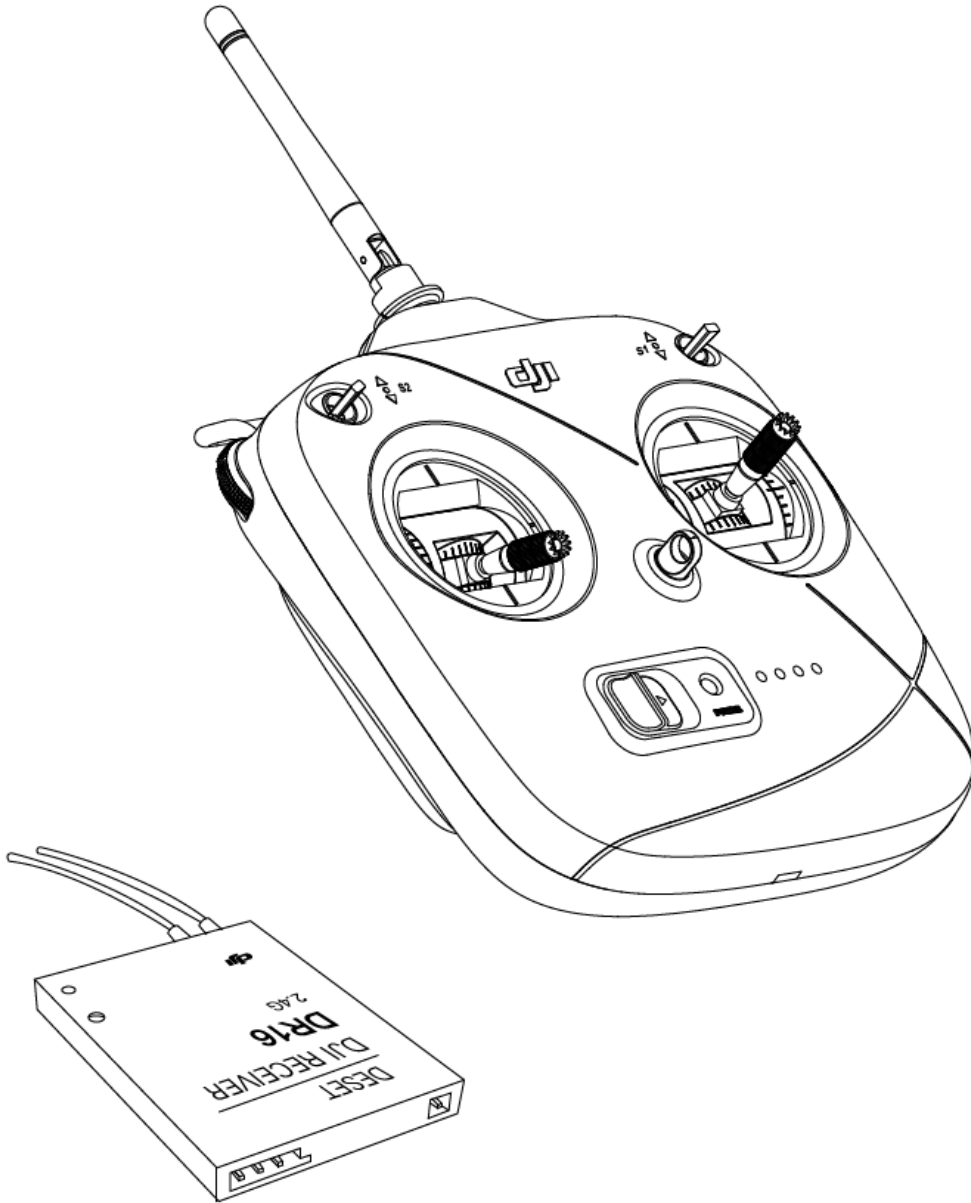


DT7&DR16 2.4GHz

遥控接收系统用户手册

V1.0 2016.09



免责声明

感谢您购买DT7 & DR16 2.4 GHz 遥控接收系统。在使用之前，请仔细阅读本声明，一旦使用，即被视为对本声明全部内容的认可和接受。请严格遵守手册安装和使用该产品。因用户不当使用、安装、改装（包括使用非指定的DJI 零配件等）造成的任何损失，DJI 将不承担法律责任。

DJI 是大疆创新所有的注册商标。本文出现的产品名称、品牌等，均为其所属公司的商标或注册商标。本产品及手册为大疆创新版权所有。未经许可，不得以任何形式复制翻印。关于不同语言版本的免责声明可能存在的语义差异，以英文版为准。

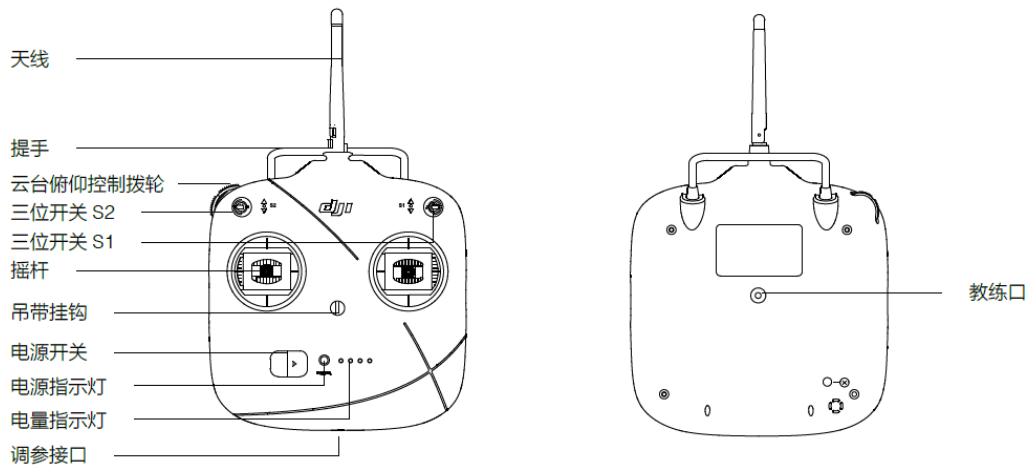
盒内物品

DT7 遥控器 ×1，DR16 接收机×1，3-Pin 连接线×1，双端圆口教练线×1。

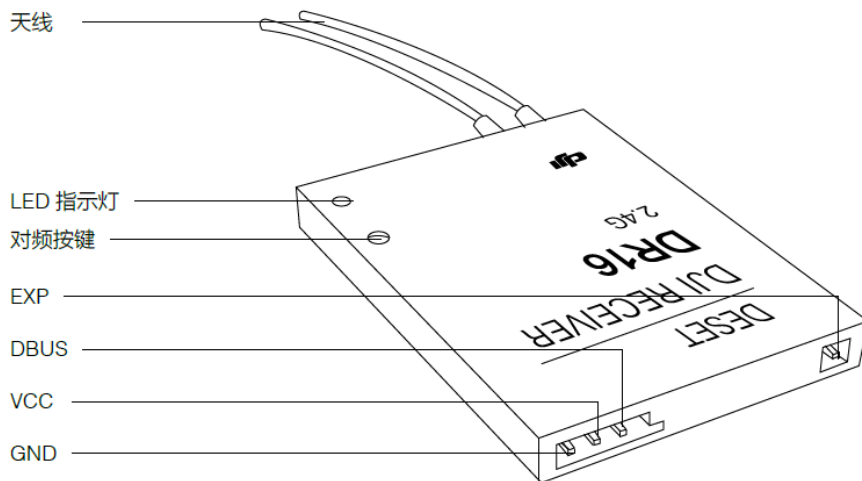
产品简介

DT7 遥控器是一款工作于2.4 GHz 频段的无线电通信设备，该遥控器仅能与DR16 接收机配合使用。

本产品可用于控制RM战车以及DJI飞行产品。开阔室外中，该遥控器最大的控制范围可达到1000 m，最长工作时间可达到12 个小时。



DR16 接收机是一款工作频率为2.4 GHz 的16 通道接收机，可配合DT7 遥控器使用。



遥控接收系统规格参数

DJI DESSET 2.4 G	工作频率2.4 GHz ISM	工作频率2.4 GHz ISM
遥控接收系统	通信距离（开阔室外） 1 km	通信距离（开阔室外） 1 km
DT7 遥控器	特性7 通道	特性7 通道
	工作电流/ 电压 120 mA @ 3.7 V	工作电流/ 电压 120 mA @ 3.7 V
	电池 3.7 V 2000 mAh 锂电池	电池 3.7 V 2000 mAh 锂电池

DR16 接收机	特性2.4 GHz D-BUS 协议	特性2.4 GHz D-BUS 协议
	接收灵敏度 (1%PER) -97 dBm	接收灵敏度 (1%PER) -97 dBm
	工作电流/ 电压145 mA @ 5 V	工作电流/ 电压145 mA @ 5 V
	电源4 - 8.4 V	电源4 - 8.4 V
	尺寸41 mm×29 mm×5 mm	尺寸41 mm×29 mm×5 mm
	重量 10 g	重量 10 g

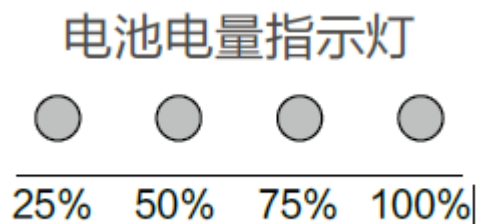
遥控器操作

开启与关闭

DT7 遥控器内置容量为2000 mAh 的锂充电电池，可通过电池电量指示灯查看当前电量。

按以下步骤开启/ 关闭遥控器：

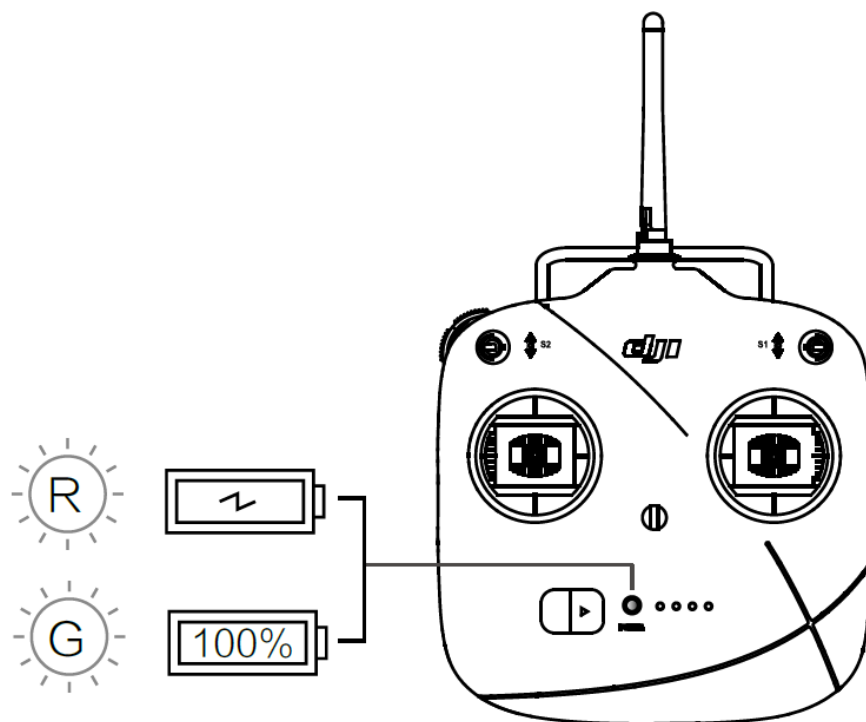
1. 向右推电源开关，开启遥控器。
2. 遥控器电源指示灯绿灯常亮表示遥控器正常工作。
3. 通过遥控器面板上的电量指示灯了解当前电量。电量指示灯的详细信息如图所示。



4. 向左推电源开关，电源指示灯熄灭，遥控器关闭。

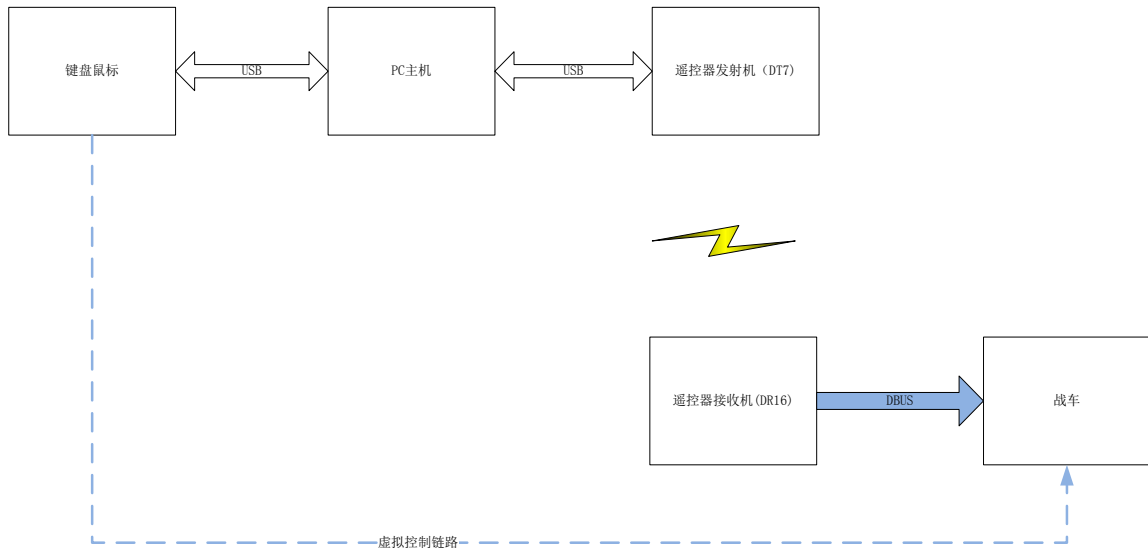
遥控器充电

用户可通过遥控器调参接口对遥控器进行充电，请注意在关机状态充电。充电时，电源指示灯为红灯常亮。充电完成后，电源指示灯为绿灯常亮。



Robomaster战车控制协议

DR16接收机输出的信号为标准的DBUS协议数据,当遥控器与接收机建立连接后，接收机每隔14ms通过DBUS发送一帧18字节数据.为了满足通过PC控制战车的需求，增加了额外的键盘鼠标控制信息，控制链路示意如图所示：



DBUS通信参数如下表所示：

DBUS 参数	数值
波特率	100Kbps
单元数据长度	8
奇偶校验位	偶校验
结束位	1
流控	无

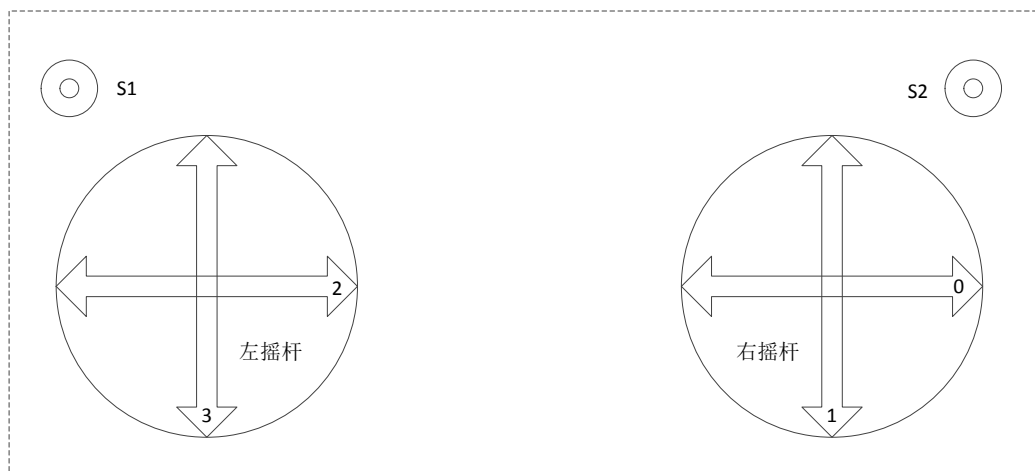
注：DBUS 信号控制电平符合 TTL，却和普通 UART 信号是相反的，所以需要在 MCU 端需要增加三极管取反电路，MCU 才能正常识别出 UART 信号

18 字节控制帧结构

域	通道 0	通道 1	通道 2	通道 3	S1	S2
偏移	0	11	22	33	44	46
长度 (bit)	11	11	11	11	2	2
符号位	无	无	无	无	无	无
范围	最大值 1684 中间值 1024 最小值 364	最大值 1684 中间值 1024 最小值 364	最大值 1684 中间值 1024 最小值 364	最大值 1684 中间值 1024 最小值 364	最大值 3 最小值 1	最大值 3 最小值 1
功能	无符号类型 遥控器通道 0	无符号类型 遥控器通道 1	无符号类型 遥控器通道 2	无符号类型 遥控器通道 3	遥控器发射机 S1 开关位	遥控器发射机 S2 开关位置

	控制信息	控制信息	控制信息	控制信息	置 1:上 2:下 3:中	1:上 2:下 3:中
域	鼠标 X 轴	鼠标 Y 轴	鼠标 Z 轴	鼠标左键	鼠标右键	按键
偏移	48	64	80	96	104	112
长度	16	16	16	8	8	16
符号位	有	有	有	无	无	无
范围	最大值 32767 最小值 -32768 静止值 0	最大值 32767 最小值 -32768 静止值 0	最大值 32767 最小值 -32768 静止值 0	最大值 1 最小值 0	最大值 1 最小值 0	位值标识
功能	鼠标在 X 轴的移动速度 负值标识往左移动 正值标识往右移动	鼠标在 Y 轴的移动速度 负值标识往左移动 正值标识往右移动	鼠标在 Z 轴的移动速度 负值标识往左移动 正值标识往右移动	鼠标左键是否按下 0 --没按下 1 --按下	鼠标右键是否按下 0 --没按下 1 --按下	每个按键对应一个 bit, 如下所示 Bit0 -- W 键 Bit1 -- S 键 Bit2 -- A 键 Bit3 -- D 键 Bit4 -- Q 键 Bit5 -- E 键 Bit6 -- Shift 键 Bit7 -- Ctrl 键
域	保留字段					
偏移	128					
长度	16					

遥控器通道与控制开关如下图所示：



协议解析示例代码见附件。

附录 (RM战车遥控器协议解析示例代码STM32F4平台)

```
/* ----- RC Channel Definition ----- */
#define RC_CH_VALUE_MIN          ((uint16_t)364 )
#define RC_CH_VALUE_OFFSET      ((uint16_t)1024)
#define RC_CH_VALUE_MAX        ((uint16_t)1684)

/* ----- RC Switch Definition ----- */
#define RC_SW_UP                 ((uint16_t)1)
#define RC_SW_MID                ((uint16_t)3)
#define RC_SW_DOWN               ((uint16_t)2)

/* ----- PC Key Definition ----- */
#define KEY_PRESSED_OFFSET_W     ((uint16_t)0x01<<0)
#define KEY_PRESSED_OFFSET_S     ((uint16_t)0x01<<1)
#define KEY_PRESSED_OFFSET_A     ((uint16_t)0x01<<2)
#define KEY_PRESSED_OFFSET_D     ((uint16_t)0x01<<3)
#define KEY_PRESSED_OFFSET_Q     ((uint16_t)0x01<<4)
#define KEY_PRESSED_OFFSET_E     ((uint16_t)0x01<<5)
#define KEY_PRESSED_OFFSET_SHIFT ((uint16_t)0x01<<6)
#define KEY_PRESSED_OFFSET_CTRL  ((uint16_t)0x01<<7)

#define RC_FRAME_LENGTH          18u

/* ----- Data Struct ----- */
typedef __packed struct
{
    struct
    {
        uint16_t ch0;
        uint16_t ch1;
        uint16_t ch2;
        uint16_t ch3;
        uint8_t  s1;
        uint8_t  s2;
    }rc;

    struct
    {
        int16_t x;
        int16_t y;
        int16_t z;
        uint8_t press_l;
        uint8_t press_r;
    }mouse;

    struct
    {
        uint16_t v;
    }key;
};
```



```

}RC_Ctl_t;

/* ----- Internal Data ----- */
volatile unsigned char sbus_rx_buffer[2][RC_FRAME_LENGTH]; //double sbus
rx buffer to save data
static RC_Ctl_t RC_CtrlData;

/* ----- Function Implements ----- */

/*****
 * @fn      RC_Init
 *
 * @brief   configure stm32 usart2 port
 *          - USART Parameters
 *          - 100Kbps
 *          - 8-N-1
 *          - DMA Mode
 *
 * @return  None.
 *
 * @note    This code is fully tested on STM32F405RGT6 Platform, You can port
it
 *          to the other platform. Using double buffer to receive data prevent
losing data.
 */
void RC_Init(void)
{
    /* ----- Enable Module Clock Source -----*/
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_DMA1, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    GPIO_PinAFConfig(GPIOA,GPIO_PinSource3, GPIO_AF_USART2);

    /* ----- Configure GPIO -----*/
    {
        GPIO_InitTypeDef gpio;
        USART_InitTypeDef usart2;

        gpio.GPIO_Pin   = GPIO_Pin_3 ;
        gpio.GPIO_Mode  = GPIO_Mode_AF;
        gpio.GPIO_OType = GPIO_OType_PP;
        gpio.GPIO_Speed = GPIO_Speed_100MHz;
        gpio.GPIO_PuPd  = GPIO_PuPd_NOPULL;
        GPIO_Init(GPIOA, &gpio);

        USART_DeInit(USART2);
        usart2.USART_BaudRate           = 100000;
        usart2.USART_WordLength         = USART_WordLength_8b;
        usart2.USART_StopBits           = USART_StopBits_1;
        usart2.USART_Parity              = USART_Parity_Even;
        usart2.USART_Mode                = USART_Mode_Rx;
        usart2.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    }
}

```

```

    USART_Init(USART2, &usart2);

    USART_Cmd(USART2, ENABLE);
    USART_DMAMCmd(USART2, USART_DMAREq_Rx, ENABLE);
}

/* ----- Configure NVIC -----*/
{
    NVIC_InitTypeDef  nvic;

    nvic.NVIC_IRQChannel          = DMA1_Stream5_IRQn;
    nvic.NVIC_IRQChannelPreemptionPriority = 1;
    nvic.NVIC_IRQChannelSubPriority   = 1;
    nvic.NVIC_IRQChannelCmd         = ENABLE;
    NVIC_Init(&nvic);
}

/* ----- Configure DMA -----*/
{
    DMA_InitTypeDef  dma;

    DMA_DeInit(DMA1_Stream5);
    dma.DMA_Channel          = DMA_Channel_4;
    dma.DMA_PeripheralBaseAddr = (uint32_t)&(USART2->DR);
    dma.DMA_Memory0BaseAddr  = (uint32_t)&sbus_rx_buffer[0][0];
    dma.DMA_DIR               = DMA_DIR_PeripheralToMemory;
    dma.DMA_BufferSize        = RC_FRAME_LENGTH;
    dma.DMA_PeripheralInc     = DMA_PeripheralInc_Disable;
    dma.DMA_MemoryInc         = DMA_MemoryInc_Enable;
    dma.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
    dma.DMA_MemoryDataSize    = DMA_MemoryDataSize_Byte;
    dma.DMA_Mode              = DMA_Mode_Circular;
    dma.DMA_Priority          = DMA_Priority_VeryHigh;
    dma.DMA_FIFOMode          = DMA_FIFOMode_Disable;
    dma.DMA_FIFOThreshold    = DMA_FIFOThreshold_1QuarterFull;
    dma.DMA_MemoryBurst       = DMA_MemoryBurst_Single;
    dma.DMA_PeripheralBurst   = DMA_PeripheralBurst_Single;
    DMA_DoubleBufferModeConfig(DMA1_Stream5, (uint32_t)&sbus_rx_buffer[1][0],
DMA_Memory_0); //first used memory configuration
    DMA_DoubleBufferModeCmd(DMA1_Stream5, ENABLE);
    DMA_Init(DMA1_Stream5, &dma);

    USART_ITConfig(USART2, USART_IT_IDLE, ENABLE); //usart rx idle
interrupt enabled
    DMA_Cmd(DMA1_Stream5, ENABLE);
}
}

/*****
* @fn      RemoteDataProcess
*
*****/

```

```

* @brief resolution rc protocol data.
* @pData a point to rc receive buffer.
* @return None.
* @note RC_CtrlData is a global variable.you can deal with it in other place.
*/
void RemoteDataProcess(uint8_t *pData)
{
    if(pData == NULL)
    {
        return;
    }

    RC_CtrlData.rc.ch0 = ((int16_t)pData[0] | ((int16_t)pData[1] << 8)) & 0x07FF;
    RC_CtrlData.rc.ch1 = (((int16_t)pData[1] >> 3) | ((int16_t)pData[2] << 5))
& 0x07FF;
    RC_CtrlData.rc.ch2 = (((int16_t)pData[2] >> 6) | ((int16_t)pData[3] << 2) |
((int16_t)pData[4] << 10)) & 0x07FF;
    RC_CtrlData.rc.ch3 = (((int16_t)pData[4] >> 1) | ((int16_t)pData[5]<<7)) &
0x07FF;

    RC_CtrlData.rc.s1 = ((pData[5] >> 4) & 0x000C) >> 2;
    RC_CtrlData.rc.s2 = ((pData[5] >> 4) & 0x0003);

    RC_CtrlData.mouse.x = ((int16_t)pData[6]) | ((int16_t)pData[7] << 8);
    RC_CtrlData.mouse.y = ((int16_t)pData[8]) | ((int16_t)pData[9] << 8);
    RC_CtrlData.mouse.z = ((int16_t)pData[10]) | ((int16_t)pData[11] << 8);

    RC_CtrlData.mouse.press_l = pData[12];
    RC_CtrlData.mouse.press_r = pData[13];

    RC_CtrlData.key.v = ((int16_t)pData[14]);// | ((int16_t)pData[15] << 8);

    //your control code ...
}

/*****
* @fn USART2_IRQHandler
*
* @brief USART2 irq, we are care of ilde interrupt that means receiving the
one frame datas is finished.
*
* @return None.
*
* @note This code is fully tested on STM32F405RGT6 Platform, You can port
it
* to the other platform.
*/
void USART2_IRQHandler (void)
{
    if(USART_GetITStatus(USART2, USART_IT_IDLE) != RESET)
    {

```

```

//clear the idle pending flag
(void)USART2->SR;
(void)USART2->DR;

//Target is Memory0
if(DMA_GetCurrentMemoryTarget(DMA1_Stream5) == 0)
{
    DMA_Cmd(DMA1_Stream5, DISABLE);
    DMA1_Stream5->NDTR = (uint16_t)RC_FRAME_LENGTH; //relocate the
dma memory pointer to the beginning position
    DMA1_Stream5->CR |= (uint32_t)(DMA_SxCR_CT); //enable the
current selected memory is Memory 1
    DMA_Cmd(DMA1_Stream5, ENABLE);
    if(DMA_GetCurrDataCounter(DMA1_Stream5) == 0) //ensure
received complete frame data.
    {
        RemoteDataProcess(sbus_rx_buffer[0]);
    }
}
//Target is Memory1
else
{
    DMA_Cmd(DMA1_Stream5, DISABLE);
    DMA1_Stream5->NDTR = (uint16_t)RC_FRAME_LENGTH; //relocate the dma
memory pointer to the beginning position
    DMA1_Stream5->CR &= ~(uint32_t)(DMA_SxCR_CT); //enable the current
selected memory is Memory 0
    DMA_Cmd(DMA1_Stream5, ENABLE);
    if(DMA_GetCurrDataCounter(DMA1_Stream5) == 0)
    {
        RemoteDataProcess(sbus_rx_buffer[1]);
    }
}
} }

```