SimpleBGC 2.3 serial protocol specification

by Aleksey Moskalenko

Revision history

rev. 0.1 - 07.05.2013: this is first revision

rev. 0.2 - 29.05.2013: modified R and W commands

rev. 0.3 – 18.06.2013: add 'r' command

rev. 0.4 – 27.06.2013: add 'g' command; add SKIP_GYRO_CAL var and re-arrange 'W' command parameters order

rev. 0.5 – 12.07.2013: add followMode, followDeadband, followExpoRate variables to 'W' command

rev. 0.6 – 24.07.2013: add FOLLOW_OFFSET; add 10 reserved bytes

rev. 0.7 – 14.08.2013: some minor errors corrected

rev. 0.8 – 09.09.2013: errors: page 2, “modulo 256”; add control command 'C'; add battery monitoring settings and command 'B'; add helper command 'H'; extended RC mapping settings; add RC_MIX settings; add command 'T'; add command 'M' and 'm'; add 'E' command; modified 'D' command;

Message format

Communications is initiated from the GUI side by sending *outgoing* commands. The controller board may do some action and send response (further named as *incoming* commands). Each command consists of the *header* and the *body*, both with checksum. Commands with the wrong header or body checksum, or with the body size that differs from expected, should be ignored.

Board can work on different serial baud rate, so the GUI should find proper baud rate by sending 'V' command on every speed ant wait for response, until valid response is received.

Make a small delay after sending each command (20ms minimal) to prevent overflows of the input buffer.

Input and output commands have the same format, described below:

Header:

character '>'
command ID - 1u
data_size – 1u, may be zero
header checksum = (command ID + data_size) modulo 256 - 1u

Body:

[array of bytes *data_size* length]
body checksum - 1u

Checksum is calculated as a sum of all bytes modulo 256.

Example: outgoing command to read Profile2:

0x3E (>)	0x52 (R)	0x01	0x53	0x01	0x01
	command id	data size	header checksum	data	body checksum
header				body	

Data type notation

- 1u – 1 byte unsigned
- 1s – 1 byte signed
- 2u – 2 byte unsigned (little-endian order)
- 2s – 2 byte signed (little-endian order)
- 4f – float
- 4s – 4 bytes signed (little-endian order)
- string – ASCII character array, first byte is array size
- Nb – byte array size N

Incoming commands

V – version information

- BOARD_VER - 1u (split into decimal digits X . X, for example 10 means 1.0)
- FIRMWARE_VER - 2u (split into decimal digits X . XX . X, for example 2305 means 2.30b5)
- DEBUG_MODE - 1u (should hide DEBUG output if DEBUG_MODE = 0)
- BOARD_FEATURES - 2u
- reserved – 12b

R – Receive parameters

Receive parameters for single profile together with general parameters .

Profile parameters:

- PROFILE_ID – 1u (ID of profile to read, starting from 0)
- for(axis in [ROLL, PITCH, YAW]) {
    - P - 1u
    - I - 1u (multiplied by 100)
    - D - 1u
    - POWER - 1u
    - INVERT – 1u   (checked=1, not checked=0)
    - POLES - 1u
- }
- EXT_FC_GAIN_ROLL - 1s
- EXT_FC_GAIN_PITCH – 1s
-
- for(axis in [ROLL, PITCH, YAW]) {
    - RC_MIN_ANGLE - 2s
    - RC_MAX_ANGLE - 2s
    - RC_MODE - 1u
    - RC_LPF – 1u
    - RC_SPEED – 1u
    - RC_FOLLOW - 1u
- }
- GYRO_TRUST – 1u
- USE_MODEL – 1u
- PWM_FREQ – 1u
- SERIAL_SPEED – 1u
- RC_TRIM_ROLL - 1s
- RC_TRIM_PITCH - 1s

- RC_TRIM_YAW - 1s
- RC_DEADBAND - 1u
- RC_EXPO_RATE - 1u
-
- RC_VIRT_MODE – 1u
- RC_MAP_ROLL – 1u
- RC_MAP_PITCH – 1u
- RC_MAP_YAW – 1u
- RC_MAP_CMD – 1u
- RC_MAP_FC_ROLL – 1u
- RC_MAP_FC_PITCH – 1u
-
- RC_MIX_FC_ROLL - 1u
- RC_MIX_FC_PITCH - 1u
-
- FOLLOW_MODE – 1u
- FOLLOW_DEADBAND – 1u
- FOLLOW_EXPO_RATE – 1u
- FOLLOW_OFFSET_ROLL – 1s
- FOLLOW_OFFSET_PITCH – 1s
- FOLLOW_OFFSET_YAW - 1s
- FOLLOW_ROLL_MIX_START - 1u
- FOLLOW_ROLL_MIX_RANGE - 1u

General parameters:

- AXIS_TOP – 1s
- AXIS_RIGHT – 1s
- GYRO_LPF – 1u
- I2C_INTERNAL_PULLUPS – 1u
- SKIP_GYRO_CALIB – 1u
-
- RC_CMD_LOW – 1u
- RC_CMD_MID – 1u
- RC_CMD_HIGH – 1u
-

- MENU_CMD_1 - 1u
- MENU_CMD_2 - 1u
- MENU_CMD_3 - 1u
- MENU_CMD_4 - 1u
- MENU_CMD_5 - 1u
- MENU_CMD_LONG - 1u
-
- OUTPUT_ROLL - 1u
- OUTPUT_PITCH – 1u
- OUTPUT_YAW – 1u
-
- BAT_THRESHOLD_ALARM – 2s
- BAT_THRESHOLD_MOTORS – 2s
- BAT_COMP_REF – 2s
-
- BEEPER_MODES – 1u
-
- RESERVED_BYTES - 6u
-
- CUR_PROFILE_ID – 1u (profile ID which is currently active in the controller)

D – real-time data

- for(axis in [ROLL, PITCH, YAW]) {
    - ACC – 2s
    - GYRO – 2s
- }
-
- DEBUG1 – 2s
- DEBUG2 – 2s
- DEBUG3 – 2s
- DEBUG4 – 2s
- RC_ROLL - 2s
- RC_PITCH - 2s
- RC_YAW - 2s
- RC_CMD – 2s

- EXT_FC_ROLL – 2s
- EXT_FC_PITCH – 2s
- ANGLE_ROLL – 2s (in 0.1 degree)
- ANGLE_PITCH – 2s
- ANGLE_YAW – 2s
- RC_ANGLE_ROLL - 2s (in 0.1 degree)
- RC_ANGLE_PITCH - 2s
- RC_ANGLE_YAW - 2s
- CYCLE_TIME - 2u
- I2C_ERROR_COUNT - 2u
- ERROR_CODE – 1u
- BAT_LEVEL - 2u
- OTHER_FLAGS - 1u
- CUR_PROFILE - 1u

C – confirmation of previous command

- CMD – 1u
- DATA – depends on CMD

Board sends confirmation on commands: A, G, P, W, etc. DATA is empty unless mentioned in command description.

I - Information about actual RC control state

- for(axis in [ROLL, PITCH, YAW]) {
    - ANGLE - 2s
    - RC_ANGLE - 2s
    - RC_SPEED - 2s
- }

Outgoing command

V – request version information

D – request real-time data

A – calibrate accelerometer

G – calibrate EXT_FC gains

F – reset to factory defaults

- PROFILE_ID – 1u – profile to reset

P – calibrate poles and direction

R – request parameters

- PROFILE_ID – 1u – profile to load

W – write parameters
Data structure is the same as for 'R' incoming command.

r – reset device

O – calibrate follow offset

B - calibrate battery (voltage sensor)

- ACTUAL_VOLTAGE - 2u

C – control gimbal movement

- CONTROL_MODE – 1u
- SPEED_ROLL – 2s
- ANGLE_ROLL – 2s
- SPEED_PITCH – 2s
- ANGLE_PITCH – 2s
- SPEED_YAW – 2s
- ANGLE_YAW – 2s

T - trigger output pin

- PIN_ID - 1u
- STATE - 1u

Confirmation is sent only if pin is not used for input and is triggered.

M - switch motors ON
Confirmation send 'M'

m - switch motors OFF
Confirmation send 'm'

E - execute menu command

- CMD_ID - 1u

H – pass helper data

- FRAME_ACC_X – 2s
- FRAME_ACC_Y – 2s
- FRAME_ACC_Z – 2s
- FRAME_ANGLE_ROLL – 2s

- FRAME_ANGLE_PITCH – 2s

I - Request information about RC control state
See the **I** incoming command.

NOTE: requests for API extension please send to alexmos@simplebgc.com

Variables description and range

Name	Type	Min	Max	Possible values, remarks
Version information ('V' command)				
BOARD_VER	1u			Multiplied by 10: 3.0 => 30
FIRMWARE_VER	1u			Multiplied by 10: 2.3 => 23

BOARD_FEATURES	2u			Bit set:
				BOARD_FEATURE_3AXIS = 1
				BOARD_FEATURE_BAT_MONITORING = 2

Parameters ('R', 'W' commands)				
PROFILE_ID	1u	0	2 (255)	profile ID to read or write. To read or write current (active) profile, specify 255.
P	1u	0	50	
I	1u	0	50	divided by 100 when displayed in the GUI
D	1u	0	50	
POWER	1u	0	255	
INVERT	1u	0	1	
POLES	1u	0	255	
EXT_FC_GAIN	1s	-127	127	
RC_MIN_ANGLE	2s	-180	180	
RC_MAX_ANGLE	2s	-180	180	

RC_MODE	1u			RC_MODE_ANGLE = 0
				RC_MODE_SPEED = 1

RC_LPF	1u	0	16	
RC_SPEED	1u	0	50	
RC_FOLLOW	1u	-127	127	
GYRO_TRUST	1u	0	255	
USE_MODEL	1u	0	1	

PWM_FREQ	1u			PWM_FREQ_LOW = 0
				PWM_FREQ_HIGH = 1

SERIAL_SPPED	1u			115200 = 0 57600 = 1

				38400 = 2 19200 = 3 9600 = 4
RC_TRIM_ROLL RC_TRIM_PITCH RC_TRIM_YAW	1s	-127	127	
RC_DEADBAND	1u	0	255	
RC_EXPO_RATE	1u	0	100	

RC_VIRT_MODE	1u			Mode of RC_ROLL input pin operation:
				RC_VIRT_MODE_NORMAL = 0
				RC_VIRT_MODE_CPPM = 1
				RC_VIRT_MODE_SBUS = 2 (BOARD_VER >= 30)
				RC_VIRT_MODE_SPEKTRUM = 3 (BOARD_VER >= 30)

RC_MAP_ROLL
RC_MAP_PITCH
RC_MAP_YAW
RC_MAP_CMD
RC_MAP_FC_ROLL
RC_MAP_FC_PITCH
	1u			Assigns pin input or virtual channel (in serial modes), and specifies input mode.
				
				INPUT_NO = 0
				PWM mode:
				RC_INPUT_ROLL = 1
				RC_INPUT_PITCH = 2
				EXT_FC_INPUT_ROLL = 3
				EXT_FC_INPUT_PITCH = 4
				RC_INPUT_YAW = 5 (BOARD_VER >= 30)
				
				Analog mode:
				BOARD_VER < 30: the same as above + 32
				RC_INPUT_ROLL = 33
				RC_INPUT_PITCH = 34
				EXT_FC_INPUT_ROLL = 35
				EXT_FC_INPUT_PITCH = 36
				
				BOARD_VER >= 30:
				ADC1 = 33
				ADC2 = 34
				ADC3 = 35
				
				Serial mode (CPPM/SBUS/SPEKTRUM):
				virtual channel (1..31) + 64 (6th bit is set)

RC_MIX_FC_ROLL RC_MIX_FC_PITCH	1u			Add FC channel to selected RC channels with given rate. bits 0..5: mix rate. For example, 0 - no mix (100% RC) 32 - 50% RC, 50% FC, 63 - 0% RC, 100% FC bits 6,7: target RC channel 0 - no mix 1 - ROLL 2 - PITCH 3 - YAW

FOLLOW_MODE	1u			FOLLOW_MODE_DISABLED=0
				FOLLOW_MODE_FC=1

				FOLLOW_MODE_PITCH=2
FOLLOW_DEADBAND	1u	0	255	
FOLLOW_EXPO_RATE	1u	0	100	
FOLLOW_OFFSET_ROLL FOLLOW_OFFSET_PITCH FOLLOW_OFFSET_YAW	1s	-127	127	
FOLLOW_ROLL_MIX_START	1u	0	90	
FOLLOW_ROLL_MIX_RANGE	1u	0	90	

AXIS_TOP
AXIS_RIGHT
	1s			X = 1
				Y = 2
				Z = 3
				-X = -1
				-Y = -2
				-Z = -3

GYRO_LPF	1u	0	5	0 means no LPF, 5 means LPF at maximum
I2C_INTERNAL_PULLUPS	1u	0	1	
SKIP_GYRO_CALIB	1u	0	1	

RC_CMD_LOW
RC_CMD_MID
RC_CMD_HIGH
 	
MENU_CMD_1..5
MENU_CMD_LONG
	1u			CMD_NO = 0
				CMD_PROFILE1 = 1
				CMD_PROFILE2 = 2
				CMD_PROFILE3 = 3
				CMD_SWAP_PITCH_ROLL = 4
				CMD_SWAP_YAW_ROLL = 5
				CMD_CALIB_ACC = 6
				CMD_RESET = 7
				CMD_SET_ANGLE = 8
				CMD_CALIB_GYRO = 9

OUTPUT_ROLL
OUTPUT_PITCH
OUTPUT_YAW
	1u			DISABLED = 0
				ROLL = 1
				PITCH = 2
				YAW = 3

BAT_THRESHOLD_ALARM	2s	-3000	3000	Negative means means alarm is disabled *Units: 0.01V*
BAT_THRESHOLD_MOTORS	2s	-3000	3000	Negative value means function is disabled *Units: 0.01V*
BAT_COMP_REF	2s	-3000	3000	Negative value means compensation is disabled. *Units: 0.01V*

BEEPER_MODES	1u			BEEPER_MODE_CALIBRATE=1
				BEEPER_MODE_CONFIRM=2
				BEEPER_MODE_ERROR=4

				BEEPER_MODE_ALARM=8
CUR_PROFILE	1u	0	2	active profile
Real-time data ('D' command)				
ACC GYRO RESERVED_SENSOR	2s			raw data from sensors
DEBUG	2s			debug variables
RC_ROLL RC_PITCH RC_YAW	2s	1000	2000	RC control channels values (PWM or normalized analog)
RC_CMD	2s	1000	2000	RC command channel value (PWM or normalized analog)
EXT_FC_ROLL EXT_FC_PITCH	2s	1000	2000	External FC PWM values. May be zero if their inputs are mapped to RC control or command.
ANGLE_ROLL ANGLE_PITCH ANGLE_YAW	2s 2s	-900 -7200 -7200	900 7200 7200	Gimbal angles. After 2 full turns, angle is cycled *Units: 0.1 degree*
CYCLE_TIME	2u			
I2C_ERROR_COUNT	2u			Number of registered errors on I2C bus
ERROR_CODE	1u			Bit set of system errors: ERR_NO_SENSOR 1<<0 ERR_CALIB_ACC 1<<1 ERR_SET_POWER 1<<2 ERR_CALIB_POLES 1<<3 ERR_SERIAL 1<<5
BAT_LEVEL	2u			Battery voltage *Units: 0.01 volt*
OTHER_FLAGS	1u			bit0 set - motors turned ON bit1..7 - reserved
CUR_PROFILE	1u	0	2	Current (active) profile
Control ('C' command)				

CONTROL_MODE*	1u			MODE_NO_CONTROL=0
				MODE_SPEED=1
				MODE_ANGLE=2
				MODE_SPEED_ANGLE=3
				* See Fig.1 below

SPEED_ROLL SPEED_PITCH SPEED_YAW	2s	- - -	- - -	Depends on the CONTROL_MODE: • MODE_SPEED – camera travels with the given speed. Angle is ignored. • MODE_ANGLE – value is ignored • MODE_SPEED_ANGLE – camera travels with the given speed while the actual angle matches

				the given angle. Additionally, PID controller keeps the given angle. This mode allows the most precise and error-proof control. *Units: 0,1220740379 degree/sec*
ANGLE_ROLL ANGLE_PITCH ANGLE_YAW	2s	-900 -7200 -7200	900 7200 7200	Depends on the CONTROL_MODE: • MODE_SPEED – value is ignored • MODE_ANGLE - camera travels to the given angle. Speed depends on SPEED setting and acceleration settings in the GUI. *Units: 0.1 degree. If angle exceed 2 full turns, it should be cycled.*
Notes: • Serial control overrides RC control. To switch back to RC, send this command with the mode=MODE_NO_CONTROL and all zeros data. • Send this command with rate 50Hz or less • See Appendix A for source code example				
Trigger pin ('T' command)				
PIN_ID	1u			Triggers pin only if it is not used for input RC_INPUT_ROLL = 1 RC_INPUT_PITCH = 2 EXT_FC_INPUT_ROLL = 3 EXT_FC_INPUT_PITCH = 4 RC_INPUT_YAW = 5 (BOARD_VER >= 30) PIN_AUX1* = 16 PIN_AUX2* = 17 PIN_AUX3* = 18 PIN_BUZZER* = 32 * On boards v1.x (based on Atmega328p) PIN_AUX1..3 are not mapped to connectors, and should be soldered to pin2, pin11, pin12 of MCU correspondingly. PIN_BUZZER is mapped to pin32 of MCU.

STATE	1u			LOW = 0
				HIGH = 1
				LOW - pin can sink up to 40mA
				HIGH - pin can source up to 40mA

RC control state ('I' command)				
ANGLE_ROLL ANGLE_PITCH ANGLE_YAW	2s	-900 -7200 -7200	900 7200 7200	Actual angle measured by IMU. After 2 full turns, angle is cycled *Units: 0.1 degree*
RC_ANGLE_ROLL RC_ANGLE_PITCH RC_ANGLE_YAW	2s	-900 -7200 -7200	900 7200 7200	Target angle that gimbal should keep. Angle is set by RC or control command 'C'. *Units: 0.1 degree*
RC_SPEED_ROLL RC_SPEED_PITCH RC_SPEED_YAW	2s	- - -	- - -	Target speed that gimbal should keep. Speed is set by RC or control command 'C'. Zero speed means control is idle (target is reached)

				Units: 0,1220740379 degree/sec
Execute menu command ('E' command)				
CMD_ID	1u			Executes a menu command (acts like the menu button or RC control channel) See the RC_CMD_LOW parameter inside the 'R' command for available menu commands.
Helper ('H' command) Pass helper data from an outer system. Used to increase precision of the stabilization				
FRAME_ACC_X FRAME_ACC_Y FRAME_ACC_Z	2s	-	-	Linear acceleration of the gimbal measured in the 'outer' system. Relationship between the outer system and the sensor's system is shown on the fig.2. Note: The **Y** axis of the outer system always points the same direction as **ROLL** axis. It means that ACC vector, measured in the ground system, should be translated to 'outer' system by rotating it around Z axis by the YAW angle. *Units: 1g/512 ≈ 0,019160156 m/s^2*
FRAME_ANGLE_ROLL FRAME_ANGLE_PITCH	2s	-900 -900 -	900 900 -	Inclination of the outer frame in the 'outer' system. *Units: 0.1 degree*
Notes: • FRAME_ANGLE is used only if “External FC Gain” setting is zero. • FRAME_ACC is used only if “Acceleration compensations” setting is disabled. • This command is useless for 3-axis systems, until YAW encoders will be implemented to know exact YAW angles. • Send this command with rate 50Hz or less				

* The difference between control modes is illustrated on the picture below:

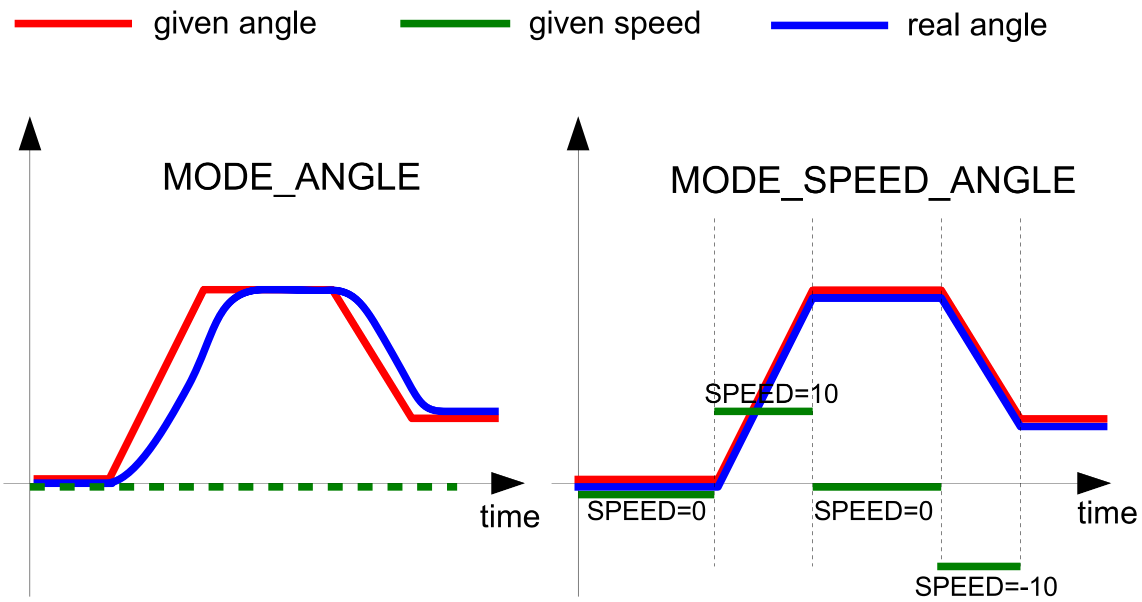

Fig.1 – Control modes

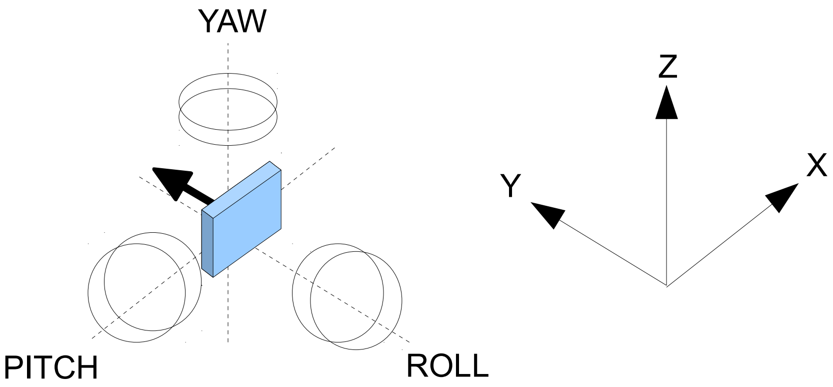

Fig.2 – relationship between the gimbal axes and the ground system axes

Appendix A: Example program (C++, Arduino) to demonstrate how to use the control command 'C'

```
/******************************************************************************
  This is example sketch for Arduino.
  Shows how to control SimpleBGC-driven gimbal via Serial API.
  API specs are available at http://www.simplebgc.com/eng/downloads/

  Connection:
         Arduino GND -> SimpleBGC GND
         Arduino TX -> SimpleBGC RX

         Power Arduino separatly or via +5V from onboard FTDI connector

  (C) Aleksey Moskalenko
*******************************************************************************/
#include <inttypes.h>


// default is 115200 but may be changed in the Advanced tab of the SimpleBGC GUI
#define SERIAL_SPEED 115200

// delay between commands, ms
#define SBGC_CMD_DELAY 20

// Some definitions required to send commands
#define SBGC_CMD_CONTROL             'C'
#define SBGC_CMD_TRIGGER             'T'

#define SBGC_CONTROL_MODE_SPEED                                  1
#define SBGC_CONTROL_MODE_ANGLE                                  2
#define SBGC_CONTROL_MODE_SPEED_ANGLE         3


// Pins that may be triggered
#define SBGC_RC_INPUT_ROLL 1
#define SBGC_RC_INPUT_PITCH 2
#define SBGC_RC_INPUT_EXT_ROLL 3
#define SBGC_RC_INPUT_EXT_PITCH 4
#define SBGC_RC_INPUT_YAW 5 // not connected in 1.0 board
#define SBGC_PIN_AUX1 16
#define SBGC_PIN_AUX2 17
#define SBGC_PIN_AUX3 18
#define SBGC_PIN_BUZZER 32



// Conversion from degree/sec to units that command understand
#define SBGC_SPEED_SCALE  (1.0f/0.1220740379f)

// Holder for command parameters
typedef struct {
         uint8_t mode;
         int16_t speedROLL;
         int16_t angleROLL;
         int16_t speedPITCH;
         int16_t anglePITCH;
         int16_t speedYAW;
         int16_t angleYAW;
} SBGC_cmd_control_data;

typedef struct {
         uint8_t pin;
         int8_t state;
} SBGC_cmd_trigger_data;




// This helper function formats and sends a command to SimpleBGC Serial API
void SBGC_sendCommand(uint8_t cmd, void *data, uint8_t size) {
         uint8_t i, checksum=0;

         // Header
         Serial.write('>');
         Serial.write(cmd);
         Serial.write(size);
         Serial.write(cmd+size);
         // Body
         for(i=0;i<size;i++) {
                  checksum+= ((uint8_t*)data)[i];
                  Serial.write(((uint8_t*)data)[i]);
         }
         Serial.write(checksum);
}
/***************************************************************************/




#define LED_ON() {   digitalWrite(13, HIGH); }
#define LED_OFF() {    digitalWrite(13, LOW); }

void blink_led(uint8_t cnt) {
         for(uint8_t i=0; i<cnt; i++) {
                  LED_OFF();
                  delay(200);
                  LED_ON();
                  delay(300);
         }
}


void setup() {
  Serial.begin(SERIAL_SPEED);
  pinMode(13, OUTPUT);

  // Take a pause to let gimbal controller to initialize
  //delay(5000);
}


void loop() {
  SBGC_cmd_control_data c = { 0, 0, 0, 0, 0, 0, 0 };
  SBGC_cmd_trigger_data t = { 0, 0 };


  // Move camera to initial position (all angles are zero)
  c.mode = SBGC_CONTROL_MODE_ANGLE;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(3000);

  // LED ON: start demo
  LED_ON();


  // Demo 1. PAN and ROLL gimbal by 60 and 30 degrees both sides and return back. Actual speed depends on PID setting.
  // Whait 5 sec to finish
  c.mode = SBGC_CONTROL_MODE_ANGLE;
  c.angleROLL = 300;
  c.angleYAW = 600;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(3000);
  c.angleROLL = -300;
  c.angleYAW = -600;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(3000);
  // .. and back
  c.angleYAW = c.angleROLL = 0;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(3000);
  blink_led(2);


         // Demo 2. Pitch gimbal down with constant speed 10 degree/sec  by 50 degree (it takes 5 sec)
         // (this is simplified version of speed control. To prevent jerks, you should add acceleration and de-acceleration
phase)
         c.mode = SBGC_CONTROL_MODE_SPEED;
         c.speedPITCH = 10 * SBGC_SPEED_SCALE;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(5000);
  // Stop
         c.speedPITCH = 0;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(1000);
  // .. and back
         c.speedPITCH = -10 * SBGC_SPEED_SCALE;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(5000);
  // Stop
         c.speedPITCH = 0;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(1000);
  blink_led(2);
  // Demo3: Return control back to RC for 5 seconds
  c.mode = 0;
  SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
  delay(5000);
  blink_led(2);



  // Demo 4. More complicated example: Pitch gimbal by 40 degrees with the full control of speed and angle.
  //     - send control command with the fixed frame rate
  //     - angle is calculated by the integration of the speed
         float speed = 0, angle = 0;
         c.mode = SBGC_CONTROL_MODE_SPEED_ANGLE;
         // acceleration phase
         while(angle < 20.0f) {
                  speed+= 0.5f;
                  c.speedPITCH = speed * SBGC_SPEED_SCALE;
                  angle+= speed * SBGC_CMD_DELAY / 1000.0f;  // degree/sec -> degree/ms
                  c.anglePITCH = (int16_t)(angle*10.0f);
           SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
                  delay(SBGC_CMD_DELAY);
         }
         // de-acceleration phase
         while(angle < 40.0f && speed > 0.0f) {
                  speed-= 0.5f;
                  c.speedPITCH = speed * SBGC_SPEED_SCALE;
                  angle+= speed * SBGC_CMD_DELAY / 1000.0f;
                  c.anglePITCH = (int16_t)(angle*10.0f);
           SBGC_sendCommand(SBGC_CMD_CONTROL, &c, sizeof(c));
                  delay(SBGC_CMD_DELAY);
         }
  blink_led(2);


         // Demo 5: Trigger AUX1 pin state HIGH and make BUZZER to buzz for 0.5 sec
         t.pin = SBGC_PIN_AUX1;
         t.state = 1;
         SBGC_sendCommand(SBGC_CMD_TRIGGER, &t, sizeof(t));
         t.pin = SBGC_PIN_BUZZER;
         t.state = 1;
         SBGC_sendCommand(SBGC_CMD_TRIGGER, &t, sizeof(t));
         delay(500);
         t.pin = SBGC_PIN_AUX1;
         t.state = 0;
         SBGC_sendCommand(SBGC_CMD_TRIGGER, &t, sizeof(t));
         t.pin = SBGC_PIN_BUZZER;
         t.state = 0;
         SBGC_sendCommand(SBGC_CMD_TRIGGER, &t, sizeof(t));
  blink_led(2);


         LED_OFF();
}
```