基于ESKF的IMU姿态融合及MATLAB代码

zhoupian

于 2021-08-28 12:57:43 发布

目录

0 前言

1 什么是ESKF

2 系统方程

2.1 状态变量

2.2 imu的测量值

2.3 预测方程及雅克比矩阵

2.4 测量方程及雅克比矩阵

3 kalman filter loop计算

4 Show me the code

5 代码下载链接

0 前言

在很多工程应用里都需要获得物体的姿态信息，而通过imu估计姿态应该最广。

仅使用imu获取姿态信息有很多算法，本文讨论基于ESKF（Error-State Kalman Filter）的算法。

本文偏向于算法的工程应用，因此省去了公式推导，但是会列出必不可少的核心公式。

1 什么是ESKF

ESKF（Error-State Kalman Filter）看起来很吓唬人，其实它和一般的KF、EKF没有什么本质的区别，只是状态量的选取不同罢了。

以位姿估计为例，普通的kalman filter的状态量（名义状态量）一般都是位置p、速度v、四元数q，而ESKF的状态量（误差状态量）是上述状态量的误差，如位置的误差δp、速度误差δv、姿态误差δq，而kalman得5个核心公式是没有区别的。

为了方便阅读，kalman的核心公式扔在这里。

​

优点（不翻译了）

2 系统方程

2.1 状态变量

1）真实状态量：

x t = [ q t ω b t ] x\_t=

[qtωbt]

[qtωbt]

x

t

​

=[

q

t

​

ω

bt

​

​

]，q t q\_tq

t

​

为真实四元数，ω b t \omega\_{bt}ω

bt

​

为陀螺仪的真实bias。

​

2）状态量（名义状态量）：

x = [ q ω b ] x=

[qωb]

[qωb]

x=[

q

ω

b

​

​

]，q qq为四元数，ω b \omega\_{b}ω

b

​

为陀螺仪的bias。

​

3）误差状态量：

δ x = [ δ q δ ω b ] \delta x=

[δqδωb]

[δqδωb]

δx=[

δq

δω

b

​

​

]，δ q \delta qδq、δ ω b \delta\omega\_bδω

b

​

分别真实状态量和名义状态量的误差。

​

那么他们之间有这样的关系：

x t = x ⊕ δ x = [ q × δ q ω b + δ ω b ] x\_t = x\oplus\delta x=

[q×δqωb+δωb]

[q×δqωb+δωb]

x

t

​

=x⊕δx=[

q×δq

ω

b

​

+δω

b

​

​

]​

其中，δ q \delta qδq和 δ θ \delta\thetaδθ的关系为：δ q = e δ θ / 2 \delta q=e^{\delta\theta/2}δq=e

δθ/2

，因为有

q ≜ E x p ( ϕ u ) = e ϕ u / 2 = c o s ( ϕ / 2 ) + u s i n ( ϕ / 2 ) = [ c o s ( ϕ / 2 ) u s i n ( ϕ / 2 ) ] q\triangleq Exp(\phi \boldsymbol{u}) =e^{\phi \boldsymbol{u}/2} = cos (\phi/2) + \boldsymbol{u}sin(\phi/2) =

[cos(ϕ/2)usin(ϕ/2)]

[cos(ϕ/2)usin(ϕ/2)]

q≜Exp(ϕu)=e

ϕu/2

=cos(ϕ/2)+usin(ϕ/2)=[

cos(ϕ/2)

usin(ϕ/2)

​

]

也就是常用的轴角公式，由于δ θ \delta\thetaδθ只需要3个变量表示，所以误差状态量又可以表示为更简洁的形式：

δ x = [ δ θ δ ω b ] \delta x=

[δθδωb]

[δθδωb]

δx=[

δθ

δω

b

​

​

]

​

2.2 imu的测量值

1）加速度计

a t = a m + a n a\_t=a\_m+a\_na

t

​

=a

m

​

+a

n

​

其中 a t a\_ta

t

​

为真值，a m a\_ma

m

​

为测量值， a n a\_na

n

​

为测量噪声。

​

2）陀螺仪

ω t = ω m + ω n \omega\_t = \omega\_m + \omega\_nω

t

​

=ω

m

​

+ω

n

​

其中 ω t \omega\_tω

t

​

为真值，ω m \omega\_mω

m

​

为测量值， ω n \omega\_nω

n

​

为测量噪声。

2.3 预测方程及雅克比矩阵

这里我们是用陀螺仪的数据对姿态进行预测。

对于名义状态量，有：

{ q ˙ = 1 2 q ⊗ ( ω m − ω b ) ω ˙ b = 0 \left \{

q˙=12q⊗(ωm−ωb)ω˙b=0

q˙=12q⊗(ωm−ωb)ω˙b=0

\right.{

q

˙

​

=

2

1

​

q⊗(ω

m

​

−ω

b

​

)

ω

˙

b

​

=0

​

对于误差状态量，有：

{ δ θ ˙ = − [ ω m − ω b ] × δ θ − δ ω b − ω n δ ω b ˙ = ω ω \left \{

δθ˙=−[ωm−ωb]×δθ−δωb−ωnδωb˙=ωω

δθ˙=−[ωm−ωb]×δθ−δωb−ωnδωb˙=ωω

\right.{

δθ

˙

=−[ω

m

​

−ω

b

​

]

×

​

δθ−δω

b

​

−ω

n

​

δω

b

​

˙

​

=ω

ω

​

​

上式中，

ω n \omega\_nω

n

​

为陀螺仪数据噪声，ω ω \omega\_{\omega}ω

ω

​

为陀螺仪零偏的噪声（仔细品味一下，象征陀螺仪零偏的稳定性）；[ ∙ ] × [\bullet]\_{\times}[∙]

×

​

为反对称矩阵。

这里，我们把误差状态的系统方程的一般形式表示为：

δ x ← f ( x , δ x , u m , i ) = F x ( x , u m ) ⋅ δ x + F i ⋅ i \delta x \gets f(x,\delta x, u\_m, i) = F\_x(x, u\_m) \cdot \delta x + F\_i \cdot iδx←f(x,δx,u

m

​

,i)=F

x

​

(x,u

m

​

)⋅δx+F

i

​

⋅i

上式中，u m u\_mu

m

​

为系统的输入，i ii为噪声。

​

那么其预测方程可以写为：

δ x ^ ← F x ( x , u m ) ⋅ δ x ^ \hat{\delta x} \gets F\_x(x, u\_m) \cdot \hat{\delta x}

δx

^

←F

x

​

(x,u

m

​

)⋅

δx

^

P ← F x P F x T + F i Q i F i T P \gets F\_xPF^T\_x + F\_iQ\_iF^T\_iP←F

x

​

PF

x

T

​

+F

i

​

Q

i

​

F

i

T

​

其中:

F x = ∂ f ∂ δ x ∣ x , u m = [ R T { ( ω m − ω b ) Δ t } − I Δ t 0 I ] F\_x = \frac{\partial f}{\partial \delta x} \lvert\_{x,u\_m} =

[RT{(ωm−ωb)Δt}0−IΔtI]

[RT{(ωm−ωb)Δt}−IΔt0I]

F

x

​

=

∂δx

∂f

​

∣

x,u

m

​

​

=[

R

T

{(ω

m

​

−ω

b

​

)Δt}

0

​

−IΔt

I

​

]

上式中，R { u } = I + s i n ϕ [ u ] × + ( 1 − c o s ϕ ) [ u ] × 2 R\{\boldsymbol{u}\} = I + sin \phi[\boldsymbol{u}]\_{\times} + (1-cos \phi)[\boldsymbol{u}]^2\_{\times}R{u}=I+sinϕ[u]

×

​

+(1−cosϕ)[u]

×

2

​

，有[ u ] × 2 = u u T − I [\boldsymbol{u}]^2\_{\times} = \boldsymbol{u}\boldsymbol{u}^T-I[u]

×

2

​

=uu

T

−I

F i = ∂ f ∂ i ∣ x , u m = [ I 0 0 I ] F\_i = \frac{\partial f}{\partial i} \lvert\_{x,u\_m}=

[I00I]

[I00I]

F

i

​

=

∂i

∂f

​

∣

x,u

m

​

​

=[

I

0

​

0

I

​

]

Q i = [ σ ω n 2 Δ t 2 I 0 0 σ ω ω 2 Δ t I ] Q\_i =

[σ2ωnΔt2I00σ2ωωΔtI]

[σωn2Δt2I00σωω2ΔtI]

Q

i

​

=[

σ

ω

n

​

2

​

Δt

2

I

0

​

0

σ

ω

ω

​

2

​

ΔtI

​

]​

需要注意的是，δ x \delta xδx不需要做预测，因为每次初始化δ x \delta xδx都为0，所以预测值肯定也是0。

有人会问，它会一直是0吗，经过几次运算迭代一直是0？

如果有这样的疑问，那就说明还没有从常规kalman filter的模式中走出来。ESKF每次迭代后，都会将最优估计得到的δ x \delta xδx叠加到x xx中，所以对于下一时刻的开始，认为x xx是没有误差的，也就是认为δ x \delta xδx为0了。

​

总结，在kalman预测这一步，每一轮新的kalman迭代，需要做的事情是：

1）对名义状态量进行预测；

3）计算雅克比矩阵

2）对误差状态量进行预测；（δ x \delta xδx的值是0，可以不计算，但是不能认为它没有意义）

2.4 测量方程及雅克比矩阵

观测方程可以表示为下面的形式

y = h ( x t ) + v y=h(x\_t)+vy=h(x

t

​

)+v

这里使用imu的加速度进行姿态的校正，假设的前提是没有机动加速度，只有重力加速度，所以有

y = a m = − R t T g + a n y=a\_m=-R^T\_t \boldsymbol{g} + a\_ny=a

m

​

=−R

t

T

​

g+a

n

​

，其中R t T R\_t^TR

t

T

​

是通过q t q\_tq

t

​

得到的旋转矩阵的转置。

​

带入R T R^TR

T

，可以把上式化简为：

y = h ( x t ) + v = − R t T g + a n = − R t T [ 0 0 g ] + a n = [ 2 ( q x q z − q w q y ) 2 ( q y q z + q w q x ) q w 2 − q x 2 − q y 2 + q z 2 ] ( − g ) + a n

y=h(xt)+v=−RTtg+an=−RTt⎡⎣⎢00g⎤⎦⎥+an=⎡⎣⎢2(qxqz−qwqy)2(qyqz+qwqx)q2w−q2x−q2y+q2z⎤⎦⎥(−g)+an

y=h(xt)+v=−RtTg+an=−RtT[00g]+an=[2(qxqz−qwqy)2(qyqz+qwqx)qw2−qx2−qy2+qz2](−g)+an

y

​

=h(x

t

​

)+v

=−R

t

T

​

g+a

n

​

=−R

t

T

​

⎣

⎡

​

0

0

g

​

⎦

⎤

​

+a

n

​

=

⎣

⎡

​

2(q

x

​

q

z

​

−q

w

​

q

y

​

)

2(q

y

​

q

z

​

+q

w

​

qx)

q

w

2

​

−q

x

2

​

−q

y

2

​

+q

z

2

​

​

⎦

⎤

​

(−g)+a

n

​

​

然后将h ( x t ) h(x\_t)h(x

t

​

)对δ x \delta xδx求偏导，即

H ≜ ∂ h ∂ δ x ∣ x = ∂ h ∂ x t ∣ x ∂ x t ∂ δ x ∣ x = H x X δ x H \triangleq \frac{\partial h}{\partial \delta x}|\_x = \frac{\partial h}{\partial x\_t}|\_x \frac{\partial x\_t}{\partial \delta x}|\_x = H\_x X\_{\delta x}H≜

∂δx

∂h

​

∣

x

​

=

∂x

t

​

∂h

​

∣

x

​

∂δx

∂x

t

​

​

∣

x

​

=H

x

​

X

δx

​

​H x = ∂ h ∂ δ x ∣ x = [ − 2 q y 2 q z − 2 q w 2 q x 2 q x 2 q w 2 q z 2 q y 2 q w − 2 q x − 2 q y 2 q z ] ( − g ) H\_x = \frac{\partial h}{\partial \delta x}|\_x =

⎡⎣⎢−2qy2qx2qw2qz2qw−2qx−2qw2qz−2qy2qx2qy2qz⎤⎦⎥

[−2qy2qz−2qw2qx2qx2qw2qz2qy2qw−2qx−2qy2qz]

(-g)H

x

​

=

∂δx

∂h

​

∣

x

​

=

⎣

⎡

​

−2q

y

​

2q

x

​

2q

w

​

​

2q

z

​

2q

w

​

−2q

x

​

​

−2q

w

​

2q

z

​

−2q

y

​

​

2q

x

​

2q

y

​

2q

z

​

​

⎦

⎤

​

(−g)

X δ x = [ ∂ ( q ⊗ δ q ) ∂ δ θ 0 0 ∂ ( ω b + δ ω b ) ∂ δ ω b ] = [ Q δ θ 0 0 I ] X\_{\delta x} =

⎡⎣∂(q⊗δq)∂δθ00∂(ωb+δωb)∂δωb⎤⎦

[∂(q⊗δq)∂δθ00∂(ωb+δωb)∂δωb]

=

[Qδθ00I]

[Qδθ00I]

X

δx

​

=[

∂δθ

∂(q⊗δq)

​

0

​

0

∂δω

b

​

∂(ω

b

​

+δω

b

​

)

​

​

]=[

Q

δ

θ

​

​

0

​

0

I

​

]

Q δ θ = 1 2 [ − q x − q y − q z q w − q z q y q z q w − q x − q y q x q w ] Q\_{\delta \theta} = \frac{1}{2}

⎡⎣⎢⎢⎢⎢−qxqwqz−qy−qy−qzqwqx−qzqy−qxqw⎤⎦⎥⎥⎥⎥

[−qx−qy−qzqw−qzqyqzqw−qx−qyqxqw]

Q

δθ

​

=

2

1

​

⎣

⎢

⎢

⎡

​

−q

x

​

q

w

​

q

z

​

−q

y

​

​

−q

y

​

−q

z

​

q

w

​

q

x

​

​

−q

z

​

q

y

​

−q

x

​

q

w

​

​

⎦

⎥

⎥

⎤

​

3 kalman filter loop计算

上面的公式太多了，看得人头皮发麻，那就赶紧进入算法实践环节，接下来就可以按照Kalman Filter的“套路”计算了，大致的流程如下。

上图中的每一个公式或者变量都可以在前文中找到定义和说明，你要做的就是按照上述的流程进行计算就行。

4 Show me the code

有人说，没有代码代码，还是太抽象，好吧，这里奉上MATLAB脚本和仿真数据。

代码文件的结构如下。

​仿真结果如下。

%% 使用IMU的数据，基于ESKF的方法估计姿态

clc;clear;close;

%% 添加路径，加载数据

addpath('ESKF')

load('imu\_log\_data.mat');

mean\_dt = mean(diff(imu.t));

%% 调用算法，记录数据

eskf.t = imu.t;

[eskf.quat, eskf.gyb, eskf.cov] = eskf\_imu(imu.t, imu.gyr, imu.acc);

[eskf.yaw, eskf.pitch, eskf.roll] = quat2angle(eskf.quat,'ZYX');

%% 画图

figure('Name', 'euler in degree');

h1 = subplot(3,1,1);

plot(eskf.t, rad2deg(eskf.roll), 'r');

grid minor;title('roll')

h2 = subplot(3,1,2);

plot(eskf.t, rad2deg(eskf.pitch), 'r');

grid minor;title('pitch')

h3 = subplot(3,1,3);

plot(eskf.t, rad2deg(eskf.yaw), 'r');

grid minor;title('yaw')

linkaxes([h1 h2 h3],'x');

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

​

function [state\_quat, state\_gyb, state\_cov] = eskf\_imu(t, gyr, acc)

%% 参数初始化

quat = quaternion(1,0,0,0);

init\_quat\_var = power(4, 2);

init\_gyb\_var = power(1e-3, 2);

% 初始化P矩阵

P(1:3,1:3) = init\_quat\_var \* eye(3);

P(4:6,4:6) = init\_gyb\_var \* eye(3);

acc\_meas\_noise\_var = power(2, 2); % 加速度计噪声方差

R = eye(3) \* acc\_meas\_noise\_var;

gyr\_noise = power(1e-2, 2);

gyr\_drift\_noise = power(1e-3, 2);

gyb = zeros(1,3);

%% 申请内存，可以加快仿真速度

len = length(gyr);

state\_quat = zeros(len,4);

state\_gyb = zeros(len,3);

state\_cov = zeros(len,6);

%% 循环计算

for index = 1:len

if (index == 1)

dt = saturate\_user(t(2) - t(1),0.001,0.05);

else

dt = saturate\_user(t(index) - t(index-1),0.001,0.05);

end

% ---- 预测 ----- %

% 预测姿态角

d\_angle = (gyr(index,:) - gyb) \* dt;

dq = quaternion(d\_angle, 'rotvec'); % 将旋转向量转化为四元数

new\_quat = quat \* dq;

quat = new\_quat.normalize;

% 预测协方差

P = ESKF\_predict\_P(dt, P, gyr\_noise, gyr\_drift\_noise);

% ---- 更新 ----- %

y = ESKF\_predict\_acc(quat); %

H = ESKF\_jacH(quat);

S = H \* P \* H' + R;

K = P \* H' / S;

P = P - K \* H \* P;

innov = K \* (acc(index,:) - y)';

dq = quaternion(innov(1:3)', 'rotvec');

new\_quat = quat \* dq;

quat = new\_quat.normalize;

gyb = gyb + innov(4:6)';

state\_quat(index,:) = quat.compact;

state\_gyb(index,:) = gyb;

state\_cov(index,:) = diag(P)';

end

end

function out = saturate\_user(in, min, max)

if (in < min)

out = min;

elseif (in > max)

out = max;

else

out = in;

end

end

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

​

function new\_P = ESKF\_predict\_P(dt, P, gyro\_noise, gyr\_drift\_noise)

Fx = eye(6);

Fx(1:3,4:6) = -eye(3) \* dt;

Fi = eye(6);

Qi(1:3, 1:3) = eye(3) \* gyro\_noise \* dt \* dt;

Qi(4:6, 4:6) = eye(3) \* gyr\_drift\_noise \* dt;

new\_P = Fx \* P \* transpose(Fx) + Fi \* Qi \* transpose(Fi);

end

1

2

3

4

5

6

7

8

9

10

​

​

function pred\_meas = ESKF\_predict\_acc(quat)

[qw, qx, qy, qz] = parts(quat);

gravity = 9.80665;

pred\_meas(1) = -gravity \* 2 \* (qx \* qz - qw \* qy);

pred\_meas(2) = -gravity \* 2 \* (qw \* qx + qy \* qz);

pred\_meas(3) = -gravity \* (qw \* qw - qx \* qx - qy \* qy + qz \* qz);

end

1

2

3

4

5

6

7

8

9

​

function H = ESKF\_jacH(quat)

[qw, qx, qy, qz] = parts(quat);

Hx = zeros(3,7);

Xdx = zeros(7,6);

Hx(1:3,1:4) = -9.80665 \* 2 \* [ -qy, qz, -qw, qx;...

qx, qw, qz, qy;...

qw, -qx, -qy, qz];

Xdx(1:4,1:3) = 0.5 \* [-qx, -qy, -qz;...

qw, -qz, qy;...

qz, qw, -qx;...

-qy, qx, qw];

Xdx(5:7,4:6) = eye(3);

H = Hx \* Xdx;

end

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

​

如果还有什么不清楚，欢迎评论区交流。

​

5 代码下载链接

可以在这里下载。链接: 点击下载。

​

————————————————

版权声明：本文为CSDN博主「zhoupian」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/zhoupian/article/details/117236240