelativeAngle.
- typedef uint32_t Sds::DroneSDK::CallbackHandle

  A handle representing a registered callback function.

Enumerations

- enum class Sds::DroneSDK::Severity { Exception = 1 , Warning , Informative }

  An enum representing the severity level of a status message.
- enum class Sds::DroneSDK::CallingLayer {
  SdsDroneSdk = 1 , Algorithm6Dof , AlgorithmRelativeBeacon , SdsCommLib ,
  NetworkConnection , GyroCalibration , AlgorithmRelativeAngle }

  An enum representing the calling layer for a status message.
- enum class Sds::DroneSDK::DroneSdkEventCode {
  UnexpectedAlgoInstance , TrackingModeNotInitalized , NetworkNotInitalized , GyroCalibrationRejected ,
  CommunicationLoopNotClosed }

  An enum representing event codes specific to the SdsDroneSdk calling layer.
- enum class Sds::DroneSDK::TrackingMode { TrackingOFF , RelativeBeacon , Sixdof , RelativeAngle }

  An enum representing different tracking modes.
- enum class Sds::DroneSDK::TrackingModeState { Initializing , Running , Error }

  An enum representing different states of the current tracking mode.

Functions

- SDS_API Version Sds::DroneSDK::getVersion ()

  Get the version of the SDS DroneSDK shared library.
- SDS_API std::string Sds::DroneSDK::to_string (TrackingMode mode)

  Convert a TrackingMode enum value to a string.
- SDS_API std::string Sds::DroneSDK::to_string (TrackingModeState state)

  Convert a TrackingModeState enum value to a string.
- SDS_API ShutterSettings Sds::DroneSDK::getShutterSettingsAuto ()

  Get shutter settings for Auto shutter mode.
- SDS_API ShutterSettings Sds::DroneSDK::getShutterSettingsFixed (uint8_t shutter)

  Get shutter settings for Fixed shutter mode.
- SDS_API ShutterSettings Sds::DroneSDK::getShutterSettingsRange (uint8_t min_shutter, uint8_t max_↩ shutter)

  Get shutter settings for Range shutter mode.
- SDS_API std::unique_ptr< DroneTrackingManager > Sds::DroneSDK::createDroneTrackingManager ()

  Create an instance of DroneTrackingManager.

5.1.1 Detailed Description

Header file containing declarations for the SDS DroneSDK library.

This file contains the declarations for classes and data structures related to the SDS DroneSDK library. It provides functionality for managing drone tracking, including retrieving pose information, managing the Sixdof sensor, and registering callbacks for various events.

5.1.2 Typedef Documentation

5.1.2.1 CallbackHandle

`Sds::DroneSDK::CallbackHandle`

A handle representing a registered callback function.

This typedef represents a callback handle used to identify registered callback functions. It is used to remove a specific callback.

5.1.2.2 PoseRelativeBeaconCollection

`Sds::DroneSDK::PoseRelativeBeaconCollection`

A structure representing a collection of PoseRelativeBeacon.

This structure contains multiple PoseRelativeBeacon estimates. All PoseRelativeBeacon estimates were calculated at the same and therefore bundled together.

This is the return type when in RelativeBeacon tracking mode.

5.1.2.3 RelativeAngleCollection

`Sds::DroneSDK::RelativeAngleCollection`

A structure representing a collection of RelativeAngle.

This structure contains multiple RelativeAngle estimates. All RelativeAngle estimates were calculated at the same and therefore bundled together.

This is the return type when in RelativeAngle tracking mode.

5.1.3 Enumeration Type Documentation

5.1.3.1 CallingLayer

`enum class Sds::DroneSDK::CallingLayer [strong]`

An enum representing the calling layer for a status message.

This enum defines the calling layers for status messages. Each layer indicates the source of a particular message.

Enumerator

SdsDroneSdk	Message from the top layer of the library. Generally indicates the library was used incorrectly.
Algorithm6Dof	Message from Sixdof tracking mode.
AlgorithmRelativeBeacon	Message from RelativeBeacon tracking mode.
SdsCommLib	Message from the UDP communication library, this is used to communicate with the sensor.
NetworkConnection	Message from the network connection stage.
GyroCalibration	Message from the gyro calibration feature.
AlgorithmRelativeAngle	Message from RelativeAngle tracking mode.

5.1.3.2 DroneSdkEventCode

`enum class Sds::DroneSDK::DroneSdkEventCode [strong]`

An enum representing event codes specific to the SdsDroneSdk calling layer.

This enum defines event codes that are specific to the SdsDroneSdk calling layer.

Enumerator

UnexpectedAlgoInstance	indicates internal error, contact Sixdof for tech support.
TrackingModeNotInitalized	indicates tracking mode was set before the corresponding initialization.
NetworkNotInitalized	indicates that the initializeNetwork function was not called.
GyroCalibrationRejected	indicates gyro calibration values were rejected.
CommunicationLoopNotClosed	communication loop is not closed, check your firewall

5.1.3.3 Severity

```
enum class Sds::DroneSDK::Severity  [strong]
```

An enum representing the severity level of a status message.

This enumeration defines severity levels for status messages.

Enumerator

Exception	Exception indicates that there was a error in operation.
Warning	Warning indicates that something went wrong but it is not critical.
Informative	Informative indicates general debugging information, these can be ignored.

5.1.3.4 TrackingMode

```
enum class Sds::DroneSDK::TrackingMode  [strong]
```

An enum representing different tracking modes.

This enum is used to set the tracking mode.

Enumerator

TrackingOFF	tracking is turned off. This is the default and when the DroneTrackingManager goes out of scope it is set to TrackingOFF automatically.
RelativeBeacon	tracking is set to RelativeBeacon mode.
Sixdof	tracking is set to Sixdof (6dof) mode.
RelativeAngle	tracking is set to RelativeAngle mode.

5.1.3.5 TrackingModeState

```
enum class Sds::DroneSDK::TrackingModeState  [strong]
```

An enum representing different states of the current tracking mode.

Enumerator

Initializing	tracking mode is currently initializing.
Running	tracking mode is currently running.
Error	tracking mode is currently in the error state.

5.1.4 Function Documentation

5.1.4.1 createDroneTrackingManager()

```
SDS_API std::unique_ptr< DroneTrackingManager > Sds::DroneSDK::createDroneTrackingManager ( )
```

Create an instance of DroneTrackingManager.

This function creates a new instance of the DroneTrackingManager. When the DroneTrackingManger goes out of scope it will stop the sensor and be cleaned up automatically.

Returns

A unique pointer to a new instance of DroneTrackingManager.

5.1.4.2 getShutterSettingsAuto()

```
SDS_API ShutterSettings Sds::DroneSDK::getShutterSettingsAuto ( )
```

Get shutter settings for Auto shutter mode.

Returns

ShutterSettings

5.1.4.3 getShutterSettingsFixed()

```
SDS_API ShutterSettings Sds::DroneSDK::getShutterSettingsFixed (
            uint8_t shutter )
```

Get shutter settings for Fixed shutter mode.

Parameters

shutter	Shutter value, between 0 and 16 inclusive.

Returns

ShutterSettings

5.1.4.4 getShutterSettingsRange()

```
SDS_API ShutterSettings Sds::DroneSDK::getShutterSettingsRange (
            uint8_t min_shutter,
            uint8_t max_shutter )
```

Get shutter settings for Range shutter mode.

Parameters

min_shutter	Min shutter value, between 0 and 16 inclusive.
max_shutter	Max shutter value, between 0 and 16 inclusive.

Returns

ShutterSettings

5.1.4.5 getVersion()

```
SDS_API Version Sds::DroneSDK::getVersion ( )
```

Get the version of the SDS DroneSDK shared library.

Returns

The Version object representing the library version.

5.1.4.6 to_string() [1/2]

```
SDS_API std::string Sds::DroneSDK::to_string (
            TrackingMode mode )
```

Convert a TrackingMode enum value to a string.

Parameters

mode	TrackingMode enum value.

Returns

string representation of the TrackingMode.

5.1.4.7 to_string() [2/2]

```
SDS_API std::string Sds::DroneSDK::to_string (
            TrackingModeState state )
```

Convert a TrackingModeState enum value to a string.

Parameters

state	TrackingModeState enum value.

Returns

string representation of the TrackingModeState.

5.2 SdsDroneSdk.h

Go to the documentation of this file.

```
00001 //Copyright © 2017-2024 Six Degrees Space Ltd.  All rights reserved.
00002 //Proprietary and Confidential.  Unauthorized use, disclosure or reproduction is strictly prohibited.
00003
00013 #pragma once
00014 #include <stdint.h>
00015 #include <map>
00016 #include <vector>
00017 #include <string>
00018 #include <memory>
00019 #include <functional>
00020
00021 #ifdef _WIN32
00022     // Windows platform
00023     #ifdef SDS_EXPORTS
00024         #define SDS_API __declspec(dllexport)
00025     #else
00026         #define SDS_API __declspec(dllimport)
00027     #endif
00028 #else
00029     // Non-Windows platform
00030     #define SDS_API __attribute__((visibility("default")))
00031 #endif
00032
00033 #if defined(__GNUC__) || defined(__clang__)
00034 #define DEPRECATED(msg) __attribute__((deprecated(msg)))
00035 #elif defined(_MSC_VER)
00036 #define DEPRECATED(msg) __declspec(deprecated(msg))
00037 #else
00038 #define DEPRECATED(msg)
00039 #endif
00040
00170 namespace Sds {
00171     namespace DroneSDK {
00172
00179         class SDS_API Version {
00180         public:
00181             Version(uint8_t major, uint8_t minor, uint8_t patch);
00182
00187             uint8_t getMajor() const;
00188
00193             uint8_t getMinor() const;
00194
00199             uint8_t getPatch() const;
00200
00209             bool isEqual(uint8_t major, uint8_t minor, uint8_t patch) const;
00210
00219             bool isAtLeast(uint8_t major, uint8_t minor, uint8_t patch) const;
00220
00227             std::string getString() const;
00228
00229         private:
00230             uint8_t _major, _minor, _patch;
00231         };
00232
00237         SDS_API Version getVersion();
00238
00245         struct SDS_API NetworkConfig {
00246             uint16_t sensor_id;
00247             std::string sensor_network_name;
00248             uint16_t udp_read_port { 0 };
00249         };
00250
00261         struct SDS_API NetworkAdvancedConfig {
00262             uint16_t sensor_id;
00263             std::string sensor_ip;
00264             uint16_t udp_read_port { 0 };
00265         };
00266
00275         struct SDS_API Beacon {
00276             double x;
00277             double y;
00278             double z;
00279             uint16_t id;
00280         };
00281
00292         struct SDS_API Pose6Dof {
00293             bool valid { false };
00294             double x;
00295             double y;
00296             double z;
00297             double qw;
00298             double qx;
00299             double qy;
00300             double qz;
00301             float var_x, var_y, var_z; // m^2
00302             float var_h, var_p, var_r; // rad^2
00303         };
```

```
00304
00312         struct SDS_API PoseRelativeBeacon {
00313             double x;
00314             double y;
00315             double z;
00316             uint16_t id;
00317         };
00318
00328         typedef std::vector<PoseRelativeBeacon> PoseRelativeBeaconCollection;
00329
00330
00338         struct SDS_API RelativeAngle {
00339             double x_angle;
00340             double z_angle;
00341             uint16_t id;
00342             double intensity;
00343             double width;
00344         };
00345
00355         typedef std::vector<RelativeAngle> RelativeAngleCollection;
00356
00363         enum class Severity {
00364             Exception = 1,
00365             Warning,
00366             Informative,
00367         };
00368
00376         enum class CallingLayer {
00377             SdsDroneSdk = 1,
00378             Algorithm6Dof,
00379             AlgorithmRelativeBeacon,
00380             SdsCommLib,
00381             NetworkConnection,
00382             GyroCalibration,
00383             AlgorithmRelativeAngle,
00384         };
00385
00392         enum class DroneSdkEventCode {
00393             UnexpectedAlgoInstance,
00394             TrackingModeNotInitalized,
00395             NetworkNotInitalized,
00396             GyroCalibrationRejected,
00397             CommunicationLoopNotClosed,
00398         };
00399
00407         struct SDS_API StatusMessage {
00408             Severity severity;
00409             CallingLayer layer;
00410             uint8_t event_code;
00411             std::string message;
00412         };
00413
00420         enum class TrackingMode {
00421             TrackingOFF,
00422             RelativeBeacon,
00423             Sixdof,
00424             RelativeAngle,
00425         };
00426
00433         SDS_API std::string to_string(TrackingMode mode);
00434
00439         enum class TrackingModeState {
00440             Initializing,
00441             Running,
00442             Error,
00443         };
00444
00451         SDS_API std::string to_string(TrackingModeState state);
00452
00460         struct SDS_API Heartbeat {
00461             bool sensor_communication_ok { false };
00462             TrackingMode current_tracking_mode { TrackingMode::TrackingOFF };
00463             TrackingModeState current_tracking_mode_state { TrackingModeState::Initializing };
00464         };
00465
00473         struct SDS_API CurrentShutter {
00474             uint8_t shutter_value;
00475         };
00476
00483         struct SDS_API GyroOffset {
00484             uint16_t sensorId { 0 };
00485             std::string datetime { "" };
00486             int8_t temperature { 0 };
00487             int16_t gyro_x { 0 };
00488             int16_t gyro_y { 0 };
00489             int16_t gyro_z { 0 };
00490             uint16_t standard_dev { 0 };
```

```
00491         };
00492
00499         struct SDS_API FieldOfViewReport {
00500             std::map<uint16_t, int> seenBeacons;
00501         };
00502
00519         struct SDS_API ShutterSettings {
00520             uint8_t min_shutter;
00521             uint8_t max_shutter;
00522         };
00523
00528         SDS_API ShutterSettings getShutterSettingsAuto();
00529
00535         SDS_API ShutterSettings getShutterSettingsFixed(uint8_t shutter);
00536
00543         SDS_API ShutterSettings getShutterSettingsRange(uint8_t min_shutter, uint8_t max_shutter);
00544
00552         struct SDS_API Config6Dof {
00553             std::vector<Beacon> map;
00554             std::vector<GyroOffset> gyroCalibration;
00555             bool skip_gyro_calibration { true };
00556             ShutterSettings shutter_settings { getShutterSettingsAuto() };
00557             bool no_gyro_mode { false };
00558             std::string sixdof_dump_path { "" };
00559         };
00560
00568         struct SDS_API ConfigRelativeBeacon {
00569             ShutterSettings shutter_settings { getShutterSettingsAuto() };
00570             std::vector<uint16_t> expected_beacon_ids;
00571             float field_of_view_cutoff_deg { 45.0 };
00572             std::string rel_lights_dump_path { "" };
00573         };
00574
00582         struct SDS_API ConfigRelativeAngle {
00583             ShutterSettings shutter_settings { getShutterSettingsAuto() };
00584             std::vector<uint16_t> expected_beacon_ids;
00585             bool matching_mode_single_beacon { false };
00586             float field_of_view_cutoff_deg { 45.0 };
00587             std::string rel_angles_dump_path { "" };
00588         };
00589
00597         struct SDS_API ConfigGyroCalibration {
00598             int duration_seconds { 30 };
00599         };
00600
00608         typedef uint32_t CallbackHandle;
00609
00632         class SDS_API DroneTrackingManager {
00633         public:
00634             virtual ~DroneTrackingManager() { }
00635
00644             virtual bool initializeNetwork(const NetworkConfig&) = 0;
00645
00654             virtual bool initializeNetwork(const NetworkAdvancedConfig&) = 0;
00655
00664             virtual bool initialize6Dof(const Config6Dof&) = 0;
00665
00674             virtual bool initializeRelativeBeacon(const ConfigRelativeBeacon&) = 0;
00675
00684             virtual bool initializeRelativeAngle(const ConfigRelativeAngle&) = 0;
00685
00691             virtual void setTrackingMode(const TrackingMode&) = 0;
00692
00696             virtual Version getFirmwareVersion() = 0;
00697
00707             virtual std::vector<GyroOffset> getGyroCalibration(const ConfigGyroCalibration&) = 0;
00708
00709             virtual Pose6Dof getPose6Dof() = 0;
00710
00711             virtual PoseRelativeBeaconCollection getPoseRelativeBeaconCollection() = 0;
00712
00713             virtual RelativeAngleCollection getRelativeAngleCollection() = 0;
00714
00715             virtual StatusMessage getStatusMessage() = 0;
00716
00717             virtual FieldOfViewReport getFieldOfViewReport() = 0;
00718
00719             virtual CurrentShutter getCurrentShutter() = 0;
00720
00721             virtual CallbackHandle registerPose6DofCallback(const std::function<void(Pose6Dof)>&) = 0;
00722
00723             virtual CallbackHandle registerPoseRelativeBeaconCallback(const
      std::function<void(PoseRelativeBeaconCollection)>&) = 0;
00724
00725             virtual CallbackHandle registerRelativeAngleCallback(const
      std::function<void(RelativeAngleCollection)>&) = 0;
00726
```

```
00727             virtual CallbackHandle registerMessageCallback(const std::function<void(StatusMessage)>&)
      = 0;
00728
00729             virtual CallbackHandle registerFieldOfViewReportCallback(const
      std::function<void(FieldOfViewReport)>&) = 0;
00730
00731             virtual CallbackHandle registerHeartbeatCallback(const std::function<void(Heartbeat)>&) =
      0;
00732
00733             virtual CallbackHandle registerCurrentShutterCallback(const
      std::function<void(CurrentShutter)>&) = 0;
00734
00735             virtual void removePose6DofCallback(const CallbackHandle&) = 0;
00736
00737             virtual void removePoseRelativeBeaconCallback(const CallbackHandle&) = 0;
00738
00739             virtual void removeRelativeAngleCallback(const CallbackHandle&) = 0;
00740
00741             virtual void removeMessageCallback(const CallbackHandle&) = 0;
00742
00743             virtual void removeFieldOfViewCallback(const CallbackHandle&) = 0;
00744
00745             virtual void removeHeartbeatCallback(const CallbackHandle&) = 0;
00746
00747             virtual void removeCurrentShutterCallback(const CallbackHandle&) = 0;
00748
00757             virtual void swapMap(const std::vector<Beacon>& map) = 0;
00758         };
00759
00768         SDS_API std::unique_ptr<DroneTrackingManager> createDroneTrackingManager();
00769     }
00770 }
```