2023 Moving target control and automatic tracking system

1. Introduction

Please read the information in the 24-way servo driver board and the stm32 introductory tutorial first to understand how to burn the program and the basics of getting started with stm32. This will make it easier to learn and burn the subsequent programs.

Usage tips: It is best to use a battery to power the servo so that the servo is in normal operation. If the data cable is used for power supply, although the servo can move, the servo voltage cannot reach 7.4v for normal operation, and the servo movement effect will be worse.

2. Equipment list

k230	x2	
24-way servo driver board	x2	
2D electric gimbal	x2	
Green laser pen	x1	Buy
7.4v battery	x1	Buy
Red laser pen	x1	Buy
Red laser pen bracket	x1	Buy
Male to male Dupont line	x2	
Aluminum alloy plate	x2	Buy

The red laser pen needs to be recognized by k230. Customers can choose the laser pen that meets the competition questions by themselves, or choose the red laser pen on the equipment list as a red laser pen that can be recognized and meets the competition questions. If the laser pen is fixed, you can find a suitable laser pen holder by yourself or choose the holder on the list. Choose the square holder 22.5mm style. Pay special attention to the fact that if you use this square holder, you need to remove it to use part of it. See the picture of the placement for usage.

3.Placement

The placement is as shown in the figure. You can choose our aluminum base frame as the base plate to fix the k230 and the gimbal, or you can fix it yourself. The distance between the k230 and the gimbal is about 7cm, and the 2D gimbal system is 1m away from the black frame as required by the question.

4.Experimental preparation

First, follow the 2D PTZ installation tutorial to build the 2D PTZ.

Hardware Wiring:

2D PTZ	24-channel Servo Driver Board
X-axis Servo	S1
Y-axis Servo	S2

K230 Vision Module	24-channel Motor Driver Board
5V	5V
GND	GND
TX	RX3
RX	TX3

Note: When inserting the servo cable, be sure to align the yellow cable with the yellow GPIO port of the 24-channel Servo Driver Board, and the brown cable with the black GPIO port of the 24-channel Servo Driver Board.

5. K230 key code analysis

- get_square.py

```
# Find inner frame
if flag_find_first_rect == False:
# The threshold parameter is used to set the detection threshold (the threshold size is related to the pixel size of the rectangle to be detected. Setting the threshold within the area of the rectangle to be detected can reduce other external noise)
# The x_gradient and y_gradient parameters are used to set the edge gradient threshold (the larger the x_gradient and y_gradient, the more accurate the recognition but the longer the time. The smaller the x_gradient and y_gradient, the faster the recognition but the greater the interference)
# Both are related to the ambient brightness
# Find rectangles and adjust the threshold, x_gradient, y_gradient values
for rect in img.find_rects(threshold = 80000, x_gradient=10, y_gradient=10):
if rect:
flag_find_first_rect = True
# Get the coordinates of the four corners of the rectangle
first_rect_corners = rect.corners()
area = rect.magnitude() # Rectangle pixel area size
print(area)
# Find the outer frame
if flag_find_first_rect == True:
if flag_find_second_rect == False:
for rect in img.find_rects(threshold = 60000, x_gradient=10, y_gradient=10):
if rect:
area = rect.magnitude()
if area < 90000:
```

```
flag_find_second_rect = True
# Get the coordinates of the four corners of the rectangle
second_rect_corners = rect.corners()
print(area)
```

This code is to get the coordinates of the four points of the rectangle. By finding the outer rectangle and inner rectangle of the black frame and then determining the center rectangle of the black frame, the coordinates of the four vertices of the center rectangle are obtained in turn.

- red_light.py

```
def color_blob(img,hano,color):#Find the largest color block of the specified color
blobs = img.find_blobs([hano],roi=light_roi,merge = True)
if blobs:
global last_cx
global last_cy
for blob in blobs:
roi = (blob.x()-5,blob.y()-5,blob.w()+10,blob.h()+10)#Get the circumscribed circle range of the bright spot
if img.find_blobs([color],roi=roi,x_stride=1, y_stride=1,
area_threshold=0, pixels_threshold=0,merge=False,margin=1):#Found the specified color
cx = blob[5]
cy = blob[6]
img.draw_rectangle(blob[0:4],color=(0,0,255)) # Draw a rectangle
img.draw_cross(blob[5], blob[6],color=(0,0,255)) # Draw a cross
last_cx = cx#Update coordinates
last_cy = cy#Update coordinates
return cx, cy, 1
return last_cx, last_cy, 0
return last_cx, last_cy, 0
```

This code is to obtain the coordinates of the target color block. By calling the color block search function of k230, the color block within the threshold is found. For this question, it is a red color block, so the threshold needs to be defined as red.

- Split coordinate function

```
def conversion_postion(x,y):
if x > 510:
x_h = 255
x_l = 255
x_z = x-255-255
elif 510>=x>=255:
x_h = 255
x_l = x-255
x_z = 0
else:
x_h = x
x_l = 0
x_z = 0
```

```
if y > 510:
y_h = 255
y_l = 255
y_z = y-255-255
elif 510>=y>=255:
y_h = 255
y_l = y-255
y_z = 0
else:
y_h = y
y_l = 0
y_z = 0
return x_h,x_l,x_z,y_h,y_l,y_z
```

Because stm32 only transmits 8-bit binary numbers, the maximum transmission data is 255, and our K230 uses 640x480, so its coordinates need to be divided and 640 is split into 3 frames of data for transmission.

6. Analysis of the key code of the first gimbal

- control.c

```
void go_to_point(uint16_t now_x, uint16_t now_y, uint16_t targ_x, uint16_t targ_y)
{
uint16_t i;
float step_num = 1000;/* 1000 steps*/
float step_x, step_y; /* step value */
float going_x = now_x, going_y = now_y; /* current target value*/

step_x = (targ_x - now_x) / step_num;
step_y = (targ_y - now_y) / step_num;

for(i = 0; i < step_num; i++)
{

kfp_x.source = step_x;
kfp_y.source = step_y;
kfp_x.out = KalmanFilter(&kfp_x, kfp_x.source);
kfp_y.out = KalmanFilter(&kfp_y, kfp_y.source);
going_x += kfp_x.out;
going_y += kfp_y.out;

set_target(going_x, going_y);

}
}
```

This code subdivides the coordinates of the two points obtained, thereby dividing the two points of the rectangle into many points, and then performing pid calculations between many points, making the servo movement more accurate, and also controlling the speed of the servo moving around the black frame.

```
uint8_t get_point1_p()
```

```
{
uint8_t lable, load_1, load_2,load_3, ret = 0;
if(DataDecode1(&Uart3_RingBuff,&lable, &load_1, &load_2, &load_3) == 0){

if(lable == 0x10){
rec_step_point[3][0] = (load_1 + load_2 + load_3);
} else if(lable == 0x11){
rec_step_point[3][1] = (load_1 + load_2 + load_3);
} else if(lable == 0x12){
rec_step_point[0][0] = (load_1 + load_2 + load_3);
} else if(lable == 0x13){
rec_step_point[0][1] = (load_1 + load_2 + load_3);
} else if(lable == 0x14){
rec_step_point[1][0] = (load_1 + load_2 + load_3);
} else if(lable == 0x15){
rec_step_point[1][1] = (load_1 + load_2 + load_3);
} else if(lable == 0x16){
rec_step_point[2][0] = (load_1 + load_2 + load_3);
} else if(lable == 0x17){
rec_step_point[2][1] = (load_1 + load_2 + load_3);
mode_flag = 4;
}

}
return ret;
}
```

This code adds together the three frames of data transmitted by k230. As mentioned earlier, the coordinate data of k230 is split into three frames and then sent separately. Here, the data is added together to form the correct coordinates of the rectangle vertices, and then the x and y coordinates of the rectangle are assigned to the two-dimensional array in sequence. When the last coordinate is obtained, the defined flag is assigned.

```
static uint8_t step_flag =0;
if(step_flag ==0)
{

USART_Cmd(USART3, ENABLE);
get_point1_p();//Get coordinates

}

if(mode_flag == 4&&step_flag==0){
USART_Cmd(USART3, DISABLE);//Open serial port 3
TIM_ITConfig(TIM1,TIM_IT_Update, ENABLE);//Open timer 1°

TIM_Cmd(TIM4, ENABLE); // Open pid timer

go_to_point(rec_step_point[0][0], rec_step_point[0][1], rec_step_point[1][0],
rec_step_point[1][1]);
go_to_point(rec_step_point[1][0], rec_step_point[1][1], rec_step_point[2][0],
rec_step_point[2][1]);
go_to_point(rec_step_point[2][0], rec_step_point[2][1], rec_step_point[3][0],
rec_step_point[3][1]);
go_to_point(rec_step_point[3][0], rec_step_point[3][1], rec_step_point[0][0],
rec_step_point[0][1]);

go_to_point(rec_step_point[0][0], rec_step_point[0][1], rec_step_point[1][0],
rec_step_point[1][1]);
go_to_point(rec_step_point[1][0], rec_step_point[1][1], rec_step_point[2][0],
rec_step_point[2][1]); go_to_point(rec_step_point[2][0], rec_step_point[2][1],
rec_step_point[3][0], rec_step_point[3][1]);
go_to_point(rec_step_point[3][0], rec_step_point[3][1], rec_step_point[0][0],
rec_step_point[0][1]);
step_flag+=1;



}
```

Note: The first gimbal must first go to the position of the function defined by the serial port to turn off the serial port 3 enable. For this code, first define the flag bit, then open the serial port 3 to run the coordinate acquisition function, after the acquisition is completed, another flag bit will be assigned, and then the two flag bits will be judged to continue execution, and then the serial port 3 will be closed to prevent the serial port from receiving data and occupying the pid calculation and servo pwm movement process, turn on the pid calculation timer and pwm movement timer, and then execute the subdivided pid movement function to control the servo to move in the rectangular frame. After the movement is completed, add 1 to the flag bit to stop the servo.

7. Analysis of the key code of the second gimbal

- control.c

```
uint8_t get_point2_p() //Coordinate processing function
{
uint8_t lable, load_1, load_2 ,load_3 ,ret=0;
if(DataDecode1(&Uart3_RingBuff,&lable, &load_1, &load_2,&load_3) == 0){
if(lable == 0x00){
rec_step_p[0][0] = load_1 + load_2 + load_3 ;
// ret=1;

} else if(lable == 0xff){

rec_step_p[0][1] = load_1 + load_2 + load_3 ;
// ret=1;

}

return ret;
}
}
```

```
```

This code adds together the three frames of data transmitted by k230. As mentioned earlier, the coordinate data of k230 is split into three frames and then sent separately. Here, the data is added together to form the correct coordinates and assign the coordinates to the two-dimensional array.

```
int a_Err=(int)x_pid.Err, b_Err=(int)y_pid.Err;

get_point2_p();
if(abs(a_Err) + abs(b_Err) < 800){/* Sound and light reminder*/
BEEP_ON;
}
else {
BEEP_OFF;
}
set_target(rec_step_p[0][0],rec_step_p[0][1]);
```

This code uses the coordinates as the pid target value based on the red laser coordinates obtained above, and then controls the servo movement, and issues a sound and light reminder when it is less than the sum of the x-axis and y-axis errors.

8.2D PTZ important code analysis

- bsp_servo.c

```
ServTypdef Xserv = {
.pwm = 1022,
.MAXPWM = 2400,
.MINPWM = 500,
.MIDPWM = 1097
};

ServTypdef Yserv = {
.pwm = 1022,
.MAXPWM = 2100,
.MINPWM = 400,
.MIDPWM = 933
};
```

This code sets the starting position of the servo. The positions of the first and second gimbals are different, so this needs to be modified to fit the vertex of the gimbal movement. Since the 0 point of x and y on the k230 is in the upper left corner, I set the initial position of the gimbal to the upper left corner, and then the gimbal can be adjusted proportionally through the coordinate changes of the k230. If you need to adjust the gimbal position and fine-tune the position, you need to adjust this position parameter.

```
void servo_ctr(ServTypdef *servo, int PidInput)
{
uint16_t CompSet = servo->MIDPWM + PidInput;

if( CompSet > servo->MAXPWM ){
```

```
CompSet = servo->MAXPWM;
} else if ( CompSet < servo->MINPWM ){
CompSet = servo->MINPWM;
}
if(servo->pwm == CompSet){/*Avoid repeated changes in comparison value*/

return;
}

if( servo == &Xserv ){

GPIO_SetBits(Servo_J1_PORT, Servo_J1_PIN ); //Set the servo level high
delay_us(CompSet); //Delay pulse value in microseconds

GPIO_ResetBits(Servo_J1_PORT, Servo_J1_PIN ); //Set the servo level low
delay_ms(20 - CompSet/1000);
}
else if( servo == &Yserv ){

GPIO_SetBits(Servo_J2_PORT, Servo_J2_PIN ); //Set the servo level high
delay_us(CompSet); //Delay pulse value in microseconds

GPIO_ResetBits(Servo_J2_PORT, Servo_J2_PIN ); //Set the servo level low
delay_ms(20 - CompSet/1000);
}
}
```

This code outputs the pid to the pwm, connects the pid to the pwm, and limits the pwm. It is also the core code for the servo movement. It controls the rotation of the servo through the delay and the change of the pwm, and connects the servo movement with the pwm.

- pid.c

```
float pid_calculate(PID_TypeDef *PID, float CurrentValue)
{
PID->Err = PID->Target - CurrentValue;
PID->Integral += PID->Err;
if(PID->Integral > 7000){
PID->Integral = 7000;
}
if(PID->Integral < -7000){
PID->Integral = -7000;
} //Points limiter
PID->PID_out = PID->Kp * PID->Err /*proportion*/
+ PID->Ki * PID->Integral /*Points*/
+ PID->Kd * (PID->Err - PID->LastErr); /*Derivative*/

PID->LastErr = PID->Err;
return PID->PID_out;
}
```

This code is the configuration of the position PID operation, including integral limiting. The position PID is calculated by the current deviation value (difference between target value and actual value), integral value (sum of past errors) and differential value (error change rate), and then the weighted addition of these three parts is used to obtain the PID output.

- bsp_usart.c

```c
uint8_t DataDecode1(RingBuff_t *ringbuff, uint8_t *data1, uint8_t *data2, uint8_t
*data3,uint8_t *data4)
{
static uint8_t uart_dec_count;
static uint8_t uart_rec_data[6];
uint8_t ret = 1;

if(Read_RingBuff(ringbuff, &uart_rec_data[uart_dec_count]) == RINGBUFF_ERR){
return 1;

}
if((uart_dec_count == 0)&&(uart_rec_data[uart_dec_count] != 0x55)) { //Frame header 0x55
uart_rec_data[uart_dec_count] = 0;
}
else if((uart_dec_count == 1)&&(uart_rec_data[uart_dec_count] != 0xaa)){ //Second frame 0xaa
uart_rec_data[uart_dec_count] = 0;
uart_rec_data[uart_dec_count-1] = 0;
uart_dec_count = 0;
}else if((uart_dec_count == 6)&&(uart_rec_data[uart_dec_count] != 0xfa)){ //Frame end 0xfa
uart_rec_data[uart_dec_count] = 0;
uart_rec_data[uart_dec_count-1] = 0;
uart_rec_data[uart_dec_count-2] = 0;
uart_rec_data[uart_dec_count-3] = 0; uart_rec_data[uart_dec_count-4] = 0;
uart_rec_data[uart_dec_count-5] = 0;
uart_dec_count = 0;
}
else{
if(uart_dec_count == 6)//Successfully received one frame of data
{
*data1 = uart_rec_data[2];
*data2 = uart_rec_data[3];
*data3 = uart_rec_data[4];
*data4 = uart_rec_data[5];
ret = 0;

}
uart_dec_count++;
if(uart_dec_count == 7)
{
uart_dec_count = 0;
}

}
return ret;
}
```

This code is for receiving k230 data. It receives data only when the frame header is 0x55, the second data is 0xaa, and the frame tail is 0xfa. It locates a frame of data through three data. Since the coordinates of the four vertices transmitted from k230 are each split into three groups of data for transmission, the transmitted data is very large. In order to ensure that the transmitted data is accurate, three data are used to confirm a frame of data.

9.Effects Display

1. Pencil frame effect display

This question requires moving around in a circle within 30 seconds without leaving the line. Our movement time is less than 30 seconds, which perfectly meets the requirements and can return to the center point.

2. Basic part frame effect display

This question requires moving in a circle within a rectangular frame within 30 seconds. From the video, we can see that the time is perfectly consistent and the movement is also relatively good. Since drawing a regular rectangle is too simple, we directly show the drawing of an arbitrary frame.

3. Partial pan-tilt tracking effect display

This question requires the red laser pen to move in a circle, and the green laser pen follows the movement. The distance between the two should not exceed 3cm. There is a sound reminder when they are within the range. From the video, we can see that the red laser pen is in good motion and the green laser pen does not exceed 3cm. There is a sound reminder when it catches up, which perfectly meets the requirements of the question.