

position_estimator_inav.cpp 思路整理及数据流

1. 为什么需要算法进行高度估计？

因为单纯的 acc.z 或者超声波/气压计都不能很好的估计实际飞行器高度。加速度高频震动，气压计会温漂，超声波模块有时候会接收不到反射的超声波。所以需要进行多种传感器融合得到尽可能准确的高度。多传感器融合技术就是用多种便宜的传感器，融合他们的信息得到准确信息。

2. inav 里的高度估计算法是什么？融合的过程是什么？

我的理解是：主要是用 acc.z 二次积分得到高度，但是 mpu6000 直接得到的 acc.z 并不能直接使用，所以用气压计算出一个矫正系数用来矫正 acc.z，然后二次积分得到高度，然后用气压计得到的高度直接矫正高度。也就是说里面有 2 次矫正。

过程是：

(1) 变量初始化

```
float z_est[2] = { 0.0f, 0.0f }; // pos, velfloat z_est[2] = { 0.0f, 0.0f }; // z 轴的高度、速度
```

```
float acc[] = { 0.0f, 0.0f, 0.0f }; //地理坐标系（NED）的加速度数据
```

```
float acc_bias[] = { 0.0f, 0.0f, 0.0f }; //机体坐标系下的加速度偏移量
```

```
float corr_baro = 0.0f; // 气压计校正系数
```

(2) 计算气压计高度的零点偏移，主要是取 200 个数据求平均。

```
bool wait_baro = true; //用来区别有没有初始化
```

```
thread_running = true;
```

```
hrt_abstime baro_wait_for_sample_time = hrt_absolute_time();
```

```
while (wait_baro && !thread_should_exit) {
```

```
    int ret = px4_poll(&fds_init[0], 1, 1000);
```

```
    if (ret < 0) {
```

```
        /* poll error */
```

```
        .....
```

```
    }
```

```
    else if (ret > 0) {
```

```
        if (fds_init[0].revents & POLLIN) {
```

```
            orb_copy(ORB_ID(sensor_combined), sensor_combined_sub, &sensor);
```

```
            .....
```

```
            if (baro_init_cnt < baro_init_num) {
```

```
                if (PX4_ISFINITE(sensor.baro_alt_meter[0])) {
```

```
                    baro_offset += sensor.baro_alt_meter[0];
```

```
                    baro_init_cnt++;
```

```
                }
```

```
            } else {
```

```
                wait_baro = false; //用来区别有没有初始化
```

```
                baro_offset /= (float) baro_init_cnt;
```

```
                .....
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

(3) 将传感器获取的机体加速度数据转换到地理坐标系下。

加速度数据要先去除矫正量(这里的 acc_bias[] 就是气压计经过一系列处理后得到的矫正量，第一次是初始值 0)(一次矫正)

```
sensor.accelerometer_m_s2[0] -= acc_bias[0];
```

```
sensor.accelerometer_m_s2[1] -= acc_bias[1];
```

```
sensor.accelerometer_m_s2[2] -= acc_bias[2];
```

然后转换坐标系

```
for (int i = 0; i < 3; i++) {  
    acc[i] = 0.0f;  
    for (int j = 0; j < 3; j++) {  
        acc[i] += PX4_R(att.R, i, j) * sensor.accelerometer_m_s2[j];  
    }  
}
```

地理坐标系下的 z 轴加速度是有重力加速度的，因此补偿上去

acc[2] += CONSTANTS_ONE_G; //z 轴的方向是向下的，即 acc[2]=-9.8，所以是加 CONSTANTS_ONE_G

(4) 计算气压计的矫正系数

```
corr_baro = baro_offset - sensor.baro_alt_meter[0] - z_est[0];
```

// baro_offset 气压计偏移量

// sensor.baro_alt_meter[0] 气压计的值

// z_est[0] 上一次计算出的最佳 z 轴位置估计值

//也就是求气压得到新高度与上一次的高度差

(5) 加速度偏移向量矫正

```
accel_bias_corr[2] -= corr_baro * params.w_z_baro * params.w_z_baro;
```

(6) 将偏移向量转换到机体坐标系

```
for (int i = 0; i < 3; i++) {  
    float c = 0.0f;  
    for (int j = 0; j < 3; j++) {  
        c += PX4_R(att.R, j, i) * accel_bias_corr[j]; //转变成机体加速度  
    }  
    if (PX4_ISFINITE(c)) {  
        acc_bias[i] += c * params.w_acc_bias * dt; //接着用于下一次 acc 矫正  
    }  
}
```

(7) 加速度推算高度

```
inertial_filter_predict(dt, z_est, acc[2]);
```

```
void inertial_filter_predict(float dt, float x[2], float acc)
```

```
{  
    if (isfinite(dt)) {  
        if (!isfinite(acc)) {  
            acc = 0.0f;  
        }  
        x[0] += x[1] * dt + acc * dt * dt / 2.0f; //位置  
        x[1] += acc * dt; //速度  
    }  
}
```

(8) 气压计校正系数进行校正(二次矫正)

```
inertial_filter_correct(corr_baro, dt, z_est, 0, params.w_z_baro);
```

```
void inertial_filter_correct(float e, float dt, float x[2], int i, float w)
```

```
{  
    if (isfinite(e) && isfinite(w) && isfinite(dt)) {  
        float ewdt = e * w * dt; // corr_baro * params.w_z_baro * dt, z_est  
        x[i] += ewdt; //位置矫正  
        if (i == 0) {  
            x[1] += w * ewdt; //速度矫正  
        }  
    }  
}
```

```

    }
}
}
也可以用超声波跟 acc 融合得到高度
if (lidar_first) {
    lidar_first = false;
    lidar_offset = dist_ground + z_est[0];
}
corr_lidar = lidar_offset - dist_ground - z_est[0];
.....
if (use_lidar) { //先判断 lidar, 所以是 lidar 的优先级高于气压计
    accel_bias_corr[2] -= corr_lidar * params.w_z_lidar * params.w_z_lidar;
} else {
    accel_bias_corr[2] -= corr_baro * params.w_z_baro * params.w_z_baro;
}

```

接下来就是和气压计的处理一样了

3. xy 轴怎么矫正的？

现在常用的是光流和 GPS。算法过程和定高过程类似，也是用传感器两次矫正，第一次矫正加速度，第二次矫正传感器对应能感知的量，比如光流就是感知速度，GPS 可以感知位置和速度。

总体的代码思路是：

(1) 变量初始化。

```

1. float z_est[2] = { 0.0f, 0.0f }; // z 轴的高度、速度
2. float acc[] = { 0.0f, 0.0f, 0.0f }; //地理坐标系 (NED) 的加速度数据
3. float acc_bias[] = { 0.0f, 0.0f, 0.0f }; // 机体坐标系下的加速度偏移量
4.
5. float corr_baro = 0.0f; // D
6. float corr_gps[3][2] = {
7.     { 0.0f, 0.0f }, // N (pos, vel)
8.     { 0.0f, 0.0f }, // E (pos, vel)
9.     { 0.0f, 0.0f }, // D (pos, vel)
10. };
11. float corr_vision[3][2] = {
12.     { 0.0f, 0.0f }, // N (pos, vel)
13.     { 0.0f, 0.0f }, // E (pos, vel)
14.     { 0.0f, 0.0f }, // D (pos, vel)
15. };
16. float corr_mocap[3][1] = {
17.     { 0.0f }, // N (pos)
18.     { 0.0f }, // E (pos)
19.     { 0.0f }, // D (pos)
20. };
21. float corr_lidar = 0.0f; //据说是超声波
22. float corr_flow[] = { 0.0f, 0.0f }; // N E
23.
24. bool gps_valid = false; // GPS is valid
25. bool lidar_valid = false; // lidar is valid
26. bool flow_valid = false; // flow is valid

```

```

27. bool flow_accurate = false;    // flow should be accurate (this flag not updated if flow_valid ==
    false)
28. bool vision_valid = false;    // vision is valid
29. bool mocap_valid = false;    // mocap is valid

```

(2)计算气压计高度的零点偏移，主要是取 200 个数据求平均。

```

1. baro_offset += sensor.baro_alt_meter;
2. baro_offset /= (float) baro_init_cnt;

```

(3)各传感器计算得带各自的修正系数和权重

```

1. corr_baro = baro_offset - sensor.baro_alt_meter[0] - z_est[0];
2. corr_lidar = lidar_offset - dist_ground - z_est[0];
3. corr_flow[0] = flow_v[0] - x_est[1]; /* velocity correction */
4. corr_flow[1] = flow_v[1] - y_est[1];
5. corr_vision[0][0] = vision.x - x_est[0]; /* calculate correction for position */
6. corr_vision[1][0] = vision.y - y_est[0];
7. corr_vision[2][0] = vision.z - z_est[0];
8. corr_vision[0][1] = vision.vx - x_est[1]; /* calculate correction for velocity */
9. corr_vision[1][1] = vision.vy - y_est[1];
10. corr_vision[2][1] = vision.vz - z_est[1];
11. corr_mocap[0][0] = mocap.x - x_est[0]; /* calculate correction for position */
12. corr_mocap[1][0] = mocap.y - y_est[0];
13. corr_mocap[2][0] = mocap.z - z_est[0];
14. corr_gps[0][0] = gps_proj[0] - est_buf[est_i][0][0]; /* calculate correction for position */
15. corr_gps[1][0] = gps_proj[1] - est_buf[est_i][1][0];
16. corr_gps[2][0] = local_pos.ref_alt - alt - est_buf[est_i][2][0];
17. corr_gps[0][1] = gps.vel_n_m_s - est_buf[est_i][0][1]; /* calculate correction for velocity */
18. corr_gps[1][1] = gps.vel_e_m_s - est_buf[est_i][1][1];
19. corr_gps[2][1] = gps.vel_d_m_s - est_buf[est_i][2][1];
20. w_gps_xy = min_eph_epv / fmaxf(min_eph_epv, gps.eph);
21. w_gps_z = min_eph_epv / fmaxf(min_eph_epv, gps.epv);

```

(4)判断是否超时

```

1. if ((flow_valid || lidar_valid) && t > (flow_time + flow_topic_timeout))
2. if (gps_valid && (t > (gps.timestamp_position + gps_topic_timeout)))
3. if (vision_valid && (t > (vision.timestamp_boot + vision_topic_timeout)))
4. if (mocap_valid && (t > (mocap.timestamp_boot + mocap_topic_timeout)))
5. if (lidar_valid && (t > (lidar_time + lidar_timeout)))

```

(5)判断是用哪一个传感器

```

1. /* use GPS if it's valid and reference position initialized */
2. bool use_gps_xy = ref_initied && gps_valid && params.w_xy_gps_p > MIN_VALID_W;
3. bool use_gps_z = ref_initied && gps_valid && params.w_z_gps_p > MIN_VALID_W;
4. /* use VISION if it's valid and has a valid weight parameter */
5. bool use_vision_xy = vision_valid && params.w_xy_vision_p > MIN_VALID_W;

```

```

6. bool use_vision_z = vision_valid && params.w_z_vision_p > MIN_VALID_W;
7. /* use MOCAP if it's valid and has a valid weight parameter */
8. bool use_mocap = mocap_valid && params.w_mocap_p > MIN_VALID_W && params.att_ext_hdg_m == mocap_heading; //check if external heading is mocap
9. if (params.disable_mocap) { //disable mocap if fake gps is used
10.     use_mocap = false;
11. }
12. /* use flow if it's valid and (accurate or no GPS available) */
13. bool use_flow = flow_valid && (flow_accurate || !use_gps_xy);
14. /* use LIDAR if it's valid and lidar altitude estimation is enabled */
15. use_lidar = lidar_valid && params.enable_lidar_alt_est;

```

(6)计算权重

```

1. float flow_q = flow.quality / 255.0f;
2. float flow_q_weight = (flow_q - params.flow_q_min) / (1.0f - params.flow_q_min);
3. w_flow = PX4_R(att.R, 2, 2) * flow_q_weight / fmaxf(1.0f, flow_dist);
4. if (!flow_accurate) {
5.     w_flow *= 0.05f;
6. }
7.
8. float w_xy_gps_p = params.w_xy_gps_p * w_gps_xy;
9. float w_xy_gps_v = params.w_xy_gps_v * w_gps_xy;
10. float w_z_gps_p = params.w_z_gps_p * w_gps_z;
11. float w_z_gps_v = params.w_z_gps_v * w_gps_z;
12.
13. float w_xy_vision_p = params.w_xy_vision_p;
14. float w_xy_vision_v = params.w_xy_vision_v;
15. float w_z_vision_p = params.w_z_vision_p;
16.
17. float w_mocap_p = params.w_mocap_p;
18. /* reduce GPS weight if optical flow is good */
19. if (use_flow && flow_accurate) {
20.     w_xy_gps_p *= params.w_gps_flow;
21.     w_xy_gps_v *= params.w_gps_flow;
22. }
23. /* baro offset correction */
24. if (use_gps_z) {
25.     float offs_corr = corr_gps[2][0] * w_z_gps_p * dt;
26.     baro_offset += offs_corr;
27.     corr_baro += offs_corr;
28. }
29. /* accelerometer bias correction for GPS (use buffered rotation matrix) */
30. float accel_bias_corr[3] = { 0.0f, 0.0f, 0.0f };

```

(7)根据使用的传感器计算加速度偏差

```

1. if (use_gps_xy) {
2.     accel_bias_corr[0] -= corr_gps[0][0] * w_xy_gps_p * w_xy_gps_p;

```

```

3.     accel_bias_corr[0] -= corr_gps[0][1] * w_xy_gps_v;
4.     accel_bias_corr[1] -= corr_gps[1][0] * w_xy_gps_p * w_xy_gps_p;
5.     accel_bias_corr[1] -= corr_gps[1][1] * w_xy_gps_v;
6. }
7.
8. if (use_gps_z) {
9.     accel_bias_corr[2] -= corr_gps[2][0] * w_z_gps_p * w_z_gps_p;
10.    accel_bias_corr[2] -= corr_gps[2][1] * w_z_gps_v;
11. }
12.
13. /* transform error vector from NED frame to body frame */
14. for (int i = 0; i < 3; i++) {
15.     float c = 0.0f;
16.
17.     for (int j = 0; j < 3; j++) {
18.         c += R_gps[j][i] * accel_bias_corr[j];
19.     }
20.
21.     if (PX4_ISFINITE(c)) {
22.         acc_bias[i] += c * params.w_acc_bias * dt;
23.     }
24. }
25.
26. /* accelerometer bias correction for VISION (use buffered rotation matrix) */
27. accel_bias_corr[0] = 0.0f;
28. accel_bias_corr[1] = 0.0f;
29. accel_bias_corr[2] = 0.0f;
30.
31. if (use_vision_xy) {
32.     accel_bias_corr[0] -= corr_vision[0][0] * w_xy_vision_p * w_xy_vision_p;
33.     accel_bias_corr[0] -= corr_vision[0][1] * w_xy_vision_v;
34.     accel_bias_corr[1] -= corr_vision[1][0] * w_xy_vision_p * w_xy_vision_p;
35.     accel_bias_corr[1] -= corr_vision[1][1] * w_xy_vision_v;
36. }
37.
38. if (use_vision_z) {
39.     accel_bias_corr[2] -= corr_vision[2][0] * w_z_vision_p * w_z_vision_p;
40. }
41.
42. /* accelerometer bias correction for MOCAP (use buffered rotation matrix) */
43. accel_bias_corr[0] = 0.0f;
44. accel_bias_corr[1] = 0.0f;
45. accel_bias_corr[2] = 0.0f;
46.
47. if (use_mocap) {
48.     accel_bias_corr[0] -= corr_mocap[0][0] * w_mocap_p * w_mocap_p;
49.     accel_bias_corr[1] -= corr_mocap[1][0] * w_mocap_p * w_mocap_p;
50.     accel_bias_corr[2] -= corr_mocap[2][0] * w_mocap_p * w_mocap_p;
51. }

```

```

52.
53. /* transform error vector from NED frame to body frame */
54. for (int i = 0; i < 3; i++) {
55.     float c = 0.0f;
56.
57.     for (int j = 0; j < 3; j++) {
58.         c += PX4_R(att.R, j, i) * accel_bias_corr[j];
59.     }
60.
61.     if (PX4_ISFINITE(c)) {
62.         acc_bias[i] += c * params.w_acc_bias * dt;
63.     }
64. }
65.
66. /* accelerometer bias correction for flow and baro (assume that there is no delay) */
67. accel_bias_corr[0] = 0.0f;
68. accel_bias_corr[1] = 0.0f;
69. accel_bias_corr[2] = 0.0f;
70.
71. if (use_flow) {
72.     accel_bias_corr[0] -= corr_flow[0] * params.w_xy_flow;
73.     accel_bias_corr[1] -= corr_flow[1] * params.w_xy_flow;
74. }
75.
76. if (use_lidar) {
77.     accel_bias_corr[2] -= corr_lidar * params.w_z_lidar * params.w_z_lidar;
78. } else {
79.     accel_bias_corr[2] -= corr_baro * params.w_z_baro * params.w_z_baro;
80. }
81.
82. /* transform error vector from NED frame to body frame */
83. for (int i = 0; i < 3; i++) {
84.     float c = 0.0f;
85.
86.     for (int j = 0; j < 3; j++) {
87.         c += PX4_R(att.R, j, i) * accel_bias_corr[j];
88.     }
89.
90.     if (PX4_ISFINITE(c)) {
91.         acc_bias[i] += c * params.w_acc_bias * dt;
92.     }
93. }

```

这里得到的 `acc_bias[]`用于前面程序(500 行左右)

```

1. /* sensor combined */
2. orb_check(sensor_combined_sub, &updated);
3.
4. if (updated) {

```

```

5.     orb_copy(ORB_ID(sensor_combined), sensor_combined_sub, &sensor);
6.
7.     if (sensor.accelerometer_timestamp[0] != accel_timestamp) {
8.         if (att.R_valid) {
9.             /* correct accel bias */
10.            sensor.accelerometer_m_s2[0] -= acc_bias[0];
11.            sensor.accelerometer_m_s2[1] -= acc_bias[1];
12.            sensor.accelerometer_m_s2[2] -= acc_bias[2];
13.
14.            /* transform acceleration vector from body frame to NED frame */
15.            for (int i = 0; i < 3; i++) {
16.                acc[i] = 0.0f;
17.
18.                for (int j = 0; j < 3; j++) {
19.                    acc[i] += PX4_R(att.R, i, j) * sensor.accelerometer_m_s2[j];
20.                }
21.            }
22.
23.            acc[2] += CONSTANTS_ONE_G;
24.
25.        } else {
26.            memset(acc, 0, sizeof(acc));
27.        }
28.
29.        accel_timestamp = sensor.accelerometer_timestamp[0];
30.        accel_updates++;
31.    }

```

这里得到修正后的加速度，之后用此加速度进行一次、二次积分得到预计速度和位置

8. 预计位置

```

1.  /* inertial filter prediction for altitude */
2.  if (can_estimate_xy) {
3.  {
4.      inertial_filter_predict(dt, x_est, acc[0]);
5.      inertial_filter_predict(dt, y_est, acc[1]);
6.  }
7.  inertial_filter_predict(dt, z_est, acc[2]);

```

函数解析

```

1.  void inertial_filter_predict(float dt, float x[2], float acc)
2.  {
3.      if (isfinite(dt)) {
4.          if (!isfinite(acc)) {
5.              acc = 0.0f;
6.          }
7.          x[0] += x[1] * dt + acc * dt * dt / 2.0f;

```



```

8.         x[1] += acc * dt;
9.     }
10. }

```

9.修正位置

利用传感器得到的速度和位置修正

```

1.  /* inertial filter correction for altitude */
2.  if (use_lidar) {
3.      inertial_filter_correct(corr_lidar, dt, z_est, 0, params.w_z_lidar);
4.  } else {
5.      inertial_filter_correct(corr_baro, dt, z_est, 0, params.w_z_baro);
6.  }
7.  if (use_gps_z) {
8.      epv = fminf(epv, gps.epv);
9.      inertial_filter_correct(corr_gps[2][0], dt, z_est, 0, w_z_gps_p);
10.     inertial_filter_correct(corr_gps[2][1], dt, z_est, 1, w_z_gps_v);
11. }
12. if (use_vision_z) {
13.     epv = fminf(epv, epv_vision);
14.     inertial_filter_correct(corr_vision[2][0], dt, z_est, 0, w_z_vision_p);
15. }
16. if (use_mocap) {
17.     epv = fminf(epv, epv_mocap);
18.     inertial_filter_correct(corr_mocap[2][0], dt, z_est, 0, w_mocap_p);
19. }
20. if (can_estimate_xy) {
21.     /* inertial filter correction for position */
22.     if (use_flow) {
23.         eph = fminf(eph, eph_flow);
24.         inertial_filter_correct(corr_flow[0], dt, x_est, 1, params.w_xy_flow * w_flow);
25.         inertial_filter_correct(corr_flow[1], dt, y_est, 1, params.w_xy_flow * w_flow);
26.     }
27.     if (use_gps_xy) {
28.         eph = fminf(eph, gps.eph);
29.         inertial_filter_correct(corr_gps[0][0], dt, x_est, 0, w_xy_gps_p);
30.         inertial_filter_correct(corr_gps[1][0], dt, y_est, 0, w_xy_gps_p);
31.         if (gps.vel_ned_valid && t < gps.timestamp_velocity + gps_topic_timeout) {
32.             inertial_filter_correct(corr_gps[0][1], dt, x_est, 1, w_xy_gps_v);
33.             inertial_filter_correct(corr_gps[1][1], dt, y_est, 1, w_xy_gps_v);
34.         }
35.     }
36.     if (use_vision_xy) {
37.         eph = fminf(eph, eph_vision);
38.         inertial_filter_correct(corr_vision[0][0], dt, x_est, 0, w_xy_vision_p);
39.         inertial_filter_correct(corr_vision[1][0], dt, y_est, 0, w_xy_vision_p);
40.         if (w_xy_vision_v > MIN_VALID_W) {
41.             inertial_filter_correct(corr_vision[0][1], dt, x_est, 1, w_xy_vision_v);

```

```

42.         inertial_filter_correct(corr_vision[1][1], dt, y_est, 1, w_xy_vision_v);
43.     }
44. }
45. if (use_mocap) {
46.     eph = fminf(eph, eph_mocap);
47.     inertial_filter_correct(corr_mocap[0][0], dt, x_est, 0, w_mocap_p);
48.     inertial_filter_correct(corr_mocap[1][0], dt, y_est, 0, w_mocap_p);
49. }
50. }
51. /* run terrain estimator */
52. terrain_estimator.predict(dt, &att, &sensor, &lidar);

```

1. 函数解析
2. e 是修正系数; dt 周期时间; $x[2]$ 是 2 个 `float` 型成员的数组, $x[0]$ 是位置, $x[1]$ 是速度;
3. i 表示修正位置还是速度, 0 是修正位置, 1 是修正速度; w 是权重系数
4. 这里 x_est 、 y_est 、 z_est 通过 `float x[2]` 传进来后, 经过函数处理直接传回来给 x_est 、 y_est 、 z_est

```

1. void inertial_filter_correct(float e, float dt, float x[2], int i, float w)
2. {
3.     if (isfinite(e) && isfinite(w) && isfinite(dt)) {
4.         float ewdt = e * w * dt;
5.         x[i] += ewdt;
6.
7.         if (i == 0) {
8.             x[1] += w * ewdt;
9.         }
10.    }
11. }

```

10.发布

```

1. /* publish local position */
2. local_pos.xy_valid = can_estimate_xy;
3. local_pos.v_xy_valid = can_estimate_xy;
4. local_pos.xy_global = local_pos.xy_valid && use_gps_xy;
5. local_pos.z_global = local_pos.z_valid && use_gps_z;
6. local_pos.x = x_est[0];
7. local_pos.vx = x_est[1];
8. local_pos.y = y_est[0];
9. local_pos.vy = y_est[1];
10. local_pos.z = z_est[0];
11. local_pos.vz = z_est[1];
12. local_pos.yaw = att.yaw;
13. local_pos.dist_bottom_valid = dist_bottom_valid;
14. local_pos.eph = eph;
15. local_pos.epv = epv;
16.

```

```

17. if (local_pos.dist_bottom_valid) {
18.     local_pos.dist_bottom = dist_ground;
19.     local_pos.dist_bottom_rate = - z_est[1];
20. }
21.
22. local_pos.timestamp = t;
23.
24. orb_publish(ORB_ID(vehicle_local_position), vehicle_local_position_pub, &local_pos);
25.
26. if (local_pos.xy_global && local_pos.z_global) {
27.     /* publish global position */
28.     global_pos.timestamp = t;
29.     global_pos.time_utc_usec = gps.time_utc_usec;
30.
31.     double est_lat, est_lon;
32.     map_projection_reproject(&ref, local_pos.x, local_pos.y, &est_lat, &est_lon);
33.
34.     global_pos.lat = est_lat;
35.     global_pos.lon = est_lon;
36.     global_pos.alt = local_pos.ref_alt - local_pos.z;
37.
38.     global_pos.vel_n = local_pos.vx;
39.     global_pos.vel_e = local_pos.vy;
40.     global_pos.vel_d = local_pos.vz;
41.
42.     global_pos.yaw = local_pos.yaw;
43.
44.     global_pos.eph = eph;
45.     global_pos.epv = epv;
46.
47.     if (terrain_estimator.is_valid()) {
48.         global_pos.terrain_alt = global_pos.alt - terrain_estimator.get_distance_to_ground();
49.         global_pos.terrain_alt_valid = true;
50.
51.     } else {
52.         global_pos.terrain_alt_valid = false;
53.     }
54.
55.     global_pos.pressure_alt = sensor.baro_alt_meter[0];
56.
57.     if (vehicle_global_position_pub == NULL) {
58.         vehicle_global_position_pub = orb_advertise(ORB_ID(vehicle_global_position), &global_pos);
59.
60.     } else {
61.         orb_publish(ORB_ID(vehicle_global_position), vehicle_global_position_pub, &global_pos);
62.     }
63. }

```

最后再把整个程序过一遍。

关于光流部分注释是：

```
1.  /* optical flow */
2.  orb_check(optical_flow_sub, &updated);
3.
4.  if (updated && lidar_valid) {
5.      orb_copy(ORB_ID(optical_flow), optical_flow_sub, &flow);
6.
7.      flow_time = t;
8.      float flow_q = flow.quality / 255.0f;
9.      //0<flow.quality<255
10.     //qual = (uint8_t)(meancount * 255 / (NUM_BLOCKS*NUM_BLOCKS));
11.     //每 2 帧图像匹配块的数量
12.     float dist_bottom = lidar.current_distance;//超声波测得的距离
13.     //离地面距离>20cm&&flow.quality>某个值&&PX4_R(att.R, 2, 2) > 0.7f(不知道什么意思)
14.     if (dist_bottom > flow_min_dist && flow_q > params.flow_q_min && PX4_R(att.R, 2, 2) > 0.7f) {
15.
16.         /* distance to surface */
17.         //float flow_dist = dist_bottom / PX4_R(att.R, 2, 2); //use this if using sonar
18.         float flow_dist = dist_bottom; //use this if using lidar
19.
20.         /* check if flow if too large for accurate measurements */
21.         /* calculate estimated velocity in body frame */
22.         float body_v_est[2] = { 0.0f, 0.0f };
23.         for (int i = 0; i < 2; i++) {
24.             body_v_est[i] = PX4_R(att.R, 0, i) * x_est[1] + PX4_R(att.R, 1, i) * y_est[1] + PX4_R(
att.R, 2, i) * z_est[1];
25.             /* 旋转矩阵第 1,2 列*北东地 xyz 轴的速度=机体 xy 轴方向的速度
26.             * x_est[]是矫正之后的速度
27.             */
28.         }
29.
30.         /* set this flag if flow should be accurate according to current velocity and attitude rat
e estimate */
31.         flow_accurate = fabsf(body_v_est[1] / flow_dist - att.rollspeed) < max_flow &&
fabsf(body_v_est[0] / flow_dist + att.pitchspeed) < max_flow;
32.         /* flow_accurate 判断光流精度能否使用
33.         * xy 轴机体速度/距离=机体角速度
34.         * 机体角速度-飞控测得的角速度<某个阈值, 表明精度可用
35.         */
36.
37.         /*calculate offset of flow-gyro using already calibrated gyro from autopilot*/
38.         flow_gyrospeed[0] = flow.gyro_x_rate_integral / (float)flow.integration_timespan * 1000000
.0f;
39.         flow_gyrospeed[1] = flow.gyro_y_rate_integral / (float)flow.integration_timespan * 1000000
.0f;
```

```

40.         flow_gyrospeed[2] = flow.gyro_z_rate_integral / (float)flow.integration_timespan * 1000000
        .0f;
41.
42.         //moving average
43.         //滑动均值滤波
44.         if (n_flow >= 100) {
45.             gyro_offset_filtered[0] = flow_gyrospeed_filtered[0] - att_gyrospeed_filtered[0];
46.             gyro_offset_filtered[1] = flow_gyrospeed_filtered[1] - att_gyrospeed_filtered[1];
47.             gyro_offset_filtered[2] = flow_gyrospeed_filtered[2] - att_gyrospeed_filtered[2];
48.             n_flow = 0;
49.             flow_gyrospeed_filtered[0] = 0.0f;
50.             flow_gyrospeed_filtered[1] = 0.0f;
51.             flow_gyrospeed_filtered[2] = 0.0f;
52.             att_gyrospeed_filtered[0] = 0.0f;
53.             att_gyrospeed_filtered[1] = 0.0f;
54.             att_gyrospeed_filtered[2] = 0.0f;
55.
56.         } else {
57.             flow_gyrospeed_filtered[0] = (flow_gyrospeed[0] + n_flow * flow_gyrospeed_filtered[0])
/ (n_flow + 1);
58.             flow_gyrospeed_filtered[1] = (flow_gyrospeed[1] + n_flow * flow_gyrospeed_filtered[1])
/ (n_flow + 1);
59.             flow_gyrospeed_filtered[2] = (flow_gyrospeed[2] + n_flow * flow_gyrospeed_filtered[2])
/ (n_flow + 1);
60.             att_gyrospeed_filtered[0] = (att.pitchspeed + n_flow * att_gyrospeed_filtered[0]) / (n
_flow + 1);
61.             att_gyrospeed_filtered[1] = (att.rollspeed + n_flow * att_gyrospeed_filtered[1]) / (n
_flow + 1);
62.             att_gyrospeed_filtered[2] = (att.yawspeed + n_flow * att_gyrospeed_filtered[2]) / (n_f
low + 1);
63.             n_flow++;
64.         }
65.
66.
67.         /*yaw compensation (flow sensor is not in center of rotation) -> params in QGC*/
68.         //yaw 方向补偿
69.         yaw_comp[0] = - params.flow_module_offset_y * (flow_gyrospeed[2] - gyro_offset_filtered[2]
);
70.         yaw_comp[1] = params.flow_module_offset_x * (flow_gyrospeed[2] - gyro_offset_filtered[2]);
71.
72.         /* flow_gyrospeed[2]光流模块上测得的 z 轴角速度
73.          * gyro_offset_filtered[2]光流模块上测得的 z 轴角速度平均值-飞控测得的 z 轴角速度平均值
74.          * flow_gyrospeed[2]-gyro_offset_filtered[2]=飞控的 z 轴角速度
75.          */
76.         /* convert raw flow to angular flow (rad/s) */
77.         //将光流转换成弧度
78.         float flow_ang[2];

```

```

79.      /* check for vehicle rates setpoint - it below threshold -> dont subtract -> better hover
      */
80.      orb_check(vehicle_rate_sp_sub, &updated);
81.      if (updated)
82.          orb_copy(ORB_ID(vehicle_rates_setpoint), vehicle_rate_sp_sub, &rates_setpoint);
83.
84.      double rate_threshold = 0.15f;
85.      //pitch 方向的角速度<某个阈值, 不用 pitch 角度的补偿
86.      if (fabs(rates_setpoint.pitch) < rate_threshold) {
87.          //warnx("[inav] test ohne comp");
88.          flow_ang[0] = (flow.pixel_flow_x_integral / (float)flow.integration_timespan * 1000000
89.              .0f) * params.flow_k; //for now the flow has to be scaled (to small)
90.      }
91.      //pitch 方向的角速度>某个阈值, 需要 pitch 角度的补偿
92.      else {
93.          //warnx("[inav] test mit comp");
94.          //calculate flow [rad/s] and compensate for rotations (and offset of flow-gyro)
95.          flow_ang[0] = ((flow.pixel_flow_x_integral - flow.gyro_x_rate_integral) / (float)flow.
96.              integration_timespan * 1000000.0f
97.              + gyro_offset_filtered[0]) * params.flow_k; //for now the flow has to be sca
98.              led (to small)
99.      }
100.
101.      if (fabs(rates_setpoint.roll) < rate_threshold) {
102.          flow_ang[1] = (flow.pixel_flow_y_integral / (float)flow.integration_timespan * 1000000
103.              .0f) * params.flow_k; //for now the flow has to be scaled (to small)
104.      }
105.
106.      /* flow.integration_timespan
107.      * 每 2 帧图像拍摄时间差的积分, 积分时间段是 I2C 读取光流的时间
108.      * accumulation timespan in microseconds
109.      * pixflow main.c 中
110.      * uint32_t deltetime = (get_boot_time_us() - lasttime);
111.      * integration_timespan += deltetime;
112.      */
113.
114.      /* flow.pixel_flow_y_integral
115.      * I2C 读取光流的时间段内每 2 帧图像间的光流和
116.      * accumulated optical flow in radians around y axis
117.      * pixflow main.c 中
118.      * accumulated_flow_x += pixel_flow_y/focal_length_px * 1.0f; //rad axis swapped to align
119.      x flow around y axis
120.
121.      *
122.      x 移动距离(图片)/焦距=弧度
123.      */
124.
125.      /* flow_ang[]是[rad/s], 光流位移转变成弧度再除以时间*/
126.      else {
127.          //calculate flow [rad/s] and compensate for rotations (and offset of flow-gyro)
128.          flow_ang[1] = ((flow.pixel_flow_y_integral - flow.gyro_y_rate_integral) / (float)flow
129.              .integration_timespan * 1000000.0f
130.              + gyro_offset_filtered[1]) * params.flow_k; //for now the flow has to be sc
131.              aled (to small)

```

```

120.     }
121.
122.     /* flow measurements vector */
123.     float flow_m[3];
124.     if (fabs(rates_setpoint.yaw) < rate_threshold) {
125.         flow_m[0] = -flow_ang[0] * flow_dist; //角速度*距离=位移速度
126.         flow_m[1] = -flow_ang[1] * flow_dist;
127.     } else {
128.         flow_m[0] = -flow_ang[0] * flow_dist - yaw_comp[0] * params.flow_k;
129.         flow_m[1] = -flow_ang[1] * flow_dist - yaw_comp[1] * params.flow_k;
130.     }
131.     flow_m[2] = z_est[1];
132.
133.     /* velocity in NED */
134.     float flow_v[2] = { 0.0f, 0.0f };
135.
136.     /* project measurements vector to NED basis, skip Z component */
137.     for (int i = 0; i < 2; i++) {
138.         for (int j = 0; j < 3; j++) {
139.             flow_v[i] += PX4_R(att.R, i, j) * flow_m[j]; //从机体坐标转换成北东地坐标，旋转矩阵的
列分别为 xyz 的旋转，i=0,1，正好跳过 z 轴
140.         }
141.     }
142.
143.     /* velocity correction */
144.     corr_flow[0] = flow_v[0] - x_est[1];
145.     corr_flow[1] = flow_v[1] - y_est[1];
146.     /* adjust correction weight */
147.     float flow_q_weight = (flow_q - params.flow_q_min) / (1.0f - params.flow_q_min); // (光流匹
配数量-最少达标光流匹配数量)/(1-最少达标光流匹配数量)
148.     w_flow = PX4_R(att.R, 2, 2) * flow_q_weight / fmaxf(1.0f, flow_dist); //X4_R(att.R, 2, 2)
是 cosA*cosB 的值
149.
150.
151.     /* if flow is not accurate, reduce weight for it */
152.     // TODO make this more fuzzy
153.     if (!flow_accurate) { //精度不可用
154.         w_flow *= 0.05f;
155.     }
156.
157.     /* under ideal conditions, on 1m distance assume EPH = 10cm */
158.     eph_flow = 0.1f / w_flow;
159.
160.     flow_valid = true;
161.
162. } else {
163.     w_flow = 0.0f;
164.     flow_valid = false;
165. }

```

```
166.  
167.     flow_updates++;  
168. }
```