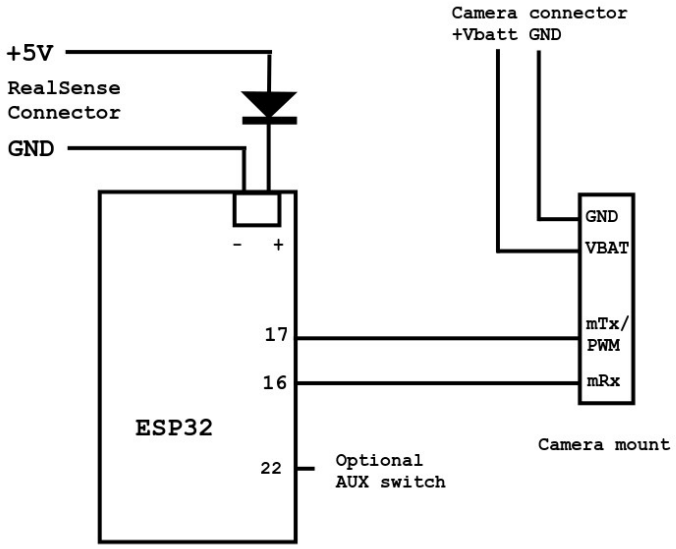
Volle Funktion der CGO3+ am Thunderbird

Um den *Pan, Tilt, Pan Mode, and Tilt Mode* zum Laufen zu bringen, werden einfach RC-Daten, die von der ST16 an die Kamera über WiFi empfangen wurden, in Gimbal-Steuermeldungen umgewandelt und über UART an den Gimbal gesendet. Um dies zu erreichen, verwende ich ein ESP32 MCU-Board. die 5V vom RealSense-Anschluss und die bestehende Kamera-Stromversorgung vom Mainboard.



Das ESP32 MCU habe ich außen angebracht, um es leicht über USB updaten zu können.

Anmerkung: Die mittlere Position des Pan Mode Schalters ist jetzt ein Winkelmodus. Er funktioniert wie bei den neueren Kameras C23 oder E90. Die Kamera zeigt auf den Winkel des Pan-Drehknopfes. Wenn du das nicht willst, kannst du das im Sketch auskommentieren.

Code:

```
/*
This device acts as controller for the CGO3+ on Thunderbird with ST16.
The processor reads the WiFi channel data from ST16 and send following
data via control message to CGO3+:
- pan
- tilt
- pan mode
- tilt mode.
Camera control, settings, and video stream all using WiFi connection.

You need a power source for the camera 12-16V (VBAT, GND) and a step-down
converter to 5V (or 3.3V depending on ESP32 module type) for the ESP32.
CGO3_TXD2 to cam mRx/PWM
CGO3_RXD2 to cam mTx
Wiring depends on HW port definition below.
*/

#define AUX_PIN 22          // Built-in LED
#define CGO3_RXD2 16        // mTx/PWM  (black)
#define CGO3_TXD2 17        // mRx      (brown)
#define sendbuffer_size 36
#define UART_speed 115200
#define panmode_F 683       // Pan mode Fixed, camera points forward
#define panmode_A 830       // Pan mode Angle - camera points to angle depending knob: 0x033E
#define panmode_P 1502      // Pan mode Position
const uint16_t X25_INIT_CRC = 0xFFFF;
const byte FEheader = 0xFE;

byte cgo3buffer[47];
byte cgo3sequno = 0;

//                                                                                                  pan------ tilt-----
pan mode- tilt mode          CRC_l CRC_h
byte cgo3send_buffer[sendbuffer_size] = {0xFE,26,0,1,0,2,0,1,0,0,0,0,0,0,0,0,0,0,0,0x37,0,8,0, 0x00,0x08,0xAB,0x02,
0,8, 0xAB,0x02,0x88,0x08, 0xF4,1, 0x00, 0x00};

static inline void crc_accumulate(uint8_t data, uint16_t *crcAccum) {
  // Accumulate one byte of data into the CRC
  uint8_t tmp;
  tmp = data ^ (uint8_t)(*crcAccum &0xFF);
  tmp ^= (tmp<<4);
  *crcAccum = (*crcAccum>>8) ^ (tmp<<8) ^ (tmp <<3) ^ (tmp>>4);
}

uint16_t UpscaleTo150 (int val, bool convert = true) {                  // Scale 683 to 3412 (100%)
  if (convert) {
    val -= panmode_F;
    if (val < 0) {val = 0;}
    float val_asfloat = val * 4095.0 / 2729.0;                          // Scale 0 to 4095 (150%)
    val = round(val_asfloat);
  }
  return (val & 0x0FFF);                                                // 12bit per channel
}

void setup() {
  btStop();
//  Serial.begin(UART_speed);                                           // Debug
  Serial2.begin(UART_speed, SERIAL_8N1, CGO3_RXD2, CGO3_TXD2);         // Camera
  pinMode(AUX_PIN, OUTPUT);
  delay(100);
  digitalWrite(AUX_PIN, LOW);
}
```

```
void loop() {
  uint16_t crc16;
  uint16_t aux;
  byte cgo3len = 0;
  byte incomingByte;
  int numreadbytes;

  uint16_t panmode = panmode_F;

  if (Serial2.available() > 1) {
    incomingByte = Serial2.read();
    cgo3len = Serial2.peek();
    if ((incomingByte == FEheader) && (cgo3len == 38)) {    // Possible a message ChannelData 5GHz.
      numreadbytes = Serial2.readBytes(cgo3buffer, cgo3len+9);
      if ((cgo3len > 0) && (numreadbytes == cgo3len+9) &&  (cgo3buffer[2] == 4) && (cgo3buffer[6] == 8)) {   //
This SysID and Message ID contains channel data

// Read pan mode first, we need it later
        panmode = ((cgo3buffer[39] & 0x0F) << 8) + cgo3buffer[40];
        aux = ((cgo3buffer[36] & 0x0F) << 8) + cgo3buffer[37];          // 12 bytes per channel, camera pan

// Change panmode Position to panmode Angle (like E90)
        if (panmode == panmode_P) {                                     // Optional: Change pan mode to "Angle"
          panmode = panmode_A;
          aux = UpscaleTo150(aux);
        }
        cgo3send_buffer[22] = lowByte(aux);                             // Camera pan
        cgo3send_buffer[23] = highByte(aux);
        cgo3send_buffer[28] = lowByte(panmode);
        cgo3send_buffer[29] = highByte(panmode);

// Read and write tilt and tilt mode
        aux = (cgo3buffer[35] << 4) + (cgo3buffer[36] >> 4);            // Camera  tilt
        cgo3send_buffer[24] = lowByte(aux);
        cgo3send_buffer[25] = highByte(aux);
        aux = (cgo3buffer[38] << 4) + (cgo3buffer[39] >> 4);            // Camera  tilt mode
        cgo3send_buffer[30] = lowByte(aux);
        cgo3send_buffer[31] = highByte(aux);

        if ((cgo3buffer[42] & 0x0F) < 8) {                              // Optional: AUX button channel 11
          digitalWrite(AUX_PIN, HIGH);
        } else {
          digitalWrite(AUX_PIN, LOW);
        }

// Prepare CGO3+ control message, add CRC16, and send it to camera
        cgo3send_buffer[2] = cgo3sequno;
        crc16 = 0xFFFF;
        for (uint8_t k=1; k<34; k++) {
          crc_accumulate(cgo3send_buffer[k], &crc16);
        }
        crc_accumulate(0, &crc16);
        cgo3send_buffer[34] = lowByte(crc16);
        cgo3send_buffer[35] = highByte(crc16);
        Serial2.write(cgo3send_buffer, sendbuffer_size);
        cgo3sequno++;
      }
    }
  }
}
```