Face Tracking

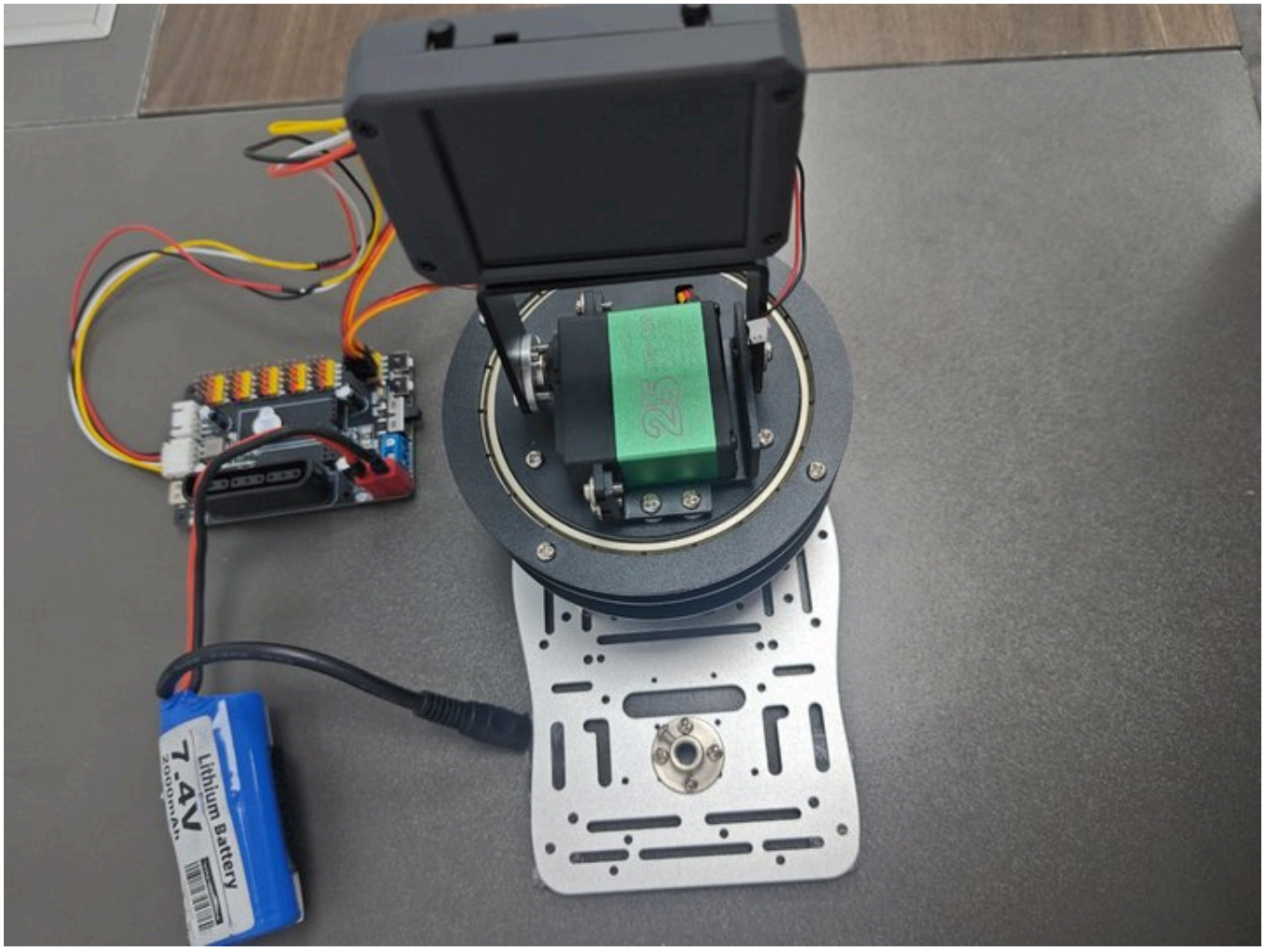
1. Introduction

Please read the information in the 24-way Servo Driver Board and STM32 Getting Started Tutorial first to understand how to burn programs and the basics of STM32. This will make it easier to learn and burn the following programs.

2. Equipment list

- k230 x1
- 24-way servo driver board x1
- 7.4v battery x1 Buy
- Aluminum alloy plate x1 Buy

3. Placement

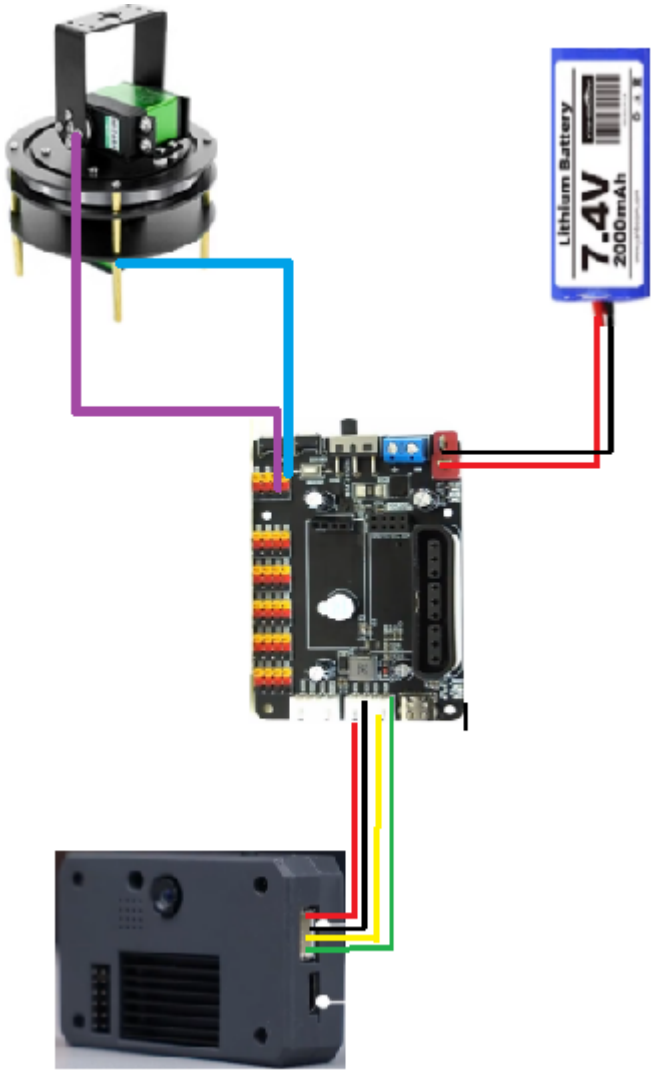


The placement is as shown in the figure. You can choose our aluminum base frame as the base plate to fix the k230 and the gimbal, or you can choose a fixed base plate. The chassis can be glued to prevent the gimbal from moving too much and causing deviation. The k230 is fixed on the two-dimensional gimbal. When using the battery, if the gimbal rotates too much, you can moderately reduce the pid parameter.

4. Experimental preparation

First, build the 2D gimbal according to the 2D gimbal installation tutorial

Hardware wiring:



2D gimbal	24-way servo driver board
X-axis servo	S1
Y-axis servo	S2

K230 vision module	24-way motor driver board
5V	5V
GND	GND
TX	RX3
RX	TX3

Note: When plugging in the servo cable, be sure to align the yellow cable with the yellow GPIO port of the 24-way servo driver board, and the brown cable with the black GPIO port of the 24-way servo driver board.

5. K230 key code analysis

- ## Import key dependencies

```
from libs.PipeLine import PipeLine, ScopedTiming
# Pipeline is a module that integrates multimedia functions like camera and LCD display,
greatly simplifying camera and display operations

# ScopedTiming is a code execution timer, refer to example code for usage

from libs.AIBase import AIBase
from libs.AI2D import Ai2d
# AIBase and AI2D are used to process the underlying logic of AI
# AIBase is the base class for all AI functionality implementations
# AI2D is used for image processing

import os
import ujson
# os and ujson provide system-related operations and JSON data-related operations
respectively, not every routine can use them
# os and ujson provide system operations and JSON data operations respectively, not required
for every example

from media.media import *
from time import *
import nncase_runtime as nn
# nncase is the core module for AI inference on K230, providing convenient methods for users
to call K230's KPU

import ulab.numpy as np
# ulab.numpy is ported from Python's numpy, used for necessary matrix operations in AI
computations

import time
import utime
import image
import random
import gc
import sys
import aidemo
# aidemo is also one of the core modules, and many AI gameplays are pre-made in the K230
firmware
# aidemo is another core module, K230 firmware includes many pre-configured AI applications
# Complex methods in these applications can be easily called through the aidemo module

import _thread
# _thread is the threading module, as detailed in previous chapters, no need to elaborate
further
```

This code is to obtain 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, and then obtaining the coordinates of the four vertices of the center rectangle.

- **Custom face detection class**

```
# Custom face detection class, inherited from AIBase base class
class FaceDetectionApp(AIBase):
def __init__(self, kmodel_path, model_input_size, anchors, confidence_threshold=0.5,
nms_threshold=0.2, rgb888p_size=[224,224], display_size=[1920,1080], debug_mode=0):
# Call parent class constructor
super().__init__(kmodel_path, model_input_size, rgb888p_size, debug_mode)

# Path to the model file
self.kmodel_path = kmodel_path

# Model input resolution
self.model_input_size = model_input_size

# Confidence threshold: minimum confidence requirement for detection results Confidence
threshold: minimum confidence requirement for detection results
self.confidence_threshold = confidence_threshold

# NMS threshold: threshold for Non-Maximum Suppression
self.nms_threshold = nms_threshold

# Anchor data: predefined boxes for object detection
self.anchors = anchors

# Image resolution from sensor to AI, width aligned to 16
self.rgb888p_size = [ALIGN_UP(rgb888p_size[0], 16), rgb888p_size[1]]

# Display resolution, width aligned to 16
self.display_size = [ALIGN_UP(display_size[0], 16), display_size[1]]

# Debug mode flag / Debug mode flag
self.debug_mode = debug_mode

# Initialize AI2D object for image preprocessing
self.ai2d = Ai2d(debug_mode)

# Set AI2D input/output format
self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT, nn.ai2d_format.NCHW_FMT, np.uint8,
np.uint8)

def config_preprocess(self, input_image_size=None):
with ScopedTiming("set preprocess config", self.debug_mode > 0):
# Get AI2D input size
ai2d_input_size = input_image_size if input_image_size else self.rgb888p_size

# Get padding parameters
top, bottom, left, right = self.get_padding_param()

# Set padding: [top, bottom, left, right], padding value [104,117,123] / Set padding: [top,
bottom, left, right], padding value [104,117,123]
self.ai2d.pad([0, 0, 0, 0, top, bottom, left, right], 0, [104, 117, 123])
```

```
# Set resize method: bilinear interpolation / Set resize method: bilinear interpolation /
self.ai2d.resize(nn.interp_method.tf_bilinear, nn.interp_mode.half_pixel)

# Build preprocessing pipeline / Build preprocessing pipeline
self.ai2d.build([1,3,ai2d_input_size[1],ai2d_input_size[0]],
[1,3,self.model_input_size[1],self.model_input_size[0]])

def postprocess(self, results):
with ScopedTiming("postprocess", self.debug_mode > 0):
# Call aidemo library for face detection post-processing / Call aidemo library for face
detection post-processing
post_ret = aidemo.face_det_post_process(self.confidence_threshold,
self.nms_threshold,
self.model_input_size[1],
self.anchors,
self.rgb888p_size,
results)
return post_ret[0] if post_ret else post_ret


def draw_result(self, pl, dets):

with ScopedTiming("display_draw", self.debug_mode > 0):
if dets:
# Clear previous frame's OSD drawing
pl.osd_img.clear()
global last_cx
global last_cy
for det in dets:
# Convert detection box coordinates to display resolution
x, y, w, h = map(lambda x: int(round(x, 0)), det[:4])
# Draw yellow detection box
pl.osd_img.draw_rectangle(x, y, w, h, color=(255, 255, 0, 255), thickness=2)
return x , y, 1
return last_cx, last_cy, 0
return last_cx, last_cy, 0

def get_padding_param(self):
# Calculate scaling ratio between model input and actual image
dst_w = self.model_input_size[0]
dst_h = self.model_input_size[1]
ratio_w = dst_w / self.rgb888p_size[0]
ratio_h = dst_h / self.rgb888p_size[1]
ratio = min(ratio_w, ratio_h)

# Calculate new dimensions after scaling
new_w = int(ratio * self.rgb888p_size[0])
new_h = int(ratio * self.rgb888p_size[1])

# Calculate padding values
dw = (dst_w - new_w) / 2
dh = (dst_h - new_h) / 2
```

```
# Return padding parameters / Return padding parameters
return (int(round(0)),
int(round(dh * 2 + 0.1)),
int(round(0)),
int(round(dw * 2 - 0.1)))
```

- **Perform face detection**

```
def exce_demo(pl):
# Declare global variable face_det
global face_det

# Get display-related parameters
display_mode = pl.display_mode # Display mode (e.g., lcd)
rgb888p_size = pl.rgb888p_size # Original image resolution
display_size = pl.display_size # Display resolution

# Set face detection model path
kmodel_path = "/sdcard/kmodel/face_detection_320.kmodel"

# Set model parameters
confidence_threshold = 0.5 # Confidence threshold
nms_threshold = 0.2 # Non-maximum suppression threshold Non-maximum suppression threshold
anchor_len = 4200 # Number of anchor boxes
det_dim = 4 # Detection dimensions (x,y,w,h)

# Load anchor box data
anchors_path = "/sdcard/utils/prior_data_320.bin"
anchors = np.fromfile(anchors_path, dtype=np.float)
anchors = anchors.reshape((anchor_len, det_dim))

try:
# Initialize face detection application instance
face_det = FaceDetectionApp(kmodel_path,
model_input_size=[320, 320],
anchors=anchors,
confidence_threshold=confidence_threshold,
nms_threshold=nms_threshold,
rgb888p_size=rgb888p_size,
display_size=display_size,
debug_mode=0)

# Configure image preprocessing
face_det.config_preprocess()

# Main loop
while True:
with ScopedTiming("total",0): # Timer
img = pl.get_frame() # Get camera frame
res = face_det.run(img) # Run face detection
```

```
face_det.draw_result(pl, res) # Draw detection results
pl.show_image() # Display processed image
x,y,ok= face_det.draw_result(pl, res)
cx_h,cx_l,cx_z,cy_h,cy_l,cy_z = conversion_postion(x,y)
uart.write(bytearray([0x55,0xaa,0x00,int(cx_h),int(cx_l),int(cx_z),0xfa]))
uart.write(bytearray([0x55,0xaa,0xff,int(cy_h),int(cy_l),int(cy_z),0xfa]))
gc.collect() # Garbage collection
time.sleep_us(10) # Brief delay

except Exception as e:
print("Face detection function exits") # Exception exit prompt
finally:
face_det.deinit() # Release resources
```

- **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. stm32 key code analysis

- ## 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 3 frames of data transmitted by k230. As mentioned earlier, the coordinate data of k230 is split into 3 frames and then sent separately. Here, the data is added together to form the correct coordinates, and the coordinates are assigned to the two-dimensional array.

```
get_point2_p();

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 face coordinates obtained above, and then controls the movement of the servo.

- ## bsp_servo.c

```
ServTypdef Xserv = {
.pwm = 2000,
.MAXPWM = 2500,
.MINPWM = 500,
.MIDPWM = 1400
};

ServTypdef Yserv = {
.pwm = 1000,
.MAXPWM = 2300,
.MINPWM = 500,
.MIDPWM = 1000
};
```

This code sets the starting position of the servo, so it needs to be modified to fit the vertex of the gimbal movement. I set the initial position of the gimbal at the upper right 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. If you want to fine-tune the code to achieve better results, you can fine-tune the PID parameters of the main function.

- ## bsp_usart.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.

7. Effect description

The k230 code defines the face type. When a face is recognized through features, the k230 transmits the face coordinates to the 2D gimbal, and then the 2D gimbal follows the face movement, thus achieving the effect of tracking the face.