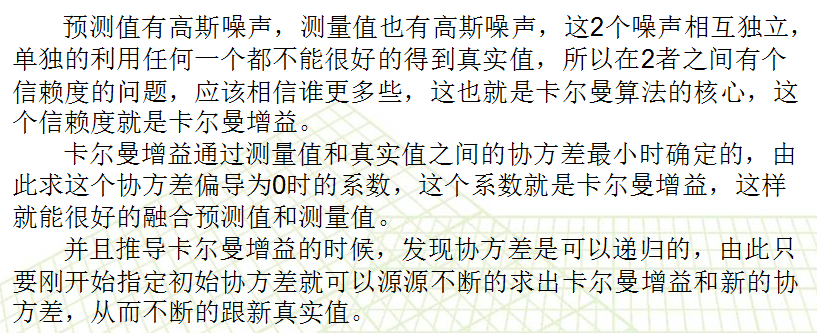
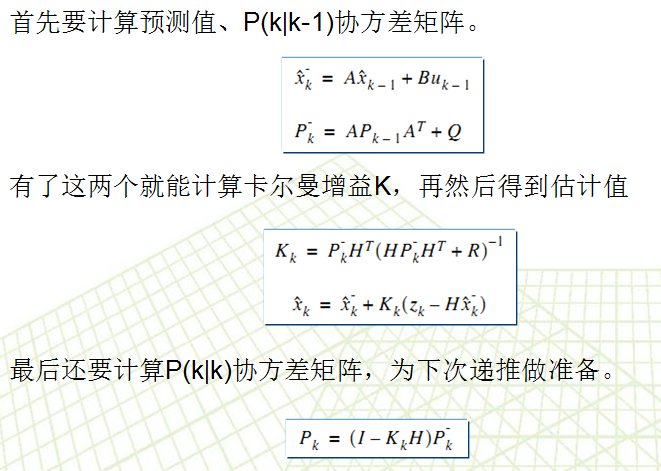
[Lacal\_position\_estimator](http://blog.csdn.net/czyv587/article/details/51814417)

1. 运行过程是什么？

看程序之前先要大概理解卡尔曼算法的原理和实现过程





int local\_position\_estimator\_thread\_main(int argc, char \*argv[])

{

warnx("starting");

using namespace control;

BlockLocalPositionEstimator est;//构造函数

thread\_running = true;

while (!thread\_should\_exit) {//不断循环

est.update();//数据计算更新

}

warnx("exiting.");

thread\_running = false;

return 0;

}

进入BlockLocalPositionEstimator est;构造函数完成矩阵初始化(构造函数是赋初始值的函数)

BlockLocalPositionEstimator::BlockLocalPositionEstimator() :

......

\_x(), \_u(), \_P()

{

......

// initialize P, x, u

initP();

\_x.setZero();

\_u.setZero();

\_flowX = 0;

\_flowY = 0;

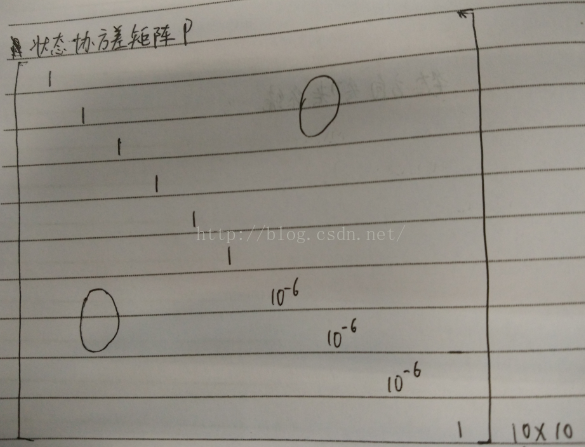
......

updateParams();

}

进入initP();

1. Matrix<**float**, n\_x, n\_x>  \_P; // state covariance matrix
2. enum {X\_x = 0, X\_y, X\_z, X\_vx, X\_vy, X\_vz, X\_bx, X\_by, X\_bz, X\_tz, n\_x};
3. enum {U\_ax = 0, U\_ay, U\_az, n\_u};
4. enum {Y\_baro\_z = 0, n\_y\_baro};
5. enum {Y\_lidar\_z = 0, n\_y\_lidar};
6. enum {Y\_flow\_x = 0, Y\_flow\_y, n\_y\_flow};
7. enum {Y\_sonar\_z = 0, n\_y\_sonar};
8. enum {Y\_gps\_x = 0, Y\_gps\_y, Y\_gps\_z, Y\_gps\_vx, Y\_gps\_vy, Y\_gps\_vz, n\_y\_gps};
9. enum {Y\_vision\_x = 0, Y\_vision\_y, Y\_vision\_z, n\_y\_vision};
10. enum {Y\_mocap\_x = 0, Y\_mocap\_y, Y\_mocap\_z, n\_y\_mocap};
11. enum {POLL\_FLOW, POLL\_SENSORS, POLL\_PARAM, n\_poll};
12. **void** BlockLocalPositionEstimator::initP()
13. {
14. \_P.setZero();
15. \_P(X\_x, X\_x) = 1;
16. \_P(X\_y, X\_y) = 1;
17. \_P(X\_z, X\_z) = 1;
18. \_P(X\_vx, X\_vx) = 1;
19. \_P(X\_vy, X\_vy) = 1;
20. \_P(X\_vz, X\_vz) = 1;
21. \_P(X\_bx, X\_bx) = 1e-6;
22. \_P(X\_by, X\_by) = 1e-6;
23. \_P(X\_bz, X\_bz) = 1e-6;
24. \_P(X\_tz, X\_tz) = 1;
25. }



进入循环est.update()

void BlockLocalPositionEstimator::update()

{

......

// get new data

updateSubscriptions();//获取传感器数据

......

predict();//一步预测，也就是模型预测

if (gpsUpdated) {

if (!\_gpsInitialized) {

gpsInit();

} else {

gpsCorrect();//矫正函数

}

}

if (baroUpdated) {

if (!\_baroInitialized) {

baroInit();

} else {

baroCorrect();//矫正函数

}

}

if (lidarUpdated) {

if (!\_lidarInitialized) {

lidarInit();

} else {

lidarCorrect();//矫正函数

}

}

if (sonarUpdated) {

if (!\_sonarInitialized) {

sonarInit();

} else {

sonarCorrect();//矫正函数

}

}

if (flowUpdated) {

if (!\_flowInitialized) {

flowInit();

} else {

perf\_begin(\_loop\_perf);// TODO

flowCorrect();//矫正函数

//perf\_count(\_interval\_perf);

perf\_end(\_loop\_perf);

}

}

if (visionUpdated) {

if (!\_visionInitialized) {

visionInit();

} else {

visionCorrect();//矫正函数

}

}

if (mocapUpdated) {

if (!\_mocapInitialized) {

mocapInit();

} else {

mocapCorrect();//矫正函数

}

}

if (\_altHomeInitialized) {

// update all publications if possible

publishLocalPos();//发布主题

publishEstimatorStatus();

if (\_canEstimateXY) {

publishGlobalPos();

}

}

......

}

进入updateSubscriptions();

1. **virtual** **void** updateSubscriptions()
2. {
3. Block::updateSubscriptions();
4. **if** (getChildren().getHead() != NULL) { updateChildSubscriptions(); }
5. }

进入Block::updateSubscriptions();

1. **void** Block::updateSubscriptions()
2. {
3. uORB::SubscriptionNode \*sub = getSubscriptions().getHead();
4. **int** count = 0;
5. **while** (sub != NULL) {
6. **if** (count++ > maxSubscriptionsPerBlock) {
7. **char** name[blockNameLengthMax];
8. getName(name, blockNameLengthMax);
9. printf("exceeded max subscriptions for block: %s\n", name);
10. **break**;
11. }
12. sub->update();
13. sub = sub->getSibling();
14. }
15. }

进入sub->update();

这里只会跟到virtual void update() = 0;然后在相对应的.cpp里面找

1. **void** SubscriptionBase::update(**void** \*data)
2. {
3. **if** (updated()) {
4. **int** ret = orb\_copy(\_meta, \_handle, data);
5. **if** (ret != PX4\_OK) { warnx("orb copy failed"); }
6. }
7. }

于是通过while (sub != NULL) {}循环，将订阅的主题都copy下来了

接下来要回到est.update()大循环中查看predict()

void BlockLocalPositionEstimator::predict()

{

// if can't update anything, don't propagate

// state or covariance

if (!\_canEstimateXY && !\_canEstimateZ) { return; }

//输入向量\_u

if (\_integrate.get() && \_sub\_att.get().R\_valid) {

Matrix3f R\_att(\_sub\_att.get().R);

Vector3f a(\_sub\_sensor.get().accelerometer\_m\_s2);

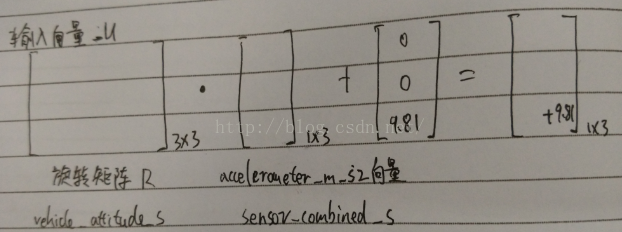
\_u = R\_att \* a;

\_u(U\_az) += 9.81f; // add g

} else {

\_u = Vector3f(0, 0, 0);

}



//动态矩阵A

// dynamics matrix

Matrix<float, n\_x, n\_x> A; // state dynamics matrix

A.setZero();

// derivative of position is velocity

A(X\_x, X\_vx) = 1;

A(X\_y, X\_vy) = 1;

A(X\_z, X\_vz) = 1;

// derivative of velocity is accelerometer acceleration

// (in input matrix) - bias (in body frame)

Matrix3f R\_att(\_sub\_att.get().R);

A(X\_vx, X\_bx) = -R\_att(0, 0);

A(X\_vx, X\_by) = -R\_att(0, 1);

A(X\_vx, X\_bz) = -R\_att(0, 2);

A(X\_vy, X\_bx) = -R\_att(1, 0);

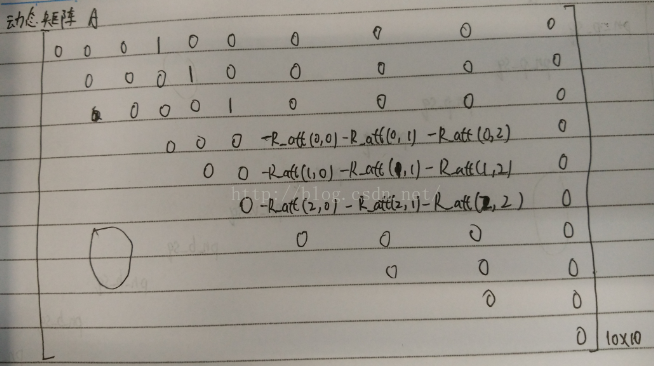
A(X\_vy, X\_by) = -R\_att(1, 1);

A(X\_vy, X\_bz) = -R\_att(1, 2);

A(X\_vz, X\_bx) = -R\_att(2, 0);

A(X\_vz, X\_by) = -R\_att(2, 1);

A(X\_vz, X\_bz) = -R\_att(2, 2);



//输入矩阵B

// input matrix

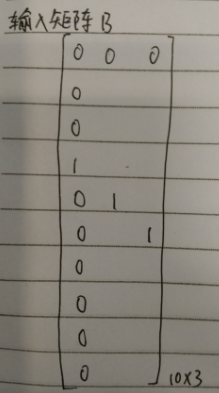
Matrix<float, n\_x, n\_u> B; // input matrix

B.setZero();

B(X\_vx, U\_ax) = 1;

B(X\_vy, U\_ay) = 1;

B(X\_vz, U\_az) = 1;



//输入噪声协方差矩阵R

// input noise covariance matrix

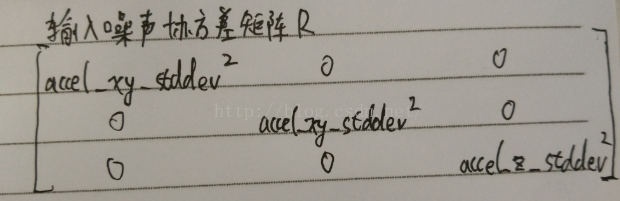
Matrix<float, n\_u, n\_u> R;

R.setZero();

R(U\_ax, U\_ax) = \_accel\_xy\_stddev.get() \* \_accel\_xy\_stddev.get();

R(U\_ay, U\_ay) = \_accel\_xy\_stddev.get() \* \_accel\_xy\_stddev.get();

R(U\_az, U\_az) = \_accel\_z\_stddev.get() \* \_accel\_z\_stddev.get();



//系统过程噪声矩阵Q

// process noise power matrix

Matrix<float, n\_x, n\_x> Q;

Q.setZero();

float pn\_p\_sq = \_pn\_p\_noise\_density.get() \* \_pn\_p\_noise\_density.get();

float pn\_v\_sq = \_pn\_v\_noise\_density.get() \* \_pn\_v\_noise\_density.get();

Q(X\_x, X\_x) = pn\_p\_sq;

Q(X\_y, X\_y) = pn\_p\_sq;

Q(X\_z, X\_z) = pn\_p\_sq;

Q(X\_vx, X\_vx) = pn\_v\_sq;

Q(X\_vy, X\_vy) = pn\_v\_sq;

Q(X\_vz, X\_vz) = pn\_v\_sq

// technically, the noise is in the body frame,

// but the components are all the same, so

// ignoring for now

float pn\_b\_sq = \_pn\_b\_noise\_density.get() \* \_pn\_b\_noise\_density.get();

Q(X\_bx, X\_bx) = pn\_b\_sq;

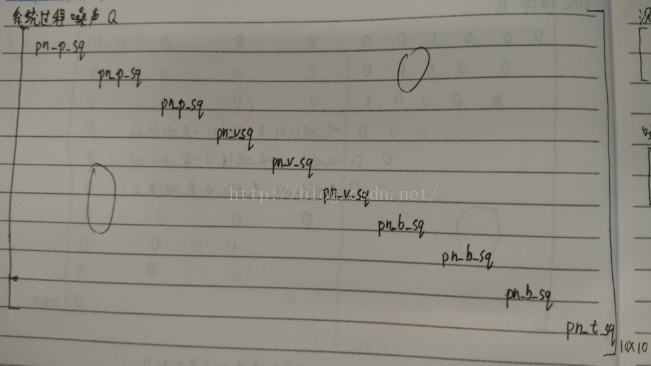
Q(X\_by, X\_by) = pn\_b\_sq;

Q(X\_bz, X\_bz) = pn\_b\_sq;

// terrain random walk noise

float pn\_t\_sq = \_pn\_t\_noise\_density.get() \* \_pn\_t\_noise\_density.get();

Q(X\_tz, X\_tz) = pn\_t\_sq;



//连续时间的卡尔曼滤波器预测值

// continuous time kalman filter prediction

Vector<float, n\_x> dx = (A \* \_x + B \* \_u) \* getDt();

//上一时刻状态推导这一时刻状态线性系统的状态



……

**//更新-x状态矩阵，-P协方差矩阵**

// propagate

**\_x += dx;//** **这个dx就是通过模型推导**

**\_P += (A \* \_P + \_P \* A.transpose() +**

**B \* R \* B.transpose() + Q) \* getDt();**



}

再回到est.update()大循环中查看Correct()，比如flowCorrect();

void BlockLocalPositionEstimator::flowCorrect()

{

// measure flow

Vector<float, n\_y\_flow> y;

if (flowMeasure(y) != OK) { return; }//获取光流数据

//光流测量矩阵C

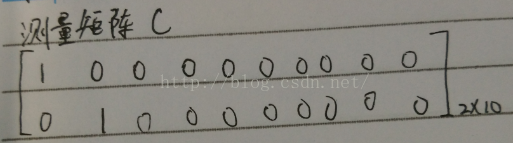
// flow measurement matrix and noise matrix

Matrix<float, n\_y\_flow, n\_x> C;

C.setZero();

C(Y\_flow\_x, X\_x) = 1;

C(Y\_flow\_y, X\_y) = 1;



//噪声矩阵R

Matrix<float, n\_y\_flow, n\_y\_flow> R;

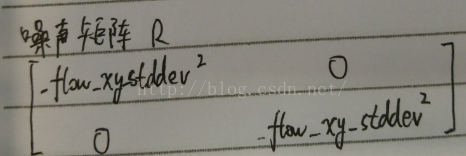
R.setZero();

R(Y\_flow\_x, Y\_flow\_x) =

\_flow\_xy\_stddev.get() \* \_flow\_xy\_stddev.get();

R(Y\_flow\_y, Y\_flow\_y) =

\_flow\_xy\_stddev.get() \* \_flow\_xy\_stddev.get();



//剩余向量r

// residual

Vector<float, 2> r = y - C \* \_x;//\_x来自上面的sonarCorrect()等

//剩余协方差(逆)

// residual covariance, (inverse)

Matrix<float, n\_y\_flow, n\_y\_flow> S\_I =

inv<float, n\_y\_flow>(C \* \_P \* C.transpose() + R);//\_P来自上面的sonarCorrect()等

//故障检测

// fault detection

float beta = (r.transpose() \* (S\_I \* r))(0, 0);

if (beta > BETA\_TABLE[n\_y\_flow]) {

if (\_flowFault < FAULT\_MINOR) {

//mavlink\_and\_console\_log\_info(&mavlink\_log\_pub, "[lpe] flow fault, beta %5.2f", double(beta));

\_flowFault = FAULT\_MINOR;

}

} else if (\_flowFault) {

\_flowFault = FAULT\_NONE;

//mavlink\_and\_console\_log\_info(&mavlink\_log\_pub, "[lpe] flow OK");

}

//光流矫正

if (\_flowFault < fault\_lvl\_disable) {

Matrix<float, n\_x, n\_y\_flow> **K =**

**\_P \* C.transpose() \* S\_I;//卡尔曼增益**

**\_x += K \* r;** //更新\_x, 状态向量

//\_x = \_x(predict求得)+K(卡尔曼增益) \* r(测量与上一状态最优值的误差)



//K =\_P \* C.transpose() \* S\_I;

// =\_P \* C.transpose() \* (C \* \_P \* C.transpose() + R)

// =协方差矩阵\*光流测量矩阵的转置\*(光流测量矩阵\*协方差矩阵\*光流测量矩阵的转置+噪声矩阵)



// r= y - C \* \_x

// =光流测量矩阵-测量矩阵\*上一状态最优值

**\_P -= K \* C \* \_P;** //更新\_P, 状态协方差矩阵

//\_P = \_P -K \* C \* \_P

// = (I-K\*C)\*\_P



} else {

// reset flow integral to current estimate of position

// if a fault occurred

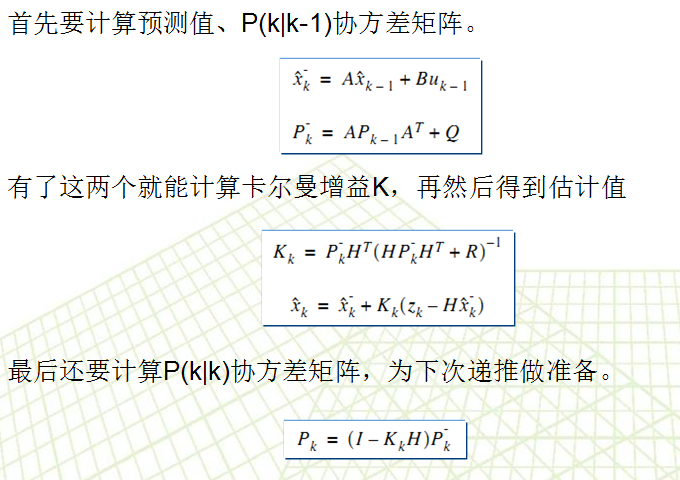
\_flowX = \_x(X\_x);

\_flowY = \_x(X\_y);

}

}

**到此卡尔曼算法完成！！其实卡尔曼算法就是以下5个公式不断迭代**



相同的，高度/GPS，求法是一样的，只是测量矩阵不一样了，噪声等“个性化”东西不一样，处理过程是一样的。

再回到est.update()大循环中发布状态

1. **if** (\_altHomeInitialized) {
2. // update all publications if possible
3. publishLocalPos();
4. publishEstimatorStatus();
5. **if** (\_canEstimateXY) {
6. publishGlobalPos();
7. }
8. }

publishLocalPos()发布这么多

1. \_pub\_lpos.get().timestamp = \_timeStamp;
2. \_pub\_lpos.get().xy\_valid = \_canEstimateXY;
3. \_pub\_lpos.get().z\_valid = \_canEstimateZ;
4. \_pub\_lpos.get().v\_xy\_valid = \_canEstimateXY;
5. \_pub\_lpos.get().v\_z\_valid = \_canEstimateZ;
6. \_pub\_lpos.get().x = \_x(X\_x);    // north
7. \_pub\_lpos.get().y = \_x(X\_y);    // east
8. \_pub\_lpos.get().z = \_x(X\_z);    // down
9. \_pub\_lpos.get().vx = \_x(X\_vx);  // north
10. \_pub\_lpos.get().vy = \_x(X\_vy);  // east
11. \_pub\_lpos.get().vz = \_x(X\_vz);  // down
12. \_pub\_lpos.get().yaw = \_sub\_att.get().yaw;
13. \_pub\_lpos.get().xy\_global = \_sub\_home.get().timestamp != 0; // need home for reference
14. \_pub\_lpos.get().z\_global = \_baroInitialized;
15. \_pub\_lpos.get().ref\_timestamp = \_sub\_home.get().timestamp;
16. \_pub\_lpos.get().ref\_lat = \_map\_ref.lat\_rad \* 180 / M\_PI;
17. \_pub\_lpos.get().ref\_lon = \_map\_ref.lon\_rad \* 180 / M\_PI;
18. \_pub\_lpos.get().ref\_alt = \_sub\_home.get().alt;
19. \_pub\_lpos.get().dist\_bottom = agl();
20. \_pub\_lpos.get().dist\_bottom\_rate = -\_x(X\_vz);
21. \_pub\_lpos.get().surface\_bottom\_timestamp = \_timeStamp;
22. \_pub\_lpos.get().dist\_bottom\_valid = \_canEstimateZ;
23. \_pub\_lpos.get().eph = sqrtf(\_P(X\_x, X\_x) + \_P(X\_y, X\_y));
24. \_pub\_lpos.get().epv = sqrtf(\_P(X\_z, X\_z));

publishEstimatorStatus()发布这么多

1. \_pub\_est\_status.get().timestamp = \_timeStamp;
2. **for** (**int** i = 0; i < n\_x; i++) {
3. \_pub\_est\_status.get().states[i] = \_x(i);
4. \_pub\_est\_status.get().covariances[i] = \_P(i, i);
5. }   \_pub\_est\_status.get().n\_states = n\_x;
6. \_pub\_est\_status.get().nan\_flags = 0;
7. \_pub\_est\_status.get().health\_flags =
8. ((\_baroFault > fault\_lvl\_disable) << SENSOR\_BARO)
9. + ((\_gpsFault > fault\_lvl\_disable) << SENSOR\_GPS)
10. + ((\_lidarFault > fault\_lvl\_disable) << SENSOR\_LIDAR)
11. + ((\_flowFault > fault\_lvl\_disable) << SENSOR\_FLOW)
12. + ((\_sonarFault > fault\_lvl\_disable) << SENSOR\_SONAR)
13. + ((\_visionFault > fault\_lvl\_disable) << SENSOR\_VISION)
14. + ((\_mocapFault > fault\_lvl\_disable) << SENSOR\_MOCAP);
15. \_pub\_est\_status.get().timeout\_flags =
16. (\_baroInitialized << SENSOR\_BARO)
17. + (\_gpsInitialized << SENSOR\_GPS)
18. + (\_flowInitialized << SENSOR\_FLOW)
19. + (\_lidarInitialized << SENSOR\_LIDAR)
20. + (\_sonarInitialized << SENSOR\_SONAR)
21. + (\_visionInitialized << SENSOR\_VISION)
22. + (\_mocapInitialized << SENSOR\_MOCAP);

publishGlobalPos()发布这么多

1. \_pub\_gpos.get().timestamp = \_timeStamp;
2. \_pub\_gpos.get().time\_utc\_usec = \_sub\_gps.get().time\_utc\_usec;
3. \_pub\_gpos.get().lat = lat;
4. \_pub\_gpos.get().lon = lon;
5. \_pub\_gpos.get().alt = alt;
6. \_pub\_gpos.get().vel\_n = \_x(X\_vx);
7. \_pub\_gpos.get().vel\_e = \_x(X\_vy);
8. \_pub\_gpos.get().vel\_d = \_x(X\_vz);
9. \_pub\_gpos.get().yaw = \_sub\_att.get().yaw;
10. \_pub\_gpos.get().eph = sqrtf(\_P(X\_x, X\_x) + \_P(X\_y, X\_y));
11. \_pub\_gpos.get().epv = sqrtf(\_P(X\_z, X\_z));
12. \_pub\_gpos.get().terrain\_alt = \_altHome - \_x(X\_tz);
13. \_pub\_gpos.get().terrain\_alt\_valid = \_canEstimateT;
14. \_pub\_gpos.get().dead\_reckoning = !\_canEstimateXY && !\_xyTimeout;
15. \_pub\_gpos.get().pressure\_alt = \_sub\_sensor.get().baro\_alt\_meter[0];
16. 对比inav和lpe算法

