Qualcomm Technologies, Inc.

Qualcomm® Snapdragon Navigator™

Developer Guide

80-P4698-20 B

August 8, 2017

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

List of Tables

1 Introduction

1.1 Purpose

This document describes how to use the Snapdragon Navigator API to develop applications that interact with Snapdragon Navigator. It is important to understand Snapdragon Navigator operation and install Snapdragon Navigator on the target before using this document. Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18).

This document assumes that the reader has a basic knowledge of UNIX.

1.2 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

Code variables appear in angle brackets, e.g., `<number>`.

Commands and command variables appear in a different font, e.g., **`copy a:*.* b:`**.

2 Functional Overview

The Snapdragon Navigator API enables external applications to interact with the internal flight controller, permitting higher-level applications written in C or C++ to control the vehicle's movement.

For basic Snapdragon Navigator API usage, refer to https://github.com/ATLFlight/snav_api_examples.

Table 2-1 lists and describes the files provided by the Snapdragon Navigator API.

Table 2-1 Snapdragon Navigator API files

Filename	Description
snapdragon_navigator.h	Header file containing Snapdragon Navigator API function declarations
snav_types.h	Header file containing Snapdragon Navigator API type declarations
snav_cached_data.h	Header file containing Snapdragon Navigator API cached data struct declarations
libsnav_arm.so	Library containing Snapdragon Navigator API implementation

3 Getting Started

3.1 Installing the Snapdragon Navigator Developer Files

Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) for instructions on how to install a Snapdragon Navigator Debian package.

4 Develop an Application

For basic Snapdragon Navigator API usage, refer to https://github.com/ATLFlight/snav_api_examples.

4.1 Writing the Source

To write a program for the applications processor that uses the Snapdragon Navigator API, include the `snapdragon_navigator.h` header file in C or C++ code. Use the example programs as a guide to using the API. Here are a few things to keep in mind:

- The vehicle can only be under API control if it is receiving commands frequently enough; this serves as a *heartbeat* to notify the vehicle that the external application is running. Commands must be sent periodically before the vehicle can enter API control.
- It is recommended to store the estimated position and yaw at startup, and use them to zero out the estimated position and yaw in the application. This sets the *origin* as the vehicle's state at takeoff.

  The application must grab the estimated position and yaw just after the propeller state transitions to starting, *not* after the props state transitions to spinning. Transient pressure data due to the ground effect can have an unexpected impact on the Z estimate.
- Knowing the estimated position and yaw is useful, but in most cases, it is best to have the application use the desired position and yaw for control. Controlling the desired position and yaw is equivalent to moving around the setpoint of the internal position controller. This allows the internal position controller to control the higher order dynamics of the system.

4.1.1 Understanding the RC Command Interface

When developing an application to compute control commands in real units, such as following a trajectory or going to a waypoint with a calculated velocity profile, it is crucial to understand how the four unitless control commands are interpreted in different flight modes.

The general meaning of the four unitless commands is summarized in Table 4-1. For the meaning of each RC command type, see Table 4-2.

Table 4-1 Snapdragon Navigator RC commands general meaning

	cmd0	cmd1	cmd2	cmd3
Positive	Forward	Left	Up	Rotate counter-clockwise
Negative	Backward	Right	Down	Rotate clockwise

Table 4-2 Snapdragon Navigator RC command details

	cmd0	cmd1	cmd2	cmd3
SN_RC_OPTIC_FLOW_POS_HOLD_CMD	Speed in vehicle-relative X direction	Speed in vehicle-relative Y direction	Vertical speed	Yaw rate
SN_RC_VIO_POS_HOLD_CMD	Speed in vehicle-relative X direction	Speed in vehicle-relative Y direction	Vertical speed	Yaw rate
SN_RC_GPS_POS_HOLD_CMD	Speed in vehicle-relative X direction	Speed in vehicle-relative Y direction	Vertical speed	Yaw rate
SN_RC_ALT_HOLD_CMD	Pitch angle	Negative roll angle	Vertical speed	Yaw rate
SN_RC_ALT_HOLD_LOW_ANGLE_CMD	Pitch angle	Negative roll angle	Vertical speed	Yaw rate
SN_RC_THRUST_ANGLE_GPS_HOVER_CMD	Pitch angle	Negative roll angle	Thrust magnitude	Yaw rate
SN_RC_THRUST_ANGLE_CMD	Pitch angle	Negative roll angle	Thrust magnitude	Yaw rate
SN_RC_RATES_CMD	Pitch rate	Negative roll rate	Thrust magnitude	Yaw rate

4.2 Building the Source

4.2.1 On Target

To build an executable on target, use `gcc` or `g++`:

```
$ g++ my_sn_api_test.cpp -o my_sn_api_test -I/home/linaro/examples/inc
-lsnav_arm
```

See the `Makefile` that builds the examples as a reference.

4.2.2 Cross Compilation with qrlSDK

Cross compilation is not yet supported but will be added in a future software release.

4.3 Running the Executable

1. Verify that Snapdragon Navigator is running.
2. Run the executable.

   For example: `$ ./my_sn_api_test`

5 Deprecated List

Global sn_get_est_accel_bias (float ∗ax_bias, float ∗ay_bias, float ∗az_bias) __SNAV_EXTERNAL_SYMBOL_ATTRIBUTE

This function will be removed in a future release.

Global sn_get_est_gyro_bias (float ∗wx_bias, float ∗wy_bias, float ∗wz_bias) __SNAV_EXTERNAL_SYMBOL_ATTRIBUTE

This function will be removed in a future release.

Global sn_is_gps_enabled (int ∗gps_enabled) __SNAV_EXTERNAL_SYMBOL_ATTRIBUTE

This function will be removed in a future release.

6 Snapdragon Navigator Interface

This chapter describes the Snapdragon Navigator interface enumerations, structures, and functions.

6.1 Commands

6.1.1 Function Documentation

6.1.1.1 int sn_update_data ( )

Updates the internal cache of flight control data.

Detailed description

This function caches the current state of all other components that can be queried. This function must be called once per control loop before querying the flight software information. This function also handles initialization of API assets and must be called at least once before calling any other API function.

Returns

- 0 for success
- -1 for failure (flight software non-functional)

Dependencies

None.

6.1.1.2 int sn_spin_props ( )

Non-blocking attempt to spin propellers.

Detailed description

This function does not guarantee that propellers start spinning. Instead, safety checks are performed and then propellers started if deemed safe.

This function introduces a time delay before the propellers spin.

Note: For this function to have effect, the following conditions must be met:

- Propellers must not be spinning – Verify using the sn_get_props_state() function.
- Vehicle must be in a flight mode, that is, a heartbeat must already be established by calling sn_send_rc_command() or a similar function.

Check SnPropsState using the sn_get_props_state() function to verify that the command executed.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.3 int sn_stop_props ( )

Non-blocking attempt to stop propellers.

Detailed description

This function does not guarantee that propellers stop spinning. Safety checks are performed internally and propellers stop if deemed safe.

Note: For this function to have effect, the following conditions must be met:

- Propellers must be spinning or starting – Verify using the sn_get_props_state() function.
- Vehicle must be in a flight mode, that is, a heartbeat must already be established by calling sn_send_rc_command() or a similar function.

Check SnPropsState using the sn_get_props_state() function to verify that the command executed.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.4 int sn_start_static_accel_calibration ( )

Non-blocking attempt to start static accelerometer calibration.

Detailed description

Calibration only starts if it is deemed safe and appropriate for vehicle to do so. Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) for instructions.

During this calibration, ensure vehicle is completely stationary on a level surface. Use the information in the GeneralStatus struct to determine whether the calibration succeeds or fails.

Note: Vehicle must be rebooted after calibration to enable flight.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.5 int sn_get_static_accel_calibration_status (SnCalibStatus ∗ *status*)

Gets the static accelerometer calibration status from the internal cache of flight control data. This function can be used to determine if calibration data exists or if the calibration procedure is in progress.

Associated data types

SnCalibStatus

Parameters

`out`	*status*	Pointer to value to be set to the status of the static accelerometer calibration.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.6 int sn_start_dynamic_accel_calibration ( )

Non-blocking attempt to start dynamic accelerometer calibration.

Detailed description

Calibration only starts if it is deemed safe and appropriate for vehicle to do so.

Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) for instructions.

Use the information in the GeneralStatus struct to determine whether the calibration succeeds or fails.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.7 int sn_get_dynamic_accel_calibration_status (SnCalibStatus * *status*)

Gets the status of dynamic accelerometer calibration from the internal cache of flight control data. This function can be used to determine if calibration data exists or if the calibration procedure is in progress.

Associated data types

SnCalibStatus

Parameters

out	*status*	Pointer to the value to be set to the status of the dynamic accelerometer calibration.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.8 int sn_start_imu_thermal_calibration ( )

Non-blocking attempt to start thermal IMU calibration.

Detailed description

Calibration only starts if it is deemed safe and appropriate for vehicle to do so. Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) for instructions.

During this calibration, ensure vehicle is completely stationary on a level surface.

Increase the vehicle temperature during this test to ensure that a large temperature range observed.

Use the information in the GeneralStatus struct to determine whether the calibration succeeds or fails.

Note: The vehicle must be rebooted after calibration to enable flight.

Note: A static calibration is required immediately after thermal calibration.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.9 int sn_get_imu_thermal_calibration_status (SnCalibStatus * *status*)

Gets the status of thermal IMU calibration from the internal cache of flight control data. This function can be used to determine if calibration data exists or if the calibration procedure is in progress.

Associated data types

SnCalibStatus

Parameters

out	*status*	Pointer to the value to be set to the status of the IMU thermal calibration.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.10 int sn_start_optic_flow_camera_yaw_calibration ( )

Non-blocking attempt to start optic flow camera yaw calibration.

Detailed description

Calibration only starts if it is deemed safe and appropriate for the vehicle. Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) for instructions on how to run this calibration.

Note: Vehicle must be rebooted after calibration to enable flight.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.11 int sn_get_optic_flow_camera_yaw_calibration_status (SnCalibStatus * *status*)

Gets the status of optic flow camera yaw calibration from the internal cache of flight control data. This function can be used to determine if calibration data exists or if the calibration procedure is in progress.

Associated data types

SnCalibStatus

Parameters

out	*status*	Pointer to the value to be set to the status of the optic flow camera yaw calibration.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.12 int sn_start_magnetometer_calibration ( )

Non-blocking attempt to start magnetometer (compass) calibration.

Detailed description

Calibration only starts if it is deemed safe and appropriate for the vehicle. Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) for instructions on how to run this calibration.

Note: The vehicle must be rebooted after calibration to enable flight.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.13 int sn_get_magnetometer_calibration_status (SnCalibStatus ∗ *status*)

Gets the magnetometer calibration status from the internal cache of flight control data. This function can be used to determine if calibration data exists or if the calibration procedure is in progress.

Associated data types

SnCalibStatus

Parameters

out	*status*	Pointer to the value to be set to the status of the magnetometer calibration.

Returns

- 0 if attempt was received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.14 int sn_send_esc_rpm (int * *rpm_data*, unsigned int *size*, int *fb_id*)

Sends RPM commands to the ESCs and requests feedback.

Parameters

in	*rpm_data*	Pointer to the array containing RPM data to be sent to ESCs. RPMs are ordered in ascending order of ESC ID, e.g., [rpm_0, rpm_1, ..., rpm_n]
in	*size*	Number of rpm_data array elements.
in	*fb_id*	ID of the ESC from which feedback is desired. If fb_id = -1, no feedback is requested.

Detailed description

Sending this command does not guarantee that the ESCs spin the motors. If the vehicle is not in flight, the flight controller forwards the RPM commands to the ESCs. The ESC identified by fb_id requests feedback.

Note: RPM commands must be sent at a rate between 100 Hz and 500 Hz.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.15 int sn_send_esc_pwm (int * *pwm_data,* unsigned int *size,* int *fb_id*)

Sends PWM commands to the ESCs and requests feedback.

Parameters

in	*pwm_data*	Pointer to int array containing PWM data in the range [-800, 800] to be sent to ESCs. PWMs are ordered in ascending order of ESC ID, e.g. [pwm_0, pwm_1, ..., pwm_n].
in	*size*	Number of pwm_data array elements.
in	*fb_id*	ID of the ESC from which feedback is desired. If fb_id = -1, no feedback is requested.

Detailed description

The pwm_data array contains ESC PWMs in the range of [-800, 800] in which 800 corresponds to 100% duty cycle and negative implies reversed direction.

Sending this command does not guarantee that the ESCs spin the motors. If the vehicle is not in flight, the flight controller forwards the PWM commands to the ESCs. The ESC identified by fb_id requests feedback.

Note: PWM commands must be sent at a rate between 100 Hz and 500 Hz.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.16 int sn_send_rc_command (SnRcCommandType *type*, SnRcCommandOptions *options*, float *cmd0*, float *cmd1*, float *cmd2*, float *cmd3*)

Sends an "RC-like" command to the flight controller.

Associated data types

SnRcCommandType
SnRcCommandOptions

Parameters

in	*type*	Specification of the desired input interpretation.
in	*options*	Options for interpreting the command.
in	*cmd0*	Value in the range [-1.0, 1.0] specifying a "forward/backward" type command in which "forward" is positive.
in	*cmd1*	Value in the range [-1.0, 1.0] specifying a "left/right" type command in which "left" is positive.
in	*cmd2*	Value in the range [-1.0, 1.0] specifying an "up/down" type command in which "up" is positive.
in	*cmd3*	Value in the range [-1.0, 1.0] specifying a "rotate" type command in which rotating counter-clockwise is positive.

Detailed description

This function sends four dimensionless control commands to the flight controller and establishes a heartbeat for the API application. The interpretation of the four commands depends on the mode, which can be obtained from the current_mode field of the GeneralStatus struct. The desired meaning of the four commands is specified with the type parameter. A heartbeat must be established by calling this function before the flight controller allows the API to take control, such as to start spinning the propellers.

See Section 4.1.1 for the meaning of the four commands in different contexts.

Suggested SnRcCommandOptions option usage:

- RC_OPT_DEFAULT_RC for intuitive joystick control, including a small deadband to prevent drift and more intuitive mapping
- RC_OPT_LINEAR_MAPPING for absolute control of outputs – Useful with the sn_apply_cmd_mapping() function

Note: RC commands must be sent at a rate of at least 50 Hz.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)
- -2 if any command is NaN

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.17 int sn_apply_cmd_mapping (SnRcCommandType *type,* SnRcCommandOptions *options,* float *input0,* float *input1,* float *input2,* float *input3,* float * *cmd0,* float * *cmd1,* float * *cmd2,* float * *cmd3*)

Converts dimensioned commands into dimensionless commands.

Associated data types

SnRcCommandType
SnRcCommandOptions

Parameters

in	*type*	Command type remapped to match the type used subsequently in the sn_send_rc_command() function.
in	*options*	Options to apply during mapping. See sn_send_rc_command().
in	*input0*	Dimensioned command to be mapped into cmd0.
in	*input1*	Dimensioned command to be mapped into cmd1.
in	*input2*	Dimensioned command to be mapped into cmd2.
in	*input3*	Dimensioned command to be mapped into cmd3.
out	*cmd0*	Mapped unitless command 0 to be sent with the sn_send_rc_command() function.
out	*cmd1*	Mapped unitless command 1 to be sent with the sn_send_rc_command() function.
out	*cmd2*	Mapped unitless command 2 to be sent with the sn_send_rc_command() function.
out	*cmd3*	Mapped unitless command 3 to be sent with the sn_send_rc_command() function.

Detailed description

This function remaps commands with real units into the appropriate dimensionless commands to be sent with sn_send_rc_command() function. Mapping is based on on the type and options parameters. See Section 4.1.1 for more information.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)

Dependencies

None.

See also

sn_send_rc_command()

6.1.1.18 const char* sn_get_enum_string (char * *type,* int *value*)

Gets a human-readable string associated with a specific enum type and value.

Parameters

in	*type*	String specifying the type of enum value provided. For example, "SnMode", "SnMotorState", etc.
in	*value*	Value to be converted to a string (based on the type provided).

Returns

Pointer to the string corresponding to the provided enum.

Dependencies

None.

6.1.1.19 const char* sn_get_cmd_name (SnRcCommandType *type*)

Gets the the name of an RC command from the type.

Parameters

in	*type*	RC command type of interest.

Returns

Pointer to the command name.

Dependencies

None.

6.1.1.20 const char* sn_get_dimensioned_units (SnRcCommandType *type,* int *index*)

Gets the the dimensioned units of an RC command from the type and index.

Associated data types

SnRcCommandType

Parameters

in	*type*	RC command type of interest.
in	*index*	Index in range [0, 3] corresponding to cmd0 through cmd3.

Returns

Pointer to the units of a particular command.

Dependencies

None.

6.1.1.21 float sn_get_min_value (SnRcCommandType *type,* int *index*)

Gets the the minimum value in real units for an RC command from the type and index.

Associated data types

SnRcCommandType

Parameters

in	*type*	RC command type of interest.
in	*index*	Index in range [0,3] corresponding to cmd0 through cmd3.

Detailed description

This function returns the smallest possible command to be applied to the system. The returned value maps to a dimensionless value of -1.0.

Use sn_get_dimensioned_units() to get a string descripton of the units.

Returns

Minimum-allowed value in real units.

Dependencies

None.

6.1.1.22 float sn_get_max_value (SnRcCommandType *type,* int *index*)

Gets the the maximum value in real units for an RC command from the type and index.

Associated data types

SnRcCommandType

Parameters

in	*type*	RC command type of interest.
in	*index*	Index in the range [0,3] corresponding to cmd0 through cmd3.

Detailed description

This function returns the largest possible command to be applied to the system. This value maps to a dimensionless value of 1.0.

Use the sn_get_dimensioned_units() function to get a string descripton of the units.

Returns

Maximum-allowed value in real units.

Dependencies

None.

6.1.1.23 int sn_send_thrust_att_ang_vel_command (float *thrust*, float *qw*, float *qx*, float *qy*, float *qz*, float *wx*, float *wy*, float *wz*)

Sends thrust, attitude, and angular velocity.

Parameters

in	*thrust*	Commanded thrust in grams.
in	*qw*	Scalar component of quaternion.
in	*qx*	X component of vector part of the quaternion.
in	*qy*	Y component of vector part of the quaternion.
in	*qz*	Z component of vector part of the quaternion.
in	*wx*	X component of angular velocity in rad/s.
in	*wy*	Y component of angular velocity in rad/s.
in	*wz*	Z component of angular velocity in rad/s.

Detailed description

This function sends the desired thrust in grams, desired attitude represented as a quaternion, and the desired angular velocity vector in rad/s to the flight controller.

The quaternion is in the following form:

$q = qw + qx*i + qy*j + qz*k$

Note: Be cautious – This function is for advanced users.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)
- -2 if any input arguments are NaN

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.24 int sn_send_trajectory_tracking_command (SnPositionController *controller,* SnTrajectoryOptions *options,* float *x,* float *y,* float *z,* float *xd,* float *yd,* float *zd,* float *xdd,* float *ydd,* float *zdd,* float *yaw,* float *yaw_rate*)

Sends position, angle, and derivatives for advanced trajectory tracking.

Parameters

`in`	*controller*	Desired position controller.
`in`	*options*	Options for trajectory tracking.
`in`	*x*	X component of the position vector in the specified frame.
`in`	*y*	Y component of the position vector in the specified frame.
`in`	*z*	Z component of the position vector in the specified frame.
`in`	*xd*	X component of the velocity vector in the specified frame.
`in`	*yd*	Y component of the velocity vector in the specified frame.
`in`	*zd*	Z component of the velocity vector in the specified frame.
`in`	*xdd*	X component of the acceleration vector in the specified frame.
`in`	*ydd*	Y component of the acceleration vector in the specified frame.
`in`	*zdd*	Z component of the acceleration vector in the specified frame.
`in`	*yaw*	Gravity-aligned yaw angle relative to the specified frame.
`in`	*yaw_rate*	Rate at which the gravity-aligned yaw angle changes.

Detailed description

This function allows precise control of the desired trajectories to use feed-forward acceleration angles.
Note: This function is intended for advanced users – A trajectory planner must be used to update these options at a minimum of 50 Hz.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)
- -2 if any input arguments are NaN
- -3 if the specified control mode is unavailable

Dependencies

None.

6.1.1.25 int sn_set_battery_voltage (float *voltage*)

Sets the battery voltage.

Parameters

in	*voltage*	Battery voltage (V).

Detailed description

This function overrides the internal battery voltage estimate. This function must be called at a rate faster than 5 Hz for the value to be considered valid, otherwise the flight controller defaults to the internal estimate of battery voltage.

Returns

- 0 if command received
- -1 for failure (flight software non-functional)
- -2 if the voltage is less than or equal to zero, or is NaN

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.26 int sn_get_flight_data_ptr (int *size_cached_struct,* SnavCachedData ** *snav_cached_data_struct*)

Gets the pointer to the Snapdragon Navigator cached data structure.

Associated data types

SnavCachedData

Parameters

in	*size_cached_struct*	Structure size. Ensures that the header file stays in sync. This argument must be sizeof(SnavCachedData).
out	*snav_cached_data_-struct*	Pointer to be filled with the cached data structure pointer values. This structure is updated with a call to sn_update_data().

Returns

- 0 if flight data pointer returns successfully
- -1 for failure to get the pointer to flight data

Dependencies

None.

6.1.1.27 int sn_set_led_colors (const uint8_t ∗ *led_colors_input_array,* int *led_colors_size,* int *led_colors_timeout_us*)

Sets the LED colors, overriding the Snapdragon Navigator LED colors.

Parameters

in	*led_colors_input_-array*	Array of RGB triplets. The range for each value is 0-255.
in	*led_colors_size*	Size of the input color array – Value must be greater than zero, less than 25, and a multiple of 3.
in	*led_colors_timeout_-us*	Timeout in microseconds for Snapdragon Navigator to take over LED control after the API color commands stop.

Detailed description

This function overrides the internal output of the LED colors. Currently only a single RGB triplet is used (first three bytes). The timeout variable specifies the time in microseconds when the LED output switches back to Snapdragon Navigator control after the API color commands stop updating. Color values are interpreted as binary (0 = Off, otherwise = On).

Returns

- 0 command received
- -1 critical failure (flight software is most likely non-functional)
- -2 bad length of the color data array
- -3 negative value provided as timeout

Dependencies

The sn_update_data() function must be called at least once prior to calling this function.

6.1.1.28 int sn_get_esc_state_feedback (SnMotorState ∗ *state_feedback,* unsigned int *size,* unsigned int ∗ *used*)

Gets the ESC state feedback data from from the internal cache of flight control data.

Associated data types

SnMotorState

Parameters

out	*state_feedback*	Pointer to the array to be filled with states from ESCs. The array is filled in ascending order of ESC IDs, e.g. [state_0, state_1, ..., state_n]
in	*size*	Number of state_feedback array elements.
out	*used*	Pointer to the value to be set to the number of elements used of the state_feedback array

Detailed description

The given array must have a number of elements equal to the number of ESCs connected to the flight controller.

If the given array is too small to hold all of the feedback data, no data copies into the array and the function returns an error code.

Note: ESC feedback data is only updated if feedback is requested. See the sn_send_esc_rpm() and sn_send_esc_pwm() functions.

Returns

- 0 for success
- -1 for failure (flight software non-functional)
- -2 if the size of the array is not big enough to hold all of the feedback data

Dependencies

The sn_update_data() function must be called to refresh the internal cache of flight control data.

See also

sn_send_esc_rpm()
sn_send_esc_pwm()

6.1.1.29 int sn_get_est_accel_bias (float ∗ *ax_bias,* float ∗ *ay_bias,* float ∗ *az_bias*)

Deprecated This function will be removed in a future release.

Query-estimated accelerometer biases.

Parameters

ax_bias	Reference to the float to be filled with the X accelerometer-estimated bias in G's.
ay_bias	Reference to the float to be filled with the Y accelerometer-estimated bias in G's.
az_bias	Reference to the float to be filled with the Z accelerometer-estimated bias in G's.

Detailed description

Biases are represented with respect to the flight controller's body frame.
Accelerometer biases are defined as follows:
compensated linear acceleration = raw linear acceleration - biases

Returns

- 0 for success
- -1 for failure (flight software is most likely non-functional)

Dependencies

The sn_update_data() function must be called to update these values.

6.1.1.30 int sn_get_est_gyro_bias (float ∗ *wx_bias,* float ∗ *wy_bias,* float ∗ *wz_bias*)

Deprecated This function will be removed in a future release.

Query-estimated gyroscope biases.

Parameters

wx_bias	Reference to the float to be filled with the X gyro-estimated bias.
wy_bias	Reference to the float to be filled with the Y gyro-estimated bias.
wz_bias	Reference to the float to be filled with the Z gyro-estimated bias.

Returns

- 0 for success
- -1 for failure (most likely indicates flight non-functional software)

Dependencies

The sn_update_data() function must be called to update these values.

6.1.1.31 int sn_is_gps_enabled (int * *gps_enabled*)

Deprecated This function will be removed in a future release.

Detects whether GPS is enabled.

Parameters

out	*gps_enabled*	Pointer to GPS enable status – 1 if GPS is enabled; 0 otherwise.

Returns

- 0 for success
- -1 for failure (flight software non-functional)

Dependencies

The sn_update_data() function must be called to refresh the internal cache of flight control data.

6.2 Datatypes

6.2.1 Data Structure Documentation

6.2.1.1 struct VersionInfo

Version information required to uniquely identify the software and device.

Data fields

Type	Parameter	Description
char	device_-identifier[20]	Version of the DSPaL library used.
char	compile_-date[16]	Null terminated string containing the compilation date.
char	compile_-time[16]	Null terminated string containing the compilation time.
char	library_-version[18]	Null terminated string representing version information.
char	library_-hash[41]	Null terminated string with a unique build identifier.
char	mac_-address[18]	Null terminated string containing wlan0 mac address if it was successfully polled.
char	dspal_-version[40]	Version of the DSPaL library used.
int32_t	esc_hw_-version[8]	Hardware revision of the ESCs.
int32_t	esc_sw_-version[8]	Software version of the ESCs.

6.2.1.2 struct MvSdkVersionInfo

Machine vision (MV) version information.

Data fields

Type	Parameter	Description
char	version_-recommended[18]	Null-terminated string containing the recommended MV SDK version.
char	version_-found[18]	Null-terminated string containing the MV SDK version found on the system.
uint8_t	strict_checking	Specifies if the recommended MV SDK version is required.

6.2.1.3 struct SensorImuApiVersionInfo

Version of the sensor_imu API.

Data fields

Type	Parameter	Description
char	version_-recommended[18]	Null-terminated string containing the recommended sensor_imu API version.
char	version_-found[18]	Null-terminated string containing the sensor_imu API version found on the system.
uint8_t	strict_checking	Specifies if the recommended sensor_imu API version is required.

6.2.1.4 struct GeneralStatus

General system state information for debugging system issues.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the most recent iteration of the main flight loop started. (Units: μs)
uint32_t	loop_cntr	Number of times the control loop has run.
int32_t	desired_mode	Cast to enum: SnMode. Desired mode that the flight controller attempts to transition to.
int32_t	current_mode	Cast to enum: SnMode. Current flight controller mode.
float	voltage	Estimated input system voltage. (Units: V)
float	current	If available, the estimated electrical current being used by the system. (Units: A)
uint8_t	is_using_-external_-voltage	1 – Voltage is being measured by the external voltage driver. 0 – Voltage is measured using ESCs.
int32_t	props_state	Cast to enum: SnPropsState. Propeller state.
uint8_t	on_ground	Flag representing flight controller's detection of the device being on the ground. 1 – Device on the ground, 0 otherwise.
int32_t	input_cmd_-type	Cast to enum: SnInputCommandType. Input command type.
char	last_error_-code[32]	Null-terminated string to represent the last error code detected. To allow detection of infrequent errors, this string persists even if cleared by an error code.

6.2.1.5 struct DataStatus

Status of various sensors and estimators.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the struct was logged. (Units: ms)
uint32_t	loop_cntr	Number of times at which the control loop has run.

Type	Parameter	Description
int32_t	imu_0_status	Cast to enum: SnDataStatus. IMU sensor status.
int32_t	baro_0_status	Cast to enum: SnDataStatus. Barometer sensor status.
int32_t	esc_feedback_-status	Cast to enum: SnDataStatus. ESC feedback status.
int32_t	mag_0_status	Cast to enum: SnDataStatus. Magnetometer sensor.
int32_t	gps_0_status	Cast to enum: SnDataStatus. Global positioning GPS sensor status.
int32_t	sonar_0_status	Cast to enum: SnDataStatus. Sonar sensor status.
int32_t	optic_flow_0_-status	Cast to enum: SnDataStatus. Optic flow (DFT) sensor status.
int32_t	spektrum_rc_0-_status	Cast to enum: SnDataStatus. Spektrum RC sensor status.
int32_t	api_rc_status	Cast to enum: SnDataStatus. API RC command status.
int32_t	rc_active_status	Cast to enum: SnDataStatus. Active RC command status.
int32_t	height_-estimator_-status	Cast to enum: SnDataStatus. Height estimator status.
int32_t	attitude_-estimator_-status	Cast to enum: SnDataStatus. Attitude estimator status.
int32_t	vio_0_status	Cast to enum: SnDataStatus. Visual inertial odometry (VIO) status.
int32_t	voa_status	Cast to enum: SnDataStatus. Visual obstacle avoidance (VOA) status.
int32_t	api_trajectory_-status	Cast to enum: SnDataStatus. Status of trajectory data.

6.2.1.6 struct UpdateRates

System and sensor update rates.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	loop_cntr	Number of times the control loop has run.
float	control_loop_-freq	Main control loop update frequency. (Units: Hz)
float	imu0_freq	IMU0 update frequency. (Units: Hz)
float	imu1_freq	IMU1 update frequency. (Units: Hz)
float	imu2_freq	IMU2 update frequency. (Units: Hz)
float	baro0_freq	Baro0 update frequency. (Units: Hz)
float	mag0_freq	Mag0 update frequency. (Units: Hz)
float	sonar0_freq	Sonar0 update frequency. (Units: Hz)
float	rc0_freq	RC0 update frequency. (Units: Hz)
float	esc_fb_freq	ESC feedback update frequency. (Units: Hz)
float	gnss0_freq	GNSS0 update frequency. (Units: Hz)
float	gnss1_freq	GNSS1 update frequency. (Units: Hz)
float	voa_freq	VOA update frequency. (Units: Hz)
float	vio0_freq	VIO0 update frequency. (Units: Hz)

6.2.1.7 struct AttitudeEstimate

Estimate of the vehicle orientation.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Counter incremented upon successful computation of the attitude estimate.
float	roll	Roll angle using Tait-Bryan ZYX. (Units: rad)
float	pitch	Pitch angle using Tait-Bryan ZYX. (Units: rad)
float	yaw	Yaw angle using Tait-Bryan ZYX. (Units: rad)
float	rotation_-matrix[9]	Rotation matrix from vehicle body to world in row-major order.
float	magnetic_yaw-_offset	Yaw angle with respect to magnetic east = yaw + magnetic_yaw_offset. (Units: rad)
float	magnetic_-declination	Yaw angle with respect to true east = yaw + magnetic_yaw_offset + magnetic_declination. (Units: rad)

6.2.1.8 struct AttitudeEstimate1

Estimate of the vehicle orientation.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Counter incremented upon successful computation of the attitude estimate.
float	roll	Roll angle using Tait-Bryan ZYX. (Units: rad)
float	pitch	Pitch angle using Tait-Bryan ZYX. (Units: rad)
float	yaw	Yaw angle using Tait-Bryan ZYX. (Units: rad)
float	rotation_-matrix[9]	Rotation matrix from the vehicle body to world in row-major order.
float	magnetic_yaw-_offset	Yaw angle with respect to magnetic east = yaw + magnetic_yaw_offset. (Units: rad)
float	magnetic_-declination	Yaw angle with respect to true east = yaw + magnetic_yaw_offset + magnetic_declination. (Units: rad)

6.2.1.9 struct AttitudeEstimate2

Estimate of the vehicle orientation.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Counter incremented upon successful computation of the attitude estimate.

Type	Parameter	Description
float	roll	Roll angle using Tait-Bryan ZYX. (Units: rad)
float	pitch	Pitch angle using Tait-Bryan ZYX. (Units: rad)
float	yaw	Yaw angle using Tait-Bryan ZYX. (Units: rad)
float	rotation_-matrix[9]	Rotation matrix from the vehicle body to world in row-major order.
float	magnetic_yaw-_offset	Yaw angle with respect to magnetic east = yaw + magnetic_yaw_offset. (Units: rad)
float	magnetic_-declination	Yaw angle with respect to true east = yaw + magnetic_yaw_offset + magnetic_declination. (Units: rad)

6.2.1.10 struct CpuStats

Apps processor CPU status.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times data was updated
uint64_t	time_apps_us	Timestamp from the Apps processor. (Units: μs)
uint64_t	time_apps_real-_us	Timestamp from the Apps processor realtime clock. (Units: μs)
float	cur_freq[4]	Current Apps processor CPU frequency. If the CPU is not online, Snapdragon Navigator reads NaN. (Units: GHz)
float	max_freq[4]	Maximum Apps processor CPU frequency – Can be throttled due to temperature. If the CPU is not online, Snapdragon Navigator reads NaN. (Units: GHz)
float	temp[22]	Temperature measurements from thermal zones. Reads NaN if the thermal zone is disabled. (Units: °C)

6.2.1.11 struct Imu0Raw

Inertial measurement unit 0 raw data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of measurements received.
float	temp	IMU temperature. (Units: °C)
float	lin_acc[3]	Linear acceleration. (Units: gravity ($\sim$9.81 m/s/s))
float	ang_vel[3]	Angular velocity. (Units: rad/s)

6.2.1.12 struct Imu1Raw

Inertial measurement unit 1 raw data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of measurements received.
float	temp	IMU temperature. (Units: °C)
float	lin_acc[3]	Linear acceleration. (Units: gravity (∼9.81 m/s/s))
float	ang_vel[3]	Angular velocity. (Units: rad/s)

6.2.1.13 struct Imu2Raw

Inertial measurement unit 2 raw data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of measurements received.
float	temp	IMU temperature. (Units: °C)
float	lin_acc[3]	Linear acceleration. (Units: gravity (∼9.81 m/s/s))
float	ang_vel[3]	Angular velocity. (Units: rad/s)

6.2.1.14 struct Imu0Compensated

Inertial measurement unit (IMU) data after compensation.

Data fields

Type	Parameter	Description
int64_t	time	Time data was received. (Units: μs)
uint32_t	cntr	Number of compensated measurements recorded.
float	temp	Temperature of IMU. (Units: °C)
float	lin_acc[3]	Linear acceleration. (Units: gravity (∼9.81 m/s/s))
float	ang_vel[3]	Angular velocity. (Units: rad/s)

6.2.1.15 struct Imu0CalibrationThermal

Data results from IMU temperature calibration.

Data fields

Type	Parameter	Description
float	accel_slope[3]	XYZ slopes for accelerometer temperature calibration. (Units: gravity/ °C)
float	accel_offset[3]	XYZ offsets for accelerometer temperature calibration. (Units: gravity)
float	accel_-residual[3]	Average squared residual for accelerometer temperature calibration. (Units: gravity^2)
float	gyro_slope[3]	XYZ slopes for gyroscope temperature calibration. (Units: (rad/s)/ °C)
float	gyro_offset[3]	XYZ offsets for gyroscope temperature calibration. (Units: (rad/s)))
float	gyro_-residual[3]	Average squared residual for gyroscope temperature calibration. (Units: (rad/s)^2)

6.2.1.16 struct Imu0CalibrationOffset

IMU sensor offset values.

Data fields

Type	Parameter	Description
char	name[16]	Type of offset calibration (e.g., static or dynamic).
float	accel_offset[3]	XYZ accelerometer offsets from accelerometer offset calibration. (Units: gravity (~9.81 m/s/s))
float	avg_thrust	Average thrust found from in flight accelerometer calibration. (Units: g)
float	roll_trim_offset	Roll trim offset from in flight accelerometer calibration. (Units: g)
float	pitch_trim_-offset	Pitch trim offset from in flight accelerometer calibration. (Units: g)

6.2.1.17 struct Barometer0Raw

Raw barometer data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times data was read.
float	pressure	Atmospheric pressure measurement. (Units: Pa)
float	temp	Temperature of the sensor. (Units: °C)

6.2.1.18 struct Sonar0Raw

Raw sonar data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times the data was read.
float	range	Range measurement. (Units: m)

6.2.1.19 struct Mag0Raw

Raw magnetometer data from the mag0 sensor.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of data packets received.
uint8_t	identifier	Type of compass sensor.
float	field[3]	XYZ components of the magnetic field in the sensor frame. (Units: μT)

6.2.1.20 struct Mag1Raw

Raw magnetometer data from the mag1 sensor.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of data packets received.
uint8_t	identifier	Type of compass sensor.
float	field[3]	XYZ components of the magnetic field in the sensor frame. (Units: μT)

6.2.1.21 struct Mag0Compensated

Mag0 sensor data after compensation.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of data packets received.

Type	Parameter	Description
uint8_t	identifier	Type of compass sensor.
float	field[3]	XYZ components of the magnetic field in the sensor frame.

6.2.1.22 struct Mag1Compensated

Mag1 sensor data after compensation.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of data packets received.
uint8_t	identifier	Type of compass sensor.
float	field[3]	XYZ components of the magnetic field in the sensor frame.

6.2.1.23 struct Mag0Calibration3D

Data results from Mag0 3D calibration.

Data fields

Type	Parameter	Description
float	matrix[9]	Scale parameters of mapping.
float	offset[3]	XYZ offset of mapping.

6.2.1.24 struct SpektrumRc0Raw

Raw Spektrum RC data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times the data was read.
int32_t	protocol	Cast to enum: SnRcReceiverMode. RC protocol identifier
uint8_t	num_channels	Number of RC channels being populated.
uint16_t	vals[16]	Raw Spektrum channel values.

6.2.1.25 struct ApiRcRaw

RC commands sent through the API.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.

Type	Parameter	Description
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times the data was read.
int32_t	cmd_type	Cast to enum: SnRcCommandType. How the command is interpreted (if possible).
int32_t	cmd_options	Cast to enum: SnRcCommandOptions. Options used to deviate from linear mapping.
float	cmd[4]	Unitless RC-type command in range [-1, 1].

6.2.1.26 struct ApiThrustAttAngVel

Thrust, attitude, and angular velocity commands sent through the API.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
float	thrust	Thrust.
float	qw	Attitude quaternion W value.
float	qx	Attitude quaternion X value.
float	qy	Attitude quaternion Y value.
float	qz	Attitude quaternion Z value.
float	ang_vel[3]	Angular velocity.

6.2.1.27 struct ApiPropsCmd

Received API spin or stop propellers commands.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
uint8_t	api_spin_props-_rcvd	1 if spin-propellers command received, 0 otherwise.
uint8_t	api_stop_props-_rcvd	1 if stop-propellers command received, 0 otherwise.

6.2.1.28 struct RcActive

RC commands for control.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times the data was read.
int32_t	source	Cast to enum: SnRcCommandSource. Source of the active RC commands.

Type	Parameter	Description
int32_t	cmd_type	Cast to enum: SnRcCommandType. Specifies how the commands are interpreted (if possible). Irrelevant if the source is Spektrum RC.
int32_t	cmd_options	Cast to enum: SnRcCommandOptions. Options used to deviate from linear mapping.
float	cmd[4]	Unitless RC-type command in range [-1, 1].

6.2.1.29 struct Camera0FrameInfo

Captured camera frame information from the downward camera.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times the data was read.
uint32_t	frame_num	Frame number received, starting at 0.
int64_t	frame_-timestamp	Timestamp of frame. (Units: μs)
int64_t	preview_-timestamp	Timestamp of preview callback. (Units: μs)
float	exposure	Normalized exposure setting used to take the frame.
float	gain	Normalized gain setting used to take the frame.
float	average_-luminance	Normalized average luminance of the frame.

6.2.1.30 struct OpticFlow0Raw

Downward facing tracker (DFT) data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	cntr	Number of times the data was read.
float	pixel_flow[2]	Pixel displacement between subsequent image frames. Pixel flow is in the opposite direction of camera (and therefore vehicle) movement. (Units: pixels)
int32_t	sample_size	Number of inliers after calculation of displacement.
float	error_sum	Error metric, sum of squared error over sample size points.

6.2.1.31 struct OpticFlow0CalibrationTilt

Downfacing camera calibration for tilt angle (optic flow camera yaw calibration).

Data fields

Type	Parameter	Description
float	x_factor	Tilt factor in X. (Units: pixels/rad)
float	y_factor	Tilt factor in Y. (Units: pixels/rad)

6.2.1.32 struct Gps0Raw

Raw GPS data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which the data was received. (Units: μs)
uint32_t	cntr	Number of complete messages received.
int32_t	identifier	Cast to enum: SnGnssReceiverType. Type of GNSS receiver.
uint32_t	num_errors	Number of CRC errors.
uint32_t	gps_week	GPS week number. (Units: weeks)
uint32_t	gps_time_sec	Time of week. (Units: sec)
uint32_t	gps_time_nsec	Time of week. (Units: ns)
int32_t	latitude	Position latitude. (Units: deg$*$10e7)
int32_t	longitude	Position longitude. (Units: deg$*$10e7)
float	altitude	Altitude at mean sea level (MSL). (Units: m)
float	lin_vel[3]	Velocity of the east-north-up (ENU) frame. (Units: m/s)
uint8_t	fix_type	Fix type/quality.
uint8_t	num_satellites	Number of satellites used in the solution.
float	horizontal_acc	Horizontal accuracy of the position estimate. (Units: m)
float	speed_acc	Horizontal speed accuracy. (Units: m/s)
uint8_t	sv_ids[32]	Satellite identification number.
uint8_t	sv_cn0[32]	Satellite signal strength. (Units: C/N0)

6.2.1.33 struct TrajectoryDataRaw

Raw API data for trajectory control input.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Number of messages received.
int32_t	controller	Cast to enum: SnPositionController. Desired position controller to use.
int32_t	options	Cast to enum: SnTrajectoryOptions. Trajectory options.
float	position[3]	Desired position for the controller to achieve. (Units: m)
float	velocity[3]	Desired velocity for the controller to achieve. (Units: m/s)
float	acceleration[3]	Desired acceleration for controller to use as a feed-forward term. (Units: m/s/s)

Type	Parameter	Description
float	yaw	Desired gravity aligned yaw angle for the controller to achieve. (Units: raw)
float	yaw_rate	Desired gravity aligned yaw angle rate for the controller to achieve. (Units: raw)

6.2.1.34 struct PosVel

Position and velocity control data.

Data fields

Type	Parameter	Description
int64_t	time	Time at which the data was logged. (Units: μs)
uint32_t	cntr	Number of times the data was logged.
float	position_-estimated[3]	Estimated position of the vehicle with respect to the estimation frame. (Units: m)
float	velocity_-estimated[3]	Estimated velocity of the vehicle with respect to the estimation frame. (Units: m/s)
float	yaw_estimated	Estimated yaw angle of the vehicle with respect to the estimation frame. (Units: rad)
float	estimate_is_-valid	Whether the estimate is valid.
int32_t	position_-estimate_type	Cast to enum: SnPosEstType. Names the dominant source of the position estimate.
float	R_eg[9]	Orientation of the GNSS ENU frame with respect to the estimation frame.
float	gnss_rotation_-is_valid	Indicates if the rotation between the estimation frame and the GNSS ENU frame is valid or not.
float	t_eg[3]	Vector from the origin of the estimation frame to the origin of the GNSS ENU frame represented with respect to the estimation frame. If the estimation frame were drift-free, this vector would be zero over long time scales (could be non-zero over short time scales due to filtering delays).
float	gnss_-translation_is_-valid	Indicates if the translation between the estimation frame and the GNSS ENU frame is valid or not.
int32_t	gnss_type	Cast to enum: SnGnssReceiverType. Type of GNSS receiver.
float	position_-desired[3]	Desired position of the vehicle with respect to the estimation frame. (Units: m)
float	velocity_-desired[3]	Desired velocity of the vehicle with respect to the estimation frame. (Units: m/s)
float	yaw_desired	Desired yaw of the vehicle with respect to the estimation frame. (Units: rad)

6.2.1.35 struct VioPosVel

VIO position and velocity control data.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Counter that is incremented with each control loop.
float	position_-estimated[3]	Estimated XYZ position. (Units: m)
float	velocity_-estimated[3]	Estimated XYZ velocity. (Units: m/s)
float	yaw_estimated	Estimated yaw angle. (Units: rad)
float	position_-desired[3]	Desired XYZ position. (Units: m)
float	velocity_-desired[3]	Desired XYZ velocity. (Units: m/s)
float	yaw_desired	Desired yaw angle. (Units: rad)
uint8_t	is_valid	Is 1 if the VIO position and velocity data is valid.

6.2.1.36 struct GpsPosVel

GPS position and velocity control data.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Counter that is incremented with each control loop.
int32_t	identifier	Cast to enum: SnGnssReceiverType. Type of GNSS receiver.
float	position_-estimated[3]	Estimated XYZ position. +X is east, +Y is north, +Z is vertically up (Units: m)
float	velocity_-estimated[3]	Estimated XYZ velocity. (Units: m/s)
float	yaw_estimated	Estimated yaw angle of the vehicle's body-fixed frame with respect to the East North Up (ENU) frame. (Units: rad)
float	position_-desired[3]	Desired XYZ position. (Units: m)
float	velocity_-desired[3]	Desired XYZ velocity. (Units: m/s)
float	yaw_desired	Desired yaw angle of the vehicle's body-fixed frame with respect to the ENU frame. (Units: rad)
uint8_t	is_enabled	If enabled (set to 1), this data is populated when the vehicle gets a GPS lock.

6.2.1.37 struct OpticFlowPosVel

Optic flow position and velocity control data.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp. (Units: μs)
uint32_t	cntr	Number of times the data was read.
float	position_-estimated[3]	Estimated XYZ position. (Units: m)
float	velocity_-estimated[3]	Estimated XYZ velocity. (Units: m/s)
float	yaw_estimated	Estimated yaw angle. (Units: rad)
float	position_-desired[3]	Desired XYZ position. (Units: m)
float	velocity_-desired[3]	Desired XYZ velocity. (Units: m/s)
float	yaw_desired	Desired yaw angle. (Units: rad)
uint8_t	is_valid	1 if the optic flow position and velocity data are valid, otherwise 0.

6.2.1.38 struct EscRaw

Raw ESC data.

Data fields

Type	Parameter	Description
uint32_t	iter	Loop iteration in which data was logged.
int64_t	time	Time at which this data was published. (Units: μs)
uint32_t	bytes_tx	Number of bytes received from ESCs. (Units: bytes)
uint32_t	bytes_rx	Number of bytes received from all ESCs. (Units: bytes)
uint32_t	errors_rx	Number of feedback read errors.
uint32_t	packet_number	Packet number.
uint8_t	packet_cntr[8]	Number of control packets received by each ESC.
uint32_t	packets_rx[8]	Number of packets received from each ESC. (Units: packets)
int16_t	rpm[8]	Motor RPM.
int8_t	power[8]	Power applied by the ESC. (Units: %)
float	voltage[8]	Voltage measured by each ESC. (Units: V)
uint8_t	states[8]	ESC state.

6.2.1.39 struct VoaData

VOA data.

Data fields

Type	Parameter	Description
int64_t	time	Time at which this data was received. (Units: μs)

Type	Parameter	Description
uint32_t	cntr	DSP counter of data received.
uint64_t	apps_proc_time	Timestamp from the applications processor. (Units: μs)
uint32_t	apps_proc_cntr	Counter of data sent from the applications processor.
int	num_points	Number of points used for VOA control.

6.2.1.40 struct VoaStatus

State of VOA processing.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp of information (Units: μs)
uint32_t	cntr	Monotonically increasing counter
uint8_t	voa_enabled	1 if VOA is enabled, 0 otherwise.
uint8_t	voa_running	1 if VOA is running and has the ability to modify control output, 0 otherwise.
uint8_t	voa_active	1 if VOA is currently modifying control output, 0 otherwise.

6.2.1.41 struct RelativeObstacleDistances

Raw obstacle distance information (if available).

Data fields

Type	Parameter	Description
int64_t	time	Timestamp of data. (Units: μs)
uint32_t	cntr	Data log counter.
float	relative_-distances[4]	Array of distances sensed in the body coordinate frame and centered around the forward +X direction. Each number corresponds to an angle specified in the parameter file. By default, each number corresponds to 0.3 radians. (Units: m)

6.2.1.42 struct GpsOrigin

GPS latitude and longitude at the time of initial GPS lock.

Data fields

Type	Parameter	Description
int64_t	time	Timestamp of initial GPS lock. (Units: μs)
int32_t	origin_latitude	Latitude at GPS lock.
int32_t	origin_-longitude	Longitude at GPS lock.

6.2.1.43 struct FiducialMarkerWorldOffsetRaw

Raw world offset computation from fiducial markers.

Data fields

Type	Parameter	Description
uint64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	frame_id	Camera frame ID used for marker detection.
int64_t	frame_-timestamp_ns	Timestamp of the camera frame used for marker detection. (Units: ns)
uint32_t	num_markers_-used	Number of marker detections in the frame used in offset computation.
float	yaw_offset_raw	Raw Yaw offset. (Units: rad)
float	pos_offset_-raw[3]	Raw XYZ offset (Units: m)

6.2.1.44 struct FiducialMarkerWorldOffsetData

Filtered world offset computation from fiducial markers.

Data fields

Type	Parameter	Description
uint64_t	time	Time at which the struct was logged. (Units: μs)
uint32_t	num_raw_-offset_updates	Number of times that the raw offsets were computed.
float	pos_offset[3]	Vector from origin of marker world frame to origin of Snapdragon Navigator world frame represented in the Snapdragon Navigator world frame. (Units: m)
float	yaw_offset	Yaw component of the rotation from the Snapdragon Navigator world frame to the marker world frame. (Units: rad)
float	pos_world[3]	Estimated position represented in the marker world frame. (Units: m)
float	yaw_world	Estimated yaw angle represented in the marker world frame. (Units: rad)

6.2.1.45 struct SimGroundTruth

Simulation ground truth.

Data fields

Type	Parameter	Description
int64_t	time	Sim current time. (Units: μs)
float	position[3]	Ground truth position of the vehicle with respect to the simulation frame. (Units: m)
float	velocity[3]	Ground truth velocity of the vehicle with respect to the simulation frame. (Units: m/s)

Type	Parameter	Description
float	R[9]	Ground truth orientation of the vehicle-fixed frame with respect to the simulation frame.

6.2.1.46 struct SnavCachedData

Snapdragon Navigator cached data for the sn_get_flight_data_ptr() function.

Data fields

Type	Parameter	Description
VersionInfo	version_info	Version information required to uniquely identify the software and device.
MvSdkVersion-Info	mv_sdk_-version_info	Machine vision (MV) version information.
SensorImuApi-VersionInfo	sensor_imu_-api_version_-info	Version of the sensor_imu API.
GeneralStatus	general_status	General system state information for debugging system issues.
DataStatus	data_status	Status of various sensors and estimators.
UpdateRates	update_rates	System and sensor update rates.
Attitude-Estimate	attitude_-estimate	Estimate of the vehicle orientation.
Attitude-Estimate1	attitude_-estimate1	Estimate of the vehicle orientation.
Attitude-Estimate2	attitude_-estimate2	Estimate of the vehicle orientation.
CpuStats	cpu_stats	Apps processor CPU status.
Imu0Raw	imu_0_raw	Inertial measurement unit 0 raw data.
Imu1Raw	imu_1_raw	Inertial measurement unit 1 raw data.
Imu2Raw	imu_2_raw	Inertial measurement unit 2 raw data.
Imu0-Compensated	imu_0_-compensated	Inertial measurement unit (IMU) data after compensation.
Imu0-Calibration-Thermal	imu_0_-calibration_-thermal	Data results from IMU temperature calibration.
Imu0-Calibration-Offset	imu_0_-calibration_-offset	IMU sensor offset values.
Barometer0-Raw	barometer_0_-raw	Raw barometer data.
Sonar0Raw	sonar_0_raw	Raw sonar data.
Mag0Raw	mag_0_raw	Raw magnetometer data from the mag0 sensor.
Mag1Raw	mag_1_raw	Raw magnetometer data from the mag1 sensor.
Mag0-Compensated	mag_0_-compensated	Mag0 sensor data after compensation.
Mag1-Compensated	mag_1_-compensated	Mag1 sensor data after compensation.

Type	Parameter	Description
Mag0-Calibration3D	mag_0_-calibration_3d	Data results from Mag0 3D calibration.
SpektrumRc0-Raw	spektrum_rc_0-_raw	Raw Spektrum RC data.
ApiRcRaw	api_rc_raw	RC commands sent through the API.
ApiThrustAtt-AngVel	api_thrust_att_-ang_vel	Thrust, attitude, and angular velocity commands sent through the API.
ApiPropsCmd	api_props_cmd	Received API spin or stop propellers commands.
RcActive	rc_active	RC commands for control.
Camera0-FrameInfo	camera_0_-frame_info	Captured camera frame information from the downward camera.
OpticFlow0-Raw	optic_flow_0_-raw	Downward facing tracker (DFT) data.
OpticFlow0-CalibrationTilt	optic_flow_0_-calibration_tilt	Downfacing camera calibration for tilt angle (optic flow camera yaw calibration).
Gps0Raw	gps_0_raw	Raw GPS data.
TrajectoryData-Raw	trajectory_data-_raw	Raw API data for trajectory control input.
PosVel	pos_vel	Position and velocity control data.
VioPosVel	vio_pos_vel	VIO position and velocity control data.
GpsPosVel	gps_pos_vel	GPS position and velocity control data.
OpticFlowPos-Vel	optic_flow_-pos_vel	Optic flow position and velocity control data.
EscRaw	esc_raw	Raw ESC data.
VoaData	voa_data	VOA data.
VoaStatus	voa_status	State of VOA processing.
Relative-Obstacle-Distances	relative_-obstacle_-distances	Raw obstacle distance information (if available).
GpsOrigin	gps_origin	GPS latitude and longitude at the time of initial GPS lock.
Fiducial-MarkerWorld-OffsetRaw	fiducial_-marker_world-_offset_raw	Raw world offset computation from fiducial markers.
Fiducial-MarkerWorld-OffsetData	fiducial_-marker_world-_offset_data	Filtered world offset computation from fiducial markers.
SimGround-Truth	sim_ground_-truth	Simulation ground truth.

6.2.2 Enumeration Type Documentation

6.2.2.1 enum SnMode

Mode ID codes returned by querying the flight system mode.

Enumerator:

SN_SENSOR_ERROR_MODE Error – flight is not possible in current state.

SN_UNDEFINED_MODE Mode is not defined in this list.
SN_WAITING_FOR_DEVICE_TO_CONNECT Waiting for an RC or DroneController to connect.
SN_EMERGENCY_KILL_MODE Propellers were stopped – Most likely due to a crash.
SN_EMERGENCY_LANDING_MODE Low fixed-thrust emergency descent.
SN_THERMAL_IMU_CALIBRATION_MODE Thermal accel/gyro calibration.
SN_STATIC_ACCEL_CALIBRATION_MODE Static accel offset calibration.
SN_OPTIC_FLOW_CAM_YAW_CALIBRATION_MODE Optic flow camera yaw calibration.
SN_MAGNETOMETER_CALIBRATION_MODE Compass (magnetometer) calibration.
SN_CALIBRATION_SUCCESS Last active calibration was successful.
SN_CALIBRATION_FAILURE Last active calibration was not successful.
SN_ESC_RPM_MODE API controls the ESC RPMs.
SN_ESC_PWM_MODE API controls the ESC PWMs.
SN_RATE_MODE Thrust, roll rate, pitch rate, yaw rate; does not auto-stabilize.
SN_THRUST_ANGLE_MODE Thrust, roll angle, pitch angle, yaw rate.
SN_ALT_HOLD_MODE Vertical velocity, roll angle, pitch angle, yaw rate.
SN_THRUST_GPS_HOVER_MODE Thrust control with lateral position hold using GPS.
SN_GPS_POS_HOLD_MODE Body-relative 3D velocity and yaw rate using GPS.
SN_OPTIC_FLOW_POS_HOLD_MODE Body-relative 3D velocity and yaw rate using optic flow.
SN_VIO_POS_HOLD_MODE Body-relative 3D velocity and yaw rate using VIO.
SN_THRUST_ATT_ANG_VEL_MODE Thrust, attitude, and angular velocity.
SN_PRESSURE_LANDING_MODE Vertical velocity-controlled descent with zero roll/pitch.
SN_PRESSURE_GPS_LANDING_MODE 3D velocity-controlled descent.
SN_GPS_GO_HOME_MODE 3D velocity-controlled return to home position.
SN_ALT_HOLD_LOW_ANGLE_MODE Vertical velocity, roll angle, pitch angle, and yaw rate with limits on roll and pitch angles.
SN_POS_HOLD_MODE Body-relative 3D velocity and yaw rate.

6.2.2.2 enum SnInputCommandType

Input command types.

Enumerator:

SN_INPUT_CMD_TYPE_NONE No input.
SN_INPUT_CMD_TYPE_RC RC-style input commands.
SN_INPUT_CMD_TYPE_API_THRUST_ATT_ANG_VEL Thrust attitude angular velocity input commands from the API.
SN_INPUT_CMD_TYPE_API_ESC ESC input commands from the API.
SN_INPUT_CMD_TYPE_API_TRAJECTORY_CONTROL Trajectory control commands from the API.

6.2.2.3 enum SnRcCommandSource

RC command input source.

Enumerator:

SN_RC_CMD_NO_INPUT No input.
SN_RC_CMD_SPEKTRUM_INPUT Spektrum.
SN_RC_CMD_API_INPUT RC commands from API.

6.2.2.4 enum SnRcCommandType

RC command. This enum specifies how the dimensionless commands sent by the sn_send_rc_command() function are interpreted and indirectly selects the desired operation mode. The actual mode can be verified with the sn_get_mode() function.

Enumerator:

SN_RC_RATES_CMD Command pitch rate, negative roll rate, thrust magnitude, and yaw rate.

SN_RC_THRUST_ANGLE_CMD Command pitch angle, negative roll angle, thrust magnitude, and yaw rate.

SN_RC_ALT_HOLD_CMD Command pitch angle, negative roll angle, Z speed, and yaw rate.

SN_RC_THRUST_ANGLE_GPS_HOVER_CMD Command pitch angle, negative roll angle, thrust magnitude, and yaw rate; holds lateral position using GPS when roll and pitch commands are zero.

SN_RC_GPS_POS_HOLD_CMD Command vehicle-relative X and Y speeds, Z speed, and yaw rate using GPS.

SN_RC_OPTIC_FLOW_POS_HOLD_CMD Command vehicle-relative X and Y speeds, Z speed, and yaw rate using optic flow.

SN_RC_VIO_POS_HOLD_CMD Command vehicle-relative X and Y speeds, Z speed, and yaw rate using visual inertial odometry (VIO).

SN_RC_ALT_HOLD_LOW_ANGLE_CMD Command pitch angle, negative roll angle, Z speed, and yaw rate with a maximum tilt angle limit.

SN_RC_POS_HOLD_CMD Command vehicle-relative X and Y speeds, Z speed, and yaw rate using any available sensors.

SN_RC_NUM_CMD_TYPES Do not use – Reserved to hold the number of RC command types.

6.2.2.5 enum SnPositionController

Position contoller. This enum contains supported position controllers to specify how to interpret position, angle, and their derivatives into the sn_send_trajectory_tracking_command() function. Each position controller uses an appropriate estimate of position and programs that use these controllers need to ensure that the correct reference frame is used.

Enumerator:

SN_POSITION_CONTROL_GPS GPS-based position control.

SN_POSITION_CONTROL_VIO VIO-based position control.

SN_POSITION_CONTROL_OF Optic flow-based position control.

SN_POSITION_CONTROL_NUM_TYPES Do not use – Reserved for the number of position control types.

6.2.2.6 enum SnTrajectoryOptions

Options for trajectory tracking to be used in the sn_send_trajectory_tracking_command() function.

Enumerator:

SN_TRAJ_DEFAULT Default options.
SN_TRAJECTORY_OPTIONS_NUM Do not use – Reserved for the number of trajectory options.

6.2.2.7 enum SnRcCommandOptions

RC command options. The options can be OR-ed to form hybrid options

Enumerator:

RC_OPT_LINEAR_MAPPING Linear control (default).
RC_OPT_ENABLE_DEADBAND Enable deadband.
RC_OPT_COMPLIANT_TRACKING Enables the flight controller to modify and smooth input commands for feasibility. Obstacle avoidance features require this bit to be set, but commands might not not be tracked precisely if this flag is set. Use this flag when stick inputs are used and disable it to track motion precisely.
RC_OPT_DEFAULT_RC Default RC.
RC_OPT_TRIGGER_LANDING Trigger landing. The vehicle determines which landing mode is appropriate based on which sensors are available and what mode is active.

6.2.2.8 enum SnPropsState

Collective state of all of the propellers – Identification code returned by querying the flight system propeller state.

Enumerator:

SN_PROPS_STATE_UNKNOWN State of propellers is unknown.
SN_PROPS_STATE_NOT_SPINNING All propellers are not spinning.
SN_PROPS_STATE_STARTING Propellers are starting to spin.
SN_PROPS_STATE_SPINNING All propellers are spinning.

6.2.2.9 enum SnDataStatus

Identification code returned by querying the sensor data status.

Enumerator:

SN_DATA_INVALID Sensor data is invalid.
SN_DATA_VALID Sensor data is valid.
SN_DATA_NOT_INITIALIZED Sensor data has not been initialized.
SN_DATA_STUCK Sensor data is unchanging.
SN_DATA_TIMEOUT Sensor data has not been updated past the data timeout threshold.
SN_DATA_UNCALIBRATED Sensor data has not been calibrated.
SN_DATA_OFFSET_UNCALIBRATED Sensor data is missing offset calibration.
SN_DATA_TEMP_UNCALIBRATED Sensor data is missing temperature calibration.
SN_DATA_STARTING Sensor is acquiring additional samples.

SN_DATA_STATUS_UNAVAILABLE Sensor data status unavailable.
SN_DATA_NOT_ORIENTED Sensor data missing the orientation parameter.
SN_DATA_NO_LOCK Sensor data unable to lock on.
SN_DATA_WARNING Sensor is in a warning state.
SN_DATA_TRANSITIONING Sensor data is transitioning.

6.2.2.10 enum SnMotorState

Individual state of a motor – Identification code returned by querying the state feedback from ESCs.

Enumerator:

SN_MOTOR_STATE_UNKNOWN State of motor is unknown.
SN_MOTOR_STATE_NOT_SPINNING Motor is not spinning.
SN_MOTOR_STATE_STARTING Motor is starting to spin.
SN_MOTOR_STATE_SPINNING_FORWARD Motor is spinning in the forward direction.
SN_MOTOR_STATE_SPINNING_BACKWARD Motor is spinning in the backward direction.

6.2.2.11 enum SnCalibStatus

Identification code returned by querying the sensor calibration status.

Enumerator:

SN_CALIB_STATUS_NOT_CALIBRATED Calibration data does not exist.
SN_CALIB_STATUS_CALIBRATION_IN_PROGRESS Calibration procedure is in progress.
SN_CALIB_STATUS_CALIBRATED Calibration data exists.

6.2.2.12 enum SnRcReceiverMode

Spektrum data transmission mode. This value is used to request binding and display the current mode.

Enumerator:

SN_RC_RECEIVER_MODE_UNKNOWN Unknown DSM mode.
SN_SPEKTRUM_MODE_DSM2_22 DSM2 22 ms (6-channel maximum, every 22 ms).
SN_SPEKTRUM_MODE_DSM2_11 DSM2 11 ms (9-channel maximum, complete packet every 22 ms).
SN_SPEKTRUM_MODE_DSMX_22 DSMX 22 ms (6-channel maximum, every 22 ms).
SN_SPEKTRUM_MODE_DSMX_11 DSMX 11 ms (9-channel maximum, complete packet every 22 ms).

6.2.2.13 enum SnGnssReceiverType

Supported GNSS receiver types.

Enumerator:

SN_GNSS_RECEIVER_TYPE_UNKNOWN Unknown receiver.
SN_GNSS_RECEIVER_TYPE_CSR_SSV CSR receiver.
SN_GNSS_RECEIVER_TYPE_QC_WGR Qualcomm WGR receiver.
SN_GNSS_RECEIVER_TYPE_UBLOX U-blox receiver.

6.2.2.14 enum SnPosEstType

Position estimate type.

Multiple sensors can be used to estimate the position. This enum specifies the dominant source of the estimate used to determine the expected performance level.

Enumerator:

SN_POS_EST_TYPE_NONE No position estimate is available.
SN_POS_EST_TYPE_GPS GPS is the dominant source of position estimate.
SN_POS_EST_TYPE_VIO VIO is the dominant source of position estimate.
SN_POS_EST_TYPE_DFT Downward facing tracker (DFT) is the dominant source of position estimate.

A Troubleshooting

A.1 Propellers are not spinning when commanded

A.1.1 Calling the sn_start_props() function

Ensure that the flight controller is not in `SN_WAITING_FOR_DEVICE_TO_CONNECT` mode.

The sn_start_props() function only takes effect if the flight controller is receiving data and has not timed out.

Propellers only start spinning if Snapdragon Navigator has determined that it is safe and appropriate. Many conditions can cause Snapdragon Navigator to deem it unsafe or inappropriate to start the propellers. Refer to *Qualcomm Snapdragon Navigator User Guide* (80-P4698-18) to review these conditions.

B Terms and Conditions

THIS TERMS AND CONDITIONS OF USE (THE "AGREEMENT") IS A LEGALLY BINDING AGREEMENT BETWEEN QUALCOMM TECHNOLOGIES, INC. ("QTI") AND YOU OR THE LEGAL ENTITY YOU REPRESENT ("You" or "Your"). QTI IS WILLING TO PROVIDE THESE INSTRUCTION SETS AND ANY ASSOCIATED DOCUMENTATION (COLLECTIVELY REFERRED TO AS THE "INSTRUCTIONS") TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT AND AGREE TO ALL OF THE TERMS AND CONDITIONS IN THIS AGREEMENT. BY DOWNLOADING AND USING THE INSTRUCTIONS YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS, QTI IS UNWILLING TO AND DOES NOT LICENSE THE INSTRUCTIONS TO YOU. IF YOU DO NOT AGREE TO THESE TERMS YOU MUST DISCONTINUE THE USE OF THE INSTRUCTIONS AND YOU SHALL NOT USE THE INSTRUCTIONS OR RETAIN ANY COPIES OF THE INSTRUCTIONS. ANY USE OR POSSESSION OF THE INSTRUCTIONS BY YOU IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

BY USING THE INSTRUCTIONS, YOU REPRESENT, WARRANT AND CERTIFY THAT: YOU ARE AN AUTHORIZED REPRESENTATIVE OF THE LEGAL ENTITY YOU REPRESENT; YOU HAVE READ THIS AGREEMENT AND UNDERSTAND IT, INCLUDING THE CIVIL CODE SECTION BELOW; YOU HAVE THE AUTHORITY TO BIND THE LEGAL ENTITY YOU REPRESENT TO THE TERMS AND CONDITIONS OF THIS AGREEMENT; AND YOU AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS.

1. **DESIGN OF YOUR PRODUCTS.** You acknowledge that QTI has had and will have no participation in or control over – and no responsibility for – the design or assembly of Your products, including drones, the integration of any of the attached Materials into Your products, including drones, or the sale or marketing of Your products, including drones.
2. **ASSUMPTION OF RISK.** You acknowledge that the operation of the Materials, alone or in a product, including drones, is a potentially dangerous activity and may result in significant harm to property or injury or death to persons. You agree to include on Your products prominent warnings of such risks, as may be required by law or regulation and as may be necessary or prudent to advise users of such risks. You, and not QTI, assume all risks and liabilities that may result from the use of the Materials, whether or not modified by You and whether or not implemented in connection with a reference design provided by QTI or any of its Affiliates.
3. **SAFETY PRECAUTIONS AND PROCEDURES.** You will operate Your products, including drones, in compliance with any and all safety procedures and precautions as are reasonable for the operation of Your products, including drones. This may include using blade guards on drones or operating any drone sufficiently far away from people, property or other hazardous structures e.g., electricity lines. In addition, You will operate Your products, including drones, in compliance with all applicable laws and regulations, including safety and operational guidelines, that apply to the use of Your products, including drones.
4. **WARRANTY.** THE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAWS, QTI AND ITS AFFILAITES EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, TITLE, FITNESS FOR A PARTICULAR PURPOSE AND WARRANTIES THAT THE MATERIALS ARE FREE FROM THE RIGHTFUL CLAIM OF ANY THIRD PARTY, BY WAY OF INFRINGEMENT OR THE LIKE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY QTI OR ITS AUTHORIZED REPRESENTATIVES SHALL CREATE OR EXTEND ANY WARRANTY.

5. **LIMITATION OF LIABILITY.** IN NO EVENT SHALL QTI OR ANY OF ITS AFFILIATES BE LIABLE TO YOU FOR ANY INCIDENTAL, CONSEQUENTIAL OR SPECIAL DAMAGES, INCLUDING, BUT NOT LIMITED TO, ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL DAMAGES ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT OR THE USE OR INABILITY TO USE, OR THE DELIVERY OF OR FAILURE TO DELIVER THE MATERIALS EVEN IF QTI OR ITS AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. QTI'S TOTAL, CUMULATIVE LIABILITY FOR DIRECT DAMAGES ARISING FROM OR IN CONNECTION WITH THIS AGREEMENT, WILL BE LIMITED TO A TOTAL AMOUNT OF ONE HUNDRED UNITED STATES DOLLARS (US $100). MULTIPLE CLAIMS WILL BE AGGREGATED TO DETERMINE THE SATISFACTION OF THIS LIMIT.
6. **INDEMNITY.** You agree to defend, indemnify and hold QTI, its Affiliates, employees, directors, agents, licensors, successors and assignees (each an "Indemnified Party") harmless from any and all claims, penalties, demands, causes of action, liabilities, lawsuits, or damages, including attorneys' fees and costs, that result from or relate to the Materials or any product, including drones, made, used, sold, imported, exported, or distributed by You which uses the Materials or any part or derivative work thereof, even where such product uses the Materials without modification and even where the design of such product is identical to the design of any reference product, including drones. This indemnification includes, without limitation, any claims for damages to property or injury or death to persons and any investigation, enforcement action, civil penalty, or other action conducted or cost imposed by the United States Federal Aviation Administration (FAA) or any governmental entity of the United States or any other government. If any third party asserts a claim or initiates an action against an Indemnified Party for which You are responsible under this Section, QTI shall promptly notify You when it becomes aware of such claim or action, provided, however, that any delay in notification shall not relieve You from Your indemnification obligations under this Agreement. QTI shall have the right to participate in the defense of such claim or action, including any related settlement negotiations. No such claim or action may be settled or compromised without QTI's express written consent, which may be conditioned upon the execution of a release of all claims against the Indemnified Parties by the party bringing such claim or action.
7. **GENERAL RELEASE.**
   (a) Release. In consideration of QTI's allowing You to use the Materials, You on behalf of Yourself, Your, affiliates, agents, successors and assigns, hereby fully and forever release and discharge QTI (and all of its officers, directors, employees, agents, successors, assigns, control persons, subsidiaries and Affiliated companies, together the "Released Parties"), from any and all claims, demands, liabilities, obligations, responsibilities, suits, actions and causes of action against any of the Released Parties, whether liquidated or unliquidated, fixed or contingent, known or unknown, presently existing or which, through the passage of time, might arise in the future, related to or arising out of Your use of the Materials.
   (b) Waiver and Acknowledgement. You hereby expressly waives all rights under Section 1542 of the Civil Code of the State of California, and under any and all similar laws of any governmental entity. You hereby confirm that you are aware that said Section 1542 of the Civil Code provides as follows: "A general release does not extend to claims which the creditor does not know or suspect to exist in his favor at the time of executing the release, which if known by him must have materially affected his settlement with the debtor."
8. **INSURANCE.** If You make or distribute drones, or participate in making or distributing drones, You agree to and will maintain (throughout the time You are using any of the Materials in connection with such drones) insurance providing adequate coverage for potential product liability, personal injury, property damage, and privacy claims and litigation associated with such drones and/or Materials.

C References

C.1 Related Documents

Title	Number
Qualcomm Technologies	
Qualcomm Snapdragon Navigator User Guide	80-P4698-18
Qualcomm Snapdragon Navigator Example API Programs	https://github.com/ATLFlight/snav_api_examples

C.2 Acronyms and Terms

Acronym or term	Definition
ESC	Electronic speed controllers
RC	Radio controller
VIO	Visual inertial odometry
VOA	Visual obstacle avoidance