[position\_estimator\_inav.cpp思路整理及数据流](http://blog.csdn.net/czyv587/article/details/51884052)

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。

1. 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

1. 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;

......

}

}

}

}

}

1. 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

1. 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

1. 加速度偏移向量矫正

accel\_bias\_corr[2] -= corr\_baro \* params.w\_z\_baro \* params.w\_z\_baro;

1. 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

}

}

1. 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

}

}

1. 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

1. 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
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
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\_inited && gps\_valid && params.w\_xy\_gps\_p > MIN\_VALID\_W;
3. **bool** use\_gps\_z = ref\_inited && 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. }
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;
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;
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. }
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. }
13. /\* transform error vector from NED frame to body frame \*/
14. **for** (**int** i = 0; i < 3; i++) {
15. **float** c = 0.0f;
17. **for** (**int** j = 0; j < 3; j++) {
18. c += R\_gps[j][i] \* accel\_bias\_corr[j];
19. }
21. **if** (PX4\_ISFINITE(c)) {
22. acc\_bias[i] += c \* params.w\_acc\_bias \* dt;
23. }
24. }
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;
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. }
38. **if** (use\_vision\_z) {
39. accel\_bias\_corr[2] -= corr\_vision[2][0] \* w\_z\_vision\_p \* w\_z\_vision\_p;
40. }
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;
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. }
53. /\* transform error vector from NED frame to body frame \*/
54. **for** (**int** i = 0; i < 3; i++) {
55. **float** c = 0.0f;
57. **for** (**int** j = 0; j < 3; j++) {
58. c += PX4\_R(att.R, j, i) \* accel\_bias\_corr[j];
59. }
61. **if** (PX4\_ISFINITE(c)) {
62. acc\_bias[i] += c \* params.w\_acc\_bias \* dt;
63. }
64. }
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;
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. }
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. }
82. /\* transform error vector from NED frame to body frame \*/
83. **for** (**int** i = 0; i < 3; i++) {
84. **float** c = 0.0f;
86. **for** (**int** j = 0; j < 3; j++) {
87. c += PX4\_R(att.R, j, i) \* accel\_bias\_corr[j];
88. }
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);
4. **if** (updated) {
5. orb\_copy(ORB\_ID(sensor\_combined), sensor\_combined\_sub, &sensor);
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];
14. /\* transform acceleration vector from body frame to NED frame \*/
15. **for** (**int** i = 0; i < 3; i++) {
16. acc[i] = 0.0f;
18. **for** (**int** j = 0; j < 3; j++) {
19. acc[i] += PX4\_R(att.R, i, j) \* sensor.accelerometer\_m\_s2[j];
20. }
21. }
23. acc[2] += CONSTANTS\_ONE\_G;
25. } **else** {
26. memset(acc, 0, **sizeof**(acc));
27. }
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表示修正是位置还是速度，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]传进来来后，经过函数处理直接传回来给x\_est、y\_est、z\_est
5. **void** inertial\_filter\_correct(**float** e, **float** dt, **float** x[2], **int** i, **float** w)
6. {
7. **if** (isfinite(e) && isfinite(w) && isfinite(dt)) {
8. **float** ewdt = e \* w \* dt;
9. x[i] += ewdt;
11. **if** (i == 0) {
12. x[1] += w \* ewdt;
13. }
14. }
15. }

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;
17. **if** (local\_pos.dist\_bottom\_valid) {
18. local\_pos.dist\_bottom = dist\_ground;
19. local\_pos.dist\_bottom\_rate = - z\_est[1];
20. }
22. local\_pos.timestamp = t;
24. orb\_publish(ORB\_ID(vehicle\_local\_position), vehicle\_local\_position\_pub, &local\_pos);
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;
31. **double** est\_lat, est\_lon;
32. map\_projection\_reproject(&ref, local\_pos.x, local\_pos.y, &est\_lat, &est\_lon);
34. global\_pos.lat = est\_lat;
35. global\_pos.lon = est\_lon;
36. global\_pos.alt = local\_pos.ref\_alt - local\_pos.z;
38. global\_pos.vel\_n = local\_pos.vx;
39. global\_pos.vel\_e = local\_pos.vy;
40. global\_pos.vel\_d = local\_pos.vz;
42. global\_pos.yaw = local\_pos.yaw;
44. global\_pos.eph = eph;
45. global\_pos.epv = epv;
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**;
51. } **else** {
52. global\_pos.terrain\_alt\_valid = **false**;
53. }
55. global\_pos.pressure\_alt = sensor.baro\_alt\_meter[0];
57. **if** (vehicle\_global\_position\_pub == NULL) {
58. vehicle\_global\_position\_pub = orb\_advertise(ORB\_ID(vehicle\_global\_position), &global\_pos);
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);
4. **if** (updated && lidar\_valid) {
5. orb\_copy(ORB\_ID(optical\_flow), optical\_flow\_sub, &flow);
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. /\* distance to surface \*/
16. //float flow\_dist = dist\_bottom / PX4\_R(att.R, 2, 2); //use this if using sonar
17. **float** flow\_dist = dist\_bottom; //use this if using lidar
19. /\* check if flow if too large for accurate measurements \*/
20. /\* calculate estimated velocity in body frame \*/
21. **float** body\_v\_est[2] = { 0.0f, 0.0f };
22. **for** (**int** i = 0; i < 2; i++) {
23. 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];
24. /\* 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
25. \* x\_est[]是矫正之后的速度
26. \*/
27. }
29. /\* set this flag if flow should be accurate according to current velocity and attitude rate estimate \*/
30. flow\_accurate = fabsf(body\_v\_est[1] / flow\_dist - att.rollspeed) < max\_flow &&
31. 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. \*/
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;
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;
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\_flow + 1);
63. n\_flow++;
64. }

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. /\* flow\_gyrospeed[2]光流模块上测得的z轴角速度
72. \* 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
73. \* flow\_gyrospeed[2]-gyro\_offset\_filtered[2]=飞控的z轴角速度
74. \*/
75. /\* convert raw flow to angular flow (rad/s) \*/
76. //将光流转换成弧度
77. **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);
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 \* 1000000.0f) \* params.flow\_k;//for now the flow has to be scaled (to small)
89. }
90. //pitch方向的角速度>某个阈值，需要pitch角度的补偿
91. **else** {
92. //warnx("[inav] test mit comp");
93. //calculate flow [rad/s] and compensate for rotations (and offset of flow-gyro)
94. flow\_ang[0] = ((flow.pixel\_flow\_x\_integral - flow.gyro\_x\_rate\_integral) / (**float**)flow.integration\_timespan \* 1000000.0f
95. + gyro\_offset\_filtered[0]) \* params.flow\_k;//for now the flow has to be scaled (to small)
96. }
98. **if** (fabs(rates\_setpoint.roll) < rate\_threshold) {
99. flow\_ang[1] = (flow.pixel\_flow\_y\_integral / (**float**)flow.integration\_timespan \* 1000000.0f) \* params.flow\_k;//for now the flow has to be scaled (to small)
100. }
101. /\* flow.integration\_timespan
102. \* 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
103. \* accumulation timespan in microseconds
104. \* pixflow main.c中
105. \* uint32\_t deltatime = (get\_boot\_time\_us() - lasttime);
106. \* integration\_timespan += deltatime;
107. \*/
108. /\* flow.pixel\_flow\_y\_integral
109. \* I2C读取光流的时间段内每2帧图像间的光流和
110. \* accumulated optical flow in radians around y axis
111. \* pixflow main.c中
112. \* accumulated\_flow\_x += pixel\_flow\_y/focal\_length\_px \* 1.0f; //rad axis swapped to align x flow around y axis
113. \*                       x移动距离(图片)/焦距=弧度
114. \*/
115. /\* flow\_ang[]是[rad/s]，光流位移转变成弧度再除以时间\*/
116. **else** {
117. //calculate flow [rad/s] and compensate for rotations (and offset of flow-gyro)
118. flow\_ang[1] = ((flow.pixel\_flow\_y\_integral - flow.gyro\_y\_rate\_integral) / (**float**)flow.integration\_timespan \* 1000000.0f
119. + gyro\_offset\_filtered[1]) \* params.flow\_k;//for now the flow has to be scaled (to small)
120. }
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];
133. /\* velocity in NED \*/
134. **float** flow\_v[2] = { 0.0f, 0.0f };
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. }
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

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. }
157. /\* under ideal conditions, on 1m distance assume EPH = 10cm \*/
158. eph\_flow = 0.1f / w\_flow;
160. flow\_valid = **true**;
162. } **else** {
163. w\_flow = 0.0f;
164. flow\_valid = **false**;
165. }
167. flow\_updates++;
168. }