V1.0.3

2025.06

Gimbal

Private Protocol

Using this Manual – Legend

 Important

 Tips

 Explanation

Revision History

Date	Document Version	Firmware Version	Protocol Version
2024.12.13	V1.0.0	V3.5 and higher	V1.0

Date	Document Version	Firmware Version	Protocol Version
2025.01.03	V1.0.1	V3.5 and higher	V1.0

Date	Document Version	Firmware Version	Protocol Version
2025.02.26	V1.0.2	V3.5 and higher	V1.0

1. Host-to-Gimbal Data Packet Structure:
   1.1 Added detailed explanation for command trigger counter (Note 2)[P3];
   1.2 Added descriptions for "start gimbal" and "stop gimbal"commands.[P3]

Date	Document Version	Firmware Version	Protocol Version
2025.06.09	V1.0.3	V3.5 and higher	V1.0

1. Add Appendix 3: Example Code.[P10]
2. Add Appendix 4: Example Data Packet and Conversion.[P21]

Catalog

Interface Configuration

Serial Port Configuration

- Voltage level: TTL
- Data bits: 8
- Stop bits: 1
- Parity: None
- Communication mode: full-duplex
- Logic: Acknowledge-based. The host sends a data packet first; the gimbal replies after receiving a valid packet.
- Baud rate: 115200
- Frequency: ⩾ 50Hz (higher frequency improves control performance).
- Byte order: Little-endian (except CRC).

Attitude Angles and Angular Velocities

In this document:

Camera attitude angles	Camera's Euler angles in the world system (NED)
Carrier attitude angles	Carrier's Euler angles in the world system (NED)
Camera attitude angular velocities	Camera's Euler angular velocities in the world system (NED)
Carrier attitude angular velocities	Carrier's Euler angular velocities in the world system (NED)
Camera relative angles	Camera's Euler angles in the carrier corrdinate system，also known as frame angles or joint angles
Camera & carrier attitude difference angles	The difference between camera attitude angles and carrier attitude angles

For coordinate system definitions, see Appendices 1 and 2.

Host-to-Gimbal Data Packet Structure

```
#pragma pack(1)
typedef struct
{
    uint8_t sync[2]; //Note 1
    struct
    {
        uint8_t trig:3; //Note 2
        uint8_t value:5; //Note 3
    } cmd;
    struct
    {
        uint8_t:3;
        int8_t fl_sens:5; //Note 4
    } aux;
    struct
    {
        uint8_t:3;
        uint8_t go_zero:1; //Note 5
        uint8_t wk_mode:2; //Note 6
        uint8_t op_type:2; //Note 7
        int16_t op_value; //Note 8
    } gbc[3]; //Note 9
    struct
    {
        uint8_t:7;
        uint8_t valid:1; //Note 10
        int16_t angle[3]; //Note 11
        int16_t accel[3]; //Note 12
    } uav; //Note 13
    struct
    {
        uint32_t vert_fov1x:7; //Note 14
        uint32_t zoom_value:24; //Note 15
        uint32_t reserved:1; //Note 16
        float target_angle[2]; //Note 17
    } cam;
    uint8_t crc[2];//Note 18
    } Gcu2GbcPkt_t;
#pragma pack()
```

Note 1: Sync header 0xA9 0x5B

Note 2: Command trigger counter. When the sent command code is the same as the previous data packet, the counter value must be modified to make the gimbal repeatedly execute the command. If the command code differs from the previous data packet, the gimbal will immediately execute the command regardless of whether the counter value is changed.

Note 3: Command code. The code is sent once and must be cleared (set to 0) after the host confirms the gimbal's response.
0 - Undefined; 1 - Gyro calibration; 2 - Start gimbal; 3 - Stop gimbal;
4 - Manual control; 5 - Click to aim

Wait for the gimbal to confirm temperature control readiness before initiating gyroscope calibration. The device must remain motionless throughout the calibration, which typically lasts several seconds.

Start gimbal: Enables gimbal motors and activates stabilization, transitioning the gimbal to normal operational mode. This command does not initiate rotation or execute tasks. The gimbal activates automatically upon power-up. Use this command only to restore normal operation after issuing a "Stop gimbal" command.

Stop gimbal: Deactivates gimbal motors and disables stabilization. This command does not halt ongoing rotation or terminate active tasks. Use this command exclusively when storing the gimbal in a powered-on state.

Note 4: FPV sensitivity, [-16, 15]. Only effective in FPV mode. Higher values increase responsiveness but reduce vibration damping.

Note 5: This data fluctuation triggers the gimbal returning its neutral position.

Note 6: Operating mode
0 - Follow. When the carrie's attitude changes, the camera rotates with the carrier while the gimbal provides stabilization to mitigate shaking within specified limits. This mode controls the camera & carrier attitude difference angles or camera attitude angular velocities. Tilt protection remains active in this mode.

1 - Lock. When the carrier's attitude changes, the camera maintains its orientation by locking the current attitude angles instead of rotating with the carrier. This mode controls the camera attitude angles or camera attitude angular velocities. Tilt protection remains active in this mode.

2 - FPV. When the carrier's attitude changes, the camera rotates with the aircraft but can mitigate a certain degree of shaking. This mode controls the camera relative angles. When any axis is set to FPV mode, the remaining axes are automatically switched to FPV mode. Tilt protection is disabled in FPV mode.

Tilt protection: When the gimbal's mounting platform tilts beyond the protection angle, the gimbal will activate protection mode and return to its neutral position, rendering it uncontrollable. The protection angle can be adjusted in the gimbal configuration software GimbalConfig.

Do not set pitch or roll axis to follow mode.

Note 7: Controlling type

0 - Angle control (available in follow, lock, and FPV modes). Controls the camera attitude angles, camera & carrier attitude difference angles or camera relative angle.

1 - Scaled angular velocity control (available in follow and lock modes). Controls the camera attitude angular velocity, but the angular velocity decreases as the camera's zoom ratio increases, ensuring consistent panning speed across zoom ratio.

2 - True angular velocity control (available in follow and lock modes). Controls the camera attitude angular velocity directly, independent of the camera's zoom ratio.

Note 8: Control value

Follow + angle control: Desired camera & carrier attitude difference angle = op_value * 0.01(deg).

Follow + scaled angular velocity control: Desired camera attitude angular velocity = op_value * 0.1 * zoom_factor (deg/s). The camera rotates with the carrier while op_value=0.

Follow + true angular velocity control: Desired camera attitude angular velocity = op_value * 0.1 (deg/s). The camera rotates with the carrier while op_value=0.

Lock + angle control: Desired camera attitude angle = op_value * 0.01(deg).

Lock + scaled angular velocity control: Desired camera attitude angular velocity = op_value * 0.1 * zoom_factor (deg/s). The camera keeps the current attitude angle while op_value=0.

Lock + true angular velocity control: Desired camera attitude angular velocity = op_value * 0.1 (deg/s). The camera keeps the current attitude angle while op_value=0.

FPV + angle control: Desired camera relative angle = op_value * 0.01(deg).

? Zoom_factor: Calculated based on the vertical field of view (vert_fov1x) at 1x zoom and the camera zoom ratio (zoom_value), Zoom_factor = arccot(zoom_value * cot(0.5 * vert_fov1x)) / 18.

Note 9: 0 – Roll; 1 – Pitch; 2 - Yaw

The data in struct gbc are available only in manual control mode.

Note 10: Carrier AHRS data validity flag

0 – Invalid; 1 - Valid

Note 11: Carrier attitude angles, unit: 0.01deg.

0 – Roll, [-18000, 18000); 1 – Pitch, [-9000, 9000]; 2 – Yaw, [-18000, 18000)

Note 12: Carrier accelerations, unit: 0.01m/s2.

0 – Northward (northward is positive); 1 - Eastward (eastward is positive); 2 – Upward (upward is positive)

Note 13: The struct uav provides the carrier data required by the gimbal. Failure to supply this data or providing incorrect data will result in inaccurate gimbal attitude angles during carrier maneuvers.

Note 14: The vertical field of view of camera at 1x zoom, unit: 1deg.

Note 15: The zoom ratio of camera, unit: 0.001x.

Note 16: Reserved

Note 17: The offsite angles of the target, which are the control values of click to aim mode. The origin is the center of the screen, rightward and downward are positive, unit: 1deg.
0 – Horizontal; 1 – Vertical

Note 18: CRC16

Gimbal Responding Data Packet Structure

```
#pragma pack(1)
typedef struct
{
    uint8_t sync[2]; //Note 19
    uint8_t fw_ver; //Note 20
    uint8_t hw_err; //Note 21
    uint8_t inv_flag:1; //Note 22
    uint8_t gbc_stat:3; //Note 23
    uint8_t tca_flag:1; //Note 24
    uint8_t:3;
    struct
    {
        uint8_t stat:3; //Note 25
        uint8_t value:5; //Note 26
    } cmd;
    int16_t cam_rate[3]; //Note 27
    int16_t cam_angle[3]; //Note 28
    int16_t mtr_angle[3]; //Note 29
    uint8_t crc[2]; //Note 30
} Gbc2GcuPkt_t;
#pragma pack()
```

Note 19: Sync Header 0xB5 0x9A

Note 20: Firmware version

Note 21: Hardware failure

Note22: Mounting direction flag

0 – Downward; 1 - Upward

Note 23: Gimbal status

0 – Undefined; 1 – Initializing; 2 – Stopped; 3 – Protected; 4 - Manual control; 5 – Click to aim

Note 24: Temperature control readiness flag

0 – Not ready; 1 – Ready

Gyro calibration should only proceed when the temperature control is ready.

Note 25: Command execution status

0 – In progress; 1 – Success; 2 - Failure

Note 26: Command code response

Note 27: Camera body angular velocities, specifically the angular velocities output by the gimbal's IMU

0 – Pitch; 1 – Roll; 2 – Yaw

Note 28: Camera attitude angles, unit: 0.01deg. For the angle conversion, refer to Appendix 2

0 – Roll; 1 – Pitch; 2 – Yaw

Note 29: Camera relative angles, unit: 0.01deg. These angles are calculated from motor encoder angles and does not rely on the carrier's AHRS data

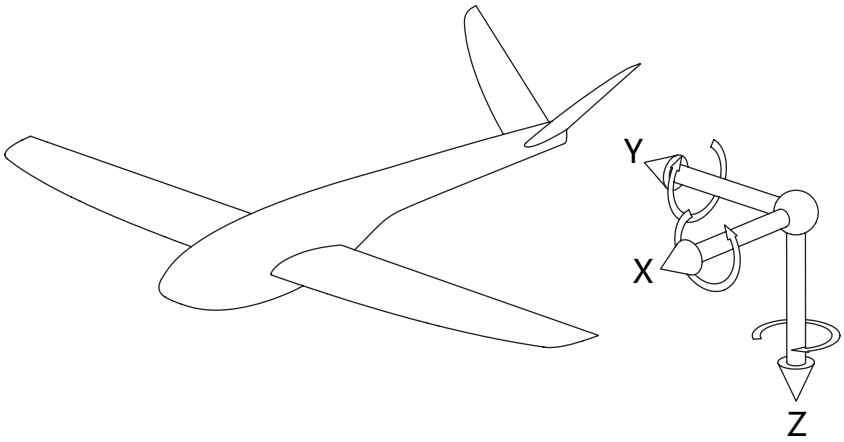
0 – Pitch; 1 – Roll; 2 – Yaw

Note 29: CRC16

CRC Function

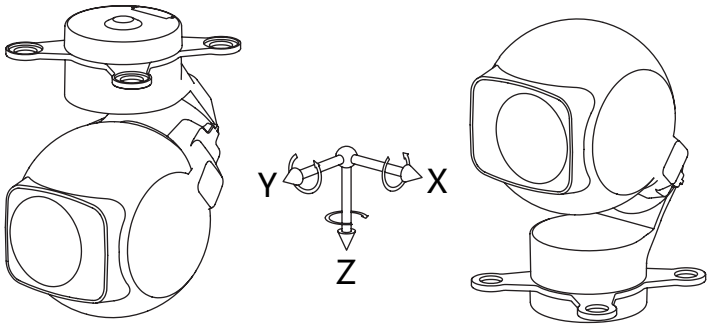
```
uint16_t CalculateCrc16(uint8_t *ptr,uint8_t len)
{
	uint16_t crc;
	uint8_t da;
	uint16_t crc_ta[16]={
	0x0000,0x1021,0x2042,0x3063,0x4084,0x50a5,0x60c6,0x70e7,
	0x8108,0x9129,0xa14a,0xb16b,0xc18c,0xd1ad,0xe1ce,0xf1ef,
	};
	crc=0;
	while(len--!=0)
	{
		da=crc>>12;
		crc<<=4;
		crc^=crc_ta[da^(*ptr>>4)];
		da=crc>>12;
		crc<<=4;
		crc^=crc_ta[da^(*ptr&0x0F)];
		ptr++;
	}
	return(crc);
}
```

Appendix 1: Definition of Carrier Coordinate System



Appendix 2: Definition of Camera Coordinate System and Rotate Order

1. Coordinate system definition：The definitions of Euler angles ϕ, θ, ψ are as illustrated in the coordinate system below.



Downward　　Upward

The damping platform should be parallel to the XOY plane of the carrier. The gimbal should be mounted as close as possible to the C.G. of the carrier

2. Rotate order: Z → Y → X.
3. Angle conversion：
   ψ+=90；
   cam_angle[0]=+θ;
   cam_angle[1]=-ϕ;
   cam_angle[2]=+ψ;

Appendix 3: Example Code

1. Development Environment

   The example execution environment is Keil5 V5.25, using the V5.06 update 6 (build 750) compiler.

2. Controller

   STM32F103C8T6 (Cortex-M3, 72MHz basic frequency).

3. Peripheral Configuration

   - UART1 (PA9-TX, PA10-RX): communicate with the gimbal, with a baud rate of 115200, 8N1.
   - UART2 (PA2-TX, PA3-RX): output debug information with a baud rate of 115200, 8N1.
   - TIM2 (50Hz interrupt for periodically sending control commands).

4. Connection

   UART1 connects the gimbal:

   - STM32's PA9-TX to gimbal’s UART_Rx.
   - STM32's PA10-RX to gimbal’s UART_Rx.
   - Common GND.

   UART2 connects the computer (debug):

   - STM32's PA2-TX to USB-TTL module’s UART_RX。
   - Common GND.

5. Code

```
#include "stm32f10x.h"
#include <stdio.h>
#include <string.h>

// -------------------- Protocol Data Packet Definition --------------------
#pragma pack(1)
typedef struct
{
  uint8_t sync[2]; // Header 0xA9 0x5B
  struct
  {
    uint8_t trig : 3; // Command trigger counter
    uint8_t value : 5; // Command code
  } cmd;
  struct
  {
    uint8_t : 3;
    int8_t fl_sens : 5; // FPV Sensitivity
  } aux;
  struct
  {
    uint8_t : 3;
    uint8_t go_zero : 1; // Neutral position trigger
    uint8_t wk_mode : 2; // Operating mode
    uint8_t op_type : 2; // Controlling type
    int16_t op_value;    // Control value
  } gbc[3];             // 0- Roll, 1- Pitch, 2- Yaw
  struct
  {
    uint8_t : 7;
    uint8_t valid : 1; // Carrier AHRS data validity flag
    int16_t angle[3];  // Carrier attitude angle
    int16_t accel[3];  // Carrier acceleration
  } uav;
  struct
  {
    uint32_t vert_fovlx : 7;  // Vertical field of view of camera at 1x
    uint32_t zoom_value : 24; // Zoom ratio
    uint32_t reserved : 1;
    float target_angle[2]; // Offsite angle of target
  } cam;
  uint8_t crc[2]; // CRC16
} Gcu2GbcPkt_t;
```

```
typedef struct
{
   uint8_t sync[2];     // Header  0xB5 0x9A
   uint8_t fw_ver;      // Firmware version
   uint8_t hw_err;      // Hardware failure
   uint8_t inv_flag : 1; // Mounting direction
   uint8_t gbc_stat : 3; // Gimbal status
   uint8_t tca_flag : 1; // Temperature control readiness flag
   uint8_t : 3;
   struct
   {
      uint8_t stat : 3; // Command execution status
      uint8_t value : 5; // Command code response
   } cmd;
   int16_t cam_rate[3]; // Camera attitude angular velocities
   int16_t cam_angle[3];  // Camera attitude angle
   int16_t mtr_angle[3];  // Camera relative angle
   uint8_t crc[2];     // CRC16
} Gbc2GcuPkt_t;
#pragma pack()

// -------------------- Global Variables --------------------
volatile uint8_t send_flag = 0; // 50Hz sending flag
uint8_t debug_output = 0;      // Debug output flag
Gbc2GcuPkt_t g_response;       // Receiving data caches
float ned_angle[3] = {0};      // Attitude angle, unit 1deg
// Simulated Carrier Data
float simulated_ned_angle[3] = {0.0f, 0.0f, 0.0f}; // Carrier attitude angle, unit
1deg
float simulated_ned_accel[3] = {0.0f, 0.0f, 0.0f}; // Carrier acceleration, unit 1m/
s2

// -------------------- Function Declarations--------------------
void UART1_Init(void);
void UART2_Init(void);
void TIM2_Init(void);
void SendControlPacket(void);
uint16_t CalculateCrc16(uint8_t *ptr, uint8_t len);
void PrintSendFrame(const Gcu2GbcPkt_t *pkt);
void PrintGimbalStatus(const Gbc2GcuPkt_t *pkt);
int fputc(int ch, FILE *f);
```

```
// -------------------- Main --------------------
int main(void)
{
  SystemInit(); // System clock initialization (72MHz)
  UART1_Init(); // UART1: gimbal communication
  UART2_Init(); // UART2: Debug output
  TIM2_Init();  // TIM2: 50Hz interrupt

  while (1)
  {
    if (send_flag)
    {
      SendControlPacket(); // Sending control command
      send_flag = 0;
      if (debug_output)
      {
        PrintGimbalStatus(&g_response);
        debug_output = 0;
      }
    }
  }
}

// -------------------- Timer Interrupt (50Hz) --------------------
void TIM2_IRQHandler(void)
{
  if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
  {
    static uint8_t count = 0;
    if (count++ % 25 == 0)
    {
      debug_output = 1; // Output debug information every 500ms
      count %= 25;     // Counter reset
    }
    send_flag = 1; // Trigger send
    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
  }
}
```

```
// ---------- Sending Control Command, Configure Operation Mode, Sending Carrier Data ----------
void SendControlPacket(void)
{
  Gcu2GbcPkt_t packet = {0};
  packet.sync[0] = 0xA9;
  packet.sync[1] = 0x5B;
  packet.cmd.value = 4; // Command code 4: manual control
  packet.cmd.trig = 0;
  // Roll: lock + angle control (0deg)
  packet.gbc[0].wk_mode = 1; // Lock
  packet.gbc[0].op_type = 0; // Angle control
  packet.gbc[0].op_value = 0;

  // Pitch: lock + true angular velocity control (0deg/s)
  packet.gbc[1].wk_mode = 1;
  packet.gbc[1].op_type = 2;
  packet.gbc[1].op_value = 0;

  // Yaw: follow + true angular velocity control (0deg/s)
  packet.gbc[2].wk_mode = 0;
  packet.gbc[2].op_type = 2;
  packet.gbc[2].op_value = 0;

  // Carrier Data (Simulated)
  packet.uav.valid = 0; // Invalid carrier data. If accurate and authentic carrier attitude angles and acceleration data are unavailable, this value must be set to zero. Otherwise, erroneous carrier data will result in incorrect gimbal attitude angles.
  // Euler Angle Converse (deg-> 0.01deg)
  packet.uav.angle[0] = (int16_t)(simulated_ned_angle[0] * 100); // Roll
  packet.uav.angle[1] = (int16_t)(simulated_ned_angle[1] * 100); // Pitch
  packet.uav.angle[2] = (int16_t)(simulated_ned_angle[2] * 100); // Yaw
  // Acceleration Converse (m/s2-> 0.01m/s2)
  packet.uav.accel[0] = (int16_t)(simulated_ned_accel[0] * 100); // Northward
  packet.uav.accel[1] = (int16_t)(simulated_ned_accel[1] * 100); // Eastward
  packet.uav.accel[2] = (int16_t)(simulated_ned_accel[2] * 100); // Upward
```

```
/// CRC16
  uint16_t crc = CalculateCrc16((uint8_t *)&packet, sizeof(packet) - 2);
  packet.crc[0] = (crc>> 8) & 0xFF;
  packet.crc[1] = crc& 0xFF;

  for (uint8_t i = 0; i<sizeof(packet); i++)
  {
    USART_SendData(USART1, *((uint8_t *)&packet + i));
    while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET)
      ;
  }
  if (debug_output)
  {
    PrintSendFrame(&packet);
  }
}

// -------------------- UART1 Receive Interrupt --------------------
void USART1_IRQHandler(void)
{
  static uint8_t rx_buffer[sizeof(Gbc2GcuPkt_t)];
  static uint8_t rx_index = 0;

  if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
  {
    rx_buffer[rx_index++] = USART_ReceiveData(USART1);
    // Complete reception of a data frame
    if (rx_index == sizeof(Gbc2GcuPkt_t))
    {
      Gbc2GcuPkt_t *pkt = (Gbc2GcuPkt_t *)rx_buffer;
      // Check header & CRC
      if (pkt->sync[0] == 0xB5 && pkt->sync[1] == 0x9A)
      {
        uint16_t crc = CalculateCrc16(rx_buffer, sizeof(Gbc2GcuPkt_t) - 2);
        if (pkt->crc[0] == (crc>> 8) && pkt->crc[1] == (crc& 0xFF))
        {
          memcpy(&g_response, pkt, sizeof(Gbc2GcuPkt_t));
        }
      }
      rx_index = 0;
    }
  }
}
```

```
// -------------------- CRC16 Function --------------------
uint16_t CalculateCrc16(uint8_t *ptr, uint8_t len)
{
  uint16_t crc = 0;
  const uint16_t crc_ta[16] = {
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
    0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF};
  while (len--)
  {
    uint8_t da = (crc>> 12);
    crc = (crc<< 4) ^ crc_ta[da ^ (*ptr>> 4)];
    da = (crc>> 12);
    crc = (crc<< 4) ^ crc_ta[da ^ (*ptr& 0x0F)];
    ptr++;
  }
  return crc;
}

// -------------------- Peripheral Initialization --------------------
void UART1_Init(void)
{
  GPIO_InitTypeDefGPIO_InitStruct;
  USART_InitTypeDefUSART_InitStruct;
  NVIC_InitTypeDefNVIC_InitStruct;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_
  USART1, ENABLE);

  // PA9(TX): Multiplexed push-pull output
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStruct);

  // PA10(RX): Floating input
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
// USART1 config: 115200, 8N1
    USART_InitStruct.USART_BaudRate = 115200;
    USART_InitStruct.USART_WordLength = USART_WordLength_8b;
    USART_InitStruct.USART_StopBits = USART_StopBits_1;
    USART_InitStruct.USART_Parity = USART_Parity_No;
    USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_InitStruct.USART_HardwareFlowControl = USART_
    HardwareFlowControl_None;
    USART_Init(USART1, &USART_InitStruct);

    // Enable receive interrupt
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    USART_Cmd(USART1, ENABLE);

    // Configure interrupt priority
    NVIC_InitStruct.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}

void UART2_Init(void)
{
    GPIO_InitTypeDefGPIO_InitStruct;
    USART_InitTypeDefUSART_InitStruct;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    // PA2(TX): Multiplexed push-pull output
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    // USART2 config: 115200, 8N1
    USART_InitStruct.USART_BaudRate = 115200;
    USART_InitStruct.USART_WordLength = USART_WordLength_8b;
    USART_InitStruct.USART_StopBits = USART_StopBits_1;
    USART_InitStruct.USART_Parity = USART_Parity_No;
    USART_InitStruct.USART_Mode = USART_Mode_Tx;
    USART_InitStruct.USART_HardwareFlowControl = USART_
    HardwareFlowControl_None;
    USART_Init(USART2, &USART_InitStruct);
    USART_Cmd(USART2, ENABLE);
```

```
void TIM2_Init(void)
{
   TIM_TimeBaseInitTypeDefTIM_InitStruct;
   NVIC_InitTypeDefNVIC_InitStruct;

   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

   // Configure timer for 50Hz output (72MHz basic frequency)
   TIM_InitStruct.TIM_Prescaler = 7200 - 1; // 72MHz / 7200 = 10kHz
   TIM_InitStruct.TIM_Period = 200 - 1;    // 10kHz / 200 = 50Hz
   TIM_InitStruct.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_InitStruct.TIM_ClockDivision = TIM_CKD_DIV1;
   TIM_TimeBaseInit(TIM2, &TIM_InitStruct);

   // Enable interrupt
   TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
   TIM_Cmd(TIM2, ENABLE);

   // Configure interrupt priority
   NVIC_InitStruct.NVIC_IRQChannel = TIM2_IRQn;
   NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 1;
   NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
   NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
   NVIC_Init(&NVIC_InitStruct);
}

// -------------------- Tool Funtion --------------------
int fputc(int ch, FILE *f)
{
   USART_SendData(USART2, (uint8_t)ch);
   while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RESET)
      ;
   return ch;
}

// -------------------- Debug Output --------------------
void PrintSendFrame(const Gcu2GbcPkt_t *pkt)
{
   printf("\nSend frame: ");
   for (uint8_t i = 0; i<sizeof(Gcu2GbcPkt_t); i++) {
      printf("%02X ", ((uint8_t *)pkt)[i]);
   }
```

```
    printf("\n");
    printf("Parse send frame: \n");
    printf("cmdTrig : %-1d\n", pkt->cmd.trig);
    printf("cmdValue : %-2d\n", pkt->cmd.value);
    printf("FPV Sens : %-2d\n", pkt->aux.fl_sens);

    // Control information
    const char *axis_names[] = {"Roll", "Pitch", "Yaw"};
    const char *wk_mode[] = {"Follow", "Lock", "FPV "};
    const char *op_type[] = {"Angle Control", "Scaled Angular Velocity Control", "
    True Angular Velocity Control "};

    for (uint8_t i = 0; i< 3; i++) {
        printf("%-5s : ", axis_names[i]);
        printf("go_zero=%-1d   ", pkt->gbc[i].go_zero);
        printf("wk_mode=%-10s   ", wk_mode[pkt->gbc[i].wk_mode<2 ? pkt-
        >gbc[i].wk_mode : 2]);
        printf("op_type=%-10s   ", op_type[pkt->gbc[i].op_type<3 ? pkt->gbc[i].
        op_ type : 3]);
        printf("op_value=%-2d  \n", pkt->gbc[i].op_value);
    }

    // Carrier information
    printf("uav valid: %-1d\n", pkt->uav.valid);
    printf("uav Angle: [%6.2f, %6.2f, %6.2f] deg\n",
        pkt->uav.angle[0]*0.01f, pkt->uav.angle[1]*0.01f, pkt->uav.angle[2]*0.01f);
    printf("uav Accel: [%6.2f, %6.2f, %6.2f] m/s²\n",
        pkt->uav.accel[0]*0.01f, pkt->uav.accel[1]*0.01f, pkt->uav.accel[2]*0.01f);

    // Camera information
    printf("cam vert_fovlx  : %-3d\n", pkt->cam.vert_fovlx);
    printf("cam zoom_value  : %-6d\n", pkt->cam.zoom_value);
    printf("cam target_angle : [%6.2f, %6.2f] deg\n",
        pkt->cam.target_angle[0], pkt->cam.target_angle[1]);
}

void PrintGimbalStatus(const Gbc2GcuPkt_t *pkt)
{
    printf("\nRecv frame: ");
    for (uint8_t i = 0; i<sizeof(Gbc2GcuPkt_t); i++)
    {
        printf("%02X ", ((uint8_t *)pkt)[i]);
    }
```

```
    printf("\n");
    printf("Parse recv frame: \n");
    printf("FW Ver: %-2d\n", pkt->fw_ver);
    printf("HW Err: %-2d\n", pkt->hw_err);
    printf("Inv Flag: %-1d\n", pkt->inv_flag);
    printf("GBC Stat: %-1d\n", pkt->gbc_stat);
    printf("TCA Flag: %-1d\n", pkt->tca_flag);
    printf("Cmd Stat: %-1d\n", pkt->cmd.stat);
    printf("Cmd Value: %-2d\n", pkt->cmd.value);

    printf("Cam Rate: [%6d , %6d , %6d]\n",
        pkt->cam_rate[0],
        pkt->cam_rate[1],
        pkt->cam_rate[2]);

    printf("Cam Angle:  [%6d , %6d , %6d]\n",
        pkt->cam_angle[0],
        pkt->cam_angle[1],
        pkt->cam_angle[2]);

    printf("Mtr Angle:  [%6d , %6d , %6d]\n",
        pkt->mtr_angle[0],
        pkt->mtr_angle[1],
        pkt->mtr_angle[2]);

    for (uint8_t i = 0; i< 3; i++)
      ned_angle[i] = pkt->cam_angle[i] * 0.01f;

    printf("NED Angle:  [%6.2f , %6.2f , %6.2f]\n\n",
        ned_angle[0],
        ned_angle[1],
        ned_angle[2]);
}
```

Appendix 4: Example Data Packet and Conversion

Example 1
The complete host-to-gimbal data packet:
A9 5B 20 00 10 00 00 90 00 00 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 61 BA

Byte	Content	Original Data	Accuracy or Binary Conversion	Hexadecimal
0	Header	0xA9	-	A9
1		0x5B	-	5B
2	[7:3] Command code	4- Manual control	0b00100	20
	[2:0] Command trigger counter		0b000	
3	[7:3] FPV Sensitivity	0	0b00000	00
	[2:0]		0b	
4	[7:6] Roll controlling type	0- Angle control	0b00	10
	[5:4] Roll operating mode	1- Lock	0b01	
	[3] Roll neutral position trigger		0b0	
	[2:0]		0b000	
5~6	Roll control value	0deg	0	00 00
7	[7:6] Pitch controlling type	2- True angular velocity control	0b10	90
	[5:4] Pitch operating mode	0-	Lock	
	[3] Pitch neutral position trigger	-	0b0	
	[2:0]	-	0b000	
8~9	Pitch control value	0deg/s	0	00 00
10	[7:6] Yaw controlling type	2- True angular velocity control	0b10	80
	[5:4] Yaw operating mode	0-	Follow	
	[3] Yaw neutral position trigger	-	0b0	
	[2:0]	-	0b000	
11~12	Yaw control value	0deg/s	0	00 00
13	[7] Carrier AHRS validity	invalid	0b0	00
	[6:0]	-	0b0000000	
14~15	Carrier roll attitude angle	0deg	0	00 00

Byte	Content	Original Data	Accuracy or Binary Conversion	Hexadecimal
16~17	Carrier pitch attitude angle	0deg	0	00 00
18~19	Carrier yaw attitude angle	0deg	0	00 00
20~21	Carrier northward acceleration	0m/s2	0	00 00
22~23	Carrier eastward acceleration	0m/s2	0	00 00
24~25	Carrier upward acceleration	0m/s2	0	00 00
26~29	[31] Reserved	-	0b0	00 00 00 00
	[30:7] Zoom ratio	0x	0b00000000000 0000000000000	
	[6:0] Vertical field of view of camera at 1x	0deg	0b0000000	
30~33	Target horizontal offsite angle	0deg	0	00 00 00 00
34~37	Target vertical offsite angle	0deg	0	00 00 00 00
38~39	CRC	-	-	61 BA

Example 2
The complete host-to-gimbal data packet:
A9 5B 20 00 90 00 00 90 0A 00 90 00 00 80 F4 01 F4 01 F4 01 0A 00 0A 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 98 DB

Byte	Content	Original Data	Accuracy or Binary Conversion	Hexadecimal
0	Header	0xA9	-	A9
1		0x5B	-	5B
2	[7:3] Command code	4- Manual control	0b00100	20
	[2:0] Command trigger counter		0b000	
3	[7:3] FPV Sensitivity	0	0b00000	00
	[2:0]		0b000	
4	[7:6] Roll controlling type	2- True angular velocity control	0b10	90
	[5:4] Roll operating mode	1- Lock	0b01	
	[3] Roll neutral position trigger		0b0	
	[2:0]		0b000	
5~6	Roll control value	0deg/s	0	00 00
7	[7:6] Pitch controlling type	2- True angular velocity control	0b10	90
	[5:4] Pitch operating mode	1- Lock	0b01	
	[3] Pitch neutral position trigger	-	0b0	
	[2:0]	-	0b000	
8~9	Pitch control value	1deg/s	10	0A 00
10	[7:6] Yaw controlling type	2- True angular velocity control	0b10	90
	[5:4] Yaw operating mode	1- Lock	0b01	
	[3] Yaw neutral position trigger	-	0b0	
	[2:0]	-	0b000	
11~12	Yaw control value	0deg/s	0	00 00
13	[7] Carrier AHRS validity	Valid	0b1	80
	[6:0]	-	0b0000000	
14~15	Carrier roll attitude angle	5deg	500	F4 01

Byte	Content	Original Data	Accuracy or Binary Conversion	Hexadecimal
16~17	Carrier pitch attitude angle	5deg	500	F4 01
18~19	Carrier yaw attitude angle	5deg	500	F4 01
20~21	Carrier northward acceleration	0.1m/s2	10	0A 00
22~23	Carrier eastward acceleration	0.1m/s2	10	0A 00
24~25	Carrier upward acceleration	0.1m/s2	10	0A 00
26~29	[31] Reserved	-	0b0	00 00 00 00
	[30:7] Zoom ratio	0x	0b00000000000 0000000000000	
	[6:0] Vertical field of view of camera at 1x	0deg	0b0000000	
30~33	Target horizontal offsite angle	0deg	-	00 00 00 00
34~37	Target vertical offsite angle	0deg	-	00 00 00 00
38~39	CRC	-	-	98 DB

Example 3
The complete gimbal responding data packet:
B5 9A 24 00 19 21 FF FF 00 00 FE FF 09 00 8C 35 B9 01 52 C9 5F 00 FE FF 52 CD

Byte	Content	Original Data (Hexadecimal)	Parsed Data
0	Header	B5	0xB5
1		9A	0x9A
2	Firmware version	24	36
3	Hardware failure	00	0
4	[7:5]	19	0b000
	[4] Temperature control readiness		0b1 1- Ready
	[3:1] Gimbal Status		0b100 4- Manual control
	[0] Mounting direction flag		0b0 Downward
5	[7:3] Command code response	21	0b00100 4- Manual control
	[2:0] Command execution status		0b001 1- Success
6~7	Camera pitch body angular velocity	FF FF	-0.1deg/s
8~9	Camera roll body angular velocity	00 00	0deg/s
10~11	Camera yaw body angular velocity	FE FF	-0.2deg/s
12~13	Camera roll attitude angle	09 00	0.09deg
14~15	Camera pitch attitude angle	8C 35	137.08deg
16~17	Camera yaw attitude angle	B9 01	4.41deg
18~19	Camera pitch relative angle	52 C9	-139.98deg
20~21	Camera roll relative angle	5F 00	0.95deg
22~23	Camera yaw relative angle	FE FF	-0.02deg
24~25	CRC	52 CD	0x52 0xCD