

Dron kompensacja masa



drone log 2026 02 01 04 28 11

1	8.60899999999674	0	0
2	8.60899999999674	0	0
3	8.639999999999418	0	0
4	8.686999999999808	0	0
5	8.718999999997322	0	0
6	8.780999999995402	0	0
	t_s	base	action

Mam drona z zaprojektowaną płytka PCB z STM32. Masa nie jest równomiernie rozłożona. Mam 4x silniki 8520 z zasilaniem 3.7V z baterii, STM steruje PWM z zakresu 0-10000 na tranzystory przepuszczające napięcie z silników. Możliwe, że niektóre silniki są mocniejsze. Pierwsza uwaga - mam dobre oznaczenia silników (LF RF RB LB), nie próbuj mnie przekonywać, że coś jest nie tak i trzeba je zmienić. Druga rzecz, mam dobre SIGN'y Pitch i Roll, tego też mi nie wmawiąj, że trzeba zmienić. Pochylanie drona w każdą stronę odpowiednio zmienia Pitch i Roll. Oto mój kod:

```
/* USER CODE BEGIN Header */
/** 
 ****
 * @file      : main.c
 * @brief     : Main program body
 ****
 */
/* USER CODE END Header */
/* Includes ----- */
#include "main.h"
#include "i2c.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----- */
/* USER CODE BEGIN Includes */
#include "MPU6050.h"
#include "Madgwick.h"
#include "protocol.h"
#include <stdio.h>
#include <string.h>
#include <math.h>
```

```
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */
typedef enum {
    CTRL_MANUAL = 0,
    CTRL_STAB  = 1
} ControlMode_t;
/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL 16.0f
#define STAB_KD_ROLL 2.0f
#define STAB_KI_ROLL 0.25f

#define STAB_KP_PITCH 16.0f
#define STAB_KD_PITCH 2.0f
#define STAB_KI_PITCH 0.25f

#define STAB_KI_YAW_RATE 0.0f

#define STAB_I_MAX 1600.0f

#define STAB_KFF_YAW 0.0f
#define STAB_YAW_SIGN 1.0f
#define STAB_YAW_I_MAX 1200.0f

#define YAW_CCW_IS_RF_LB 1

#define IMU_SWAP_ROLL_PITCH 1

#define STAB_ROLL_SIGN 1.0f
```

```

#define STAB_PITCH_SIGN -1.0f

#define CMD_TILT_SLEW_DEG_PER_S 80.0f
#define CMD_YAW_SLEW_DPS_PER_S 900.0f
#define BASE_SLEW_PWM_PER_S 12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S 6000.0f

#define YAW_MIX_FRONT_CCW_REAR_CW 1

#define MOTOR_GAIN_LF 1.000f
#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.150f
#define MOTOR_GAIN_RB 1.100f

#define MOTOR_OFFSETS_LF 0.0f
#define MOTOR_OFFSETS_LB 0.0f
#define MOTOR_OFFSETS_RF 0.0f
#define MOTOR_OFFSETS_RB 0.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG 0.0f
#define PITCH_TRIM_DEG 0.0f

#define LEVEL_CALIB_SAMPLES 200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM 200
#define CG_ROLL_BIAS_PWM -700

#define CG_BIAS_START_PWM 2800u
#define CG_BIAS_FULL_PWM 4800u

#define I_ENABLE_BASE_PWM 3200u
#define I_DISABLE_BASE_PWM 3000u

// =====
// PROFILE SWITCH
// =====
#define CONTROL_PROFILE_GROUND_TEST 0

#if CONTROL_PROFILE_GROUND_TEST
#define NAV_TILT_DEG 20.0f
#define NAV_YAW_RATE_DPS 360.0f
#define NAV_THRUST_BIAS 700
#define NAV_MIX_GAIN 2.0f

#define STAB_U_MAX 2800.0f
#define STAB_YAW_MAX 1200.0f
#define STAB_KP_YAW_RATE 1.0f

```

```

#else
#define NAV_TILT_DEG      10.0f
#define NAV_YAW_RATE_DPS  180.0f
#define NAV_THRUST_BIAS   0
#define NAV_MIX_GAIN      1.0f

#define STAB_U_MAX        3000.0f
#define STAB_YAW_MAX      600.0f
#define STAB_KP_YAW_RATE   1.5f
#endif
/* USER CODE END PD */

/* Private variables -----*/
/* USER CODE BEGIN PV */
MPU6050_t imu;
Madgwick_t filter;

static uint32_t lastTick = 0;

static ImuPayload_t g_imu_last;
static uint8_t g_imu_ready = 0;
static uint32_t g_imu_last_update = 0;

static volatile uint8_t g_tilt_kill = 0;
static uint32_t g_tilt_kill_since = 0;
static uint32_t g_tilt_unkill_since = 0;

static float ax_f = 0.0f, ay_f = 0.0f, az_f = 1.0f;

static PwmPayload_t g_last_pwm_cmd;
static uint8_t g_have_pwm_cmd = 0;

static volatile ControlMode_t g_ctrl_mode = CTRL_MANUAL;
static volatile uint8_t g_armed = 0;

static float i_roll = 0.0f, i_pitch = 0.0f, i_yaw = 0.0f;
static uint8_t g_i_enabled = 0;

static volatile float g_roll_deg = 0.0f;
static volatile float g_pitch_deg = 0.0f;
static volatile float g_gyro_roll_dps = 0.0f;
static volatile float g_gyro_pitch_dps = 0.0f;
static volatile float g_gyro_yaw_dps = 0.0f;
static volatile uint8_t g_att_valid = 0;

static volatile float g_cmd_roll_deg = 0.0f;
static volatile float g_cmd_pitch_deg = 0.0f;
static volatile float g_cmd_yaw_rate_dps = 0.0f;
static volatile uint16_t g_stab_base_pwm = 0;
static volatile int32_t g_nav_bias = 0;

```

```

static volatile float g_target_cmd_roll_deg = 0.0f;
static volatile float g_target_cmd_pitch_deg = 0.0f;
static volatile float g_target_cmd_yaw_rate_dps = 0.0f;
static volatile uint16_t g_target_base_pwm = 0;
static volatile uint8_t g_last_nav_action = NAV_STOP;

static volatile uint8_t g_imu_new = 0;

static volatile float g_level_roll_off = 0.0f;
static volatile float g_level_pitch_off = 0.0f;
static volatile uint8_t g_level_calib_done = 0;
static float g_level_sum_roll = 0.0f;
static float g_level_sum_pitch = 0.0f;
static uint16_t g_level_n = 0;
/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);

/* USER CODE BEGIN PFP */
#define UART_RX_RING_SIZE 256

static volatile uint8_t uart_rx_ring[UART_RX_RING_SIZE];
static volatile uint16_t uart_rx_head = 0;
static volatile uint16_t uart_rx_tail = 0;
static uint8_t uart_rx_it_byte;

static void UartRing_Push(uint8_t b)
{
    uint16_t next = (uint16_t)((uart_rx_head + 1) % UART_RX_RING_SIZE);
    if (next != uart_rx_tail)
    {
        uart_rx_ring[uart_rx_head] = b;
        uart_rx_head = next;
    }
}

static int UartRing_Pop(uint8_t *out)
{
    if (uart_rx_tail == uart_rx_head) return 0;
    *out = uart_rx_ring[uart_rx_tail];
    uart_rx_tail = (uint16_t)((uart_rx_tail + 1) % UART_RX_RING_SIZE);
    return 1;
}

static inline float lpf_alpha(float dt, float cutoff_hz)
{
    const float RC = 1.0f / (2.0f * 3.1415926535f * cutoff_hz);
    return dt / (dt + RC);
}

```

```

static inline void lpf3(float dt, float cutoff_hz,
                      float ax, float ay, float az,
                      float *ox, float *oy, float *oz)
{
    float a = lpf_alpha(dt, cutoff_hz);
    *ox += a * (ax - *ox);
    *oy += a * (ay - *oy);
    *oz += a * (az - *oz);
}

static inline int accel_norm_ok(float ax, float ay, float az, float g_min, float g_max)
{
    float n = sqrtf(ax*ax + ay*ay + az*az);
    return (n >= g_min && n <= g_max) ? 1 : 0;
}

static inline uint16_t clamp_u16(int32_t v, int32_t lo, int32_t hi)
{
    if (v < lo) return (uint16_t)lo;
    if (v > hi) return (uint16_t)hi;
    return (uint16_t)v;
}

static inline float clamp_f(float v, float lo, float hi)
{
    if (v < lo) return lo;
    if (v > hi) return hi;
    return v;
}

static inline float slew_f(float cur, float target, float max_rate_per_s, float dt)
{
    float max_step = max_rate_per_s * dt;
    float d = target - cur;
    if (d > max_step) d = max_step;
    if (d < -max_step) d = -max_step;
    return cur + d;
}

static inline int32_t slew_i32(int32_t cur, int32_t target, float max_rate_per_s, float dt)
{
    float max_step_f = max_rate_per_s * dt;
    int32_t max_step = (int32_t)lroundf(max_step_f);
    int32_t d = target - cur;
    if (d > max_step) d = max_step;
    if (d < -max_step) d = -max_step;
    return cur + d;
}

static float Mixer_ComputeScale(int32_t base, float c_if, float c_rf, float c_lb, float c_rb)
{

```

```

float k = 1.0f;
float c[4] = { c_lf, c_rf, c_lb, c_rb };

for (int i = 0; i < 4; ++i)
{
    float ci = c[i];
    if (ci > 0.0f)
    {
        float lim = ((float)PWM_MAX - (float)base) / ci;
        if (lim < k) k = lim;
    }
    else if (ci < 0.0f)
    {
        float lim = ((float)PWM_STAB_FLOOR - (float)base) / ci;
        if (lim < k) k = lim;
    }
}

if (k > 1.0f) k = 1.0f;
if (k < 0.0f) k = 0.0f;
return k;
}

static float CgBiasScale(uint16_t base_pwm)
{
    if (base_pwm <= CG_BIAS_START_PWM) return 0.0f;
    if (base_pwm >= CG_BIAS_FULL_PWM) return 1.0f;
    return ((float)base_pwm - (float)CG_BIAS_START_PWM) /
        ((float)CG_BIAS_FULL_PWM - (float)CG_BIAS_START_PWM);
}
/* USER CODE END PFP */

/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len)
{
    HAL_UART_Transmit(&huart6, (uint8_t*)ptr, len, HAL_MAX_DELAY);
    return len;
}

static void ESP_LeaveSendMode(void)
{
    const char* cmd = "+";
    HAL_UART_Transmit(&huart6, (uint8_t*)cmd, strlen(cmd), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart6, (uint8_t*)cmd, strlen(cmd), HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart6, (uint8_t*)cmd, strlen(cmd), HAL_MAX_DELAY);
    HAL_Delay(1000);
}

static void ESP_Send(const char *cmd, uint32_t delay_ms)
{
    HAL_UART_Transmit(&huart6, (uint8_t*)cmd, strlen(cmd), HAL_MAX_DELAY);
}

```

```

HAL_UART_Transmit(&huart6, (uint8_t*)"\\r\\n", 2, HAL_MAX_DELAY);
HAL_Delay(delay_ms);
}

static int ESP_WaitFor(const char *expected, uint32_t timeout_ms)
{
    uint32_t t0 = HAL_GetTick();
    uint16_t idx = 0;
    char buf[64] = {0};
    uint8_t c;

    while (HAL_GetTick() - t0 < timeout_ms)
    {
        if (HAL_UART_Receive(&huart6, &c, 1, 10) == HAL_OK)
        {
            if (idx < sizeof(buf) - 1)
            {
                buf[idx++] = (char)c;
                buf[idx] = 0;
            }
            if (strstr(buf, expected)) return 1;
        }
    }
    return 0;
}

static void ESP_ConnectWithRetry(void)
{
    while (1)
    {
        ESP_Send("AT+CIPSTART=\"TCP\",\"192.168.8.100\",3333", 0);
        if (ESP_WaitFor("OK", 3000) || ESP_WaitFor("ALREADY CONNECTED", 1000)) break;
        HAL_Delay(500);
        HAL_Delay(500);
    }
}

void ESP_Init(void)
{
    ESP_LeaveSendMode();
    ESP_Send("AT+RST", 2000);
    ESP_Send("ATE0", 200);

    ESP_Send("AT+CWMODE=1", 200);
    ESP_Send("AT+CIPSTA=\"192.168.8.127\",\"192.168.8.1\",\"255.255.255.0\"", 500);
    ESP_Send("AT+CWJAP=\"HUAWEI-7975\",\"40720250\"", 500);

    ESP_Send("AT+CIPMUX=0", 200);
    ESP_ConnectWithRetry();

    ESP_Send("AT+CIPMODE=1", 200);
}

```

```

ESP_Send("AT+CIPSEND", 0);

if (!ESP_WaitFor(">", 2000)) { while (1) {} }

uint8_t dump;
while (HAL_UART_Receive(&huart6, &dump, 1, 10) == HAL_OK) {}

static void IMU_Init(void)
{
    if (MPU6050_Init(&hi2c1) != HAL_OK) Error_Handler();
    if (MPU6050_CalibrateGyro(&hi2c1, &imu, 1000) != HAL_OK) Error_Handler();

    Madgwick_Init(&filter, 0.05f);
    filter.q0 = 1.0f; filter.q1 = 0.0f; filter.q2 = 0.0f; filter.q3 = 0.0f;

    ax_f = 0.0f; ay_f = 0.0f; az_f = 1.0f;

    lastTick = HAL_GetTick();
    g_imu_last_update = lastTick;

    g_tilt_kill = 0;
    g_tilt_kill_since = 0;
    g_tilt_unkill_since = 0;

    g_cmd_roll_deg = 0.0f;
    g_cmd_pitch_deg = 0.0f;
    g_cmd_yaw_rate_dps = 0.0f;

    g_target_cmd_roll_deg = 0.0f;
    g_target_cmd_pitch_deg = 0.0f;
    g_target_cmd_yaw_rate_dps = 0.0f;

    g_stab_base_pwm = 0;
    g_target_base_pwm = 0;

    g_nav_bias = 0;
    g_last_nav_action = NAV_STOP;

    i_roll = 0.0f;
    i_pitch = 0.0f;
    i_yaw = 0.0f;
    g_i_enabled = 0;

    g_imu_new = 0;

    g_level_roll_off = 0.0f;
    g_level_pitch_off = 0.0f;
    g_level_calib_done = 0;
    g_level_sum_roll = 0.0f;
    g_level_sum_pitch = 0.0f;
}

```

```

g_level_n = 0;

    HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 0);
}

static void PWM_StartAll(void)
{
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
    __HAL_TIM_MOE_ENABLE(&htim1);
}

static void PWM_SetSafe(void)
{
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);
}

static void ReadPwm(PwmPayload_t *p)
{
    p->motor_lb = __HAL_TIM_GET_COMPARE(&htim1, TIM_CHANNEL_1);
    p->motor_if = __HAL_TIM_GET_COMPARE(&htim1, TIM_CHANNEL_2);
    p->motor_rf = __HAL_TIM_GET_COMPARE(&htim1, TIM_CHANNEL_3);
    p->motor_rb = __HAL_TIM_GET_COMPARE(&htim1, TIM_CHANNEL_4);
}

static void SetPwm(const PwmPayload_t *p)
{
    g_last_pwm_cmd = *p;
    g_have_pwm_cmd = 1;

    if (g_tilt_kill)
    {
        PWM_SetSafe();
        return;
    }

    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, p->motor_lb);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, p->motor_if);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3, p->motor_rf);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, p->motor_rb);
}

static void IMU_UpdateContinuous(void)
{
    uint32_t now = HAL_GetTick();
    if ((now - g_imu_last_update) < IMU_UPDATE_PERIOD_MS) return;
}

```

```

g_imu_last_update = now;

MPU6050_ReadRaw(&hi2c1, &imu);
MPU6050_ComputeScaled(&imu);

if (lastTick == 0) { lastTick = now; return; }

float dt = (now - lastTick) * 0.001f;
lastTick = now;

if (dt <= 0.0f) return;
if (dt > 0.02f) dt = 0.02f;

lpf3(dt, ACC_LPF_CUTOFF_HZ, imu.acc_x, imu.acc_y, imu.acc_z, &ax_f, &ay_f, &az_f);
int acc_ok = accel_norm_ok(ax_f, ay_f, az_f, ACC_G_MIN, ACC_G_MAX);

float gx = imu.gyro_x * DEG2RAD;
float gy = imu.gyro_y * DEG2RAD;
float gz = imu.gyro_z * DEG2RAD;

if (acc_ok) Madgwick_UpdateIMU(&filter, gx, gy, gz, ax_f, ay_f, az_f, dt);
else      Madgwick_UpdateIMU(&filter, gx, gy, gz, 0.0f, 0.0f, 0.0f, dt);

float roll = atan2f(
    2.0f * (filter.q0 * filter.q1 + filter.q2 * filter.q3),
    1.0f - 2.0f * (filter.q1 * filter.q1 + filter.q2 * filter.q2)
) * RAD2DEG;

float s = 2.0f * (filter.q0 * filter.q2 - filter.q3 * filter.q1);
s = clamp_f(s, -1.0f, 1.0f);
float pitch = asinf(s) * RAD2DEG;

float yaw = atan2f(
    2.0f * (filter.q0 * filter.q3 + filter.q1 * filter.q2),
    1.0f - 2.0f * (filter.q2 * filter.q2 + filter.q3 * filter.q3)
) * RAD2DEG;

g_imu_last.roll = (int16_t)(roll * 100.0f);
g_imu_last.pitch = (int16_t)(pitch * 100.0f);
g_imu_last.yaw = (int16_t)(yaw * 100.0f);

g_roll_deg = roll;
g_pitch_deg = pitch;

g_gyro_roll_dps = imu.gyro_x;
g_gyro_pitch_dps = imu.gyro_y;
g_gyro_yaw_dps = imu.gyro_z;

g_att_valid = 1;

if (!g_level_calib_done && !g_armed)

```

```

{

    int gyro_ok = (fabsf(imu.gyro_x) < LEVEL_CALIB_GYRO_MAX_DPS) &&
        (fabsf(imu.gyro_y) < LEVEL_CALIB_GYRO_MAX_DPS) &&
        (fabsf(imu.gyro_z) < LEVEL_CALIB_GYRO_MAX_DPS);

    int acc_req_ok = 1;
#if LEVEL_CALIB_ACC_OK_REQUIRED
    acc_req_ok = acc_ok;
#endif

    if (gyro_ok && acc_req_ok)
    {
        g_level_sum_roll += roll;
        g_level_sum_pitch += pitch;
        g_level_n++;

        if (g_level_n >= LEVEL_CALIB_SAMPLES)
        {
            g_level_roll_off = g_level_sum_roll / (float)g_level_n;
            g_level_pitch_off = g_level_sum_pitch / (float)g_level_n;
            g_level_calib_done = 1;

            HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 1);
            HAL_Delay(60);
            HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 0);
        }
    }
    else
    {
        g_level_sum_roll = 0.0f;
        g_level_sum_pitch = 0.0f;
        g_level_n = 0;
    }
}

float aroll = fabsf(roll);
float apitch = fabsf(pitch);

if (!g_tilt_kill)
{
    if (aroll > TILT_KILL_DEG || apitch > TILT_KILL_DEG)
    {
        if (g_tilt_kill_since == 0) g_tilt_kill_since = now;
        if ((now - g_tilt_kill_since) >= TILT_KILL_DEBOUNCE_MS)
        {
            g_tilt_kill = 1;
            g_tilt_unkill_since = 0;
            HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 1);
            PWM_SetSafe();
            g_ctrl_mode = CTRL_MANUAL;
        }
    }
}

```

```

    }
    else g_tilt_kill_since = 0;
}
else
{
    if (aroll < TILT_UNKILL_DEG && apitch < TILT_UNKILL_DEG)
    {
        if (g_tilt_unkill_since == 0) g_tilt_unkill_since = now;
        if ((now - g_tilt_unkill_since) >= TILT_UNKILL_HOLD_MS)
        {
            g_tilt_kill = 0;
            g_tilt_kill_since = 0;
            HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 0);
        }
    }
    else g_tilt_unkill_since = 0;
}

g_imu_new = 1;
g_imu_ready = 1;
}

static void ControlStep_Stabilize(void)
{
    static uint32_t last_ctrl = 0;
    uint32_t now = HAL_GetTick();
    float dt = 0.01f;

    if (last_ctrl != 0)
    {
        dt = (now - last_ctrl) * 0.001f;
        if (dt < 0.001f) dt = 0.001f;
        if (dt > 0.05f) dt = 0.05f;
    }
    last_ctrl = now;

    if (g_tilt_kill) { PWM_SetSafe(); return; }
    if (!g_att_valid) return;

    if (g_target_base_pwm > PWM_MAX) g_target_base_pwm = PWM_MAX;

    g_stab_base_pwm = (uint16_t)lroundf(
        slew_f((float)g_stab_base_pwm, (float)g_target_base_pwm, BASE_SLEW_PWM_PER_S, dt)
    );

    g_cmd_roll_deg = slew_f(g_cmd_roll_deg, g_target_cmd_roll_deg, CMD_TILT_SLEW_DEG_PER_S,
                           dt);
    g_cmd_pitch_deg = slew_f(g_cmd_pitch_deg, g_target_cmd_pitch_deg,
                           CMD_TILT_SLEW_DEG_PER_S, dt);
    g_cmd_yaw_rate_dps = slew_f(g_cmd_yaw_rate_dps, g_target_cmd_yaw_rate_dps,
                           CMD_YAW_SLEW_DPS_PER_S, dt);
}

```

```

int32_t target_bias = 0;
if(g_last_nav_action == NAV_FORWARD || g_last_nav_action == NAV_BACK ||
   g_last_nav_action == NAV_LEFT   || g_last_nav_action == NAV_RIGHT)
{
    target_bias = (int32_t)NAV_THRUST_BIAS;
}
g_nav_bias = slew_i32(g_nav_bias, target_bias, NAV_BIAS_SLEW_PWM_PER_S, dt);

if (!g_i_enabled)
{
    if (g_stab_base_pwm >= I_ENABLE_BASE_PWM) g_i_enabled = 1;
}
else
{
    if (g_stab_base_pwm <= I_DISABLE_BASE_PWM) g_i_enabled = 0;
}

float roll_meas_raw, pitch_meas_raw;
float roll_rate_raw, pitch_rate_raw;

#if IMU_SWAP_ROLL_PITCH
roll_meas_raw = (float)g_pitch_deg - (float)g_level_pitch_off;
pitch_meas_raw = (float)g_roll_deg - (float)g_level_roll_off;

roll_rate_raw = (float)g_gyro_pitch_dps;
pitch_rate_raw = (float)g_gyro_roll_dps;
#else
roll_meas_raw = (float)g_roll_deg - (float)g_level_roll_off;
pitch_meas_raw = (float)g_pitch_deg - (float)g_level_pitch_off;

roll_rate_raw = (float)g_gyro_roll_dps;
pitch_rate_raw = (float)g_gyro_pitch_dps;
#endif

float roll_meas = roll_meas_raw * STAB_ROLL_SIGN;
float pitch_meas = pitch_meas_raw * STAB_PITCH_SIGN;

float roll_rate = roll_rate_raw * STAB_ROLL_SIGN;
float pitch_rate = pitch_rate_raw * STAB_PITCH_SIGN;

float cmd_roll = g_cmd_roll_deg + ROLL_TRIM_DEG;
float cmd_pitch = g_cmd_pitch_deg + PITCH_TRIM_DEG;

float err_roll = cmd_roll - roll_meas;
float err_pitch = cmd_pitch - pitch_meas;

float u_roll_pd = STAB_KP_ROLL * err_roll - STAB_KD_ROLL * roll_rate;
float u_pitch_pd = STAB_KP_PITCH * err_pitch - STAB_KD_PITCH * pitch_rate;

u_roll_pd = clamp_f(u_roll_pd, -STAB_U_MAX, STAB_U_MAX);

```

```

u_pitch_pd = clamp_f(u_pitch_pd, -STAB_U_MAX, STAB_U_MAX);

float yaw_rate = (float)g_gyro_yaw_dps * STAB_YAW_SIGN;
float yaw_cmd = (float)g_cmd_yaw_rate_dps * STAB_YAW_SIGN;
float yaw_rate_err = yaw_cmd - yaw_rate;

float u_yaw = STAB_KP_YAW_RATE * yaw_rate_err
    + STAB_KI_YAW_RATE * i_yaw
    + STAB_KFF_YAW * yaw_cmd;

u_yaw = clamp_f(u_yaw, -STAB_YAW_MAX, STAB_YAW_MAX);

float u_yaw_total = u_yaw + YAW_TRIM;
u_yaw_total = clamp_f(u_yaw_total, -STAB_YAW_MAX, STAB_YAW_MAX);

float u_roll = u_roll_pd + STAB_KI_ROLL * i_roll;
float u_pitch = u_pitch_pd + STAB_KI_PITCH * i_pitch;

u_roll = clamp_f(u_roll, -STAB_U_MAX, STAB_U_MAX);
u_pitch = clamp_f(u_pitch, -STAB_U_MAX, STAB_U_MAX);

int32_t base = (int32_t)g_stab_base_pwm;

float c_lf = (-u_pitch + u_roll);
float c_rf = (-u_pitch - u_roll);
float c_lb = (u_pitch + u_roll);
float c_rb = (u_pitch - u_roll);

float nav = (float)g_nav_bias * NAV_MIX_GAIN;
switch ((uint8_t)g_last_nav_action)
{
    case NAV_FORWARD: c_lb += nav; c_rb += nav; c_lf -= nav; c_rf -= nav; break;
    case NAV_BACK: c_lb -= nav; c_rb -= nav; c_lf += nav; c_rf += nav; break;
    case NAV_LEFT: c_rb += nav; c_rf += nav; c_lb -= nav; c_lf -= nav; break;
    case NAV_RIGHT: c_lb += nav; c_lf += nav; c_rb -= nav; c_rf -= nav; break;
    default: break;
}

float cgk = CgBiasScale(g_stab_base_pwm);
float pitch_bias = (float)CG_PITCH_BIAS_PWM * cgk;
float roll_bias = (float)CG_ROLL_BIAS_PWM * cgk;

c_lf += pitch_bias;
c_rf += pitch_bias;
c_lb -= pitch_bias;
c_rb -= pitch_bias;

c_lf += roll_bias;
c_lb += roll_bias;
c_rf -= roll_bias;
c_rb -= roll_bias;

```

```

#ifndef YAW_CCW_IS_RF_LB
    c_rf += u_yaw_total;
    c_lb += u_yaw_total;
    c_lf -= u_yaw_total;
    c_rb -= u_yaw_total;
#else
    c_lf += u_yaw_total;
    c_rb += u_yaw_total;
    c_rf -= u_yaw_total;
    c_lb -= u_yaw_total;
#endif

float k = Mixer_ComputeScale(base, c_lf, c_rf, c_lb, c_rb);
int sat = (k < 0.999f) ? 1 : 0;

if (!sat && g_i_enabled)
{
    i_roll += err_roll * dt;
    i_pitch += err_pitch * dt;
    i_yaw += yaw_rate_err * dt;

    i_roll = clamp_f(i_roll, -STAB_I_MAX, STAB_I_MAX);
    i_pitch = clamp_f(i_pitch, -STAB_I_MAX, STAB_I_MAX);
    i_yaw = clamp_f(i_yaw, -STAB_YAW_I_MAX, STAB_YAW_I_MAX);
}
else
{
    if (!g_i_enabled)
    {
        i_roll *= (1.0f - 0.8f * dt);
        i_pitch *= (1.0f - 0.8f * dt);
        i_yaw *= (1.0f - 0.8f * dt);
    }
}

float mix_lf = k * c_lf;
float mix_rf = k * c_rf;
float mix_lb = k * c_lb;
float mix_rb = k * c_rb;

float out_lf_f = (float)base + mix_lf * MOTOR_GAIN_LF + MOTOR_OFFSET_LF;
float out_rf_f = (float)base + mix_rf * MOTOR_GAIN_RF + MOTOR_OFFSET_RF;
float out_lb_f = (float)base + mix_lb * MOTOR_GAIN_LB + MOTOR_OFFSET_LB;
float out_rb_f = (float)base + mix_rb * MOTOR_GAIN_RB + MOTOR_OFFSET_RB;

int32_t lf = (int32_t)lroundf(out_lf_f);
int32_t rf = (int32_t)lroundf(out_rf_f);
int32_t lb = (int32_t)lroundf(out_lb_f);
int32_t rb = (int32_t)lroundf(out_rb_f);

```

```

PwmPayload_t out;
out.motor_lf = clamp_u16(lf, PWM_STAB_FLOOR, PWM_MAX);
out.motor_rf = clamp_u16(rf, PWM_STAB_FLOOR, PWM_MAX);
out.motor_lb = clamp_u16(lb, PWM_STAB_FLOOR, PWM_MAX);
out.motor_rb = clamp_u16(rb, PWM_STAB_FLOOR, PWM_MAX);

    SetPwm(&out);
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();

    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_TIM1_Init();
    MX_USART6_UART_Init();

    PWM_StartAll();
    PWM_SetSafe();

    ESP_Init();
    IMU_Init();
    Protocol_Init();

    uart_rx_head = 0;
    uart_rx_tail = 0;

    HAL_UART_AbortReceive(&huart6);
    __HAL_UART_CLEAR_OREFLAG(&huart6);
    HAL_UART_Receive_IT(&huart6, &uart_rx_it_byte, 1);

    while (1)
    {
        IMU_UpdateContinuous();

        if (!g_armed || g_tilt_kill)
            PWM_SetSafe();
        else if (g_ctrl_mode == CTRL_STAB)
        {
            if (g_imu_new)
            {
                g_imu_new = 0;
                ControlStep_Stabilize();
            }
        }

        uint8_t b;
        while (UartRing_Pop(&b))
        {

```

```

ProtoResult_t r = Protocol_PushByte(b);

if(r == PROTO_OK)
{
    const ProtoRxFrame_t *rx = Protocol_GetFrame();

    uint8_t tx[64];
    uint8_t tx_len = 0;

    if(rx->cmd == CMD_PING)
    {
        tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_PONG, NULL, 0, tx, sizeof(tx));
    }
    else if(rx->cmd == CMD_PWM_READ)
    {
        PwmPayload_t pwm;
        ReadPwm(&pwm);
        tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_PWM_DATA,
                                      (const uint8_t*)&pwm, PWM_PAYLOAD_SIZE,
                                      tx, sizeof(tx));
    }
    else if(rx->cmd == CMD_PWM_SET)
    {
        if(rx->len != PWM_PAYLOAD_SIZE)
        {
            tx_len = Protocol_BuildStatusFrame(ST_ERR_BAD_LEN, tx, sizeof(tx));
        }
        else
        {
            PwmPayload_t pwm;
            memcpy(&pwm, rx->data, PWM_PAYLOAD_SIZE);

            g_ctrl_mode = CTRL_MANUAL;

            g_target_cmd_roll_deg = 0.0f;
            g_target_cmd_pitch_deg = 0.0f;
            g_target_cmd_yaw_rate_dps = 0.0f;

            g_cmd_roll_deg = 0.0f;
            g_cmd_pitch_deg = 0.0f;
            g_cmd_yaw_rate_dps = 0.0f;

            g_last_nav_action = NAV_STOP;
            g_nav_bias = 0;

            i_roll = 0.0f;
            i_pitch = 0.0f;
            i_yaw = 0.0f;
            g_i_enabled = 0;

            SetPwm(&pwm);
        }
    }
}

```

```

tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_PWM_ACK, NULL, 0, tx, sizeof(tx));
}

else if (rx->cmd == CMD_IMU_READ)
{
    if (!g_imu_ready)
    {
        tx_len = Protocol_BuildStatusFrame(ST_ERR_IMU_NOT_READY, tx, sizeof(tx));
    }
    else
    {
        ImuPayload_t snap = g_imu_last;
        tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_IMU_DATA,
                                      (const uint8_t*)&snap, IMU_PAYLOAD_SIZE,
                                      tx, sizeof(tx));
    }
}

else if (rx->cmd == CMD_TELEM_READ)
{
    if (!g_imu_ready)
    {
        tx_len = Protocol_BuildStatusFrame(ST_ERR_IMU_NOT_READY, tx, sizeof(tx));
    }
    else
    {
        TelemetryPayload_t tp;
        tp.imu = g_imu_last;
        ReadPwm(&tp.pwm);

        tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_TELEM_DATA,
                                      (const uint8_t*)&tp, TELEMETRY_PAYLOAD_SIZE,
                                      tx, sizeof(tx));
    }
}

else if (rx->cmd == CMD_STAB_SET)
{
    if (rx->len != STAB_PAYLOAD_SIZE)
    {
        tx_len = Protocol_BuildStatusFrame(ST_ERR_BAD_LEN, tx, sizeof(tx));
    }
    else
    {
        StabPayload_t sp;
        memcpy(&sp, rx->data, STAB_PAYLOAD_SIZE);

        if (sp.base_pwm > PWM_MAX) sp.base_pwm = PWM_MAX;

        ControlMode_t prev_mode = g_ctrl_mode;
        uint8_t prev_armed = g_armed;

        q_target_base_pwm = sp.base_pwm;
    }
}

```

```

g_ctrl_mode = CTRL_STAB;
if (g_level_calib_done) g_armed = 1;
else g_armed = 0;

g_target_cmd_roll_deg = 0.0f;
g_target_cmd_pitch_deg = 0.0f;
g_target_cmd_yaw_rate_dps = 0.0f;

g_last_nav_action = NAV_STOP;
g_nav_bias = 0;

if (prev_mode != CTRL_STAB || !prev_armed) {
    i_roll = 0.0f;
    i_pitch = 0.0f;
    i_yaw = 0.0f;
    g_i_enabled = 0;
}

tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_STAB_ACK,
                             (const uint8_t*)&sp, STAB_PAYLOAD_SIZE,
                             tx, sizeof(tx));
}
}
else if (rx->cmd == CMD_STAB_OFF)
{
g_ctrl_mode = CTRL_MANUAL;
g_last_nav_action = NAV_STOP;
g_nav_bias = 0;

i_roll = 0.0f;
i_pitch = 0.0f;
i_yaw = 0.0f;
g_i_enabled = 0;

PWM_SetSafe();
tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_STAB_OFF_ACK, NULL, 0, tx, sizeof(tx));
}
else if (rx->cmd == CMD_NAV_SET)
{
if (rx->len != NAV_PAYLOAD_SIZE)
{
    tx_len = Protocol_BuildStatusFrame(ST_ERR_BAD_LEN, tx, sizeof(tx));
}
else
{
    NavPayload_t np;
    memcpy(&np, rx->data, NAV_PAYLOAD_SIZE);

    if (np.base_pwm > PWM_MAX) np.base_pwm = PWM_MAX;
}
ControlMode_t prev_mode = g_ctrl_mode;

```

```

uint8_t prev_armed = g_armed;

g_target_base_pwm = np.base_pwm;
g_ctrl_mode = CTRL_STAB;
g_armed = 1;

g_last_nav_action = np.action;
g_nav_bias = 0;

if (prev_mode != CTRL_STAB || !prev_armed) {
    i_roll = 0.0f;
    i_pitch = 0.0f;
    i_yaw = 0.0f;
    g_i_enabled = 0;
}

g_target_cmd_roll_deg = 0.0f;
g_target_cmd_pitch_deg = 0.0f;
g_target_cmd_yaw_rate_dps = 0.0f;

switch ((uint8_t)np.action)
{
    case NAV_STOP: break;
    case NAV_FORWARD: g_target_cmd_pitch_deg = -NAV_TILT_DEG; break;
    case NAV_BACK:   g_target_cmd_pitch_deg = +NAV_TILT_DEG; break;
    case NAV_LEFT:   g_target_cmd_roll_deg = -NAV_TILT_DEG; break;
    case NAV_RIGHT:  g_target_cmd_roll_deg = +NAV_TILT_DEG; break;

    // FIX: yaw left/right were swapped in behavior -> swap signs here
    case NAV_YAW_LEFT: g_target_cmd_yaw_rate_dps = +NAV_YAW_RATE_DPS; break;
    case NAV_YAW_RIGHT: g_target_cmd_yaw_rate_dps = -NAV_YAW_RATE_DPS; break;

    default: break;
}

tx_len = Protocol_BuildFrame(DIR_STM_TO_PC, CMD_NAV_ACK,
                             (const uint8_t*)&np, NAV_PAYLOAD_SIZE,
                             tx, sizeof(tx));
}

else
{
    tx_len = Protocol_BuildStatusFrame(ST_ERR_UNKNOWN_CMD, tx, sizeof(tx));
}

if (tx_len > 0)
    HAL_UART_Transmit(&huart6, tx, tx_len, HAL_MAX_DELAY);
}
else if (r == PROTO_ERROR)
{
    uint8_t tx[16];
}

```

```

        uint8_t tx_len = Protocol_BuildStatusFrame(Protocol_GetLastError(), tx, sizeof(tx));
        if (tx_len > 0)
            HAL_UART_Transmit(&huart6, tx, tx_len, HAL_MAX_DELAY);
    }
}

HAL_Delay(1);
}
}

void SystemClock_Config(void)
{
RCC_OscInitTypeDef RCC_OscInitStruct = {0};
RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

__HAL_RCC_PWR_CLK_ENABLE();
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) { Error_Handler(); }

RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
    |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK) { Error_Handler(); }

/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if (huart == &huart6)
    {
        UartRing_Push(uart_rx_it_byte);
        HAL_UART_Receive_IT(&huart6, &uart_rx_it_byte, 1);
    }
}

void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
    if (huart == &huart6)
    {
        __HAL_UART_CLEAR_OREFLAG(&huart6);
        __HAL_UART_CLEAR_NEFLAG(&huart6);
        __HAL_UART_CLEAR_FEFLAG(&huart6);
        __HAL_UART_CLEAR_PEFLAG(&huart6);
    }
}

```

```

    HAL_UART_AbortReceive(&huart6);
    HAL_UART_Receive_IT(&huart6, &uart_rx_it_byte, 1);
}
}
/* USER CODE END 4 */

```

```

void Error_Handler(void)
{
    __disable_irq();
    while (1) {}
}

#ifndef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
    (void)file;
    (void)line;
}
#endif

```

A to apka desktopowa:

```

import socket
import struct
import threading
import time
import queue
import tkinter as tk
from tkinter import ttk
from tkinter.scrolledtext import ScrolledText
from tkinter import filedialog, messagebox
from collections import deque
import csv

from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

HOST = "192.168.8.100"
PORT = 3333

```

```

FRAME_START = 0xAA
FRAME_END   = 0x55

```

```

DIR_PC_TO_STM = 0x01
DIR_STM_TO_PC = 0x00

```

```

CMD_PING    = 0x01
CMD_PONG    = 0x81

```

```

CMD_PWM_READ = 0x10
CMD_PWM_DATA = 0x90

```

```

CMD_PWM_SET = 0x11
CMD_PWM_ACK = 0x91

CMD_IMU_READ = 0x20
CMD_IMU_DATA = 0xA0

CMD_TELEM_READ = 0x21
CMD_TELEM_DATA = 0xA1

CMD_STATUS = 0xE0

CMD_NAV_SET = 0x14
CMD_NAV_ACK = 0x94

CMD_STAB_OFF = 0x13
CMD_STAB_OFF_ACK = 0x93

NAV_STOP = 0
NAV_FORWARD = 1
NAV_RIGHT = 2
NAV_BACK = 3
NAV_LEFT = 4
NAV_YAW_RIGHT = 5
NAV_YAW_LEFT = 6

NAV_STR = {
    NAV_STOP: "STOP",
    NAV_FORWARD: "FORWARD",
    NAV_RIGHT: "RIGHT",
    NAV_BACK: "BACK",
    NAV_LEFT: "LEFT",
    NAV_YAW_RIGHT: "YAW_RIGHT",
    NAV_YAW_LEFT: "YAW_LEFT"
}

STATUS_STR = {
    0x01: "BAD_START",
    0x02: "BAD_END",
    0x03: "BAD_CRC",
    0x04: "BAD_DIR",
    0x05: "BAD_LEN",
    0x06: "UNKNOWN_CMD",
    0x07: "IMU_NOT_READY"
}

def calc_crc(*args):
    crc = 0
    for a in args:
        crc ^= a
    return crc & 0xFF

```

```

def build_frame(cmd, payload=b ""):
    length = len(payload)
    crc = calc_crc(DIR_PC_TO_STM, cmd, length, *payload)
    return bytes([FRAME_START, DIR_PC_TO_STM, cmd, length]) + payload + bytes([crc, FRAME_END])

def parse_stream(buf):
    frames = []
    while len(buf) >= 6:
        if buf[0] != FRAME_START:
            buf = buf[1:]
            continue

        ln = buf[3]
        frame_len = 6 + ln
        if frame_len > 128:
            buf = buf[1:]
            continue

        if len(buf) < frame_len:
            break

        frame = buf[:frame_len]
        buf = buf[frame_len:]

        _, dir_, cmd, ln = frame[:4]
        payload = frame[4:4+ln]
        crc = frame[4+ln]
        end = frame[5+ln]

        if end != FRAME_END:
            continue

        if calc_crc(dir_, cmd, ln, *payload) != crc:
            continue

        frames.append((cmd, payload))
    return frames, buf

class App:
    def __init__(self, root):
        self.root = root
        self.root.title("Drone Control (TCP server)")
        self.log_q = queue.Queue()

        self.srv = None
        self.conn = None
        self.running = False
        self.rx_buf = b ""

        self.rx_thread = None

```

```

self.tx_thread = None

self.tx_q = queue.Queue(maxsize=2000)
self.tx_stop = threading.Event()

self.pwm_var = tk.IntVar(value=0)

self.desired_action = NAV_STOP
self.current_base_sent = 0

self.takeoff_on = False
self.takeoff_stop = threading.Event()
self.takeoff_thread = None

self.pwm_poll_on = True
self.pwm_poll_interval_s = 0.05 # 20 Hz
self.pwm_poll_thread = None
self.pwm_poll_stop = threading.Event()

self.t0 = time.monotonic()
self.hist_len = 300
self.t_hist = deque(maxlen=self.hist_len)
self.pwm_hist = {
    "LF": deque(maxlen=self.hist_len),
    "RF": deque(maxlen=self.hist_len),
    "LB": deque(maxlen=self.hist_len),
    "RB": deque(maxlen=self.hist_len),
}
self.last_pwm = {"LF": 0, "RF": 0, "LB": 0, "RB": 0}

self.roll_var = tk.StringVar(value="ROLL: --.-°")
self.pitch_var = tk.StringVar(value="PITCH: --.-°")
self.yaw_var = tk.StringVar(value="YAW: --.-°")

# ---- DATA LOGGING ----
self.data_lock = threading.Lock()
self.data_rows = []
self.last_imu = {"roll": None, "pitch": None, "yaw": None}

self._build_ui()
self._ui_poll()
self._plot_update()

def _build_ui(self):
    top = ttk.Frame(self.root, padding=8)
    top.pack(fill="both", expand=True)

    left = ttk.Frame(top)
    left.pack(side="left", fill="y")

    right = ttk.Frame(top)

```

```

right.pack(side="right", fill="both", expand=True)

conn_box = ttk.LabelFrame(left, text="Connection", padding=8)
conn_box.pack(fill="x")

self.btn_start = ttk.Button(conn_box, text="Start server", command=self.start_server)
self.btn_start.pack(fill="x")

self.btn_ping = ttk.Button(conn_box, text="Ping", command=self.send_ping, state="disabled")
self.btn_ping.pack(fill="x", pady=(6,0))

self.lbl_state = ttk.Label(conn_box, text="State: DISCONNECTED")
self.lbl_state.pack(fill="x", pady=(6,0))

slider_box = ttk.LabelFrame(left, text="Base PWM", padding=8)
slider_box.pack(fill="y", pady=(10,0), expand=True)

self.lbl_pwm = ttk.Label(slider_box, text="0")
self.lbl_pwm.pack()

self.slider = ttk.Scale(
    slider_box,
    from_=10000, to=0,
    orient="vertical",
    command=self._on_slider
)
self.slider.set(0)
self.slider.pack(fill="y", expand=True, pady=6)
self.slider.bind("<ButtonRelease-1>", self._on_slider_release)

self.btn_set_manual = ttk.Button(slider_box, text="Manual (PWM)",
command=self.send_manual_pwm, state="disabled")
self.btn_set_manual.pack(fill="x", pady=(6,0))

self.btn_takeoff = ttk.Button(slider_box, text="TAKEOFF (smooth)",
command=self.toggle_takeoff, state="disabled")
self.btn_takeoff.pack(fill="x", pady=(6,0))

self.btn_export = ttk.Button(slider_box, text="Export CSV", command=self.export_csv,
state="disabled")
self.btn_export.pack(fill="x", pady=(12,0))

top_right = ttk.Frame(right)
top_right.pack(fill="both", expand=True)

controls = ttk.Frame(top_right)
controls.pack(side="left", anchor="n")

self.btn_yawl = ttk.Button(controls, text="yaw", width=6, command=lambda:
self.send_nav(NAV_YAW_LEFT), state="disabled")
self.btn_fwd = ttk.Button(controls, text="↑", width=6, command=lambda:

```

```

self.send_nav(NAV_FORWARD), state="disabled")
    self.btn_yawr = ttk.Button(controls, text="↻", width=6, command=lambda:
self.send_nav(NAV_YAW_RIGHT), state="disabled")

    self.btn_left = ttk.Button(controls, text="←", width=6, command=lambda:
self.send_nav(NAV_LEFT), state="disabled")
    self.btn_stop = ttk.Button(controls, text="STOP", width=6, command=lambda:
self.send_nav(NAV_STOP), state="disabled")
    self.btn_right= ttk.Button(controls, text="→", width=6, command=lambda:
self.send_nav(NAV_RIGHT), state="disabled")

    self.btn_back = ttk.Button(controls, text="↓", width=6, command=lambda:
self.send_nav(NAV_BACK), state="disabled")

self.btn_yawl.grid(row=0, column=0, padx=4, pady=4)
self.btn_fwd.grid(row=0, column=1, padx=4, pady=4)
self.btn_yawr.grid(row=0, column=2, padx=4, pady=4)

self.btn_left.grid(row=1, column=0, padx=4, pady=4)
self.btn_stop.grid(row=1, column=1, padx=4, pady=4)
self.btn_right.grid(row=1, column=2, padx=4, pady=4)

self.btn_back.grid(row=2, column=1, padx=4, pady=4)

imu_box = ttk.Frame(controls)
imu_box.grid(row=3, column=0, columnspan=3, sticky="w", padx=4, pady=(8, 0))

ttk.Label(imu_box, textvariable=self.roll_var).pack(anchor="w")
ttk.Label(imu_box, textvariable=self.pitch_var).pack(anchor="w")
ttk.Label(imu_box, textvariable=self.yaw_var).pack(anchor="w")

plot_box = ttk.LabelFrame(top_right, text="PWM (live)", padding=6)
plot_box.pack(side="left", fill="both", expand=True, padx=(10,0))

fig = Figure(figsize=(6.0, 2.8), dpi=100)
self.ax = fig.add_subplot(111)
self.ax.set_xlabel("time (s)")
self.ax.set_ylabel("PWM")

self.line_lf, = self.ax.plot([], [], label="LF")
self.line_rf, = self.ax.plot([], [], label="RF")
self.line_lb, = self.ax.plot([], [], label="LB")
self.line_rb, = self.ax.plot([], [], label="RB")

self.ax.legend(loc="upper right")

self.canvas = FigureCanvasTkAgg(fig, master=plot_box)
self.canvas.get_tk_widget().pack(fill="both", expand=True)

log_box = ttk.LabelFrame(right, text="Logs", padding=8)
log_box.pack(fill="both", expand=True, pady=(10,0))

```

```

self.log = ScrolledText(log_box, height=12, wrap="word", state="disabled")
self.log.pack(fill="both", expand=True)

def _on_slider(self, _=None):
    val = int(float(self.slider.get()) / 100) * 100
    self.pwm_var.set(val)
    self.lbl_pwm.config(text=str(val))

def _on_slider_release(self, _event=None):
    if not self.conn:
        return
    if self.takeoff_on:
        return

    base = int(self.pwm_var.get())
    payload = struct.pack("<HB", base, int(self.desired_action))
    self.send_frame(CMD_NAV_SET, payload)

    self.log_put(f"TX: NAV (slider release) act={self.desired_action} base={base}")

def log_put(self, msg):
    self.log_q.put(msg)

def _ui_poll(self):
    try:
        while True:
            msg = self.log_q.get_nowait()
            self.log.configure(state="normal")
            self.log.insert("end", msg + "\n")
            self.log.see("end")
            self.log.configure(state="disabled")
    except queue.Empty:
        pass
    self.root.after(50, self._ui_poll)

def _plot_update(self):
    try:
        if len(self.t_hist) >= 2:
            t = list(self.t_hist)
            t0 = t[-1]
            x = [ti - t0 for ti in t]

            y_lf = list(self.pwm_hist["LF"])
            y_rf = list(self.pwm_hist["RF"])
            y_lb = list(self.pwm_hist["LB"])
            y_rb = list(self.pwm_hist["RB"])

            self.line_lf.set_data(x, y_lf)
            self.line_rf.set_data(x, y_rf)
            self.line_lb.set_data(x, y_lb)
    
```

```

        self.line_rb.set_data(x, y_rb)

        self.ax.relim()
        self.ax.autoscale_view()

        self.canvas.draw_idle()
    except Exception:
        pass

    self.root.after(100, self._plot_update)

def set_connected(self, connected: bool):
    self.lbl_state.config(text="State: CONNECTED" if connected else "State: DISCONNECTED")
    state = "normal" if connected else "disabled"
    for b in [self.btn_ping, self.btn_set_manual, self.btn_takeoff,
              self.btn_yawl, self.btn_fwd, self.btn_yawr,
              self.btn_left, self.btn_stop, self.btn_right, self.btn_back,
              self.btn_export]:
        b.config(state=state)

def start_server(self):
    if self.running:
        return

    self.running = True
    self.btn_start.config(state="disabled")
    self.log_put(f"Starting TCP server on {HOST}:{PORT} ...")

def worker():
    try:
        self.srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.srv.bind((HOST, PORT))
        self.srv.listen(1)
        self.log_put("Waiting for ESP connection...")

        self.conn, addr = self.srv.accept()
        self.conn.settimeout(0.05)
        self.log_put(f"Connected from {addr}")

        self.tx_stop.clear()
        self.tx_thread = threading.Thread(target=self.tx_loop, daemon=True)
        self.tx_thread.start()

        self.pwm_poll_stop.clear()
        self.pwm_poll_thread = threading.Thread(target=self.pwm_poll_loop, daemon=True)
        self.pwm_poll_thread.start()

        self.root.after(0, lambda: self.set_connected(True))

        self.rx_loop()
    
```

```

except Exception as e:
    self.log_put(f"Server error: {e}")
finally:
    self.running = False

    self.takeoff_stop.set()
    self.takeoff_on = False

    self.tx_stop.set()
    self.pwm_poll_stop.set()

try:
    if self.conn:
        self.conn.close()
except:
    pass
try:
    if self.srv:
        self.srv.close()
except:
    pass

self.conn = None
self.srv = None

self.root.after(0, lambda: self.set_connected(False))
self.root.after(0, lambda: self.btn_start.config(state="normal"))
self.log_put("Server stopped.")

self.rx_thread = threading.Thread(target=worker, daemon=True)
self.rx_thread.start()

def send_frame(self, cmd, payload=b ""):
    frame = build_frame(cmd, payload)
    if not self.conn:
        return
    try:
        self.tx_q.put_nowait(frame)
    except queue.Full:
        self.log_put("TX queue full: dropping frame")

def tx_loop(self):
    while not self.tx_stop.is_set():
        try:
            data = self.tx_q.get(timeout=0.1)
        except queue.Empty:
            continue

        if not self.conn:
            continue

```

```

try:
    self.conn.sendall(data)
except Exception as e:
    self.log_put(f"Send failed: {e}")
    break

def pwm_poll_loop(self):
    while not self.pwm_poll_stop.is_set():
        if self.conn and self.pwm_poll_on:
            self.send_frame(CMD_TELEM_READ)
        time.sleep(self.pwm_poll_interval_s)

def send_ping(self):
    self.send_frame(CMD_PING)
    self.log_put("TX: PING")

def send_manual_pwm(self):
    if self.takeoff_on:
        return
    val = int(self.pwm_var.get())
    payload = struct.pack("<HHHH", val, val, val, val)
    self.send_frame(CMD_PWM_SET, payload)
    self.log_put(f"TX: MANUAL PWM={val}")

def send_nav(self, action):
    if not self.conn:
        return

    self.desired_action = int(action)

    base = self.current_base_sent if self.takeoff_on else int(self.pwm_var.get())
    payload = struct.pack("<HB", base, int(self.desired_action))
    self.send_frame(CMD_NAV_SET, payload)

    self.log_put(f"TX: NAV act={self.desired_action} base={base}")

def toggle_takeoff(self):
    if not self.conn:
        return

    if not self.takeoff_on:
        self.takeoff_on = True
        self.takeoff_stop.clear()
        self.desired_action = NAV_STOP
        self.btn_takeoff.config(text="DISARM (STAB_OFF)")
        self.log_put("TAKEOFF: start smooth ramp (action stays changeable)")

    def ramp_worker():
        base = self.current_base_sent
        step = 80

```

```

interval = 0.03

while not self.takeoff_stop.is_set() and self.conn:
    target = int(self.pwm_var.get())

    if base < target:
        base = min(base + step, target)
    elif base > target:
        base = max(base - step, target)

    self.current_base_sent = base
    payload = struct.pack("<HB", base, int(self.desired_action))
    self.send_frame(CMD_NAV_SET, payload)

    time.sleep(interval)

    self.log_put("TAKEOFF: ramp stopped")

self.takeoff_thread = threading.Thread(target=ramp_worker, daemon=True)
self.takeoff_thread.start()

else:
    self.takeoff_on = False
    self.takeoff_stop.set()

    self.send_frame(CMD_STAB_OFF)
    self.log_put("TX: STAB_OFF (PWM_SetSafe)")

    self.btn_takeoff.config(text="TAKEOFF (smooth)")

def _append_telem_row(self, t_s, roll, pitch, yaw, lf, rf, lb, rb):
    with self.data_lock:
        self.data_rows.append({
            "t_s": t_s,
            "base": int(self.current_base_sent),
            "action": int(self.desired_action),
            "action_name": NAV_STR.get(int(self.desired_action), str(int(self.desired_action))),
            "takeoff_on": int(self.takeoff_on),

            "roll_deg": float(roll),
            "pitch_deg": float(pitch),
            "yaw_deg": float(yaw),

            "pwm_lf": int(lf),
            "pwm_rf": int(rf),
            "pwm_lb": int(lb),
            "pwm_rb": int(rb),
        })

def export_csv(self):
    with self.data_lock:

```

```

rows = list(self.data_rows)

if not rows:
    messagebox.showinfo("Export CSV", "Brak danych do eksportu (jeszcze nic nie przyszło z telemetrii).")
    return

default_name = time.strftime("drone_log_%Y-%m-%d_%H-%M-%S.csv")
path = filedialog.asksaveasfilename(
    defaultextension=".csv",
    initialfile=default_name,
    filetypes=[("CSV files", "*.csv"), ("All files", "*.*")]
)
if not path:
    return

fieldnames = [
    "t_s",
    "base",
    "action",
    "action_name",
    "takeoff_on",
    "roll_deg",
    "pitch_deg",
    "yaw_deg",
    "pwm_lf",
    "pwm_rf",
    "pwm_lb",
    "pwm_rb",
]
try:
    with open(path, "w", newline="", encoding="utf-8") as f:
        w = csv.DictWriter(f, fieldnames=fieldnames)
        w.writeheader()
        for r in rows:
            w.writerow(r)

    self.log_put(f"EXPORT: zapisano {len(rows)} wierszy do: {path}")
except Exception as e:
    messagebox.showerror("Export CSV", f"Nie udało się zapisać CSV:\n{e}")

def rx_loop(self):
    while self.running and self.conn:
        try:
            data = self.conn.recv(256)
            if not data:
                self.log_put("Connection closed by peer.")
                break
            self.rx_buf += data
        except socket.timeout:

```

```

    pass
except Exception as e:
    self.log_put(f"RX error: {e}")
    break

frames, self.rx_buf = parse_stream(self.rx_buf)
for cmd, payload in frames:
    if cmd == CMD_PONG:
        self.log_put("RX: PONG ✓")

    elif cmd == CMD_PWM_ACK:
        self.log_put("RX: PWM ACK ✓")

    elif cmd == CMD_NAV_ACK:
        if len(payload) == 3:
            base, act = struct.unpack("<HB", payload)
            self.current_base_sent = base
            self.log_put(f"RX: NAV ACK ✓ base={base} act={act}")
        else:
            self.log_put("RX: NAV ACK ✓")

    elif cmd == CMD_PWM_DATA:
        if len(payload) == 8:
            lb, lf, rf, rb = struct.unpack("<HHHH", payload)
            self.last_pwm = {"LF": lf, "RF": rf, "LB": lb, "RB": rb}

            t = time.monotonic() - self.t0
            self.t_hist.append(t)
            self.pwm_hist["LF"].append(lf)
            self.pwm_hist["RF"].append(rf)
            self.pwm_hist["LB"].append(lb)
            self.pwm_hist["RB"].append(rb)
        else:
            self.log_put(f"RX: PWM DATA bad len={len(payload)}")

    elif cmd == CMD_STAB_OFF_ACK:
        self.log_put("RX: STAB_OFF ACK ✓")

    elif cmd == CMD_STATUS and len(payload) == 1:
        err = payload[0]
        self.log_put("RX STATUS: " + STATUS_STR.get(err, f"0x{err:02X}"))

    elif cmd == CMD_IMU_DATA and len(payload) == 6:
        roll, pitch, yaw = struct.unpack("<hhh", payload)
        r = roll / 100.0
        p = pitch / 100.0
        y = yaw / 100.0
        self.last_imu = {"roll": r, "pitch": p, "yaw": y}
        self.log_put(f"RX IMU: R={r:.1f} P={p:.1f} Y={y:.1f}")

    elif cmd == CMD_TELEM_DATA:

```

```

if len(payload) == 14:
    roll, pitch, yaw, lb, lf, rf, rb = struct.unpack("<hhHHHH", payload)

    r = roll / 100.0
    p = pitch / 100.0
    y = yaw / 100.0
    self.last_imu = {"roll": r, "pitch": p, "yaw": y}

    self.root.after(0, lambda rr=r, pp=p, yy=y: (
        self.roll_var.set(f"ROLL: {rr:+.1f}°"),
        self.pitch_var.set(f"PITCH: {pp:+.1f}°"),
        self.yaw_var.set(f"YAW: {yy:+.1f}°")
    ))

    self.last_pwm = {"LF": lf, "RF": rf, "LB": lb, "RB": rb}

    t = time.monotonic() - self.t0
    self.t_hist.append(t)
    self.pwm_hist["LF"].append(lf)
    self.pwm_hist["RF"].append(rf)
    self.pwm_hist["LB"].append(lb)
    self.pwm_hist["RB"].append(rb)

    self._append_telem_row(t, r, p, y, lf, rf, lb, rb)
else:
    self.log_put(f"RX: TELEMETRY bad len={len(payload)}")

time.sleep(0.005)

def main():
    root = tk.Tk()
    app = App(root)
    root.mainloop()

if __name__ == "__main__":
    main()

```

Załączam Ci również plik z logami. Obecnie, dron nie chce stabilnie się wznieść, cały czas się przechyla praktycznie od momentu wzniesienia się nawet delikatnie nad ziemię, nie chce trzymać balansu. Co mam zrobić? Jeżeli wszystkie ustawienia silników dam na domyślne (gain'y, cg), to dron przechyla mi się od razu na prawo i praktycznie się przewraca na plecy

Jasne — nie ruszam ani mapowania **LF/RF/LB/RB**, ani Twoich **SIGN** dla pitch/roll. Z logów da się wyciągnąć bardzo konkretną rzecz: kontroler **musi na stałe “dopompuwać” prawą stronę**, żeby w ogóle próbować utrzymać poziom.

Co widać w Twoim logu (twarde liczby)

W fragmencie najbardziej "hoverowym" (STOP, base ~4500–6500, silniki pracują) średnio wychodzi:

- **LF ≈ base – 606**
- **LB ≈ base – 879**
- **RF ≈ base + 975**
- **RB ≈ base + 631**

Czyli prawa strona dostaje sumarycznie ok. **+1500 PWM więcej na silnik** względem lewej (to jest ogromny stały trim). To idealnie pasuje do Twojego opisu "na domyślnych ustawieniach przechyla się od razu na prawo" — **prawa strona jest realnie słabsza / cięższa**, więc kontroler podbija PWM na RF/RB.

Dodatkowo: w logu masz moment, gdzie roll rośnie bardzo szybko i wpada w tilt-kill:

- ok. **t ≈ 39.25 → 39.44 s** roll idzie z $\sim 12^\circ$ do $> 40^\circ$ i wtedy PWM leci do zera (kill). To wygląda jak sytuacja, gdzie zabrakło "zapasów" kontroli (albo yaw/obciążenie/CG robią Ci potężne sprzężenie).

Najważniejsze: zrób z tego "stałą kompensację", a nie ciągłą walkę PID-em

Masz do tego idealny mechanizm w kodzie: **MOTOR_OFFSET_*** oraz **CG_*_BIAS_PWM**. U Ciebie te offsety są 0, więc cały stały błąd musi odrabiać regulator (a to robi się niestabilne przy startie i przy zakłóceniach).

Krok 1 — ustaw "trim per silnik" na bazie logu

Na start (pierwsza iteracja) wpisz:

```
c

#define MOTOR_OFFSET_LF    (-600.0f)
#define MOTOR_OFFSET_LB    (-880.0f)
#define MOTOR_OFFSET_RF    (+975.0f)
#define MOTOR_OFFSET_RB    (+630.0f)
```

To jest dokładnie to, co log pokazuje jako średnie odchyłki od base w locie.

Uwaga praktyczna: takie offsety warto *wygaszać przy niskim base*, bo przy startie mogą Ci "przydusić" lewą stronę zanim dron się odklei.

Najprostszy, bardzo skuteczny trik: przemnóż offsety przez **cgk** (masz już gotową skalę **CgBiasScale()**):

W miejscu gdzie liczysz wyjścia PWM zmień:

```
c

float out_lf_f = (float)base + mix_lf * MOTOR_GAIN_LF + MOTOR_OFFSET_LF;
...
```

na:

```
c

float trimk = CgBiasScale(g_stab_base_pwm); // 0..1
float out_lf_f = (float)base + mix_lf * MOTOR_GAIN_LF + MOTOR_OFFSET_LF * trimk;
```

```
float out_rf_f = (float)base + mix_rf * MOTOR_GAIN_RF + MOTOR_OFFSET_RF * trimk;
float out_lb_f = (float)base + mix_lb * MOTOR_GAIN_LB + MOTOR_OFFSET_LB * trimk;
float out_rb_f = (float)base + mix_rb * MOTOR_GAIN_RB + MOTOR_OFFSET_RB * trimk;
```

Efekt: przy małym base trim prawie nie działa, a przy hoverze działa w pełni.

Krok 2 — dopiero potem kręć CG_ROLL_BIAS_PWM

Twoje **CG_ROLL_BIAS_PWM** = -700 daje Ci za małą kompensację jak na to, co widać w logu. Po wprowadzeniu **MOTOR_OFFSET_*** zwykle się okazuje, że **CG_ROLL_BIAS_PWM** można mocno zmniejszyć albo nawet wyzerować, bo rolę "stałego balansu" przejmują offsety.

Ja bym zrobił tak:

1. Najpierw wprowadź **MOTOR_OFFSET_*** + mnożenie przez **trimk**

2. Ustaw tymczasowo:
 - **CG_ROLL_BIAS_PWM** = 0
 - **CG_PITCH_BIAS_PWM** = 0

3. Zrób krótki test i zobacz czy nadal "ciągnie" w prawo i jak bardzo.

Potem dopiero wracaj do CG-biasów, jeśli nadal trzeba.

Drugi killer stabilności: stały błąd jest większy niż "wygoda" P/I

U Ciebie I-term jest dość delikatny (bo **KI=0.25** i **I_MAX=1600**), a do tego integracja jest blokowana, gdy mixer uzna saturację (**k < 0.999**). Przy tak dużym stałym trymie łatwo wpadasz w sytuację, gdzie:

- regulator cały czas pracuje "pod górkę"
- pojawia się większe zakłócenie (oderwanie od ziemi / podmuch / yaw)
- roll zaczyna uciekać szybciej niż korekcja nadąża
- dobija do tilt-kill

Właśnie dlatego przeniesienie stałej kompensacji do **MOTOR_OFFSET / CG-bias** robi ogromną różnicę.

Trzeci temat: yaw potrafi Ci wzbudzać roll (widać to w logu)

Ponieważ Ty w logu często masz **action = YAW_RIGHT/YAW_LEFT**, a roll potrafi wtedy szybko uciec, to do testów stabilizacji:

- testuj najpierw **tylko STOP** (żadnego yaw),
- dopiero jak hover jest stabilny, dokładaj yaw i ewentualnie ogranicz:

W praktyce na początek:

- zmniejsz **NAV_YAW_RATE_DPS** (np. z 180 na 90)
- i/lub zmniejsz **STAB_YAW_MAX** (np. z 600 na 300)

To nie jest "leczenie objawów" — po prostu yaw jest kolejnym obciążeniem dla zapasu sterowania, a Ty teraz walczysz o samą bazową stabilność.

Co z PCB/CG i "mocniejszymi silnikami"

Jeśli masa jest przesunięta, to software to skompensuje, ale:

- będzie większy pobór,
- mniejszy zapas kontroli,
- większa szansa na wywrotkę przy startie.

Jeśli możesz choć trochę: **przesuń baterię** tak, żeby zmniejszyć potrzebny stały trim (nawet 2-3 mm robi różnicę w micro). To po prostu "darmowa stabilność".

Szybka procedura strojenia (żeby nie błędzić)

1. Dron zabezpieczony (tether / koszyk / przynajmniej miękka przestrzeń).
2. **CG_*_BIAS_PWM = 0**, wprowadź **MOTOR_OFFSET_*** i skalowanie przez **trimk**.
3. Test hover na **STOP**.
4. Zrób nowy log i sprawdź średnie **pwm_i - base**.
Jeśli po zmianach średnie delta są blisko zera (np. ±200..300), jesteś w domu.
5. Dopiero potem dodaj yaw i w razie czego zmniejsz **NAV_YAW_RATE_DPS / STAB_YAW_MAX**.

Jeśli chcesz, mogę policzyć Ci "gotowe" nowe offsety po kolejnym teście

Wystarczy, że wrzucisz następny CSV po wprowadzeniu **MOTOR_OFFSET_*** (i najlepiej z testem głównie STOP). Ja Ci z niego policzę:

- nowe **MOTOR_OFFSET_***,
- czy pitch też wymaga stałego trymu,
- i czy moment ucieczki roll wygląda jak brak zapasu czy raczej coś "mechanicznego" (np. jeden silnik siada pod obciążeniem).

Na teraz: **najbardziej opłacalna zmiana to przeniesienie stałego trymu do MOTOR_OFFSET i skalowanie go z base**. To dokładnie odpowiada temu, co Twój log pokazuje.



drone log 2026 02 01 04 43 58

1	44.07800000000134	0	0	S
2	44.07800000000134	0	0	S
3	44.07800000000134	0	0	S
4	44.10899999999674	0	0	S
5	44.17100000000209 5	0	0	S
6	44.20300000000134	0	0	S
	t_s	base	action	#

Okej, masz nowe logi. Wprowadziłem do kodu wszystkie Twoje poprawki (motor_offs_* i trimk). Dalej go zrzuciło do prawej przy startie

Dzięki — przeanalizowałem nowy CSV i mam dla Ciebie **konkretnie, "logami poparte"** wnioski. Nie ruszam mapowania **LF/RF/LB/RB** ani Twoich **SIGN**.

1) Największy błąd na startie: startujesz zanim skończy się level-calib (a NAV_SET i tak "armuje")

W Twoim kodzie level-calib potrzebuje:

- LEVEL_CALIB_SAMPLES = 200
- IMU update co 10 ms

Czyli ~2 sekundy spokojnego stania *bez armingu*.

W nowym logu:

- telemetria startuje ok. **t=44.08 s**
- takeoff zaczyna się ok. **t=44.64 s**
- pierwsze PWM > 0 masz ok. **t=45.39 s**

Czyli zaczynasz "odkręcać" silniki **po ~1.3 s**, a to zwykle **za wcześnie** na Twoją kalibrację (filter + gyro bias + level offsets jeszcze się stabilizują).

A kluczowe: w **CMD_NAV_SET** masz:

```
c
g_ctrl_mode = CTRL_STAB;
g_armed = 1;
```

Czyli **omijasz** cały mechanizm "nie uzbrajaj dopóki level-calib nie zrobione", który masz w **CMD_STAB_SET**.

Poprawka (najbardziej wartościowa na start)

W `CMD_NAV_SET` zrób to samo co w `CMD_STAB_SET`:

```
c

g_ctrl_mode = CTRL_STAB;

if (g_level_calib_done) g_armed = 1;
else {
    g_armed = 0;
    PWM_SetSafe();
    // opcjonalnie: odeślij STATUS (IMU_NOT_READY albo własny kod)
}
```

To naprawdę potrafi zmienić wszystko, bo inaczej kontroler walczy z "pseudo-tiltem" z nieskalibrowanego poziomu i od razu zaczyna budować moment w złą stronę.

2) W logu wciąż widać ogromny stały "prawy trim" przy hoverze (czyli prawa strona nadal jest słabsza / bardziej obciążona)

W przedziale, gdzie silniki pracują stabilnie (base ~4800–6500) średnie odchyłki `PWM` - `base` są nadal bardzo podobne do poprzednich:

- **LF ~ -668**
- **LB ~ -1008**
- **RF ~ +1054**
- **RB ~ +636**

To znaczy: nawet po Twoich zmianach kontroler **ciągle musi pompować prawą stronę**.

To nie jest "jakiś szum" — to stały bias.

3) Dlaczego mimo trimk dalej "zrzaca przy startie": trim i I włączają się za późno względem momentu oderwania

Masz dwa mechanizmy, które *pomagają na stałe biasy*, ale oba działają dopiero "od pewnego base":

- `trimk = CgBiasScale()` startuje dopiero od `CG_BIAS_START_PWM = 2800`
- integrator dopiero od `I_ENABLE_BASE_PWM = 3200`

Jeżeli realne **oderwanie** wypada np. ~2200–3200 (w mikrusach to częste), to w tej krytycznej fazie:

- trim jeszcze jest słaby albo 0
- I jeszcze OFF
- zostaje tylko PD → a PD nie utrzyma stałego momentu bez błędu (musi "pozwolić się przechylić", żeby wygenerować korekcję)

Szybka, bezpieczna poprawka

Przesuń progi w dół (do testu):

```
c

#define CG_BIAS_START_PWM 1200u
#define CG_BIAS_FULL_PWM 2800u

#define I_ENABLE_BASE_PWM 2000u
#define I_DISABLE_BASE_PWM 1800u
```

To powoduje, że:

- kompensacja asymetrii zaczyna działać **jeszcze zanim dron się odklei**
- I-term zaczyna łapać stałe biasy wcześniej

4) Masz wyraźny “pitch runaway” przed kill — to wygląda jak zjadanie zapasu przez stały pitch-bias

W nowym logu kill następuje, bo **pitch rośnie do ~50°** (roll jest wtedy blisko zera). Tuż przed kill przy base ~7k+ masz stale:

- suma **front** (LF+RF) jest ok. **+600...700** większa niż suma **rear** (LB+RB)

A Ty dodatkowo masz:

```
c

#define CG_PITCH_BIAS_PWM 200
```

który **zawsze** pcha front w górę i rear w dół (po cgk). To potrafi:

- zabrać headroom na korekcję pitch
- wzbudzać pitching przy większym ciągu / ground effect / flex ramy

Na test:

Na czas diagnostyki ustaw:

```
c

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0 // też warto na 1 próbę wyzerować po zmianach trimów
```

i dopiero potem wracaj do CG-biasów od zera, bazując na nowych logach.

5) Jeden ważny miksterski detal: **MOTOR_GAIN** obecnie skaluje tylko “mix”, a nie “base” (to często za mało gdy silnik jest realnie

słabszy)

Masz teraz:

```
c

out_rf = base + mix_rf * MOTOR_GAIN_RF + offs_rf*trimk;
```

Jeżeli RF/RB są słabsze "w ogóle" (thrust/PWM), to one potrzebują wzmacnienia **też na base**, nie tylko na korekcjach.

W praktyce lepiej działa (dla nierównych silników):

```
c

out_rf = (base + mix_rf) * MOTOR_GAIN_RF + offs_rf*trimk;
```

Analogicznie dla wszystkich 4.

To daje Ci:

- większy "prawy ciąg" już od pierwszych PWM (gdy trimk=0 i I=OFF)
- mniejsze ryzyko, że PD nie zdąży złapać przy oderwaniu

Proponowana kolejność (żebyś nie strzelał w ciemno)

Test 1 — tylko "bezpieczny start"

1. Zablokuj arming w **NAV_SET**, jeśli **!g_level_calib_done** (punkt 1).
2. Ustaw **CG_PITCH_BIAS_PWM = 0**, **CG_ROLL_BIAS_PWM = 0** (punkt 4).
3. Zrób próbę startu i nowy log.

Jeśli po tym dalej "od razu leci w prawo", przechodzisz do Testu 2.

Test 2 — kompensacja wcześniejszej

4. Zjedź programami **CG_BIAS_*** i **I_ENABLE_*** w dół (punkt 3).
5. Zrób próbę i log.

Test 3 — wzmacnij base na słabzych motorach

6. Zmień formułę na **(base + mix) * MOTOR_GAIN + offs*trimk** (punkt 5).
7. Log.

Jeżeli wrzucisz kolejny CSV po **Test 1** (najlepiej 15–25 s próby, samo STOP), to ja Ci policzę:

- nowe docelowe **MOTOR_OFFSET_*** po zmianach,
- czy prawa strona nadal wymaga aż takiego biasu,

- i czy pitch runaway zniknął po wyzerowaniu CG_PITCH_BIAS.

Na ten moment: **najbardziej podejrzane** i jednocześnie “najtańsze do naprawy” są:

1. **arming przez NAV_SET zanim skończy się level-calib,**
2. **za późno startujący trim/I względem oderwania,**
3. **CG_PITCH_BIAS zjadający zapas i wzbudzający pitch.**



drone log 2026 02 01 04 59 54

t_s	base	action	a
1	6.56199999998079	0	0
2	6.56199999998079	0	0
3	6.59399999997322	0	0
4	6.6089999999674	0	0
5	6.65599999995402	0	0
6	6.70299999994063	0	0

Okej, wprowadziłem wszystkie Twoje poprawki do kodu. Sprawdź logi i sam zobacz jak się zachował, bo generalnie zjebanie dosyć

Sprawdziłem ten nowy CSV i... tak, w logach widać dokładnie „dlaczego to się sypie” — to nie jest subtelna rzecz, tylko **kilka twardych, powtarzalnych mechanizmów**, które robią z tego katapultę.

Nie ruszam Twojego **LF/RF/LB/RB** ani **SIGN**.

Co dokładnie widać w logu (konkretnie liczby)

1) Masz wiele tilt-kill i po każdym killu dron sam sobie robi ponowny start na wysokim base

W logu masz kilka przejść „PWM pracuje → nagle wszystko 0” (tilt_kill):

- **t ≈ 21.56 s**: roll rośnie do ~44.7° → PWM=0 (kill)
- **t ≈ 32.20 s**: pitch rośnie do ~50.0° → PWM=0 (kill)
- **t ≈ 42.73 s**: roll rośnie do ~39.9° → PWM=0 (kill)

- $t \approx 46.51$ s: pitch rośnie do $\sim 48.3^\circ \rightarrow \text{PWM}=0$ (kill)

I teraz najważniejsze: po killu PWM jest 0, ale **base w logu dalej jest bardzo wysokie** (np. 6800, 7300, 7560, 8800), a po chwili znowu wraca praca silników — czyli robisz „re-arm + spool-up” na wysokim gazie praktycznie z miejsca. To jest przepis na „zjebanie”.

2) Prawa strona dalej jest ogromnie „głodna” PWM

W próbkach, gdzie roll/pitch są małe (czyli niby „wisi / próbuje wisieć”), średnie **PWM** - **base** nadal są gigantyczne:

- LF ~ **-708**
- LB ~ **-965**
- RF ~ **+1694**
- RB ~ **+1098**

To oznacza, że **średnio prawa strona** jest ok. **+2300 PWM** wyżej niż lewa (średnia różnica prawa-lewa ≈ 2290). To jest potężna stała nierównowaga (albo CG, albo słabsze prawe silniki/zasilanie/power-path).

3) Saturacja: RF/RB bardzo często dobijają do 10000

W samym tym logu (gdy silniki pracują):

- RF ma **275** próbek na **10000**
- RB ma **127** próbek na **10000**
- łącznie $\sim 39\%$ czasu lotu masz **co najmniej jeden silnik na 10000**

Czyli kontroler non-stop jedzie na limiterze i łatwo o runaway \rightarrow tilt-kill.

Co zrobić, żeby to przestało się rozwalać (największy efekt najszybciej)

A) Zrób “HARD LATCH” tilt-kill: po killu ma być DISARM + base=0 i koniec, bez auto-unkill na wysokim gazie

Teraz masz auto-unkill po 300 ms, a Twoja apka dalej wysyła NAV_SET z wysokim base \rightarrow to się samo wznowia i znowu wywraca.

Wprowadź zasadę:

- jeśli tilt_kill złapany \rightarrow natychmiast: **g_armed=0, g_target_base_pwm=0, g_stab_base_pwm=0**
- odblokowanie tylko jeśli: baza jest niska + świadome ponowne uzbrojenie (np. kolejne STAB_SET / NAV_SET gdy base=0)

Minimalna zmiana w Twoim IMU_UpdateContinuous() w miejscu gdzie ustawiasz kill:

```
c

if ((now - g_tilt_kill_since) >= TILT_KILL_DEBOUNCE_MS)
{
    g_tilt_kill = 1;
    g_tilt_unkill_since = 0;

    g_ctrl_mode = CTRL_MANUAL;
    g_armed = 0;

    g_target_base_pwm = 0;
    g_stab_base_pwm = 0;

    PWM_SetSafe();
    HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 1);
}
```

A w gałęzi "unkill" dodaj warunek, że **nie wolno unkill, jeśli baza nie jest niska**:

```
c

if (g_target_base_pwm <= 200) // albo 0
{
    if ((now - g_tilt_unkill_since) >= TILT_UNKILL_HOLD_MS)
    {
        g_tilt_kill = 0;
        g_tilt_unkill_since = 0;
        HAL_GPIO_WritePin(LED_SIGNAL_GPIO_Port, LED_SIGNAL_Pin, 0);
    }
}
```

To samo w sobie zwykle zmienia "katapulta" → "da się testować".

B) Dodaj "throttle desaturation" po policzeniu wyjść (żeby nie było brutalnego clipowania na 10000)

Teraz jak RF/RB dobiją do 10000, a reszta nie, to **miks się deformuje** i robi się chaos (runaway jest wtedy dużo łatwiejszy).

Po wyliczeniu **lf/rf/lb/rb** (int32) zrób przesunięcie wspólne (odejmij od wszystkich, jeśli ktoś przekracza max; dodaj do wszystkich, jeśli ktoś spada poniżej floor), dopiero potem clamp.

Wstaw tuż przed **clamp_u16()**:

```
c

int32_t lf = (int32_t)lroundf(out_lf_f);
int32_t rf = (int32_t)lroundf(out_rf_f);
int32_t lb = (int32_t)lroundf(out_lb_f);
int32_t rb = (int32_t)lroundf(out_rb_f);

int32_t maxv = lf;
if (rf > maxv) maxv = rf;
if (lb > maxv) maxv = lb;
if (rb > maxv) maxv = rb;

int32_t minv = lf;
if (rf < minv) minv = rf;
if (lb < minv) minv = lb;
if (rb < minv) minv = rb;

if (maxv > PWM_MAX)
```

```
{
    int32_t d = maxv - PWM_MAX;
    lf -= d; rf -= d; lb -= d; rb -= d;
}

if (minv < PWM_STAB_FL00R)
{
    int32_t d = PWM_STAB_FL00R - minv;
    lf += d; rf += d; lb += d; rb += d;
}
```

Potem dopiero:

```
c

out.motor_lf = clamp_u16(lf, PWM_STAB_FL00R, PWM_MAX);
...
...
```

To nie "naprawi" słabej prawej strony, ale **drastycznie** zmniejsza ryzyko, że clipowanie spowoduje nagły skręt/flip.

C) Apka PC: jak wykryjesz kill (PWM=0 przy base>0), natychmiast STOP ramp + wyślij STAB_OFF

W logu widać, że ramp leci dalej mimo kill.

W `rx_loop` przy `TELEM_DATA` możesz dorzucić prostą detekcję:

- jeśli `takeoff_on==1` i `current_base_sent > 1000` i wszystkie PWM są 0 → uznaj to za kill → zatrzymaj ramp i wyślij STAB_OFF.

Np. po odebraniu TELEM:

```
python

if self.takeoff_on and self.current_base_sent > 1000:
    if lf == 0 and rf == 0 and lb == 0 and rb == 0:
        self.takeoff_on = False
        self.takeoff_stop.set()
        self.send_frame(CMD_STAB_OFF)
        self.log_put("AUTO: detected KILL -> stop ramp + STAB_OFF")
        self.root.after(0, lambda: self.btn_takeoff.config(text="TAKEOFF (smooth)"))
```

To Ci od razu ucina pętlę „kill → re-spool na wysokim base → kill”.

I teraz brutalna prawda z logów

Nawet po tych zabezpieczeniach: w Twoich danych prawa strona wygląda na **realnie słabszą** (albo elektrycznie niedokarmioną), bo:

- różnica prawa-lewa ~2.3k PWM jest gigantyczna,
- RF/RB bardzo często dobija do 10000,
- a mimo to i tak łapiesz runaway roll/pitch.

Jeśli prawa strona potrzebuje 10000, żeby dogonić lewą, to soft nie wyczaruje większego thrustu.

Najszybszy test izolujący przyczynę (bez zmiany mapowania nazw)

Zrób test „czy problem idzie za silnikiem czy za ramieniem/zasilaniem”:

- Zamień fizycznie **silnik RF z LF** (tylko silniki na ramionach), **nie zmieniaj** nic w kodzie ani nazwach.
- Jeśli po zamianie dron zacznie „ciągnąć” w drugą stronę → winny silnik.
- Jeśli nadal ciągnie w prawo → winne ramię/power-path/CG (np. spadek napięcia na ścieżkach/tranzystorach po prawej).

To jest mega diagnostyczne i zajmuje 2 minuty.

Najlepsza kolejność działań (żebyś od razu poczuł różnicę)

1. **Latch tilt-kill** (DISARM + base=0, brak auto-unkill na wysokim base)
2. **Desaturation** (common-mode shift)
3. **Auto stop w apce PC** przy wykryciu kill
4. Test zamiany silników RF↔LF (lub RB↔LB)

Jeśli wrzucisz kolejny log po punktach **1–3** (nawet krótki, 20–30 sekund), to ja Ci powiem:

- czy zniknęły „samozwrotki” po killu,
- czy saturacja RF/RB spadła,
- i czy runaway roll/pitch zaczyna się od braku thrustu czy od miksu/sterowania.



Znaczy się ja chcę takie parametry, żeby dron stabilnie uniósł się nad ziemią nawet parę centymetrów i nie spierał mi w żadną stronę

Rozumiem dokładnie, o co Ci chodzi: **ma się spokojnie oderwać na kilka cm i stać w miejscu**, bez uciekania w prawo/lewo/przód/tył. Da się to osiągnąć *parametrami*, ale musisz zrobić jedną kluczową rzecz: **zabić stały błąd (asymetrię) zanim PID zacznie cokolwiek stabilizować**. U Ciebie z logów wynika, że prawa strona potrzebuje absurdalnie dużo więcej PWM, więc PID startuje już “pod ścianą” i kończy się flipem.

Poniżej masz **konkretny zestaw parametrów + 2 małe zasady startu**, które w praktyce dają “hover 2–5 cm bez spierania” nawet przy nierównym CG/silnikach.

1) Najważniejszy „pakiet hover” – wklej i testuj na STOP

A) Wyłącz CG biasy na czas ustawiania hover

```
C

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0
```

B) Przenieś asymetrię na per-motor TRIM (to robi robotę)

Z Twoich ostatnich logów wyszło mniej więcej (PWM – base) przy "niby hover":

- LF ~ -708
- LB ~ -965
- RF ~ +1694
- RB ~ +1098

To znaczy, że kontroler musi na stałe dopompowywać prawą stronę. Zrób to jawnie trimem:

```
C

#define MOTOR_OFFSET_LF (-700.0f)
#define MOTOR_OFFSET_LB (-950.0f)
#define MOTOR_OFFSET_RF (+1700.0f)
#define MOTOR_OFFSET_RB (+1100.0f)
```

I koniecznie zostaw to z Twoim **trimk** (skalowanie od base), **ale** przesuń jego zakres dużo niżej, żeby działało już przy odklejaniu:

```
C

#define CG_BIAS_START_PWM 600u
#define CG_BIAS_FULL_PWM 1800u
```

Wtedy trim zaczyna działać **zanim** dron się oderwie (to jest krytyczne).

C) Integrator też musi wejść wcześniejszej (żeby „dociągnąć” stały błąd)

```
C

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u
```

D) Na czas "hover only" wyłącz yaw (bo yaw Ci tylko zabiera zapas)

```
C

#define STAB_YAW_MAX 0.0f
#define STAB_KP_YAW_RATE 0.0f
```

(na chwilę — potem wróćmy)

2) PID: daj mu mniej agresji na starcie (żeby nie robił „szarpnięcia”)

Twoje **KP=16** jest dość ostre jak na mikrusa z brushed 8520 i “ground effect” przy startie. Na 2–5 cm ziemia robi turbulencję i P potrafi wywołać „kopnięcie”.

Na pierwsze stabilne oderwanie ustaw:

```
c

#define STAB_KP_ROLL    9.0f
#define STAB_KD_ROLL    2.5f
#define STAB_KI_ROLL    0.55f

#define STAB_KP_PITCH   9.0f
#define STAB_KD_PITCH   2.5f
#define STAB_KI_PITCH   0.55f

#define STAB_U_MAX      2200.0f
#define STAB_I_MAX      2200.0f
```

Dlaczego tak:

- niższe P → mniej “odskoku” w ground effect,
- trochę wyższe D → tłumi szybkie ucieczki po oderwaniu,
- I większe + wchodzi wcześniej → łapie stały bias zamiast wymuszać przechył.

3) Dwie zasady startu, bez których “parametry” dalej mogą wyglądać jak losowanie

Zasada 1: start tylko w NAV_STOP i bez rampowania do kosmosu

Na test hover:

- tylko **NAV_STOP**
- rampuj base wolno i nie przekraczaj np. 4000–5500 na siłę (zależy od Twojej mechaniki)

Zasada 2: po tilt-kill ma być twardy disarm i base=0

Jeśli po killu apka dalej wysyła NAV_SET z wysokim base, to zawsze będziesz miał “katapultę”.

Minimalnie: po killu wymuś w STM:

- g_armed=0**
- g_target_base_pwm=0**
- g_stab_base_pwm=0**

To nie jest “ficzer” – to jest warunek sensownego strojenia.

4) Jeśli po tym dalej ciągnie w prawo przy samym odklejeniu

Wtedy znaczy, że masz problem z **prawą stroną jako napędem**, a nie z PID-em. I tu masz jedną zmianę, która często jest game-changerem:

Zmień sposób użycia MOTOR_GAIN tak, żeby działał też na BASE

Zamiast:

```
c

out_rf = base + mix_rf * GAIN_RF + offs_rf*trimk;
```

zrób:

```
c

out_rf = (base + mix_rf) * GAIN_RF + offs_rf*trimk;
```

I ustaw prawą stronę na mocniejsze wzmacnienie (na start):

```
c

#define MOTOR_GAIN_LF 1.00f
#define MOTOR_GAIN_LB 1.00f
#define MOTOR_GAIN_RF 1.18f
#define MOTOR_GAIN_RB 1.14f
```

To sprawia, że prawa strona dostaje “więcej pary” **od pierwszego PWM**, a nie dopiero gdy pojawi się błąd i miiks.

5) Jak poznać, że jesteś blisko celu (prosty check z logu)

Po 10–20 sekundach testu hover w STOP:

- średnie (**pwm_i - base**) dla wszystkich 4 powinny być mniej więcej w granicach **±200...400**
- nie powinieneś widzieć dobijania do **10000** na RF/RB przez dłuższy czas
- roll/pitch powinny kręcić się w okolicach **0...±5°**, a nie narastać.

Podsumowanie: dokładny zestaw, który ma największą szansę dać “2–5 cm hover bez spierdalania”

1. **MOTOR_OFFSET_*** jak wyżej + **CG_BIAS_START/FULL** dużo niżej
2. I-term wchodzi wcześniej
3. Yaw na 0 na czas hover
4. PID spokojniejszy (P↓, D↑, I↑)

5. opcjonalnie: GAIN działa na (base+mix), nie tylko na mix

Jeśli wrzucisz **jeszcze jeden** log po tych ustawieniach (tylko STOP, 15–25 s, bez yaw), to ja Ci już wtedy policzę "na twardo" nowe **MOTOR_OFFSET_*** tak, żebyś docelowo miał te delty blisko zera (i powiem, czy prawa strona nadal dobija do limitu).



drone log 2026 02 01 14 00 08

1	6.016000000000531	0	0	S
2	6.016000000000531	0	0	S
3	6.016000000000531	0	0	S
4	6.04700000000048	0	0	S
5	6.078000000000429	0	0	S
6	6.125	0	0	S
	t_s	base	action	a

```
/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL  9.0f
#define STAB_KD_ROLL  2.5f
#define STAB_KI_ROLL  0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
```

```
#define STAB_KI_PITCH 0.55f

#define STAB_KI_YAW_RATE 0.0f

#define STAB_KFF_YAW    0.0f
#define STAB_YAW_SIGN   1.0f
#define STAB_YAW_I_MAX  1200.0f

#define YAW_CCW_IS_RF_LB 1

#define IMU_SWAP_ROLL_PITCH 1

#define STAB_ROLL_SIGN  1.0f
#define STAB_PITCH_SIGN -1.0f

#define CMD_TILT_SLEW_DEG_PER_S   80.0f
#define CMD_YAW_SLEW_DPS_PER_S   900.0f
#define BASE_SLEW_PWM_PER_S     12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S 6000.0f

#define YAW_MIX_FRONT_CCW_REAR_CW 1

#define MOTOR_GAIN_LF 1.000f
#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.180f
#define MOTOR_GAIN_RB 1.140f

#define MOTOR_OFFSETS_LF -700.0f
#define MOTOR_OFFSETS_LB -950.0f
#define MOTOR_OFFSETS_RF +1700.0f
#define MOTOR_OFFSETS_RB +1100.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG 0.0f
#define PITCH_TRIM_DEG 0.0f

#define LEVEL_CALIB_SAMPLES    200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0

#define CG_BIAS_START_PWM 600u
#define CG_BIAS_FULL_PWM 1800u

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u

// =====
```

```

// PROFILE SWITCH
// =====
#define CONTROL_PROFILE_GROUND_TEST 0

#if CONTROL_PROFILE_GROUND_TEST
#define NAV_TILT_DEG    20.0f
#define NAV_YAW_RATE_DPS 360.0f
#define NAV_THRUST_BIAS  700
#define NAV_MIX_GAIN    2.0f

#define STAB_U_MAX      2800.0f
#define STAB_I_MAX      1600.0f
#define STAB_YAW_MAX    1200.0f
#define STAB_KP_YAW_RATE 1.0f
#else
#define NAV_TILT_DEG    10.0f
#define NAV_YAW_RATE_DPS 180.0f
#define NAV_THRUST_BIAS  0
#define NAV_MIX_GAIN    1.0f

#define STAB_U_MAX      2200.0f
#define STAB_I_MAX      2200.0f
#define STAB_YAW_MAX    0.0f
#define STAB_KP_YAW_RATE 0.0f
#endif
/* USER CODE END PD */

```

Okej, to moje parametry, dorzucam logi. Prawa strona jest teraz chyba za mocna, dron zaczął mi się lekko obracać w kierunku CCW (odwrotnie do wskazówek zegara) w trybie STOP (0), po czym prawa strona się już podniosła, gdy lewa dalej była na ziemi i no obrócił mi się do boku, przez co przekroczył bezpieczne 35 stopni i się wyłączyły śmigła. Zweryfikuj, czy logi mówią to samo

Tak — log mówi dokładnie to, co opisujesz: **przegięłeś kompensację na prawą stronę** i dodatkowo (bo masz teraz yaw wyłączony) **dron sobie spokojnie “odjeżdża” w yaw (CCW)**, aż w końcu wchodzi w duży przechył i łapie tilt-kill.

Poniżej twarde fakty z Twojego CSV `drone_log_2026-02-01_14-00-08.csv`.

1) Yaw: w logu jest wyraźny obrót CCW w STOP

W przedziale, gdzie silniki realnie pracują (STOP, PWM>0, base ~3000–6500) yaw:

- rośnie o ok. **+99.6° w ~14.9 s → średnio ~+4.84°/s**
To jest stały dryf yaw (u Ciebie “CCW”) i to jest **dokładnie to, co widzisz**.

I to ma sens, bo masz:

c

```
#define STAB_YAW_MAX 0.0f
#define STAB_KP_YAW_RATE 0.0f
```

czyli **brak jakiekolwiek stabilizacji yaw**. Jak tylko momenty reakcyjne silników nie są idealnie równe, dron będzie się obracał.

2) Prawa strona jest teraz “za mocna” już od startu — log to pokazuje wprost

Już na bardzo niskim base masz próbki typu:

- t=8.25 s, base=1200
LF=851, LB=722, RF=2269, RB=1917

Czyli prawa strona dostaje ~+1100...1500 PWM więcej niż lewa *praktycznie od razu*.

To się idealnie zgadza z Twoim opisem: **prawa strona podnosi się pierwsza**, lewa jeszcze “trzyma ziemię” → moment boczny → tilt-kill.

3) Skąd to się wzięło: Twoje GAIN-y na RF/RB najpewniej mnożą także BASE (i to robi gigantyczną stałą dopłatę)

W tym logu (w “hover bandzie”) średnie (PWM - base) wyszły mi mniej więcej:

- LF ≈ -722
- LB ≈ -931
- RF ≈ +2634
- RB ≈ +1866

To wygląda “kosmicznie”, ale to bardzo pasuje do sytuacji, gdzie masz formułę w stylu:

```
(base + mix) * MOTOR_GAIN + MOTOR_OFFSET*trimk
```

Bo wtedy sam gain na RF=1.18 daje Ci dopłatę ok. 0.18*base:

- przy base ~5200 to jest ~+950 PWM zawsze,
a do tego dochodzi jeszcze Twoje MOTOR_OFFSET_RF (i trimk≈1), i robi się to, co widać w logu.

Krótko: **to nie PID “każe” podbijać prawą stronę — to Twoje stałe korekty robią z niej rakietę**.

Co zmienić, żeby w STOP nie obracał się i nie podrywał prawej strony

A) Włącz minimalną stabilizację yaw (bez tego zawsze będzie dryf)

Nie musisz robić agresywnie — tylko tyle, żeby trzymało $yaw rate \approx 0$.

Proponuję na start:

```
c

#define STAB_YAW_MAX      350.0f
#define STAB_KP_YAW_RATE  1.0f
#define STAB_KI_YAW_RATE  0.15f
#define STAB_YAW_I_MAX    600.0f
```

To zwykle wystarcza, żeby STOP nie kręcił się CCW/CW.

B) Cofnij “GAIN na base” — wróć do gainu działającego tylko na MIX

To jest najważniejsze, żeby prawa strona przestała podrywać pierwsza.

Czyli zamiast:

```
c

out = (base + mix) * gain + offs*trimk;
```

wróć do:

```
c

out = base + mix * gain + offs*trimk;
```

Jeśli zostawisz gain mnożący base, to nawet “małe” różnice gainów robią ogromną stałą asymetrię już od startu.

C) Wywal ujemne offsety na lewej stronie (one tylko psują start)

Te:

```
c

MOTOR_OFFSET_LF   -700
MOTOR_OFFSET_LB   -950
```

w praktyce **dusz a lewą stronę** w momencie, gdy i tak walczysz o oderwanie się od ziemi. To zwiększa ryzyko “podniesie się jedna strona”.

Na następny test ustaw:

```
c

#define MOTOR_OFFSET_LF 0.0f
#define MOTOR_OFFSET_LB 0.0f
#define MOTOR_OFFSET_RF 450.0f
#define MOTOR_OFFSET_RB 300.0f
```

I daj dużo łagodniejszy zakres trimk (żeby to NIE waliło już przy 900–1400):

```
c

#define CG_BIAS_START_PWM 1400u
#define CG_BIAS_FULL_PWM 3200u
```

D) Zredukuj gainy prawej strony (po cofnięciu “gain na base”)

Skoro teraz prawa strona wychodzi za mocno, to w kolejnym kroku nie dawaj 1.18/1.14. Zacznij od delikatnie:

```
c

#define MOTOR_GAIN_LF 1.000f
#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.040f
#define MOTOR_GAIN_RB 1.030f
```

Jak zrobić test, żeby wynik był jednoznaczny

1. Tylko **STOP**, żadnych NAV ruchów/yaw.
2. Ramp base powoli do momentu “2–5 cm hover” i trzymaj ~5–10 s.
3. Patrz w logu na:
 - czy yaw przestaje dryfować (po włączeniu yaw stabilizacji),
 - czy w okolicy odklejenia **RF/RB nie są dużo wyżej** niż LF/LB.

Jedno ważne zastrzeżenie (bez wciskania czegokolwiek)

Jeżeli po włączeniu yaw stabilizacji dron dalej będzie miał bardzo mocny stały dryf yaw (albo będzie wymagał ekstremalnych różnic PWM), to wtedy problem może być **czysto sprzętowy** (różne momenty/brushedy, różne śmigła, zasilanie prawej strony, itp.). Ale softem i tak da się to “przykryć”, tylko trzeba mieć yaw aktywne.

Jeśli wrzucisz kolejny log po zmianach A+B+C (to są 3 najważniejsze), to ja policzę Ci od razu:

- czy prawa strona dalej “wstaje pierwsza” (na podstawie wczesnych próbek base 800–2000),
- czy yaw rate został zgaszony,
- i jakie *finalne* **MOTOR_OFFSET_RF/RB** mają sens (już małe, a nie tysiące).

**drone log 2026 02 01 14 26 38**

1	7.5	0	0	S
2	7.5	0	0	S
3	7.5	0	0	S
4	7.530999999999949	0	0	S
5	7.563000000000102	0	0	S
6	7.60999999999673	0	0	S
	t_s	base	action	a

```
/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL  9.0f
#define STAB_KD_ROLL  2.5f
#define STAB_KI_ROLL  0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
#define STAB_KI_PITCH 0.55f

#define STAB_KFF_YAW   0.0f

#define STAB_YAW_SIGN  1.0f
#define STAB_ROLL_SIGN 1.0f
```

```
#define STAB_PITCH_SIGN -1.0f

#define STAB_U_MAX      2200.0f
#define STAB_I_MAX      2200.0f
#define STAB_YAW_MAX    350.0f
#define STAB_YAW_I_MAX  600.0f

#define STAB_KP_YAW_RATE 1.0f
#define STAB_KI_YAW_RATE 0.15f

#define CMD_TILT_SLEW_DEG_PER_S 80.0f
#define CMD_YAW_SLEW_DPS_PER_S 900.0f
#define BASE_SLEW_PWM_PER_S   12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S 6000.0f

#define IMU_SWAP_ROLL_PITCH 1
#define YAW_MIX_FRONT_CCW_REAR_CW 1
#define YAW_CCW_IS_RF_LB 1

#define MOTOR_GAIN_LF 1.000f
#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.040f
#define MOTOR_GAIN_RB 1.030f

#define MOTOR_OFFSETS_LF 0.0f
#define MOTOR_OFFSETS_LB 0.0f
#define MOTOR_OFFSETS_RF 450.0f
#define MOTOR_OFFSETS_RB 300.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG 0.0f
#define PITCH_TRIM_DEG 0.0f

#define LEVEL_CALIB_SAMPLES 200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0

#define CG_BIAS_START_PWM 1400u
#define CG_BIAS_FULL_PWM 3200u

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u

#define NAV_TILT_DEG     10.0f
#define NAV_YAW_RATE_DPS 180.0f
#define NAV_THRUST_BIAS  0
#define NAV_MIX_GAIN     1.0f
```

```
/* USER CODE END PD */
```

Okej, to obecne ustawienia i do nich logi. Dron obraca się wokół silnika RF, przy próbie wzbicia się nad ziemię przewraca mi się do przodu już na samym starcie. Sprawdź logi i to zweryfikuj

Tak — log mówi bardzo podobnie do tego, co opisujesz, tylko doprecyzowuje **oś i który róg dostaje "potyliku"**.

Co widać w logu drone_log_2026-02-01_14-26-38.csv

1) "Przewraca do przodu" = u Ciebie w logu idzie głównie ROLL, nie PITCH

W całym locie/podrywaniu:

- **max |pitch| ≈ 9.2°** (czyli pitch nie "ucieka" mocno),
- za to **roll dochodzi do ~-37.7°** przy **t ≈ 52.17 s** → to przebija TILT_KILL 35° i po chwili masz odcięcie.

Jeżeli fizycznie mówisz "do przodu", a w logu rośnie "roll", to jest spójne z Twoim ustawieniem **IMU_SWAP_ROLL_PITCH = 1** (czyli osie w regulatorze są "zamienione" względem tego, co widzisz jako roll/pitch w telemetrii).

2) W momencie wywrotki PWM pokazują, że regulator mocno podbija RF i mocno ucina LB

Przykład tuż przed kill:

- **t ≈ 52.078 s, base=7400**
 - LF=8105
 - **RF=8833 (najwyższy)**
 - **LB=6450 (najniższy)**
 - RB=6979

Czyli to nie wygląda jak "RF słaby i na nim się przewraca", tylko raczej: **regulator wali dużą korekcję po przekątnej RF↔LB**, co daje efekt "obrót/zwrot wokół tego narożnika" (w praktyce róg z najmniejszym ciągiem najczęściej zostaje "na ziemi" i robi się pivot).

3) Yaw dalej dryfuje (STOP, cmd=0)

Yaw w aktywnej fazie rośnie średnio ok. **~2.9°/s** (CCW) – czyli stabilizacja yaw nie trzyma jeszcze "zera" tak, jak powinna, ale to w tym logu **nie jest triggerem kill** (kill wynika z roll).

Najbardziej logiczny wniosek z tego logu

Masz złą “poziomicę” (level offset) o kilka stopni, przez co dron od startu próbuje lecieć z przechyłem i w końcu się przewraca.

To w logu widać po tym, że w zakresie „w miarę normalnego ciągu” (base ~4500–6500) średnio:

- roll $\approx -2.1^\circ$
- pitch $\approx +4.4^\circ$

To jest za dużo jak na “STOP hover kilka cm” – on powinien być blisko 0° , a nie stać z odchykiem kilku stopni (bo wtedy na ziemi zaczyna się ślizg/pivot i flip).

Co teraz ustaw (konkretne liczby)

Masz dwie drogi — wybierz jedną. Parametrami da się to “zabić” od razu.

Opcja A (najlepsza): skalibruj level na IDEALNIE poziomej powierzchni

To często rozwiązuje temat bez żadnych trimów. Warunki:

- dron nieruchomo,
- równa powierzchnia,
- poczekaj aż zakończy się Twoje **LEVEL_CALIB_SAMPLES** (u Ciebie 200 $\rightarrow \sim 2$ s).

Jeśli to zrobisz porządnie, trims mogą zostać 0.

Opcja B (najszybsza, bo z logu): dodaj TRIM-y w stopniach

Ponieważ masz **IMU_SWAP_ROLL_PITCH = 1**, to:

- **ROLL_TRIM_DEG** koryguje głównie to, co w telemetrii widzisz jako **pitch**,
- **PITCH_TRIM_DEG** koryguje głównie to, co w telemetrii widzisz jako **roll** (z uwagi na **STAB_PITCH_SIGN = -1**).

Z logu (średnie odchylenia):

- pitch $\approx +4.4^\circ \rightarrow$ daj **ROLL_TRIM_DEG $\approx -4.5^\circ$**
- roll $\approx -2.1^\circ \rightarrow$ daj **PITCH_TRIM_DEG $\approx -2.0^\circ$**

Czyli na start:

```
c

#define ROLL_TRIM_DEG (-4.5f)
#define PITCH_TRIM_DEG (-2.0f)
```

To ma bardzo dużą szansę sprawić, że w STOP przestanie “pchać” w jedną stronę już przy odklejaniu.

Drobna korekta motorów (bo jest OK, ale można doszlifować)

W tym logu przy base 2000–6000 średnio wychodzi:

- LF/LB blisko base (lejko na minus),
- RF ~ +450, RB ~ +300 względem base.

To nie jest już dramat (to wygląda sensownie), więc **ja bym na razie nie ruszał MOTOR_OFFSETS_RF/RB i MOTOR_GAIN_RF/RB**, dopóki nie wyzerujesz tego stałego przechyłu (trim/level-calib), bo inaczej będziesz gonił dwa problemy naraz.

Minimalny test po zmianie trimów

1. Tylko STOP.
2. Ramp base wolno do pierwszego "odklejenia" i trzymaj 5–10 s.
3. W logu oczekujesz:

- roll i pitch kręczą się w okolicy $\sim 0\ldots \pm 2\text{--}3^\circ$,
- brak narastania roll do kilkunastu stopni,
- PWM czterech motorów podobne (różnice rzędu setek, nie tysięcy).

Jeśli wrzucisz kolejny log po ustawieniu tych trimów (albo po ponownej kalibracji na poziomie), to Ci powiem:

- czy przechył "bazowy" faktycznie spadł do okolic zera,
- czy nadal w momencie odklejania pojawia się mocna korekcja RF↔LB,
- i dopiero wtedy dobierzemy subtelnie **MOTOR_OFFSETS_RF/RB** (jeśli będzie potrzebne).



drone log 2026 02 01 14 50 23

1	7.95299999999952	0	0	S
2	7.95299999999952	0	0	S
3	7.95299999999952	0	0	S
4	8.0	0	0	S
5	8.07799999999952	0	0	S
6	8.109000000000378	0	0	S
	t_s	base	action	a

```
#define ROLL_TRIM_DEG -4.5f
#define PITCH_TRIM_DEG -2.0f
```

Okej, dokonałem tych poprawek, tu masz logi. Dalej mi się trochę przechyla do przodu / w kierunku RF

Tak — log **potwierdza Twoje odczucie**: dalej masz tendencję do “pchania” w stronę **RF / przodu** (w Twojej percepcji), ale w telemetrii to wygląda jak **narastający ROLL w kierunku ujemnym** + umiarkowany dodatni PITCH, aż do **tilt-kill**.

Poniżej twarde fakty z CSV `drone_log_2026-02-01_14-50-23.csv`.

Co dokładnie mówi log

1) Kill jest od ROLL, nie od PITCH

W aktywnej fazie (PWM>0) masz:

- **max |roll| = 38.64°** przy **t ≈ 33.062 s** → to przebija 35° i chwilę później PWM idzie na 0
- **max |pitch| = 13.77°** (czyli pitch nie jest tym, co zabija)

Czyli główna ucieczka to **ROLL**.

2) Jak wygląda moment “wywrotki”

W oknie **32.5–33.1 s** (tuż przed odcięciem):

- roll spada: **-0.85 → -38.64°**
- pitch trzyma okolice **~8–14°**
- PWM pokazuje klasyczną odpowiedź na przekątnej:
 - **RF idzie najwyżej (~9.1k)**

- LB schodzi najniżej (~6.6–7.3k)

To się zgadza z Twoim opisem „ciągnie w kierunku RF” — w logu widać, że regulator już robi duże korekty, ale przechył i tak narasta.

3) Masz nadal stały bias w STOP (jeszcze zanim zacznie się dramat)

W stabilniejszym fragmencie (base ~3500–6500, t < 30 s) średnie wynoszą:

- **roll_mean ≈ -1.73°**
- **pitch_mean ≈ +1.11°**

Czyli nawet w “spokojnym” zakresie nie jesteś wyzerowany — a na ziemi to wystarczy, żeby przy odklejaniu zaczął się pivot i później lawina.

Dlaczego Twoje nowe trimy nie zadziałyły tak, jak miały

Przy Twoich ustawieniach:

- **IMU_SWAP_ROLL_PITCH = 1**
- **STAB_ROLL_SIGN = +1**
- **STAB_PITCH_SIGN = -1**

w trybie STOP (cmd=0) najprostsza zasada do ustawienia trimów jest taka:

- żeby wyzerować **pitch_deg** (telemetria) → ustaw **ROLL_TRIM_DEG ≈ mean(pitch_deg)**
- żeby wyzerować **roll_deg** (telemetria) → ustaw **PITCH_TRIM_DEG ≈ -mean(roll_deg)**

A u Ciebie w stabilnym fragmencie:

- $\text{mean}(\text{pitch_deg}) \approx +1.1^\circ \rightarrow \text{ROLL_TRIM_DEG}$ powinno być **około +1.1**, nie -4.5
- $\text{mean}(\text{roll_deg}) \approx -1.7^\circ \rightarrow \text{PITCH_TRIM_DEG}$ powinno być **około +1.7**, nie -2.0

Czyli Twoje poprzednie trimy były **w przeciwną stronę** względem tego, co pokazuje log.

Konkret: ustaw teraz te trimy (na podstawie Twojego logu)

Zmień na:

```
c

#define ROLL_TRIM_DEG    (+1.1f)
#define PITCH_TRIM_DEG   (+1.7f)
```

To jest dobrane z “czystego” fragmentu (przed runaway), gdzie dron jeszcze nie wpadł w chaos.

Co z motorami (RF/RB) — zostaw jak jest, na razie

W tym logu prawa strona nadal ma średnio większe PWM niż lewa (to wygląda jak kompensacja słabszej/cięższej prawej strony), ale **to nie jest już główny błąd do naprawy w tej iteracji.**

Najpierw wyzeruj "spoczynkowy" przechył trimami. Dopiero potem dopieszczamy **MOTOR_OFFSET_***.

Jak zrobić test, żeby wynik był jednoznaczny

1. Tylko **STOP** (żadnych NAV, żadnego yaw).
2. Ramp base powoli i zatrzymaj się przy pierwszym odklejeniu (parę cm) na 5–10 s.
3. W logu chcesz widzieć:
 - **roll_deg** i **pitch_deg** kręczą się blisko **$0 \pm 2\text{--}3^\circ$**
 - brak stałego trendu typu "roll cały czas bardziej ujemny"
 - PWM nie robią gwałtownej przekątnej RF↑ LB↓ bez realnej przyczyny

Jeśli po tych trimach dalej będzie "uciekał w stronę RF" dopiero przy wysokim base

Wtedy problem jest raczej **throttle-dependent** (prawa strona siada przy większym prądzie / spadek napięcia / tranzystory / ścieżki), i wtedy kolejnym krokiem będzie **base-zależny bias na roll** (łagodny, skalowany z base), ale najpierw zróbmy to, co log mówi wprost: wyzerujmy stały błąd.

Wrzuć proszę kolejny log po zmianie na (+1.1, +1.7) (wystarczy 15–25 s, STOP, spokojny ramp). Z niego powiem Ci już precyźniej:

- czy bias zniknął,
- czy runaway nadal startuje od roll,
- i czy trzeba dodać delikatny roll-bias zależny od base czy tylko doszlifować **MOTOR_OFFSET_RF/RB**.



drone log 2026 02 01 15 11 50

1	8.484000000000378	0	0	S
2	8.484000000000378	0	0	S
3	8.515999999999622	0	0	S
4	8.561999999999898	0	0	S
5	8.609000000000378	0	0	S
6	8.70299999999952	0	0	S
	t_s	base	action	a

```
/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL  9.0f
#define STAB_KD_ROLL  2.5f
#define STAB_KI_ROLL  0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
#define STAB_KI_PITCH 0.55f

#define STAB_KFF_YAW   0.0f

#define STAB_YAW_SIGN  1.0f
#define STAB_ROLL_SIGN 1.0f
#define STAB_PITCH_SIGN -1.0f

#define STAB_U_MAX     2200.0f
#define STAB_I_MAX     2200.0f
#define STAB_YAW_MAX   350.0f
```

```

#define STAB_YAW_I_MAX      600.0f

#define STAB_KP_YAW_RATE   1.0f
#define STAB_KI_YAW_RATE   0.15f

#define CMD_TILT_SLEW_DEG_PER_S    80.0f
#define CMD_YAW_SLEW_DPS_PER_S    900.0f
#define BASE_SLEW_PWM_PER_S      12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S  6000.0f

#define IMU_SWAP_ROLL_PITCH 1
#define YAW_MIX_FRONT_CCW_REAR_CW 1
#define YAW_CCW_IS_RF_LB 1

#define MOTOR_GAIN_LF  1.000f
#define MOTOR_GAIN_LB  1.000f
#define MOTOR_GAIN_RF  1.040f
#define MOTOR_GAIN_RB  1.030f

#define MOTOR_OFFSETS_LF 0.0f
#define MOTOR_OFFSETS_LB 0.0f
#define MOTOR_OFFSETS_RF 450.0f
#define MOTOR_OFFSETS_RB 300.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG +1.1f
#define PITCH_TRIM_DEG +1.7f

#define LEVEL_CALIB_SAMPLES     200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0

#define CG_BIAS_START_PWM 1400u
#define CG_BIAS_FULL_PWM 3200u

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u

#define NAV_TILT_DEG      10.0f
#define NAV_YAW_RATE_DPS 180.0f
#define NAV_THRUST_BIAS   0
#define NAV_MIX_GAIN      1.0f
/* USER CODE END PD */

```

Okej, łap log dla tych ustawień. Niestety, dalej widzę, że przy PWM, który jako tako go już powinien podnosić, kręci się wokół silnika RF i pochyla do przodu. Zweryfikuj, czy tak samo wynika z logów

Tak — w tych logach **widać dokładnie to, co opisujesz**: przy większym base pojawia się **stały przechył w kierunku „przód + RF”** (w Twoich odczuciach), a kontroler próbuje to ratować głównie **podbijając RF**. To jest klasyczny obraz: **kanał RF jest niedowydajny (słabszy thrust / większy spadek napięcia / większe straty), więc mimo że dostaje najwyższe PWM, róg RF i tak “siada” i robi się pivot wokół niego.**

Poniżej twarde rzeczy z CSV `drone_log_2026-02-01_15-11-50.csv`:

1) Postawa: przechył jest realny i rośnie wraz z base

W czasie gdy silniki pracują (STOP, PWM>0):

- **pitch** dochodzi do ok. **+10.65°**
- **roll** schodzi do ok. **-6.55°**
- i w “wysokim gazi” pitch ma wyraźny trend rosnący (u mnie wychodzi $\sim+0.46^\circ/\text{s}$ w odcinku z $\text{base} \geq 6000$)

To jest spójne z “pochyla do przodu”, tylko pamiętaj, że przy **IMU_SWAP_ROLL_PITCH=1** Twoje subiektywne “przód” często mapuje się na tę kombinację +pitch / -roll w telemetrii.

2) PWM: RF konsekwentnie jest najwyższy, a mimo to tilt idzie “w stronę RF”

W pasmie, gdzie już “powinno podnosić” (base 3000–6000), średnie odchyłki **PWM – base**:

- **LF ≈ +1**
- **LB ≈ +51**
- **RF ≈ +398**
- **RB ≈ +296**

A przy większym base (≥ 6500) sytuacja jest podobna: RF nadal odstaje najwyżej.

To jest dokładnie wzorzec „RF jest najsłabszy → trzeba mu dawać najwięcej PWM → a i tak róg RF zostaje *relatywnie* najniżej i robi pivot”.

3) Yaw: kontrola działa, ale i tak masz stały dryf

Yaw w tym logu zjeżdża średnio około **$-1.5^\circ/\text{s}$** (czyli obrót CW). To nie jest powód pivotu, ale na ziemi (częściowo dociążony róg) yaw potrafi “podcinać” stabilizację i pogłębiać przewrotkę.

Wniosek z logu

Twoje obecne kompensacje na RF są nadal za małe (w sensie: za mało “thrustu” z RF przy tym samym PWM), więc kontroler próbuje to nadrobić i widać to w PWM, ale mechanicznie i tak róg RF “przegrywa”.

Co zmienić teraz (jedna, mała iteracja — bez rozwalania reszty)

A) Podbij tylko RF (bo to on najbardziej odstaje)

Zostaw RB jak jest, a RF zwiększą krokowo:

```
c

#define MOTOR_GAIN_RF 1.070f // było 1.040
#define MOTOR_OFFSET_RF 650.0f // było 450
```

RB zostaw na:

```
c

#define MOTOR_GAIN_RB 1.030f
#define MOTOR_OFFSET_RB 300.0f
```

To jest celowo mały krok (żeby nie wrócić do sytuacji “prawa strona za mocna”).

B) Zmniejsz wpływ TRIM-ów na tę iterację (bo one mogą Ci przesuwać “celowy kąt”)

Twoje trymy teraz są niezerowe, a log pokazuje, że problem jest throttle-dependent (narasta z base). Na test “hover kilka cm” lepiej, żeby kontroler celował w **0°**, a nie w sztuczny offset.

Na jedną próbę ustaw:

```
c

#define ROLL_TRIM_DEG 0.0f
#define PITCH_TRIM_DEG 0.0f
```

Jeśli po tym dron przestanie “uciekać do przodu/RF”, to znaczy że trymy faktycznie przesuwały target i pogarszały sytuację przy większym ciągu.

C) Jeśli nadal “przód” rośnie z base – dodaj bardzo mały pitch CG-bias (tylko jeśli trzeba)

Masz CG biasy wyłączone. Jeżeli po poprawce RF dalej będzie “przód”, dodaj minimalnie:

```
c

#define CG_PITCH_BIAS_PWM (-80) // delikatnie wspiera tył (LB/RB)
#define CG_ROLL_BIAS_PWM 0
```

To jest mały ruch — nie 200+, tylko 80.

Najlepszy test, żeby log był jednoznaczny

1. Tylko **STOP**, żadnych NAV, żadnego yaw.
2. Ramp base wolno do pierwszego odklejenia (2–5 cm) i trzymaj **5–8 s**.
3. W logu sprawdzamy:
 - czy pitch przestaje rosnąć w czasie,
 - czy RF przestaje być “ciągle najbardziej dopompowany”,
 - czy roll/pitch trzymają się w okolicach $0 \pm$ kilka stopni.

Jedna rzecz “na 60 sekund”, która potwierdzi hipotezę na 100%

Jeśli masz możliwość: **zamień fizycznie silnik RF z LF** (bez zmiany mapowania w kodzie).

- Jeśli problem “idzie za silnikiem” → RF motor jest słabszy (zużyty / gorsza szczotka / gorszy śmigielko).
- Jeśli problem zostaje na ramieniu RF → to jest kanał zasilania/tranzystor/ścieżka (spadki napięcia pod obciążeniem).

To by idealnie wyjaśniało, czemu RF ciągle ma najwyższe PWM, a i tak pivotujesz wokół niego.

Jeśli wrzucisz log po zmianach **A + (B na test)**, to ja Ci powiem już “wprost z liczb”:

- czy pivot w stronę RF się zmniejszył,
- czy pitch trend (ten rosnący z base) zniknął,
- i czy warto dopiero wtedy dorzucać mały **CG_PITCH_BIAS_PWM**.



drone log 2026 02 01 19 11 18

	t_s	base	action	a
1	45.07900000000154	0	0	S
2	45.07900000000154	0	0	S
3	45.09400000000096	0	0	S
4	45.125	0	0	S
5	45.1720000000023	0	0	S
6	45.26600000000326	0	0	S

```
/* USER CODE BEGIN PD */  
#define DEG2RAD 0.0174532925f  
#define RAD2DEG 57.2957795f  
  
#define IMU_UPDATE_PERIOD_MS 10u  
  
#define ACC_LPF_CUTOFF_HZ 20.0f  
#define ACC_G_MIN      0.40f  
#define ACC_G_MAX      2.00f  
  
#define TILT_KILL_DEG   35.0f  
#define TILT_UNKILL_DEG 25.0f  
#define TILT_KILL_DEBOUNCE_MS 50u  
#define TILT_UNKILL_HOLD_MS 300u  
  
#define PWM_MIN 0  
#define PWM_MAX 10000  
#define PWM_STAB_FLOOR 0  
  
#define STAB_KP_ROLL  9.0f  
#define STAB_KD_ROLL  2.5f  
#define STAB_KI_ROLL  0.55f  
  
#define STAB_KP_PITCH 9.0f  
#define STAB_KD_PITCH 2.5f  
#define STAB_KI_PITCH 0.55f  
  
#define STAB_KFF_YAW  0.0f  
  
#define STAB_YAW_SIGN 1.0f  
#define STAB_ROLL_SIGN 1.0f  
#define STAB_PITCH_SIGN -1.0f  
  
#define STAB_U_MAX    2200.0f  
#define STAB_I_MAX    2200.0f  
#define STAB_YAW_MAX   350.0f  
#define STAB_YAW_I_MAX 600.0f  
  
#define STAB_KP_YAW_RATE 1.0f  
#define STAB_KI_YAW_RATE 0.15f  
  
#define CMD_TILT_SLEW_DEG_PER_S 80.0f  
#define CMD_YAW_SLEW_DPS_PER_S 900.0f  
#define BASE_SLEW_PWM_PER_S 12000.0f  
#define NAV_BIAS_SLEW_PWM_PER_S 6000.0f  
  
#define IMU_SWAP_ROLL_PITCH 1  
#define YAW_MIX_FRONT_CCW_REAR_CW 1  
#define YAW_CCW_IS_RF_LB 1  
  
#define MOTOR_GAIN_LF 1.000f
```

```

#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.070f
#define MOTOR_GAIN_RB 1.030f

#define MOTOR_OFFSET_LF 0.0f
#define MOTOR_OFFSET_LB 0.0f
#define MOTOR_OFFSET_RF 650.0f
#define MOTOR_OFFSET_RB 300.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG +0.0f
#define PITCH_TRIM_DEG +0.0f

#define LEVEL_CALIB_SAMPLES 200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM -80
#define CG_ROLL_BIAS_PWM 0

#define CG_BIAS_START_PWM 1400u
#define CG_BIAS_FULL_PWM 3200u

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u

#define NAV_TILT_DEG 10.0f
#define NAV_YAW_RATE_DPS 180.0f
#define NAV_THRUST_BIAS 0
#define NAV_MIX_GAIN 1.0f
/* USER CODE END PD */

```

Okej, to moje obecne ustawienia i log, sam zobacz co zrobił

Tak — ten log **mówią dokładnie to samo**, co widzisz na żywo: przy „prawie-hoverowym” PWM dron zaczyna pivotować wokół narożnika RF i jednocześnie idzie w przekątnej “przód+RF” (w telemetrii to jest kombinacja **roll mocno na minus + pitch na plus**), po czym łapie tilt-kill.

Poniżej konkretny z `drone_log_2026-02-01_19-11-18.csv`:

Co pokazuje log

1) Kill jest od roll (i jest gwałtowny)

W oknie tuż przed odcięciem:

- **t = 61.17 → 61.28 s**: roll spada **-23.9° → -43.9°**

- **t = 61.33 s:** roll **-46.1°** i PWM lecą na **0** (tilt-kill)

Pitch w tym samym momencie jest mały (okolice **~4–5°**), więc to **nie pitch** Cię zabija — tylko **roll runaway**.

2) Kontroler “walczy po przekątnej”, ale to wygląda jak pivot na ziemi

Tuż przed kill masz (przykład t=61.25–61.28 s):

- RF jest **najwyżej** (np. **7701 / 7240**)
- LB jest **najniżej** (np. **4885 / 4921**)

To jest klasyczny wzorzec: kontroler próbuje zbić przechył, ale jeśli jeden róg nadal “trzyma” tarcie o ziemię (albo podnosi się inny róg szybciej), to robi się **pivot** i kąt ucieka, mimo że PWM już mocno różnicujesz.

3) Jeszcze przed dramatem masz stały bias “przód + RF”

W “spokojniejszym” fragmencie (base 3000–6000, zanim roll zacznie odjeżdżać na kilkadziesiąt stopni):

- średnio **roll ≈ -1.95°**
- średnio **pitch ≈ +4.14°**

Czyli nawet zanim zacznie się flip, dron już “chce” iść w stronę **RF/przodu**.

Dodatkowo, w tym samym fragmencie średnie (**PWM - base**):

- **LF ≈ -153**
- **LB ≈ -4**
- **RF ≈ +613**
- **RB ≈ +419**

Czyli prawa strona nadal dostaje wyraźnie więcej PWM — i to jest spójne z Twoją historią (prawa strona jest realnie słabsza / bardziej obciążona), ale na ziemi to potrafi generować “podnosi się jedno ramię → pivot → flip”.

Wniosek

Log potwierdza pivot wokół RF + “przód/RF bias”.

I to nie wygląda jak brak headroomu (nie dobijasz do 10000), tylko jak **problem startu blisko ziemi + stały bias**, który wchodzi zanim całość równo oderwie się od podłoża.

Co bym zmienił w kolejnej iteracji (żeby wreszcie odrywała równo)

1) Wzmocnij kompensację “przodu” (bo **pitch_mean** nadal **+4°**)

Twoje **CG_PITCH_BIAS_PWM = -80** jest po prostu za małe.

Daj krokowo (nie od razu mega):

```
c

#define CG_PITCH_BIAS_PWM (-200)
```

To podbije tył (LB/RB) względem przodu (LF/RF) i powinno wyraźnie zmniejszyć "przód nurkuje przy startie".

2) Dodaj mały roll-bias na prawą stronę (bo roll_mean nadal ~-2°)

Żeby nie gonić tego samymi gainami RF/RB:

```
c

#define CG_ROLL_BIAS_PWM (-140)
```

(U Ciebie znak ujemny daje boost prawej stronie w Twoim mikserze.)

3) Zostaw RF gain/offs, ale nie dokręcaj dużo więcej na raz

Na tym logu RF i RB już są wyraźnie dopompowane. Dalsze "dokręcanie" RF łatwo kończy się tym, że **prawa strona wstaje pierwsza** i znowu pivot.

Na następną próbę zostawiłem:

```
c

#define MOTOR_GAIN_RF 1.070f
#define MOTOR_OFFSET_RF 650.0f
#define MOTOR_GAIN_RB 1.030f
#define MOTOR_OFFSET_RB 300.0f
```

i najpierw wyprostował "przód/prawo" bias przez **CG_PITCH/CG_ROLL** (to jest bardziej "gładkie" i przewidywalne przy startie).

Najważniejsze (i to jest game-changer przy takich objawach): "takeoff blend"

To nie jest zmiana znaków ani mapowania — to jest zabezpieczenie przed tym, że na ziemi regulator zaczyna różnicować PWM, a dron zamiast się podnieść równo, robi pivot.

Zrób tak: **poniżej pewnego base** ogranicz siłę stabilizacji (roll/pitch), a dopiero gdy jest bliżej hovera, wpuść pełny regulator.

Minimalnie (w **ControlStep_Stabilize**, zaraz po policzeniu **u_roll** i **u_pitch**, przed mikserem):

```
c

#define TAKEOFF_BLEND_START 1800.0f
#define TAKEOFF_BLEND_FULL 4200.0f

float blend;
```

```

if (g_stab_base_pwm <= TAKEOFF_BLEND_START) blend = 0.25f;
else if (g_stab_base_pwm >= TAKEOFF_BLEND_FULL) blend = 1.0f;
else {
    blend = ( (float)g_stab_base_pwm - TAKEOFF_BLEND_START ) /
        ( TAKEOFF_BLEND_FULL - TAKEOFF_BLEND_START );
    blend = 0.25f + 0.75f * blend;
}
u_roll *= blend;
u_pitch *= blend;

```

Efekt: przy odklejaniu regulator nie robi "szarpnięcia po przekątnej", które powoduje pivot na jednym rogu.

Jak sprawdzimy, że jest lepiej w logu

Po zmianach (CG_PITCH -200, CG_ROLL -140, takeoff blend):

- w zakresie base 3000–6000 średnio **pitch** powinien spaść w okolice **0...±2°**
- **roll** też blisko **0...±2°**
- i co kluczowe: nie powinno być takiej sekwencji jak w tym logu: **roll -20 → -45 w 0.2 s.**

Jeśli chcesz, to podeśle Ci też gotowe "docelowe" wartości CG_PITCH/CG_ROLL policzone stricte z Twoich danych (żeby nie robić tego na oko) — ale już teraz log jasno potwierdza Twoją diagnozę: **pivot w stronę RF + przód** i potem ucieczka roll do kill.



Okej szybkie pytanko - czy cg pitch nie powinno być dodatnie?

W Twoim **konkretnym mikserze** znak **CG_PITCH_BIAS_PWM** działa tak:

Masz:

```

c

c_lf += pitch_bias;
c_rf += pitch_bias;
c_lb -= pitch_bias;
c_rb -= pitch_bias;

```

Czyli:

- gdy **CG_PITCH_BIAS_PWM** jest **dodatnie** → **pitch_bias > 0**
LF/RF dostają więcej, LB/RB mniej
 ⇒ **przód dostaje więcej ciągu** (nose idzie raczej w dół / bardziej "ciągnie do przodu" przy startie)
- gdy **CG_PITCH_BIAS_PWM** jest **ujemne** → **pitch_bias < 0**
LF/RF dostają mniej, LB/RB więcej
 ⇒ **tył dostaje więcej ciągu** (nose ma tendencję iść w górę / przestaje "nurkować do przodu")

Więc jeśli u Ciebie dron **nurkował do przodu** przy startie (nose down), to dokładnie wtedy **CG_PITCH_BIAS_PWM** powinno iść w stronę **ujemną**, bo chcesz dopompować tył (LB/RB) i odciążyć przód (LF/RF).

To nie jest "uniwersalnie dodatnie/ujemne" — wynika 1:1 z tych czterech linijek w mikserze. W Twoim układzie:

ujemne = więcej ciągu na tył, dodatnie = więcej ciągu na przód.



```
static void ControlStep_Stabilize(void)
{
    static uint32_t last_ctrl = 0;
    uint32_t now = HAL_GetTick();
    float dt = 0.01f;

    if (last_ctrl != 0)
    {
        dt = (now - last_ctrl) * 0.001f;
        if (dt < 0.001f) dt = 0.001f;
        if (dt > 0.05f) dt = 0.05f;
    }
    last_ctrl = now;

    if (g_tilt_kill) { PWM_SetSafe(); return; }
    if (!g_att_valid) return;

    if (g_target_base_pwm > PWM_MAX) g_target_base_pwm = PWM_MAX;

    g_stab_base_pwm = (uint16_t)lroundf(
        slew_f((float)g_stab_base_pwm, (float)g_target_base_pwm, BASE_SLEW_PWM_PER_S, dt)
    );

    g_cmd_roll_deg = slew_f(g_cmd_roll_deg, g_target_cmd_roll_deg, CMD_TILT_SLEW_DEG_PER_S,
                           dt);
    g_cmd_pitch_deg = slew_f(g_cmd_pitch_deg, g_target_cmd_pitch_deg,
                           CMD_TILT_SLEW_DEG_PER_S, dt);
    g_cmd_yaw_rate_dps = slew_f(g_cmd_yaw_rate_dps, g_target_cmd_yaw_rate_dps,
                           CMD_YAW_SLEW_DPS_PER_S, dt);

    int32_t target_bias = 0;
    if (g_last_nav_action == NAV_FORWARD || g_last_nav_action == NAV_BACK ||
        g_last_nav_action == NAV_LEFT || g_last_nav_action == NAV_RIGHT)
    {
        target_bias = (int32_t)NAV_THRUST_BIAS;
    }
    g_nav_bias = slew_i32(g_nav_bias, target_bias, NAV_BIAS_SLEW_PWM_PER_S, dt);

    if (!g_i_enabled)
    {
```

```

    if (g_stab_base_pwm >= I_ENABLE_BASE_PWM) g_i_enabled = 1;
}
else
{
    if (g_stab_base_pwm <= I_DISABLE_BASE_PWM) g_i_enabled = 0;
}

float roll_meas_raw, pitch_meas_raw;
float roll_rate_raw, pitch_rate_raw;

#if IMU_SWAP_ROLL_PITCH
roll_meas_raw = (float)g_pitch_deg - (float)g_level_pitch_off;
pitch_meas_raw = (float)g_roll_deg - (float)g_level_roll_off;

roll_rate_raw = (float)g_gyro_pitch_dps;
pitch_rate_raw = (float)g_gyro_roll_dps;
#else
roll_meas_raw = (float)g_roll_deg - (float)g_level_roll_off;
pitch_meas_raw = (float)g_pitch_deg - (float)g_level_pitch_off;

roll_rate_raw = (float)g_gyro_roll_dps;
pitch_rate_raw = (float)g_gyro_pitch_dps;
#endif

float roll_meas = roll_meas_raw * STAB_ROLL_SIGN;
float pitch_meas = pitch_meas_raw * STAB_PITCH_SIGN;

float roll_rate = roll_rate_raw * STAB_ROLL_SIGN;
float pitch_rate = pitch_rate_raw * STAB_PITCH_SIGN;

float cmd_roll = g_cmd_roll_deg + ROLL_TRIM_DEG;
float cmd_pitch = g_cmd_pitch_deg + PITCH_TRIM_DEG;

float err_roll = cmd_roll - roll_meas;
float err_pitch = cmd_pitch - pitch_meas;

float u_roll_pd = STAB_KP_ROLL * err_roll - STAB_KD_ROLL * roll_rate;
float u_pitch_pd = STAB_KP_PITCH * err_pitch - STAB_KD_PITCH * pitch_rate;

u_roll_pd = clamp_f(u_roll_pd, -STAB_U_MAX, STAB_U_MAX);
u_pitch_pd = clamp_f(u_pitch_pd, -STAB_U_MAX, STAB_U_MAX);

float yaw_rate = (float)g_gyro_yaw_dps * STAB_YAW_SIGN;
float yaw_cmd = (float)g_cmd_yaw_rate_dps * STAB_YAW_SIGN;
float yaw_rate_err = yaw_cmd - yaw_rate;

float u_yaw = STAB_KP_YAW_RATE * yaw_rate_err
    + STAB_KI_YAW_RATE * i_yaw
    + STAB_KFF_YAW * yaw_cmd;

u_yaw = clamp_f(u_yaw, -STAB_YAW_MAX, STAB_YAW_MAX);

```

```

float u_yaw_total = u_yaw + YAW_TRIM;
u_yaw_total = clamp_f(u_yaw_total, -STAB_YAW_MAX, STAB_YAW_MAX);

float u_roll = u_roll_pd + STAB_KI_ROLL * i_roll;
float u_pitch = u_pitch_pd + STAB_KI_PITCH * i_pitch;

u_roll = clamp_f(u_roll, -STAB_U_MAX, STAB_U_MAX);
u_pitch = clamp_f(u_pitch, -STAB_U_MAX, STAB_U_MAX);

int32_t base = (int32_t)g_stab_base_pwm;

float c_lf = (-u_pitch + u_roll);
float c_rf = (-u_pitch - u_roll);
float c_lb = (u_pitch + u_roll);
float c_rb = (u_pitch - u_roll);

float nav = (float)g_nav_bias * NAV_MIX_GAIN;
switch ((uint8_t)g_last_nav_action)
{
    case NAV_FORWARD: c_lb += nav; c_rb += nav; c_lf -= nav; c_rf -= nav; break;
    case NAV_BACK: c_lb -= nav; c_rb -= nav; c_lf += nav; c_rf += nav; break;
    case NAV_LEFT: c_rb += nav; c_rf += nav; c_lb -= nav; c_lf -= nav; break;
    case NAV_RIGHT: c_lb += nav; c_lf += nav; c_rb -= nav; c_rf -= nav; break;
    default: break;
}

float cgk = CgBiasScale(g_stab_base_pwm);
float pitch_bias = (float)CG_PITCH_BIAS_PWM * cgk;
float roll_bias = (float)CG_ROLL_BIAS_PWM * cgk;

c_lf += pitch_bias;
c_rf += pitch_bias;
c_lb -= pitch_bias;
c_rb -= pitch_bias;

c_lf += roll_bias;
c_lb += roll_bias;
c_rf -= roll_bias;
c_rb -= roll_bias;

#if YAW_CCW_IS_RF_LB
    c_rf += u_yaw_total;
    c_lb += u_yaw_total;
    c_lf -= u_yaw_total;
    c_rb -= u_yaw_total;
#else
    c_lf += u_yaw_total;
    c_rb += u_yaw_total;
    c_rf -= u_yaw_total;
    c_lb -= u_yaw_total;

```

#endif

```

float k = Mixer_ComputeScale(base, c_lf, c_rf, c_lb, c_rb);
int sat = (k < 0.999f) ? 1 : 0;

if (!sat && g_i_enabled)
{
    i_roll += err_roll * dt;
    i_pitch += err_pitch * dt;
    i_yaw += yaw_rate_err * dt;

    i_roll = clamp_f(i_roll, -STAB_I_MAX, STAB_I_MAX);
    i_pitch = clamp_f(i_pitch, -STAB_I_MAX, STAB_I_MAX);
    i_yaw = clamp_f(i_yaw, -STAB_YAW_I_MAX, STAB_YAW_I_MAX);
}
else
{
    if (!g_i_enabled)
    {
        i_roll *= (1.0f - 0.8f * dt);
        i_pitch *= (1.0f - 0.8f * dt);
        i_yaw *= (1.0f - 0.8f * dt);
    }
}

float mix_lf = k * c_lf;
float mix_rf = k * c_rf;
float mix_lb = k * c_lb;
float mix_rb = k * c_rb;

float trimk = CgBiasScale(g_stab_base_pwm);

float out_lf_f = (float)base + mix_lf * MOTOR_GAIN_LF + MOTOR_OFFSET_LF * trimk;
float out_rf_f = (float)base + mix_rf * MOTOR_GAIN_RF + MOTOR_OFFSET_RF * trimk;
float out_lb_f = (float)base + mix_lb * MOTOR_GAIN_LB + MOTOR_OFFSET_LB * trimk;
float out_rb_f = (float)base + mix_rb * MOTOR_GAIN_RB + MOTOR_OFFSET_RB * trimk;

int32_t lf = (int32_t)lroundf(out_lf_f);
int32_t rf = (int32_t)lroundf(out_rf_f);
int32_t lb = (int32_t)lroundf(out_lb_f);
int32_t rb = (int32_t)lroundf(out_rb_f);

PwmPayload_t out;
out.motor_lf = clamp_u16(lf, PWM_STAB_FLOOR, PWM_MAX);
out.motor_rf = clamp_u16(rf, PWM_STAB_FLOOR, PWM_MAX);
out.motor_lb = clamp_u16(lb, PWM_STAB_FLOOR, PWM_MAX);
out.motor_rb = clamp_u16(rb, PWM_STAB_FLOOR, PWM_MAX);

SetPwm(&out);
}

```

Okej, to wkleisz mi cały ControlStep_Stabilize z tą zmianą blend'u?

Pewnie — poniżej masz cały **ControlStep_Stabilize()** z dodanym **takeoff blend** (skaluje **u_roll** i **u_pitch** przed mikserem). Zostawiłem Twoją logikę 1:1, tylko wstawiłem blok blend po wyliczeniu **u_roll** i **u_pitch** (po clampach), **przed** liczeniem **c_lf/c_rf/c_lb/c_rb**.

Wartości progów blend możesz potem łatwo dostroić: **TAKEOFF_BLEND_START_PWM**, **TAKEOFF_BLEND_FULL_PWM**, oraz **TAKEOFF_BLEND_MIN**.

```

c

static void ControlStep_Stabilize(void)
{
    static uint32_t last_ctrl = 0;
    uint32_t now = HAL_GetTick();
    float dt = 0.01f;

    if (last_ctrl != 0)
    {
        dt = (now - last_ctrl) * 0.001f;
        if (dt < 0.001f) dt = 0.001f;
        if (dt > 0.05f) dt = 0.05f;
    }
    last_ctrl = now;

    if (g_tilt_kill) { PWM_SetSafe(); return; }
    if (!g_att_valid) return;

    if (g_target_base_pwm > PWM_MAX) g_target_base_pwm = PWM_MAX;

    g_stab_base_pwm = (uint16_t)lroundf(
        slew_f((float)g_stab_base_pwm, (float)g_target_base_pwm, BASE_SLEW_PWM_PER_S, dt)
    );

    g_cmd_roll_deg = slew_f(g_cmd_roll_deg, g_target_cmd_roll_deg, CMD_TILT_SLEW_DEG_PER_S, dt);
    g_cmd_pitch_deg = slew_f(g_cmd_pitch_deg, g_target_cmd_pitch_deg, CMD_TILT_SLEW_DEG_PER_S,
    dt);
    g_cmd_yaw_rate_dps = slew_f(g_cmd_yaw_rate_dps, g_target_cmd_yaw_rate_dps,
CMD_YAW_SLEW_DPS_PER_S, dt);

    int32_t target_bias = 0;
    if (g_last_nav_action == NAV_FORWARD || g_last_nav_action == NAV_BACK ||
        g_last_nav_action == NAV_LEFT || g_last_nav_action == NAV_RIGHT)
    {
        target_bias = (int32_t)NAV_THRUST_BIAS;
    }
    g_nav_bias = slew_i32(g_nav_bias, target_bias, NAV_BIAS_SLEW_PWM_PER_S, dt);

    if (!g_i_enabled)
    {
        if (g_stab_base_pwm >= I_ENABLE_BASE_PWM) g_i_enabled = 1;
    }
    else
    {
        if (g_stab_base_pwm <= I_DISABLE_BASE_PWM) g_i_enabled = 0;
    }

    float roll_meas_raw, pitch_meas_raw;
    float roll_rate_raw, pitch_rate_raw;

#if IMU_SWAP_ROLL_PITCH
    roll_meas_raw = (float)g_pitch_deg - (float)g_level_pitch_off;
    pitch_meas_raw = (float)g_roll_deg - (float)g_level_roll_off;

    roll_rate_raw = (float)g_gyro_pitch_dps;
    pitch_rate_raw = (float)g_gyro_roll_dps;
#else

```

```

roll_meas_raw = (float)g_roll_deg - (float)g_level_roll_off;
pitch_meas_raw = (float)g_pitch_deg - (float)g_level_pitch_off;

roll_rate_raw = (float)g_gyro_roll_dps;
pitch_rate_raw = (float)g_gyro_pitch_dps;
#endif

float roll_meas = roll_meas_raw * STAB_ROLL_SIGN;
float pitch_meas = pitch_meas_raw * STAB_PITCH_SIGN;

float roll_rate = roll_rate_raw * STAB_ROLL_SIGN;
float pitch_rate = pitch_rate_raw * STAB_PITCH_SIGN;

float cmd_roll = g_cmd_roll_deg + ROLL_TRIM_DEG;
float cmd_pitch = g_cmd_pitch_deg + PITCH_TRIM_DEG;

float err_roll = cmd_roll - roll_meas;
float err_pitch = cmd_pitch - pitch_meas;

float u_roll_pd = STAB_KP_ROLL * err_roll - STAB_KD_ROLL * roll_rate;
float u_pitch_pd = STAB_KP_PITCH * err_pitch - STAB_KD_PITCH * pitch_rate;

u_roll_pd = clamp_f(u_roll_pd, -STAB_U_MAX, STAB_U_MAX);
u_pitch_pd = clamp_f(u_pitch_pd, -STAB_U_MAX, STAB_U_MAX);

float yaw_rate = (float)g_gyro_yaw_dps * STAB_YAW_SIGN;
float yaw_cmd = (float)g_cmd_yaw_rate_dps * STAB_YAW_SIGN;
float yaw_rate_err = yaw_cmd - yaw_rate;

float u_yaw = STAB_KP_YAW_RATE * yaw_rate_err
    + STAB_KI_YAW_RATE * i_yaw
    + STAB_KFF_YAW * yaw_cmd;

u_yaw = clamp_f(u_yaw, -STAB_YAW_MAX, STAB_YAW_MAX);

float u_yaw_total = u_yaw + YAW_TRIM;
u_yaw_total = clamp_f(u_yaw_total, -STAB_YAW_MAX, STAB_YAW_MAX);

float u_roll = u_roll_pd + STAB_KI_ROLL * i_roll;
float u_pitch = u_pitch_pd + STAB_KI_PITCH * i_pitch;

u_roll = clamp_f(u_roll, -STAB_U_MAX, STAB_U_MAX);
u_pitch = clamp_f(u_pitch, -STAB_U_MAX, STAB_U_MAX);

// =====
// TAKEOFF BLEND (anti-pivot near ground)
// =====
#define TAKEOFF_BLEND_START_PWM 1800.0f
#define TAKEOFF_BLEND_FULL_PWM 4200.0f
#define TAKEOFF_BLEND_MIN 0.25f

float blend;
float base_f = (float)g_stab_base_pwm;

if (base_f <= TAKEOFF_BLEND_START_PWM)
{
    blend = TAKEOFF_BLEND_MIN;
}
else if (base_f >= TAKEOFF_BLEND_FULL_PWM)
{
    blend = 1.0f;
}
else
{
    float t = (base_f - TAKEOFF_BLEND_START_PWM) /
        (TAKEOFF_BLEND_FULL_PWM - TAKEOFF_BLEND_START_PWM);
    blend = TAKEOFF_BLEND_MIN + (1.0f - TAKEOFF_BLEND_MIN) * t;
}

u_roll *= blend;
u_pitch *= blend;

int32_t base = (int32_t)g_stab_base_pwm;
float c_lf = (-u_pitch + u_roll);

```

```

float c_rf = (-u_pitch - u_roll);
float c_lb = (u_pitch + u_roll);
float c_rb = (u_pitch - u_roll);

float nav = (float)g_nav_bias * NAV_MIX_GAIN;
switch ((uint8_t)g_last_nav_action)
{
    case NAV_FORWARD: c_lb += nav; c_rb += nav; c_lf -= nav; c_rf -= nav; break;
    case NAV_BACK:    c_lb -= nav; c_rb -= nav; c_lf += nav; c_rf += nav; break;
    case NAV_LEFT:   c_rb += nav; c_rf += nav; c_lb -= nav; c_lf -= nav; break;
    case NAV_RIGHT:  c_lb += nav; c_lf += nav; c_rb -= nav; c_rf -= nav; break;
    default: break;
}

float cgk = CgBiasScale(g_stab_base_pwm);
float pitch_bias = (float)CG_PITCH_BIAS_PWM * cgk;
float roll_bias  = (float)CG_ROLL_BIAS_PWM * cgk;

c_lf += pitch_bias;
c_rf += pitch_bias;
c_lb -= pitch_bias;
c_rb -= pitch_bias;

c_lf += roll_bias;
c_lb += roll_bias;
c_rf -= roll_bias;
c_rb -= roll_bias;

#if YAW_CCW_IS_RF_LB
c_rf += u_yaw_total;
c_lb += u_yaw_total;
c_lf -= u_yaw_total;
c_rb -= u_yaw_total;
#else
c_lf += u_yaw_total;
c_rb += u_yaw_total;
c_rf -= u_yaw_total;
c_lb -= u_yaw_total;
#endif

float k = Mixer_ComputeScale(base, c_lf, c_rf, c_lb, c_rb);
int sat = (k < 0.999f) ? 1 : 0;

if (!sat && g_i_enabled)
{
    i_roll += err_roll * dt;
    i_pitch += err_pitch * dt;
    i_yaw   += yaw_rate_err * dt;

    i_roll = clamp_f(i_roll, -STAB_I_MAX, STAB_I_MAX);
    i_pitch = clamp_f(i_pitch, -STAB_I_MAX, STAB_I_MAX);
    i_yaw   = clamp_f(i_yaw, -STAB_YAW_I_MAX, STAB_YAW_I_MAX);
}
else
{
    if (!g_i_enabled)
    {
        i_roll *= (1.0f - 0.8f * dt);
        i_pitch *= (1.0f - 0.8f * dt);
        i_yaw   *= (1.0f - 0.8f * dt);
    }
}

float mix_lf = k * c_lf;
float mix_rf = k * c_rf;
float mix_lb = k * c_lb;
float mix_rb = k * c_rb;

float trimk = CgBiasScale(g_stab_base_pwm);

float out_lf_f = (float)base + mix_lf * MOTOR_GAIN_LF + MOTOR_OFFSET_LF * trimk;
float out_rf_f = (float)base + mix_rf * MOTOR_GAIN_RF + MOTOR_OFFSET_RF * trimk;
float out_lb_f = (float)base + mix_lb * MOTOR_GAIN_LB + MOTOR_OFFSET_LB * trimk;
float out_rb_f = (float)base + mix_rb * MOTOR_GAIN_RB + MOTOR_OFFSET_RB * trimk;

```

```

int32_t lf = (int32_t)lroundf(out_lf_f);
int32_t rf = (int32_t)lroundf(out_rf_f);
int32_t lb = (int32_t)lroundf(out_lb_f);
int32_t rb = (int32_t)lroundf(out_rb_f);

PwmPayload_t out;
out.motor_lf = clamp_u16(lf, PWM_STAB_FLOOR, PWM_MAX);
out.motor_rf = clamp_u16(rf, PWM_STAB_FLOOR, PWM_MAX);
out.motor_lb = clamp_u16(lb, PWM_STAB_FLOOR, PWM_MAX);
out.motor_rb = clamp_u16(rb, PWM_STAB_FLOOR, PWM_MAX);

SetPwm(&out);
}

```

Jeśli chcesz, mogę też dopisać wariant blendu, który:

- jest **jeszcze delikatniejszy** (np. min 0.15),
- albo który dodatkowo ogranicza **tylko różnicę między silnikami** (anti-pivot stricte), zamiast skaliu **u_roll/u_pitch**.



drone log 2026 02 01 19 21 22

1	39.1709999999846	0	0	S
2	39.1709999999846	0	0	S
3	39.20300000000134	0	0	S
4	39.23400000000038	0	0	S
5	39.2809999999904	0	0	S
6	39.32800000000134	0	0	S
	t_s	base	action	a

```

/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

```

```

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL  9.0f
#define STAB_KD_ROLL  2.5f
#define STAB_KI_ROLL  0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
#define STAB_KI_PITCH 0.55f

#define STAB_KFF_YAW    0.0f

#define STAB_YAW_SIGN   1.0f
#define STAB_ROLL_SIGN  1.0f
#define STAB_PITCH_SIGN -1.0f

#define STAB_U_MAX      2200.0f
#define STAB_I_MAX      2200.0f
#define STAB_YAW_MAX    350.0f
#define STAB_YAW_I_MAX  600.0f

#define STAB_KP_YAW_RATE 1.0f
#define STAB_KI_YAW_RATE 0.15f

#define CMD_TILT_SLEW_DEG_PER_S   80.0f
#define CMD_YAW_SLEW_DPS_PER_S   900.0f
#define BASE_SLEW_PWM_PER_S     12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S 6000.0f

#define IMU_SWAP_ROLL_PITCH 1
#define YAW_MIX_FRONT_CCW_REAR_CW 1
#define YAW_CCW_IS_RF_LB 1

#define MOTOR_GAIN_LF 1.000f
#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.070f
#define MOTOR_GAIN_RB 1.030f

#define MOTOR_OFFSETS_LF 0.0f
#define MOTOR_OFFSETS_LB 0.0f
#define MOTOR_OFFSETS_RF 650.0f
#define MOTOR_OFFSETS_RB 300.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG +0.0f
#define PITCH_TRIM_DEG +0.0f

#define LEVEL_CALIB_SAMPLES    200u

```

```
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM -200
#define CG_ROLL_BIAS_PWM -140

#define CG_BIAS_START_PWM 1400u
#define CG_BIAS_FULL_PWM 3200u

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u

#define NAV_TILT_DEG 10.0f
#define NAV_YAW_RATE_DPS 180.0f
#define NAV_THRUST_BIAS 0
#define NAV_MIX_GAIN 1.0f

#define TAKEOFF_BLEND_START 1800.0f
#define TAKEOFF_BLEND_FULL 4200.0f
/* USER CODE END PD */
```

Znowu pochylił mi się do przodu

Tak — log **potwierdza**, że z tym zestawem parametrów znowu masz wyraźny „przód” (u Ciebie to wychodzi jako **pitch dodatni**) i potem to przechodzi w **runaway roll** i tilt-kill.

Co konkretnie widać w `drone_log_2026-02-01_19-21-22.csv`

- W „w miarę stabilnym” fragmencie (base > ~3000, zanim zacznie się dramat) masz średnio:
 - **pitch ≈ +5.5°** (to jest dużo jak na STOP-hover kilka cm)
 - **roll ≈ ~0°** (tu jest w miarę ok)
- Co ważniejsze: w tym samym fragmencie średnio **tył ma wyższe PWM niż przód**:
 - **front_avg - rear_avg ≈ -265 PWM** (czyli **rear > front**)
- A potem dwa razy robisz identyczny schemat:
 - roll leci w dół do okolic **-39...-41°** (np. ~56.76 s i ~67.00 s) i wchodzi tilt-kill.

Dlaczego to się stało (najprościej)

U Ciebie `CG_PITCH_BIAS_PWM = -200` w praktyce **pompuje tył**, bo w mikserze masz:

```
c

c_lf += pitch_bias;
c_rf += pitch_bias;
c_lb -= pitch_bias;
c_rb -= pitch_bias;
```

Gdy **pitch_bias** jest **ujemny**, to:

- LF/RF dostają **mniej**
- LB/RB dostają **więcej**

A log już pokazuje, że **tył i tak jest wyżej niż przód** o ~265 PWM → więc ten bias **dokłada dokładnie w złą stronę** i „przód” robi się jeszcze bardziej nieunikniony.

Co zmienić teraz (jedna, konkretna korekta)

Odwróć znak pitch-biasu:

```
c

#define CG_PITCH_BIAS_PWM    (+200)
```

Zostaw **CG_ROLL_BIAS_PWM = -140** na razie (bo roll średnio jest blisko zera – to nie jest Twoje główne źródło „przodu”).

Jeśli po tym „przód” będzie za mały/za duży:

- nadal nurkuje → podbij do **+260 / +320**
- zaczyna ciągnąć „w tył” → zejdź do **+120 / +80**

Kroki rób po **60-100**.

Dodatkowo (opcjonalnie, ale sensowne)

Jeśli masz teraz takeoff-blend z minimalnym 0.25, a dron nadal potrafi zrobić pivot przy odklejaniu, to możesz dać trochę większą “kontrolę” na dole:

- ustaw **TAKEOFF_BLEND_MIN** na **0.35** zamiast 0.25
(żeby stabilizacja miała więcej siły zanim całkiem równo oderwie się od ziemi)
-

Jeśli wrzucisz log po samej zmianie **CG_PITCH_BIAS_PWM** na **+200** (reszta bez zmian), to Ci powiem z liczb:

- czy **pitch_mean** spadł z ~+5.5° w okolice 0...±2°
- i czy zniknęły te szybkie ucieczki roll do kill.



drone log 2026 02 01 19 29 12

1	5.922000000002299	0	0	S
2	5.922000000002299	0	0	S
3	5.953000000001339	0	0	S
4	5.985000000000582	0	0	S
5	6.0309999999904	0	0	S
6	6.078000000001339	0	0	S
	t_s	base	action	a

```
/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL  9.0f
#define STAB_KD_ROLL  2.5f
#define STAB_KI_ROLL  0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
#define STAB_KI_PITCH 0.55f

#define STAB_KFF_YAW   0.0f

#define STAB_YAW_SIGN  1.0f
#define STAB_ROLL_SIGN 1.0f
#define STAB_PITCH_SIGN -1.0f

#define STAB_U_MAX     2200.0f
#define STAB_I_MAX     2200.0f
#define STAB_YAW_MAX   350.0f
```

```
#define STAB_YAW_I_MAX      600.0f

#define STAB_KP_YAW_RATE   1.0f
#define STAB_KI_YAW_RATE   0.15f

#define CMD_TILT_SLEW_DEG_PER_S    80.0f
#define CMD_YAW_SLEW_DPS_PER_S    900.0f
#define BASE_SLEW_PWM_PER_S     12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S  6000.0f

#define IMU_SWAP_ROLL_PITCH  1
#define YAW_MIX_FRONT_CCW_REAR_CW 1
#define YAW_CCW_IS_RF_LB  1

#define MOTOR_GAIN_LF  1.000f
#define MOTOR_GAIN_LB  1.000f
#define MOTOR_GAIN_RF  1.070f
#define MOTOR_GAIN_RB  1.030f

#define MOTOR_OFFSETS_LF  0.0f
#define MOTOR_OFFSETS_LB  0.0f
#define MOTOR_OFFSETS_RF  650.0f
#define MOTOR_OFFSETS_RB  300.0f

#define YAW_TRIM  0.0f

#define ROLL_TRIM_DEG +0.0f
#define PITCH_TRIM_DEG +0.0f

#define LEVEL_CALIB_SAMPLES    200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM  200
#define CG_ROLL_BIAS_PWM -140

#define CG_BIAS_START_PWM  1400u
#define CG_BIAS_FULL_PWM  3200u

#define I_ENABLE_BASE_PWM  900u
#define I_DISABLE_BASE_PWM 700u

#define NAV_TILT_DEG       10.0f
#define NAV_YAW_RATE_DPS  180.0f
#define NAV_THRUST_BIAS    0
#define NAV_MIX_GAIN      1.0f

#define TAKEOFF_BLEND_START_PWM 1800.0f
#define TAKEOFF_BLEND_FULL_PWM 4200.0f
#define TAKEOFF_BLEND_MIN    0.35f
/* USER CODE END PD */
```

Okej, łap ten log i zweryfikuj, co się wydarzyło

Tak — z tego logu **wynika dokładnie, że “przód” robi Ci się od samego biasowania**, a nie dlatego, że PID “źle liczy”. I co gorsza: przy tych wartościach **CG_PITCH=+200 i CG_ROLL=-140 robisz sobie ogólną stałą różnicę PWM między rogami**, przez co przy odklejaniu powstaje pivot (typowo wokół RF/LB), a potem pitch leci w kosmos i łąpie tilt-kill.

Co pokazuje log drone_log_2026-02-01_19-29-12.csv

1) Dwa razy wchodzisz w tilt-kill przez PITCH (przód)

Masz dwa wyraźne zdarzenia “PWM>0 → nagle wszystko 0” przy **base ~7200**, gdzie **pitch skacze do ~50°**:

- około **t≈32.28s**: pitch **~49.9°** → PWM = 0
- około **t≈49.92s**: pitch **~48.7°** → PWM = 0

To jest wprost “pochylił się do przodu i wyłączyło”.

2) Stały bias PWM mówi, że przód+prawo jest sztucznie dopompowane

W zakresie, gdzie powinno się dać już “hover” (base 3000–6000), średnie (**PWM - base**) są takie:

- **LF ≈ +74**
- **RF ≈ +1022**
- **LB ≈ -354**
- **RB ≈ +220**

To jest gigantyczna, **stała** asymetria: RF dostaje ponad +1000 względem base, a LB jest dławione ~ -350 .

I to w 99% pochodzi z Twoich biasów CG:

- **CG_PITCH_BIAS_PWM = +200** w Twoim mikserze **odejmuje** z tyłu (LB/RB) i **dodaje** na przodzie (LF/RF)
- **CG_ROLL_BIAS_PWM = -140** **odejmuje** z lewej (LF/LB) i **dodaje** na prawej (RF/RB)

Suma dla każdego rogu (przy $cgk \approx 1$) wychodzi mniej więcej:

- **LF: +200 -140 = +60** (pasuje do $\sim +74$)
- **RF: +200 +140 = +340** (a potem jeszcze dochodzi **MOTOR_OFFSET_RF=650** → robi się $\sim +1000$)
- **LB: -200 -140 = -340** (pasuje do ~ -354)
- **RB: -200 +140 = -60** (a potem **MOTOR_OFFSET_RB=300** → robi się +220)

Czyli: **to nie PID robi “przód” — to stałe biasy ustawiają Ci geometrię thrustu tak, że LB jest przyduszone, RF dopompowane, i na ziemi zaczyna się pivot.**

Co zmienić teraz (konkretna korekta, żeby przestało nurkować na startie)

Krok 1: wyłącz oba CG biasy na czas startu/hover

Na najbliższy test ustaw:

```
c

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0
```

Masz już motor-gainy i motor-offsety — one powinny robić kompensację “mocy silników”. CG biasy w tej chwili są po prostu za agresywne i psują start.

Krok 2: zmniejsz trochę RF offset (bo RF jest przewalone)

Skoro po wyłączeniu CG biasów RF dalej będzie najwyższej, to od razu zrób mały krok:

```
c

#define MOTOR_OFFSET_RF 450.0f // było 650
#define MOTOR_OFFSET_RB 300.0f // zostaw
```

Gainy możesz zostawić jak masz (1.07 / 1.03), na razie.

Krok 3: blend zostaw (to akurat jest dobry ruch)

`TAKEOFF_BLEND_MIN = 0.35` jest sensowne — to ogranicza szarpanie różnicą PWM na ziemi.

Co powinieneś zobaczyć po tej zmianie (w logu)

W base 3000–6000:

- (`pwm_lf-base`), (`pwm_lb-base`) powinny być podobne ($\pm 100\ldots 200$)
- RF nadal może być wyżej, ale nie o +1000 — raczej kilkaset
- pitch nie powinien “narastać w czasie” aż do kilkudziesięciu stopni

Dlaczego nie “dokładać dalej CG_PITCH”, tylko na chwilę go wyłączyć?

Bo w tym logu CG biasy **dominują** nad tym, co robi regulator i motor-offsety — one ustawiają Ci “pochylenie bazowe” jeszcze zanim PID zdąży zareagować. Najpierw musimy mieć start, gdzie dron nie próbuje od razu robić pivotu na ziemi.

Jeśli wrzucisz log po: **CG_PITCH=0, CG_ROLL=0, MOTOR_OFFSET_RF=450** (reszta bez zmian), to ja Ci powiem już bardzo precyźnie:

- czy “przód” dalej wynika z thrust-asymetrii czy z osi/regulatora,

- i ile dokładnie dać na **MOTOR_OFFSET_RF/RB**, żeby wyrównać bez powrotu do pivotu.



drone log 2026 02 01 19 41 46

1	81.984000000000038	0	0	S
2	81.984000000000038	0	0	S
3	82.0	0	0	S
4	82.0469999999866	0	0	S
5	82.09400000000096	0	0	S
6	82.1559999999904	0	0	S
	t_s	base	action	a

```
/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN      0.40f
#define ACC_G_MAX      2.00f

#define TILT_KILL_DEG   35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

#define STAB_KP_ROLL  9.0f
#define STAB_KD_ROLL  2.5f
#define STAB_KI_ROLL  0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
#define STAB_KI_PITCH 0.55f

#define STAB_KFF_YAW   0.0f

#define STAB_YAW_SIGN  1.0f
```

```

#define STAB_ROLL_SIGN 1.0f
#define STAB_PITCH_SIGN -1.0f

#define STAB_U_MAX      2200.0f
#define STAB_I_MAX       2200.0f
#define STAB_YAW_MAX     350.0f
#define STAB_YAW_I_MAX   600.0f

#define STAB_KP_YAW_RATE 1.0f
#define STAB_KI_YAW_RATE 0.15f

#define CMD_TILT_SLEW_DEG_PER_S 80.0f
#define CMD_YAW_SLEW_DPS_PER_S 900.0f
#define BASE_SLEW_PWM_PER_S 12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S 6000.0f

#define IMU_SWAP_ROLL_PITCH 1
#define YAW_MIX_FRONT_CCW_REAR_CW 1
#define YAW_CCW_IS_RF_LB 1

#define MOTOR_GAIN_LF 1.000f
#define MOTOR_GAIN_LB 1.000f
#define MOTOR_GAIN_RF 1.070f
#define MOTOR_GAIN_RB 1.030f

#define MOTOR_OFFSETS_LF 0.0f
#define MOTOR_OFFSETS_LB 0.0f
#define MOTOR_OFFSETS_RF 450.0f
#define MOTOR_OFFSETS_RB 300.0f

#define YAW_TRIM 0.0f

#define ROLL_TRIM_DEG +0.0f
#define PITCH_TRIM_DEG +0.0f

#define LEVEL_CALIB_SAMPLES 200u
#define LEVEL_CALIB_GYRO_MAX_DPS 2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

#define CG_PITCH_BIAS_PWM 0
#define CG_ROLL_BIAS_PWM 0

#define CG_BIAS_START_PWM 1400u
#define CG_BIAS_FULL_PWM 3200u

#define I_ENABLE_BASE_PWM 900u
#define I_DISABLE_BASE_PWM 700u

#define NAV_TILT_DEG 10.0f
#define NAV_YAW_RATE_DPS 180.0f
#define NAV_THRUST_BIAS 0

```

```
#define NAV_MIX_GAIN      1.0f

#define TAKEOFF_BLEND_START_PWM 1800.0f
#define TAKEOFF_BLEND_FULL_PWM  4200.0f
#define TAKEOFF_BLEND_MIN      0.35f
/* USER CODE END PD */

Okej, co się teraz stało?
```

Sprawdziłem log `drone_log_2026-02-01_19-41-46.csv` — i tu jest jasne, **co się stało** po wyzerowaniu CG biasów i zmniejszeniu `MOTOR_OFFSET_RF`.

Co mówi log (konkretnie)

1) Nie zabił Cię “przód” — zabił Cię ROLL

Jest jedno wyraźne zdarzenie tilt-kill:

- **$t \approx 106.625$ s, $base \approx 6560$**
- **roll = -38.59°** (czyli przekroczenie 35°)
- **pitch = $+1.33^\circ$** (małe – nie to było przyczyną)
- zaraz potem wszystkie PWM lecą na **0** (kill)

To jest dokładnie schemat “pivot na ziemi → roll runaway → kill”.

2) Co robiły silniki tuż przed kill

W ostatniej próbce przed odcięciem (jeszcze pracuje):

- **LF ≈ 7703**
- **RF ≈ 7728**
- **LB ≈ 6163** (najniżej)
- **RB ≈ 6367**

Czyli sterownik próbuje “ratować” sytuację różnicą po przekątnej, ale roll i tak ucieka. To wygląda jak sytuacja, gdzie **jeden róg nadal trzyma tarcie o podłożę** i zamiast się wyrównać w powietrzu, dron robi obrót/pivot.

3) Nadal masz stały “przód” w danych (ale dużo mniejszy niż wcześniej)

W zakresie base **3000–6000** (jeszcze przed dramatem):

- średnio **pitch ≈ $+4.34^\circ$**
- średnio **roll ≈ -0.48°**

Czyli przód dalej trochę “ciągnie”, ale w tym logu **nie on zabił lot** (kill poszedł od roll).

4) Prawa strona dalej dostaje więcej PWM (RF/RB)

W tym samym zakresie base 3000–6000 średnie (PWM - base):

- LF ≈ -50
- LB ≈ -37
- RF ≈ +456
- RB ≈ +336

To dalej wspiera Twoją obserwację “kręci/pivotuje w okolicy RF” — bo prawa strona jest stale dopompowana.

Wniosek “co się teraz stało”

Po wyłączeniu CG biasów **zniknęło Ci to brutalne wymuszanie przodu/tyłu**, ale dalej masz:

1. **lekki stały pitch bias** (ok. +4°), oraz
2. **pivot przy odklejaniu**, który kończy się runaway **roll** i tilt-kill.

To pivot jest teraz największym problemem, nie same PID-y.

Co bym zrobił jako następny krok (żeby przestał pivotować na ziemi)

Masz już blend, ale on skaluje tylko **u_roll/u_pitch**. Przy pivotowaniu często trzeba jeszcze **twardo ograniczyć różnice PWM na ziemi**.

Opcja A: jeszcze mocniej ogranicz korekty przy ziemi (najprostsza)

Zmień:

- TAKEOFF_BLEND_MIN z **0.35** na **0.20**
- i ewentualnie podnieś FULL na **4800** (żeby pełna stabilizacja wchodziła później)

To zmniejszy różnicowanie silników zanim całość faktycznie wisi.

Opcja B: “delta limit” — limit różnicy między silnikami dopóki base jest niskie (najskuteczniejsze)

Po policzeniu **lf/rf/lb/rb** (int32, przed clamp) dodaj:

- jeśli **base < TAKEOFF_BLEND_FULL_PWM**, to **maksymalna różnica** między największym i najmniejszym PWM nie może przekroczyć np. **DELTA = 900** (na start).
- Jak przekracza, to “ściągasz” max w dół i min w górę symetrycznie.

To potrafi zabić pivot praktycznie od razu.

Dodatkowo: pitch bias +4° jest nadal widoczny

To nie jest sprawa CG biasów (masz 0), tylko najczęściej:

- kalibracji poziomu (`g_level_*_off`) robionej gdy dron nie leżała idealnie równo,
- albo stałej asymetrii thrust/CG.

Jeśli chcesz to złapać "parametrem", najbezpieczniej jest wrócić do **małego CG_PITCH_BIAS_PWM** (np. ±60), ale dopiero po ogarnięciu pivotu — bo inaczej znowu będziesz dokładał moment na ziemi.

Jeśli chcesz, to wkleję Ci gotowy fragment **delta-limit** (w Twoim stylu, bez zmiany mapowania ani signów) dokładnie w miejsce przed `clamp_u16()`. To jest jedna z tych zmian, które naprawdę robią różnicę przy "podrywa się nierówno i robi pivot" na brushedach 8520.



Okej, może teraz zamiast pytać o każdy test daj mi znać co robić w jakiej sytuacji. Jak się przechyla do przodu co zwiększać. Jak się przechyla do prawej co zwiększać. Jak się kręci wokół jednego konkretnego silnika to co zwiększać

Jasne — poniżej masz **ściągę "co ruszać gdy..."** dla Twojego miksu (LF/RF/LB/RB, Twoje SIGN-y i `IMU_SWAP_ROLL_PITCH=1` zostają). To jest podejście "diagnostyka → jeden pokrętło → oczekiwany efekt", żebyś nie musiał pytać po każdym teście.

Zasada 0: najpierw rozróżnij 2 przypadki

A) Problem pojawia się jeszcze zanim się odklei (pivot na ziemi)

Objawy:

- jeden róg się podnosi, inny "trzyma tarcie", dron się obraca/wywraca zanim wisie.

Wtedy **nie dokręcaj PID**, tylko:

1. zwiększ `TAKEOFF_BLEND_MIN` lub
2. dodaj **limit różnicy PWM** przy niskim base (delta-limit),
3. dopiero potem stroisz biasy/gainy.

B) Problem pojawia się dopiero jak już wisie (2-10 cm nad ziemią)

Wtedy działa strojenie biasów / gainów / trimów, bo dron jest w powietrzu i miks ma sens.

1) "Przechyla się do przodu" (nose down) — co zwiększać?

U Ciebie "przód/tyle" sterujesz głównie **CG_PITCH_BIAS_PWM** (bo to daje stały moment pitch zależny od base przez **cgk**).

Masz w mikserze:

- **CG_PITCH_BIAS_PWM > 0** → **więcej na przód (LF/RF), mniej na tył (LB/RB)**
- **CG_PITCH_BIAS_PWM < 0** → **więcej na tył, mniej na przód**

Jeśli nurkuje do przodu (nose down):

→ zmniejsz **CG_PITCH_BIAS_PWM** (czyli idź w stronę **ujemną**)

Przykład kroków:

- 0 → -60 → -120 → -180 (kroki po 60)

Jeśli ciągnie w tył (nose up):

→ zwiększ **CG_PITCH_BIAS_PWM** (w stronę dodatnią)

2) "Przechyla się w prawo" — co zwiększać?

Prawo/lewo robisz **CG_ROLL_BIAS_PWM**.

Masz:

- **CG_ROLL_BIAS_PWM > 0** → **więcej na lewą stronę (LF/LB), mniej na prawą**
- **CG_ROLL_BIAS_PWM < 0** → **więcej na prawą stronę (RF/RB), mniej na lewą**

Jeśli przechyla się w prawo (prawa strona siada):

→ zmniejsz **CG_ROLL_BIAS_PWM** (czyli bardziej **ujemny**)

Kroki:

- 0 → -60 → -120 → -180

Jeśli przechyla się w lewo:

→ zwiększ **CG_ROLL_BIAS_PWM** (bardziej dodatni)

3) "Kręci się (yaw) sam z siebie w STOP" — co zwiększać?

To nie CG. To jest **yaw regulator**.

- Jeśli obraca się CCW/CW w STOP:
 - zwiększ **STAB_KP_YAW_RATE** (najpierw)
 - jeśli nadal dryfuje wolno: zwiększ **STAB_KI_YAW_RATE** (bardzo małe kroki)

Przykład:

- **KP_YAW**: 1.0 → 1.4 → 1.8
- **KI_YAW**: 0.15 → 0.22 → 0.30

Jeśli zacznie "myszkować" (szybkie oscylacje yaw), to KP jest za duże.

4) "Kręci się wokół konkretnego silnika" (pivot wokół rogu) — co to znaczy i co ruszać?

To jest najczęściej **ziemia + nierówne odklejenie**, nie klasyczna stabilizacja. Pivot wokół rogu oznacza zwykle:

- ten róg jest **najtwardszym punktem kontaktu** (tarcie) albo
- ten róg jest **najtwardszy/cięższy** i zostaje na ziemi, gdy reszta już ciągnie.

Najpierw (zawsze) anti-pivot:

- ➡ **TAKEOFF_BLEND_MIN** w górę jeśli pivot robi się od razu (np. 0.35 → 0.45 → 0.55) albo jeszcze lepiej:
- ➡ **delta-limit**: limit maksymalnej różnicy PWM między silnikami przy niskim base.

Dopiero gdy pivot zniknie, stroisz "moc rogu".

Jeśli mimo anti-pivot w powietrzu dalej "ciągnie" w stronę rogu:

Wtedy to jest realna asymetria thrust i robisz to "na rogi":

- **pivot / ucieka w stronę RF** → zwiększasz RF (albo prawą stronę)
 - **MOTOR_OFFSET_RF** +100
 - ewentualnie **MOTOR_GAIN_RF** +0.02
- **pivot / ucieka w stronę LF** → zwiększasz LF
 - **MOTOR_OFFSET_LF** +100 (u Ciebie masz 0, można dać +100...+300)
 - **MOTOR_GAIN_LF** +0.02 (rzadziej)
- **pivot / ucieka w stronę RB** → zwiększasz RB
 - **MOTOR_OFFSET_RB** +100
 - **MOTOR_GAIN_RB** +0.02
- **pivot / ucieka w stronę LB** → zwiększasz LB
 - **MOTOR_OFFSET_LB** +100
 - **MOTOR_GAIN_LB** +0.02

Zasada: najpierw offsety (setki), dopiero potem gainy (setne).

Gainy ruszaj małymi krokami, bo one wpływają też na zachowanie podczas korekcji.

5) Jak poznać czy to "CG/bias" czy "słaby silnik/zasilanie"

To jest super proste po logach:

Jeśli potrzebujesz stale większego PWM na danym rogu już od niskich base:

- ➡ to zwykle **CG/bias** (albo offset level/trim).

Jeśli różnice PWM robią się dopiero przy dużym base (np. >6000):

- ➡ to często **spadek napięcia / tranzystor / ścieżka / słabnący silnik pod obciążeniem.**

Wtedy bardziej pomaga:

- **MOTOR_GAIN_*** (bo rośnie z mixem),
 - albo rozwiązanie sprzętowe (grubsze ścieżki, lepszy MOSFET, krótsze przewody).
-

6) Gotowy "drzewkowy" schemat decyzji (bez filozofii)

1. Pivot na ziemi? (przed odklejeniem)

- TAK → podnieś **TAKEOFF_BLEND_MIN** albo dodaj delta-limit → dopiero potem reszta.
- NIE → przejdź dalej.

2. Nose down/up?

- nose down → **CG_PITCH_BIAS_PWM** w stronę **ujemną**
- nose up → w stronę **dodatnią**

3. Prawo/lewo?

- w prawo → **CG_ROLL_BIAS_PWM** bardziej **ujemny**
- w lewo → bardziej **dodatni**

4. Yaw drift w STOP?

- zwiększ **STAB_KP_YAW_RATE**, potem **STAB_KI_YAW_RATE**

5. Ucieka w stronę konkretnego rogu w powietrzu?

- zwiększa **MOTOR_OFFSETS_<ten róg>** o +100, obserwuj
- jeśli dalej → +0.02 do **MOTOR_GAIN_<ten róg>**

Jeśli chcesz, wkleję Ci od razu gotowy **delta-limit** do Twojego **ControlStep_Stabilize()** (3-4 linijki logiki po wyliczeniu lf/rf/lb/rb), bo to jest najczęstsza rzecz, która kończy "pivot wokół rogu przy odklejaniu".



Okej, a krótsza instrukcja?

Jasne — **krótka ściąga** (dla Twojego miksu):

1) Przechyla się do przodu / nurkuje

→ zmniejsz CG_PITCH_BIAS_PWM (idź w stronę ujemną)

Kroki: 0 → -60 → -120 → -180

2) Przechyla się do tytu

→ zwiększ CG_PITCH_BIAS_PWM (w stronę dodatnią)

0 → +60 → +120

3) Przechyla się w prawo (prawa strona siada)

→ zmniejsz CG_ROLL_BIAS_PWM (bardziej ujemny)

0 → -60 → -120

4) Przechyla się w lewo

→ zwiększ CG_ROLL_BIAS_PWM (bardziej dodatni)

0 → +60 → +120

5) Kręci się w yaw w STOP (sam obrót CW/CCW)

→ zwiększa STAB_KP_YAW_RATE (np. 1.0 → 1.4 → 1.8)

Jeśli dalej powoli dryfuje: → zwiększa STAB_KI_YAW_RATE (0.15 → 0.22 → 0.30)

6) Pivot / kręci się wokół jednego rogu przy odklejaniu (ziemia)

→ najpierw anti-pivot, nie PID:

- podnieś TAKEOFF_BLEND_MIN (0.35 → 0.45 → 0.55)
albo (lepsze) dodaj limit różnicy PWM na dole.

Dopiero jeśli w powietrzu nadal ucieka w stronę rogu:

→ podbij MOTOR_OFFSET_<ten róg> o +100 (max parę kroków)

a jak trzeba dalej: → MOTOR_GAIN_<ten róg> +0.02

To wszystko. Jeśli chcesz, napiszę Ci 5-liniowy "delta-limit" do wklejenia (najlepszy na pivot przy startie).



Okej, daj mi proszę wszystkie parametry user code begin PD takie startowe, które mogę sobie teraz stroić zgodnie z tym co obserwuję

Pewnie — poniżej masz **startowy zestaw /* USER CODE BEGIN PD */** taki, żeby:

- dało się spokojnie robić testy “STOP hover kilka cm”,
- nic nie było “na pałę” dopompowane (CG=0 na start),
- yaw trzymał w miarę stabilnie,
- był anti-pivot (blend),
- i żebyś mógł stroić wg Twojej ściągi (CG_PITCH/CG_ROLL, OFFS/GAIN, yaw PID).

To jest baza do strojenia, nie “docelowe złote ustawienia”.

```
c

/* USER CODE BEGIN PD */
#define DEG2RAD 0.0174532925f
#define RAD2DEG 57.2957795f

#define IMU_UPDATE_PERIOD_MS 10u

#define ACC_LPF_CUTOFF_HZ 20.0f
#define ACC_G_MIN 0.40f
#define ACC_G_MAX 2.00f

#define TILT_KILL_DEG 35.0f
#define TILT_UNKILL_DEG 25.0f
#define TILT_KILL_DEBOUNCE_MS 50u
#define TILT_UNKILL_HOLD_MS 300u

#define PWM_MIN 0
#define PWM_MAX 10000
#define PWM_STAB_FLOOR 0

// =====
// STAB PID (startowe pod hover test)
// =====
#define STAB_KP_ROLL 9.0f
#define STAB_KD_ROLL 2.5f
#define STAB_KI_ROLL 0.55f

#define STAB_KP_PITCH 9.0f
#define STAB_KD_PITCH 2.5f
#define STAB_KI_PITCH 0.55f

#define STAB_U_MAX 2200.0f
#define STAB_I_MAX 2200.0f

// =====
// YAW HOLD (żeby STOP nie kręcił)
// =====
#define STAB_KFF_YAW 0.0f
#define STAB_YAW_SIGN 1.0f
#define STAB_YAW_MAX 350.0f
#define STAB_YAW_I_MAX 600.0f
#define STAB_KP_YAW_RATE 1.2f
#define STAB_KI_YAW_RATE 0.20f

// =====
// Twoje osie / znaki (zostaw)
// =====
#define IMU_SWAP_ROLL_PITCH 1
#define STAB_ROLL_SIGN 1.0f
#define STAB_PITCH_SIGN -1.0f

#define YAW_CCW_IS_RF_LB 1
#define YAW_MIX_FRONT_CCW_REAR_CW 1

// =====
// Slew / reakcja na komendy
```

```

// =====
#define CMD_TILT_SLEW_DEG_PER_S      80.0f
#define CMD_YAW_SLEW_DPS_PER_S       900.0f
#define BASE_SLEW_PWM_PER_S          12000.0f
#define NAV_BIAS_SLEW_PWM_PER_S      6000.0f

// =====
// Motor kompensacje (stroisz OFFS/GAIN)
// =====
#define MOTOR_GAIN_LF    1.000f
#define MOTOR_GAIN_LB    1.000f
#define MOTOR_GAIN_RF    1.060f
#define MOTOR_GAIN_RB    1.030f

#define MOTOR_OFFSET_LF   0.0f
#define MOTOR_OFFSET_LB   0.0f
#define MOTOR_OFFSET_RF   450.0f
#define MOTOR_OFFSET_RB   300.0f

// =====
// Trims (na start 0, stroisz dopiero po level-calib)
// =====
#define YAW_TRIM         0.0f
#define ROLL_TRIM_DEG    0.0f
#define PITCH_TRIM_DEG   0.0f

// =====
// Level calibracja
// =====
#define LEVEL_CALIB_SAMPLES        200u
#define LEVEL_CALIB_GYRO_MAX_DPS  2.5f
#define LEVEL_CALIB_ACC_OK_REQUIRED 1

// =====
// CG biasy (NA START = 0, potem stroisz wg ściągnięcia)
// =====
#define CG_PITCH_BIAS_PWM     0
#define CG_ROLL_BIAS_PWM      0

// kiedy biasy/offsety mają się "włączać" (dla trimk/cgk)
#define CG_BIAS_START_PWM    1400u
#define CG_BIAS_FULL_PWM     3200u

// =====
// Integrator enable (żeby łapał stały błąd po oderwaniu)
// =====
#define I_ENABLE_BASE_PWM    900u
#define I_DISABLE_BASE_PWM   700u

// =====
// NAV (na testach hover i tak używaj STOP)
// =====
#define NAV_TILT_DEG         10.0f
#define NAV_YAW_RATE_DPS     180.0f
#define NAV_THRUST_BIAS      0
#define NAV_MIX_GAIN          1.0f

// =====
// Anti-pivot takeoff blend
// =====
#define TAKEOFF_BLEND_START_PWM 1800.0f
#define TAKEOFF_BLEND_FULL_PWM 4200.0f
#define TAKEOFF_BLEND_MIN     0.35f
/* USER CODE END PD */

```

Co teraz stroisz (w tej kolejności)

1. **TAKEOFF_BLEND_MIN** jeśli pivot na ziemi: **0.35 → 0.45 → 0.55**
2. **CG_PITCH_BIAS_PWM / CG_ROLL_BIAS_PWM** (kroki po 60-100) gdy już wisi kilka cm

3. **MOTOR_OFFSET_*** (kroki po 100) jeśli ciągle ucieka w stronę konkretnego rogu

4. **Yaw: STAB_KP_YAW_RATE**, potem **STAB_KI_YAW_RATE** jeśli dryfuje

Jak chcesz, to dorzucę też "startowy delta-limit" (5 linijek), bo to często lepsze niż samo blend, ale powyższy blok jest już gotową bazą do samodzielnego strojenia wg obserwacji.



Okej warto wspomnieć, mam IMU domyślnie przekrzywione o -1.65 roll, pitch o +1.9. Mam coś z tym zrobić?

Tak — warto i **zdecydowanie zróć z tym porządek**, bo te ~2° "na stałe" potrafią Ci robić dokładnie to, co widzisz: przy startie na ziemi kontroler już "walczy", robi różnicę PWM, pojawia się pivot i potem runaway.

Masz dwa sensowne podejścia. Najlepsze to A, a B jest szybkim obejściem.

A) Najlepsze: popraw "level offset" (Twoje g_level_*_off)

Ty już masz automatyczną kalibrację:

- zbiera **roll** i **pitch** gdy **nieuzbrojony** i żyroskopy są spokojne,
- zapisuje do **g_level_roll_off** / **g_level_pitch_off**.

Jeśli mimo tego stale widzisz **roll ≈ -1.65°**, **pitch ≈ +1.9°**, to znaczy najczęściej:

- kalibracja odbyła się, gdy dron **nie leżał idealnie równo** (stół, frame, pianka, krzywa powierzchnia),
- albo IMU jest mechanicznie przekrzywione i **chcesz ten offset kompensować** programowo.

Co zrobić:

1. Położ drona na możliwie poziomej, twardej powierzchni.
2. Włącz, nie ruszaj go przez parę sekund aż skończy się Twoje **LEVEL_CALIB_SAMPLES**.
3. Sprawdź, czy po kalibracji telemetria w spoczynku jest blisko **0/0**.

Jeśli po porządnym "level-calib" nadal masz te same wartości, to przejdź do B.

B) Szybkie i pewne: ustaw TRIM-y tak, żeby target był "taki jak IMU"

W Twoim kodzie TRIM działa w *zadaniu kąta*, czyli ustawia "jakie 0° chcesz mieć". To jest idealne do skompensowania stałego montażowego odchyłu IMU.

Masz **IMU_SWAP_ROLL_PITCH = 1** oraz **STAB_PITCH_SIGN = -1**, więc najsensowniej zrobić to **empirycznie prostą zasadą**:

- Jeśli w spoczynku (STOP) widzisz:

- `roll = -1.65°`
- `pitch = +1.9°`

To ustaw:

```
c

#define ROLL_TRIM_DEG  (+1.9f)
#define PITCH_TRIM_DEG (+1.65f)
```

Dlaczego tak "na krzyż"? Bo przy `IMU_SWAP_ROLL_PITCH=1` Twój regulator traktuje `g_pitch_deg` jako `roll_meas`, a `g_roll_deg` jako `pitch_meas`. Te TRIM-y mają wyzerować błąd przy cmd=0.

Co powinieneś zobaczyć po tym:

- w spoczynku, w STOP, przy pracujących silnikach (ale bez odklejania), regulator przestaje "pchać" w jedną stronę,
- w logu średnie roll/pitch bliżej 0.

Co wybrać?

- Jeśli IMU jest zamontowane krzywo **mechanicznie** i tak już zostanie → **B (TRIM)** jest praktyczne i wystarczające.
- Jeśli chcesz "czysto" i żeby trymy były 0 → **A (level-calib na poziomie)** i ewentualnie przymocowanie IMU prosto.

Mini-uwaga praktyczna (ważna przy Twoich objawach)

Te 1–2° offsetu to niby mało, ale przy startie na ziemi to potrafi wywołać **stałą różnicę PWM** (bo P-regulator od razu reaguje), a to właśnie uruchamia pivot. Więc tak: **warto to wyzerować**.

Jeśli chcesz, podeślę Ci też "twardą" metodę: automatyczne ustawienie `ROLL_TRIM_DEG/PITCH_TRIM_DEG` na podstawie średniej z pierwszych 2–3 sekund po uzbrojeniu (tylko gdy cmd=0), ale powyższe dwa podejścia zwykle wystarczają.



Okej, poproszę jeszcze raz instrukcję co teraz stroić w tych define'ach. Co jak dron do przodu / tyłu się przekyla; co jak do prawej / lewej się przekyla; co jak się kręci wokół jednego z silników; co jak cały się obraca w jakimś kierunku

Jasne — oto **instrukcja 1:1 "co stroić w define'ach"** dla Twojego kodu.

0) Najpierw: IMU offset (żeby nie walczył już na ziemi)

Jeśli w spoczynku masz np. roll $\approx -1.65^\circ$, pitch $\approx +1.9^\circ$:

- ustaw trymy tak, żeby kontroler celował w "prawdziwe 0":

```
c

#define ROLL_TRIM_DEG  (+1.9f)
#define PITCH_TRIM_DEG (+1.65f)
```

To jest baza — bez tego bardzo łatwo o pivot przy startie.

1) Dron przechyla się DO PRZODU / DO TYŁU (nose down / nose up)

Stroisz: **CG_PITCH_BIAS_PWM**

Masz w miksie:

- ujemne CG_PITCH_BIAS_PWM** → więcej ciągu **tył (LB/RB)**
- dodatnie CG_PITCH_BIAS_PWM** → więcej ciągu **przód (LF/RF)**

Jeśli nurkuje do przodu (nose down):

→ dawaj **CG_PITCH_BIAS_PWM** bardziej **ujemne**

Kroki: $0 \rightarrow -60 \rightarrow -120 \rightarrow -180 \rightarrow -240$

Jeśli ciągnie w tył (nose up):

→ dawaj **CG_PITCH_BIAS_PWM** bardziej **dodatnie**

$0 \rightarrow +60 \rightarrow +120 \rightarrow +180$

2) Dron przechyla się W PRAWO / W LEWO

Stroisz: **CG_ROLL_BIAS_PWM**

Masz:

- ujemne CG_ROLL_BIAS_PWM** → więcej ciągu **prawa strona (RF/RB)**
- dodatnie CG_ROLL_BIAS_PWM** → więcej ciągu **lewa strona (LF/LB)**

Jeśli przechyla się w prawo (prawa strona siada):

→ **CG_ROLL_BIAS_PWM** bardziej **ujemne**

$0 \rightarrow -60 \rightarrow -120 \rightarrow -180$

Jeśli przechyla się w lewo:

→ CG_ROLL_BIAS_PWM bardziej **dodatnie**

0 → +60 → +120 → +180

3) "Kręci się wokół jednego silnika" (pivot wokół rogu)

To prawie zawsze jest **problem odklejania od ziemi** (tarcie + za duża różnica PWM).

3A) Jeśli pivot dzieje się PRZY ODKLEJANIU (jeszcze o ziemię zahacza):

Stroisz: **anti-pivot**

1. TAKEOFF_BLEND_MIN w góre

0.35 → 0.45 → 0.55

2. (Jeśli nadal pivotuje) podnieś TAKEOFF_BLEND_FULL_PWM

4200 → 4800

Dopiero potem ruszasz rogi.

3B) Jeśli pivot dzieje się JUŻ W POWIETRZU (wisi kilka cm):

To znaczy, że konkretny róg jest "słabszy". Stroisz **motor** dla tego rogu:

→ Najpierw **offset** tego rogu:

- **MOTOR_OFFSET_<róg> += 100** (max 2–4 kroki)

Jeśli nadal:

→ potem **gain** tego rogu:

- **MOTOR_GAIN_<róg> += 0.02**

Mapa:

- ucieka/pivotuje w stronę **LF** → zwiększMOTOR_OFFSET_LF / MOTOR_GAIN_LF
 - w stronę **RF** → zwiększMOTOR_OFFSET_RF / MOTOR_GAIN_RF
 - w stronę **LB** → zwiększMOTOR_OFFSET_LB / MOTOR_GAIN_LB
 - w stronę **RB** → zwiększMOTOR_OFFSET_RB / MOTOR_GAIN_RB
-

4) Cały dron obraca się (yaw) CW/CCW w STOP (bez komendy)

To stroisz **yaw rate hold**:

- jeśli obraca się w którąś stronę:

→ zwiększMOTOR_KP_YAW_RATE

1.2 → 1.5 → 1.8

- jeśli nadal powoli dryfuje po czasie:

→ zwiększMOTOR_KI_YAW_RATE

0.20 → 0.26 → 0.32

Jeśli yaw zaczyna “pływąć/oscylować”, KP jest za duże — cofnij o jeden krok.

Najlepsza kolejność strojenia (żeby nie gonić kilku rzeczy naraz)

1. **TRIM-y** (żeby spoczynek był blisko 0)
2. **TAKEOFF_BLEND** (żeby nie pivotował na ziemi)
3. **CG_PITCH / CG_ROLL** (żeby nie uciekał przód/tyle/prawo/lewo)
4. **MOTOR_OFFSETS / MOTOR_GAIN** (jeśli konkretny róg nadal słaby)
5. **Yaw KP/KI** (żeby STOP nie kręcił)

Jeśli chcesz, mogę Ci to przepisać jeszcze bardziej “w 6 linijkach”, ale to powyżej to już wersja maksymalnie praktyczna pod Twoje **#define**.



Okej, w strukturze, gdy mam drona na ziemi i nic nie ruszam na razie, widzę, że IMU mi wysyła pomiary ROLL z zakresu -1.8 : -2.0; a PITCH z zakresu -1.0 : -1.2; jak powiniennem dostosować te parametry?

```
#define YAW_TRIM      0.0f
#define ROLL_TRIM_DEG +1.90f
#define PITCH_TRIM_DEG +1.65f
```

Masz teraz na ziemi (STOP, nic nie ruszasz):

- **ROLL ≈ -1.8...-2.0°**
- **PITCH ≈ -1.0...-1.2°**

Chcesz, żeby w tym stanie kontroler “celował” w takie kąty, żeby błąd był ~0. Najprostsza reguła dla Twojego układu z **IMU_SWAP_ROLL_PITCH = 1** jest taka:

- **ROLL_TRIM_DEG ustawiasz na przeciwność tego, co widzisz jako PITCH**
- **PITCH_TRIM_DEG ustawiasz na przeciwność tego, co widzisz jako ROLL**

Czyli bierzemy średnie z zakresów:

- **roll_mean ≈ -1.9°**
- **pitch_mean ≈ -1.1°**

I ustaw:

```
c

#define YAW_TRIM      0.0f
#define ROLL_TRIM_DEG (+1.10f) // bo PITCH jest ~ -1.1
```

```
#define PITCH_TRIM_DEG  (+1.90f) // bo ROLL jest ~ -1.9
```

Dlaczego tak?

Bo przy `IMU_SWAP_ROLL_PITCH=1` w Twoim regulatorze:

- “roll_meas” bierze się z `g_pitch_deg`
- “pitch_meas” bierze się z `g_roll_deg` (a potem jeszcze przechodzi przez `STAB_PITCH_SIGN = -1`)

Więc TRIM-y muszą być dobrane “na krzyż”, jak wyżej.

Jak stroić dalej (2 szybkie kroki)

1. Ustaw powyższe wartości i odpal.
2. Jeśli w spoczynku nadal widzisz np. PITCH = -0.4, to:
 - dodaj do `ROLL_TRIM_DEG` jeszcze **+0.4**
Analogicznie dla ROLL:
 - jeśli ROLL = -0.6, dodaj do `PITCH_TRIM_DEG` **+0.6**

Kroki po **0.2–0.3°** są w sam raz.

Jeśli podeślesz kolejny log po tej zmianie (nawet 10–15 sekund na ziemi), mogę Ci policzyć dokładnie średnie i zaproponować “docelowe” trymy z dokładnością do 0.1°.