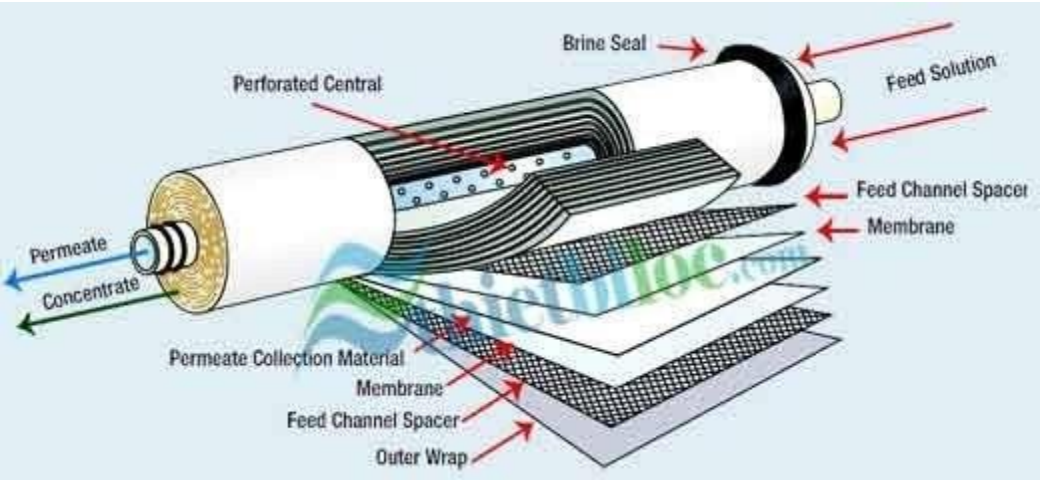
Kalman filter hay Complementary filter

ĐẶNG LÂM TÙNG·THỨ TƯ, 1 THÁNG 5, 2019

Bài viết này sẽ cung cấp một số khái niệm về Kalman và Complementary , hai bộ lọc phổ biến và đơn giản nhất

I. Vì sao ta cần một bộ lọc

Nếu như con người có 5 giác quan để quan sát và cảm nhận thế giới thì máy tính, hệ thống điều khiển, bla bla gì đó,... nếu như mắt người có quáng gà, tai người có lãng tai thì máy móc hẳn nhiên cũng phải có sai sót :>, đó là chưa kể đến môi trường có những thứ có thể ảnh hưởng đến sensor và có thể gây ra sai sót gì đó. Chính vì thế ta cần một bộ lọc

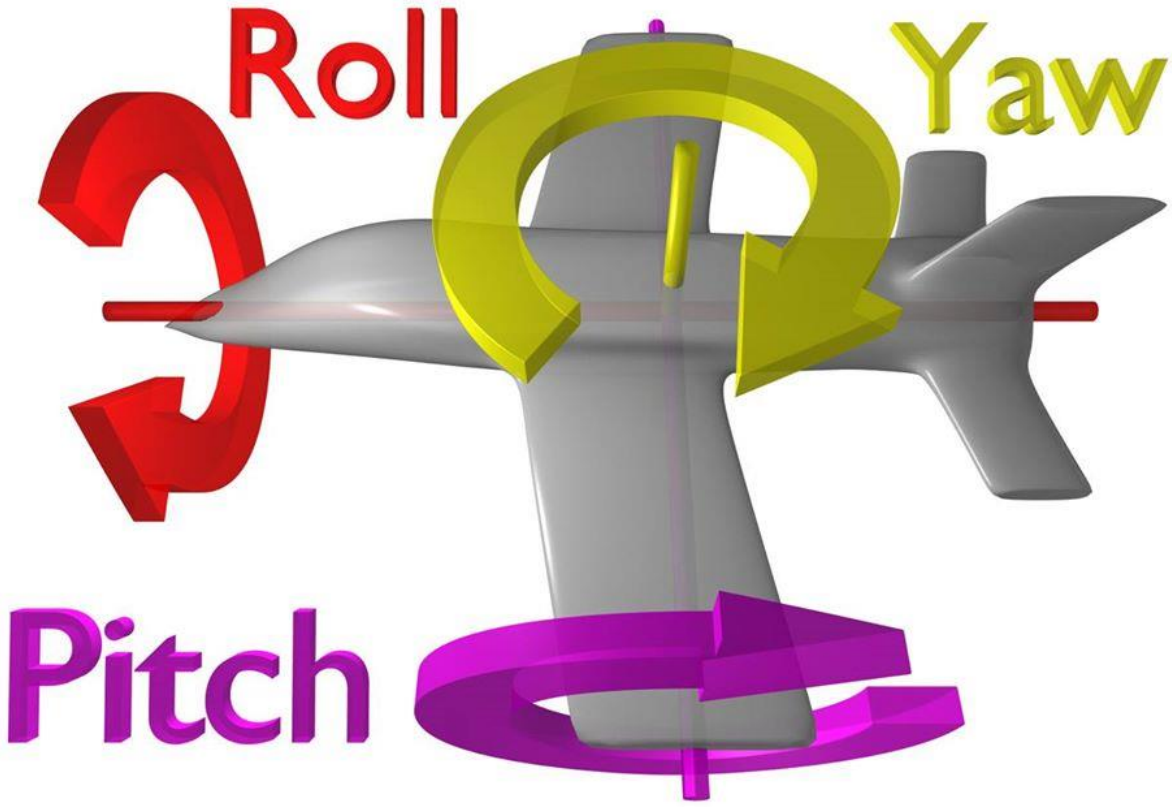


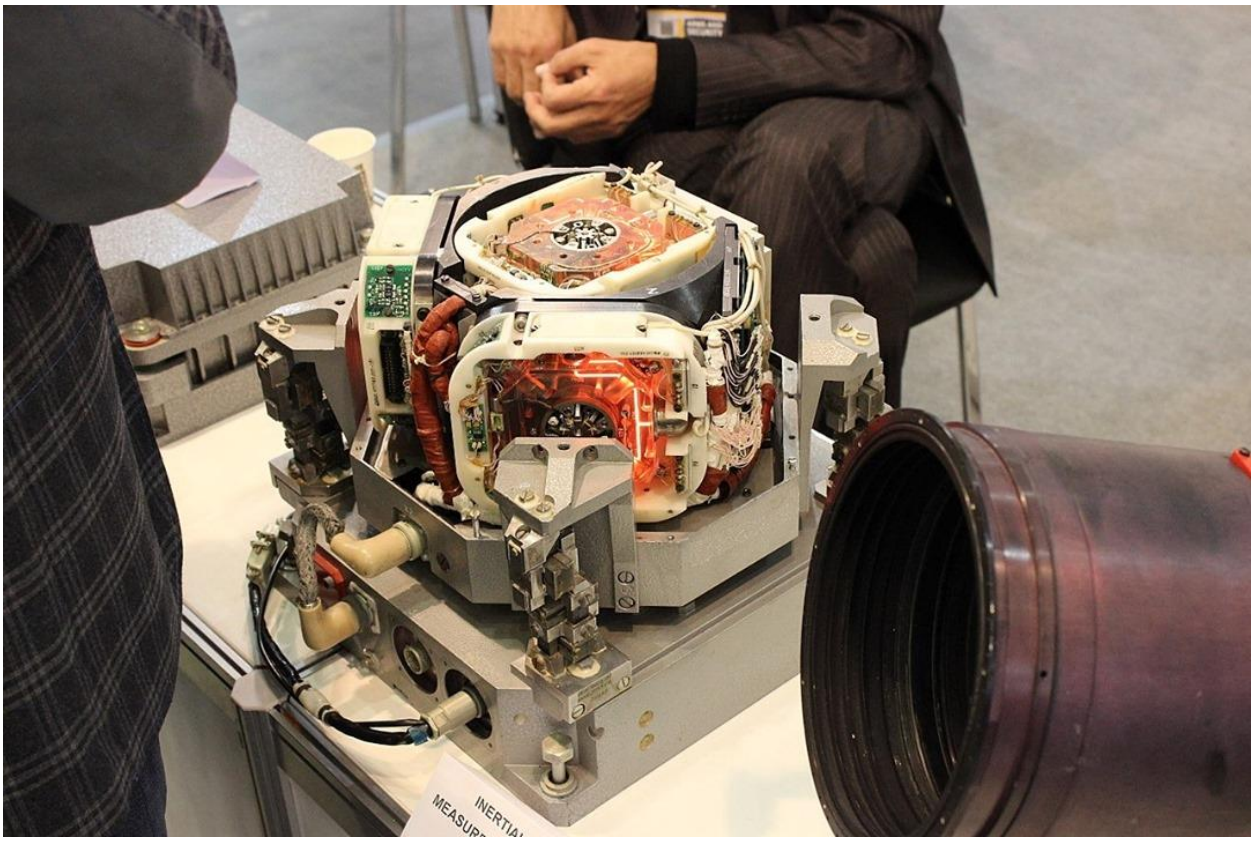
Bộ lọc RO này khá là tốt

Nếu bạn đã trải qua vài kì học ở BK, chắc hẳn bạn đã biết đến lọc thông cao, thông thấp, trung vị, butterworth, bla bla,... thì chúc mừng, bạn đã biết lọc là cái gì.

Vì vậy chủ đề chính bắt đầu nào Ò w Ó

Chú ý nho nhỏ, ví dụ về bộ lọc cho bài này là về bài toán IMU - Inertial measurement unit hay nói cho đơn giản là bộ cảm biến xác đinh vị trí nội tại của vật trong không gian, khác với SLAM ở chỗ nó chỉ cho thấy vị trí nội tại(tìm các góc yaw, pitch, roll) chứ không phải là vị trí trong không gian của vật, một điển hình ứng dụng là hệ Gimbal và ứng dụng trong các hệ GPS, trên tàu vũ trụ





Một bộ IMU trên tàu vũ trụ

II. Giới thiệu sơ lược về các cảm biến trong hệ IMU

Hệ IMU thường có 2 loại cảm biến là Gyroscope và Accelerometer, thỉnh thoảng có thêm Magnetometer.

1. Về Gyroscope:

Nếu ai đã từng chơi con quay thì đều biết là chỉ cần cung cấp một vận tốc đủ nhanh thì con quay sẽ tiếp tục giữ được vị trí trục của nó trong không gian, từ đó ta có thể xác định được *tốc độ góc* của một vật thể trong không gian

https://www.youtube.com/watch?v=DsOU0WEc6OU

Một video nói về sự vip của Gyro OwO

Gần đây với sự phát triển của công nghệ, đặc biệt là Microelectromechanical systems (hệ thống cơ điện siêu nhỏ), những chiếc gyroscope cỡ nhỏ được đưa vào sử dụng trên điện thoại, drone, thiết bị GPS cầm tay.

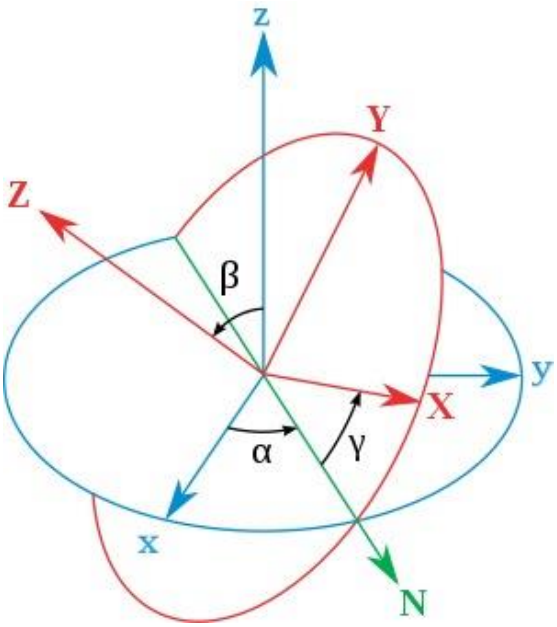
Hiểu cho đơn giản thì MEMS gyro là một quả nặng được thiết kế có hình dạng xác định, sẽ giao động với 1 tần số xác định, khi ta quay cảm biến theo 1 góc nào đó thì nó sẽ thay đổi dòng điện Piezo (dòng rung, giống như dòng trong thạch anh https://en.wikipedia.org/wiki/Piezoelectricity) và từ đó ta đo được tốc độ góc

https://www.youtube.com/watch?v=zwe6LEYF0j8

Để hiểu thêm về mấy cái MEMS gyro có thể xem thêm video này

https://www.youtube.com/watch?v=5BWerr7rJmU

Tính chất của gyroscope là tính đáp ứng nhanh, đồng thời không nhạy cảm với gia tốc như Accelerometer. Vì gyroscope đo vận tốc góc nên ta cần tích phân giá trị của Gyroscope theo thời gian để được giá trị góc Euler. Đã nghe đến chứ *tích là sẽ có thêm lỗi đó xD mình nghe nói là hiện tượng “trôi” góc là do khâu tích phân này gây ra 🫠.
Ngoài ra nhiều khi cuộc sống nó không giống cuộc đời, quả nặng của Gyro nó bị điêu điêu, sẽ sinh ra Gyro bias, tức điểm 0 bị lệch đi, trạng thái đứng yên của Gyro thực chất lại đo ra số :> nên phải đo và trừ đi bias



Góc Euler

2. Accelerometer

Gia tốc kế lại là một câu chuyện khác OwO. Nghe gia tốc chắc ai cũng biết nó đo gia tốc rồi đúng ko :> nếu các bạn đã học vật lý thì sẽ biết gia tốc trọng trường là g ~ 9.8m/s^2, nếu ta chiếu góc của gia tốc ấy lên 3 trục x,y,z thì theo nguyên tắc cộng vector, ta có thể tính được góc của vật thể (cái này là lý lớp 10 nhé :>) à mà đằng nào cũng nên show phương trình ra cho vui OwO

Chiếu 3 gia tốc trọng trường lên 3 trục là x,y,z:

$$\boldsymbol{G}_p = \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \boldsymbol{R}(\boldsymbol{g} - \boldsymbol{a}_r)$$

Gp là vector gia tốc trọng trường, R là ma trận chuyển cơ sở từ hệ tọa độ Descartes sang hệ tọa độ góc Euler, g là gia tốc trọng trường, ar là gia tốc tuyến tính (cái này khi sensor di chuyển sẽ đo ra gia tốc này, méo có cách trừ nó đi nên ta coi nó = 0)

Theo từng trục ta có các ma trận chuyển cơ sở sau:

$$\boldsymbol{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}$$

$$\boldsymbol{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$\boldsymbol{R}_z(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Nó nói chung là mấy cái ma trận quay thôi :> học đstt rồi :>0

Vì đang xét 3 trục, và gia tốc trọng trường hướng tâm nên ta có gia tốc theo z = g, vì vậy ma trận quay là

$$\boldsymbol{R}_{xyz}\begin{pmatrix}0\\0\\1\end{pmatrix} = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)\begin{pmatrix}0\\0\\1\end{pmatrix} \qquad \textit{Eqn. 6}$$

$$= \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\theta\sin\phi \\ \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix} \qquad \textit{Eqn. 7}$$

$$= \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \qquad \textit{Eqn. 8}$$

Thực ra quay theo xyz khác zxy yxz,... lắm mà lười nên vậy nhá xD
Thay cái của nợ Rxyz vào phương trình đầu tiên, ta sẽ được:

$$\boldsymbol{R}_{xyz}\begin{pmatrix}0\\0\\1\end{pmatrix} = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)\begin{pmatrix}0\\0\\1\end{pmatrix} \qquad \textit{Eqn. 6}$$

$$= \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\theta\sin\phi \\ \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\theta\cos\phi \end{pmatrix}\begin{pmatrix}0\\0\\1\end{pmatrix} \qquad \textit{Eqn. 7}$$

$$= \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \qquad \textit{Eqn. 8}$$

Chú ý là g vế phải chuyển qua là thành cái căn dưới mẫu

$$\frac{\boldsymbol{G}_p}{\|\boldsymbol{G}_p\|} = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \Rightarrow \frac{1}{\sqrt{G_{px}{}^2 + G_{py}{}^2 + G_{pz}{}^2}} \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} \qquad \textbf{Eqn. 24}$$

Đến đây tách từng hành ra và giải, thì có

$$\tan\phi_{xyz} = \left(\frac{G_{py}}{G_{pz}}\right) \qquad \textbf{Eqn. 25}$$

Là lấy 2 hàng cuối chia 2 vế cho nhau

$$\tan\theta_{xyz} = \left(\frac{-G_{px}}{G_{py}\sin\phi + G_{pz}\cos\phi}\right) = \frac{-G_{px}}{\sqrt{G_{py}{}^2 + G_{pz}{}^2}} \qquad \textbf{Eqn. 26}$$

Đoạn này là hmm :> chắc tự nghĩ đi choi vui xD

Bằng 1 mớ não ta đã tìm ra được góc bằng Accelerometer :> nhưng vấn đề là thêm 1 cái gia tốc vào là nát bấy :> vì vậy ta cần thêm vào cái góc của Gyro, và đó là lý do mà 2 cái này nó hay đi với nhau.

À.. nguyên lý hoạt động là nó có 1 cái quả nặng thiết kế đặc biệt, khi có gia tốc vào thì nó sẽ làm thay đổi khoảng cách đến mấy phần khác và thay đổi dung kháng, thành ra đo được gia tốc:

https://www.youtube.com/watch?v=T_iXLNkkjFo

3. Magnetometer

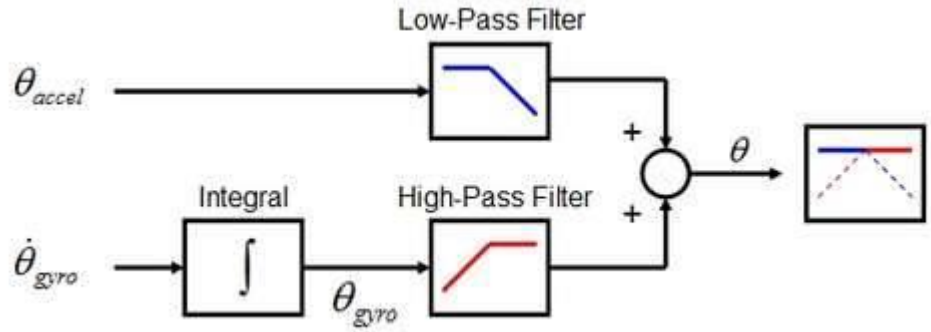
Cái này mình chưa làm nên lười chém lắm OwO hy vọng có ai tài trợ cho cái làm chơi OwO

III. Complementary filter

Nếu đã đọc đến đây thì chắc các bạn cũng hiểu vì sao phải lọc liếc các kiểu tùm lum rồi :>, mỗi sensor có một cái mạnh riêng, và nó cần kết hợp với nhau để cho kết quả *chấp nhận được*.

Nguyên nhân gây nhiễu rất nhiều, có thể là nhà sản xuất làm sai cái gì đó, vận chuyển hư mịa cái quả nặng trong cảm biến, trái gió trở trời, bla bla, nên nói chung là lọc nhẹ đi :>

Ý tưởng của Complementary filter là kết hợp 2 cái giá trị góc của cảm biến lại, một cảm biến là khâu vi phân, 1 cảm biến là khâu tỉ lệ, Complementary filter sẽ kết hợp lọc thông cao và thông thấp lại trên 2 tín hiệu này:



Ò thì thông cao thông thấp thôi :> và cách cài là:

angle_k = (angle_k-1 + gyro_k-1*dt)*alpha + accel_angle_k-1*(1-alpha)

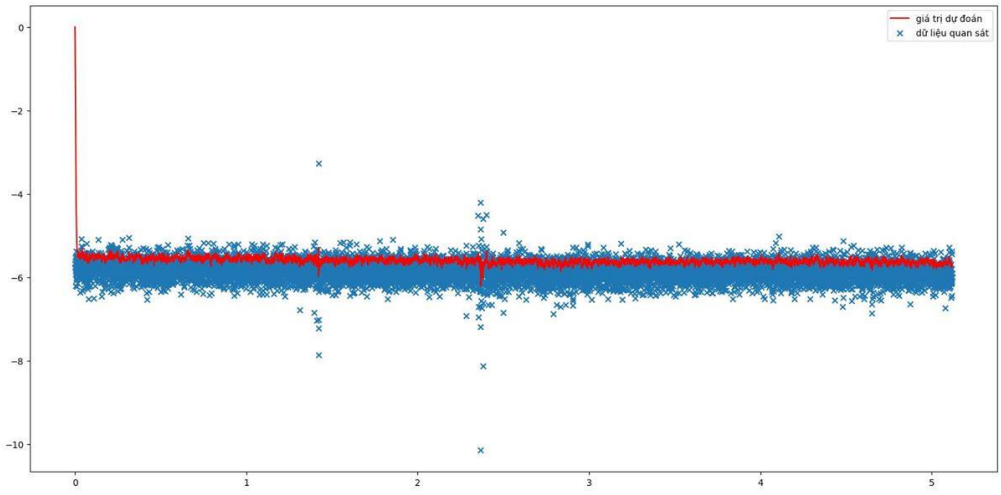
Với angle là góc dự đoán trước, cộng với gyro_k-1*dt để dự đoán góc tiếp theo, accel_angle_k-1 là góc roll hoặc pitch đã tính theo công thức ở trên.

alpha ở đây được tính dựa trên bộ lọc thông thấp như sau:

$$\tau = \frac{a \cdot dt}{1-a} \leftrightarrow a = \frac{\tau}{\tau + dt}$$

Lúc đầu mình đã nghĩ t là thời hằng cho bộ lọc thông thấp, tuy nhiên ý nghĩa của t trong đây lại rất khác so với thời hằng của lọc thông thấp.

Trong tài liệu này https://drive.google.com/file/d/0B9rLLz1XQKmaLVJLSkRwMTU0b0E/view có giải thích như sau: “Thời hằng thể hiện việc bạn *tin tưởng* vào cảm biến nào hơn, ví dụ như thời hằng nhỏ hơn 0.5s sẽ giảm drift của gyro và lọc được nhiễu trục ngang của accel, việc chọn thời hằng thực chất phải dựa vào kinh nghiệm, ví dụ như gyro trượt tầm 2 độ/s thì thời hằng phải <1s để giảm thiểu việc trượt của gyro, nhưng mà t quá nhỏ lại dẫn đến nhiều nhiễu ngang qua hơn”



Nhiễu ngang

Nói chung là khá không giống với low-pass, theo mình nghĩ thì alpha đại diện mức “bias” cho từng cảm biến, càng bias cho gyro thì càng drift và ngược lại càng bias cho accel thì lại càng bị nhiễu ngang.

Việc thiết kế lọc bù này cần thí nghiệm :>

*Một ví dụ về lọc bù:

Phần cứng:

MPU 6050 6 DOF IMU gồm accel và gyro

MPU 6050 đã tích hợp sẵn ADC,bộ lọc và giao tiếp I2C nên việc của chúng ta chỉ là đọc I2C bằng một dòng vđk bất kì

MCU mình chọn là Atmegea8 PU, chạy với tần số 1MHZ, tần số đọc I2C là 50kHz, giao tiếp với máy tính thông qua UART baud 9600

Lưu ý là module 6050 đã có sẵn trở kéo nên đoạn này không cần quan tâm

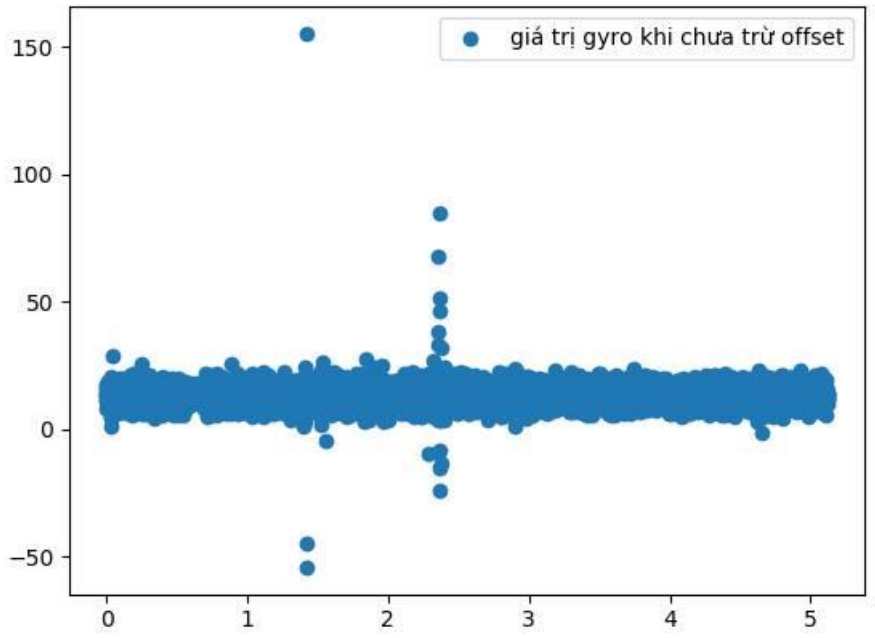
Vì bản chất của I2C và thời gian đo của MPU không chắc nên mình dùng 1 timer để đo thời gian lấy mẫu và trả về của MPU, lưu ý là thời gian trả về luôn vì đến lúc đọc được số liệu ta mới làm việc được. Về việc đọc MPU không nằm trong phạm vi bài này. Thời gian lấy mẫu đo được là: dt = 0.000512 (s)

Phần mềm:

Mình dùng AVR code trên Atmel Studio và Python để vẽ các đồ thị, đọc Serial :> (thói quen khi làm machine learning rồi :>)

Các bước thiết kế lọc bù trong trường hợp này có thể tổng kết như sau:

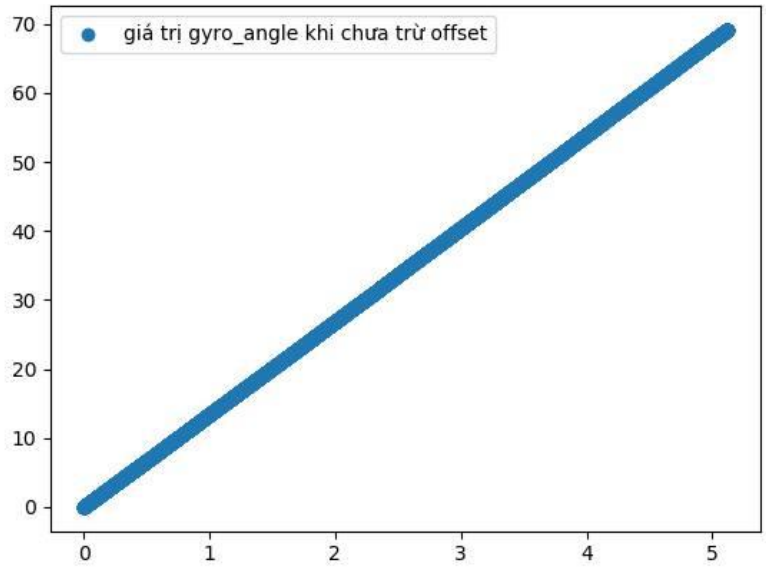
1. Thực hiện thí nghiệm đo thời gian lấy mẫu (dt), đã giới thiệu ở trên
2. Thực hiện đo các thông số Gyro tĩnh để tìm bias, xác định mức độ nhiễu ngang của accel, đồ thị dưới đây mô tả giá trị của Gyro trong 10000 lấy mẫu ở trạng thái tĩnh. Các bạn có thể thấy là raw data không hề bị drift. Dùng hàm của numpy (tụ gg nhé ahihi) thì tìm được (học xstk rồi luôn nhé ahhihi):



gyro_mean = 13.502737000000003 lấy đây làm offset (bias)

gyro_std = 3.6784756678318535 giá trị này sẽ cần cho Kalman

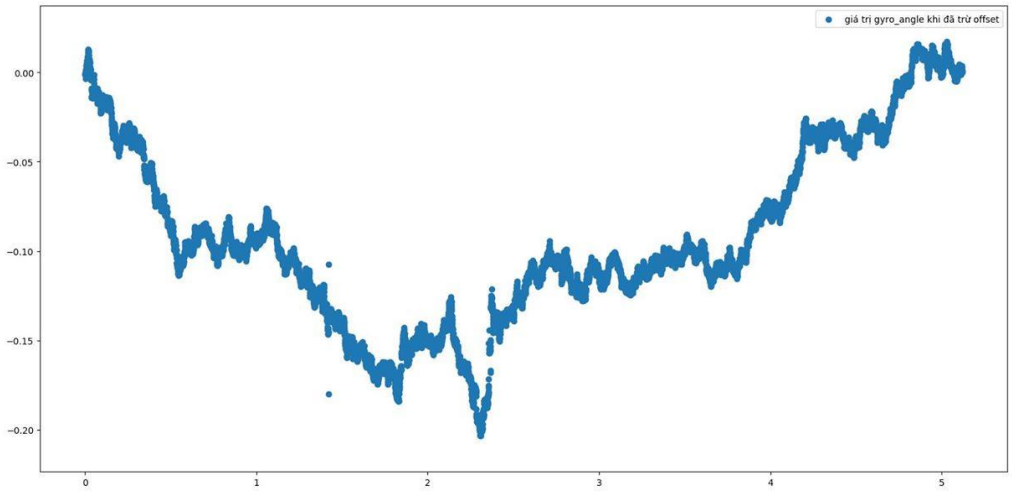
Và đây là khi tích phân cái giá trị này lên :



Cuộc sống khó khăn quá :"<

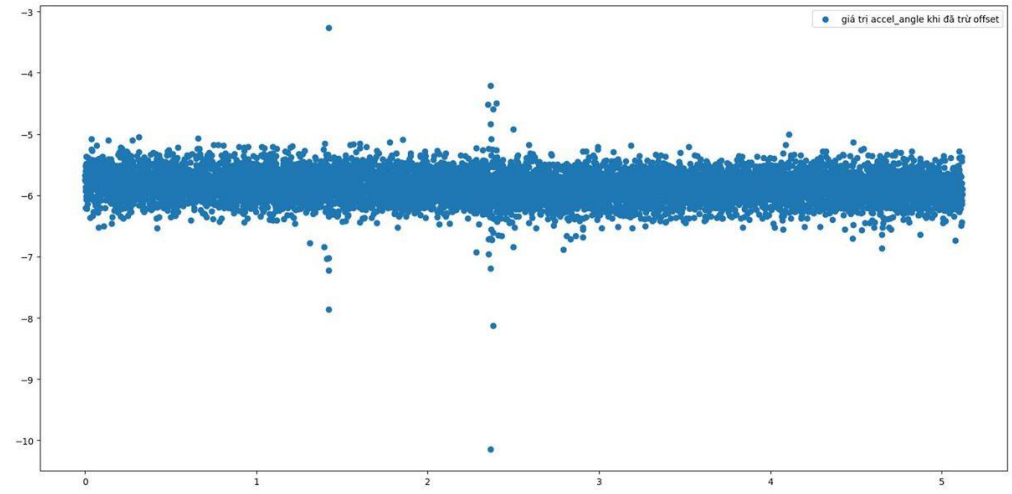
Thấy đó :> drift vkl

Vậy giờ trừ offset:



Đỡ khó khăn rồi OwO

Tiếp theo là giá trị của accel:



accel_mean = -5.854612

accel_std = 0.2393770027717784

Góc accel cũng bị trượt đi tầm 5 độ (cuộc sống khó quá :”<)

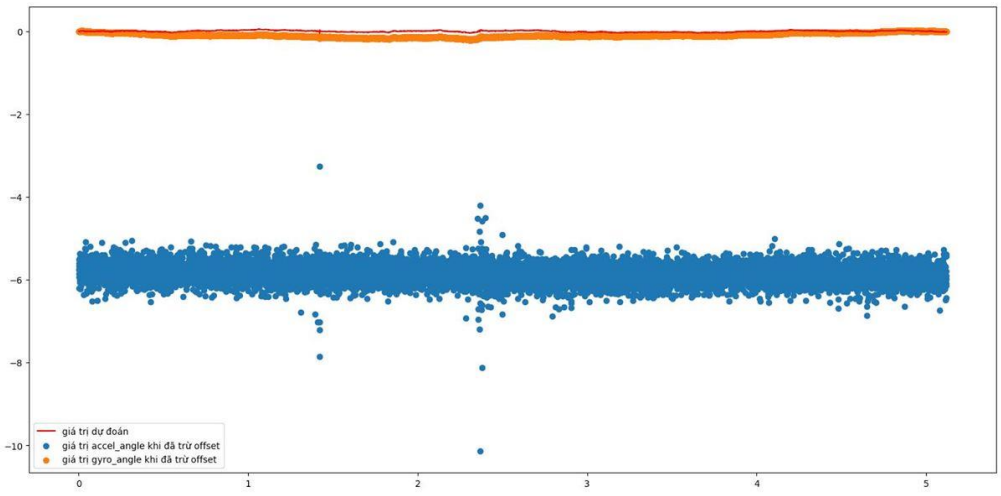
Nhưng có thể thấy là std không quá cao nên ko sao lắm OwO

3. Tính giá trị của cái bộ lọc thôi

Thấy trừ xong drift ít quá :> thôi mình cứ tính bth thôi nào :> chọn t = 0.5 => alpha = t/(t+dt) = 0.5/(0.5+0.000512) = 0.9989770475033566 OwO hơi bự nhẻ

Thôi kệ cứ quất đại, đằng nào cũng thấy gyro đúng hơn

Kết quả sau khi đã trừ bias và cả cái offset của accel:

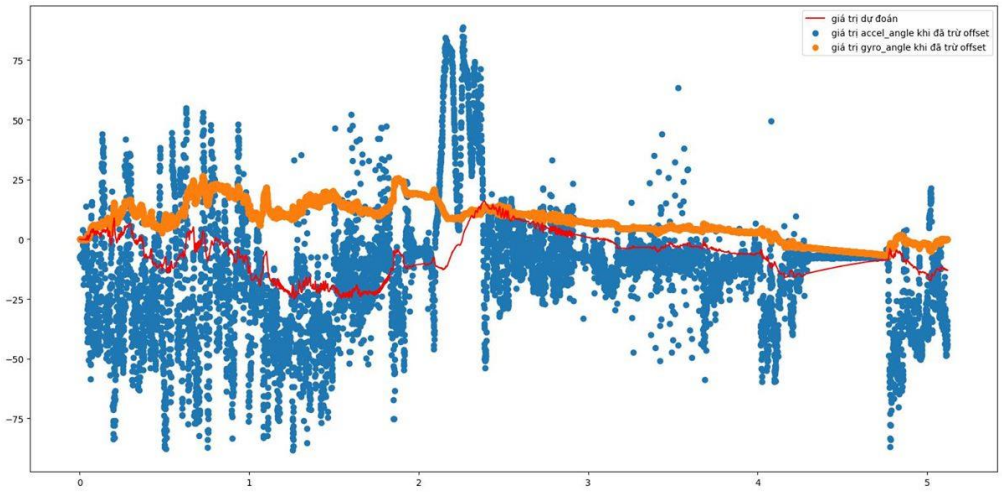


Quá ẹp nhỉ OwO

Mean của góc sau lọc là 0.0016521173549994828

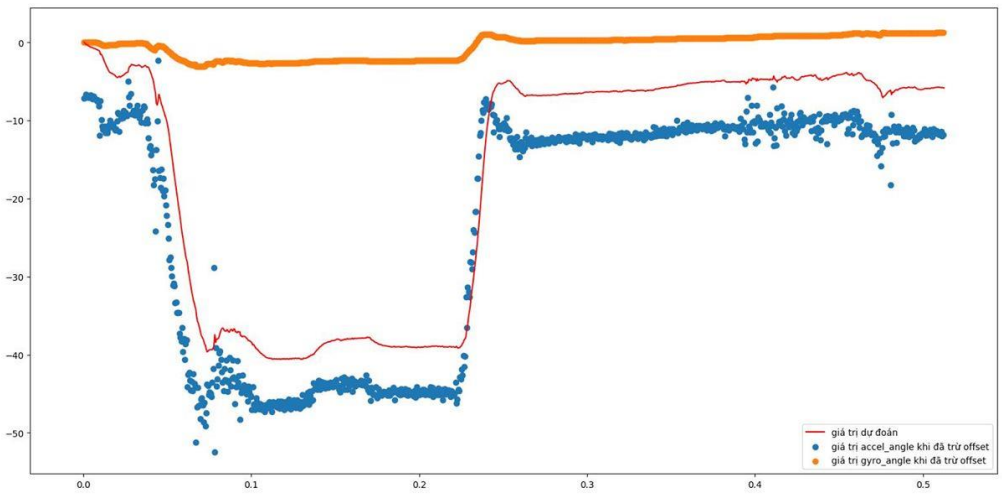
Std là 0.018080954221819167

Thử với bộ data khác, bộ này mình không rõ nó biến đổi sao vì ném lên ném xuống rất nhiều :>



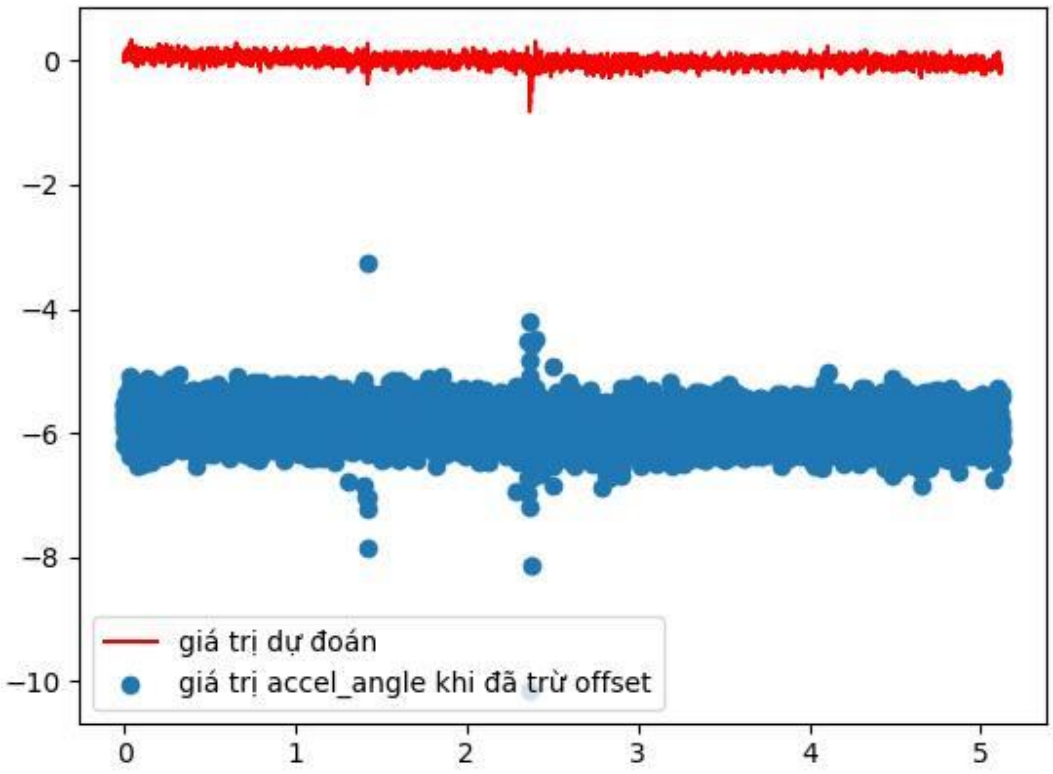
Hmm có một số chỗ vẫn chưa ổn lắm :> so với accel_angle, nên mình chưa tin lắm vào giá trị này, nhất là khi mà có 1 đoạn là góc 0 nhưng lại bị lệch, đồng thời có vẻ như quá nặng bias về phía gyro.

Thử thay lại bằng một bộ số khác: alpha = 0.8:



Có thể thấy góc lọc thiên hơn về accel, đồng thời nhiễu ngang cũng được triệt tiêu bớt, có vẻ như là ổn hơn chút :>

Với bộ data ở góc 0:



Mean: 3.375895036377464e-05

Std: 0.08114437730301427

Như vậy mình kết luận là có cái gì đó không ổn lắm cho hai cái gí trị gyro và accel này :>, nên mình chọn kết quả thiên về accel hơn

Kết quả bộ lọc với alpha = 0.9, có thể thấy là góc của gyro bị sai sai sao đó :>, cơ mà tổng quan thì nhờ accel ổn định hơn nên cũng không đến nỗi :> có lẽ việc dùng lọc bù phụ thuộc khá nhiều vào kinh nghiệm :>, và có vẻ gyro hơi có vấn đề (drift kinh quá)

IV. Kalman Filter

Để nói về Kalman Filter thì lại phải nói về lịch sử :>

Kalman Filter được sử dụng trong các bộ IMU trên tàu Apollo 11:

Hoạt động ở xung nhịp 1MHz, có 4 thanh ghi và người ta dùng nó để lên được mặt trăng :>

Và đóng góp của bộ lọc Kalman trong cái máy tính này khá là lớn à :>

Để bắt đầu, mình sẽ đi vào bản chất xác suất của bộ lọc này

Giả sử có 1 vật , trạng thái của nó được xác định dựa vào vị trí và vận tốc, kí hiệu là x:

$$x = \begin{bmatrix} P \\ P' \end{bmatrix}$$

Với P là vị trí của x, P’ là đạo hàm của vị trí tức vận tốc

Thực hiện 1 quan sát lên x, có công thức là

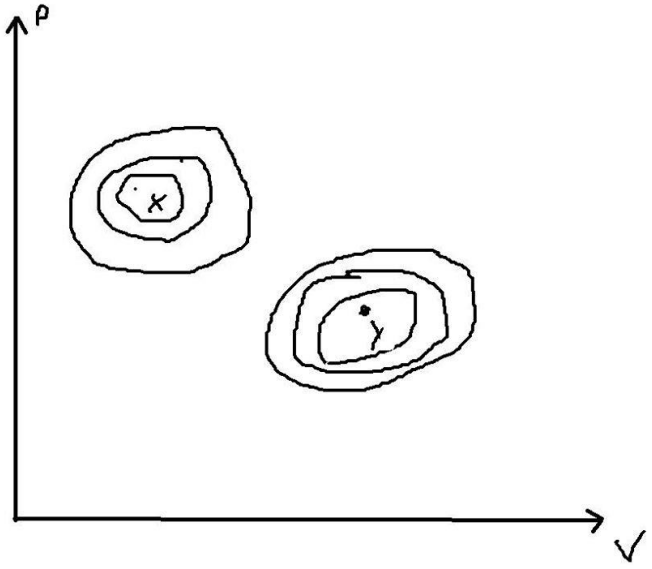
y = H*x + w

Với H là ma trận quan sát, w là nhiễu quan sát

Bởi vì cuộc sống nó không có giống cuộc đời, ta đã thấy ở trên là cái gyro với accel cũng đủ bán cả tấn hành nên tất nhiên phải có nhiễu phát sinh ra :>

Ta coi như x ~ N(u_x,Q) nào đó với Q là ma trận hiệp phương sai mấy cái độ lệch chuẩn của x

y ~ N(u_y,R1) cứ coi là vậy đi

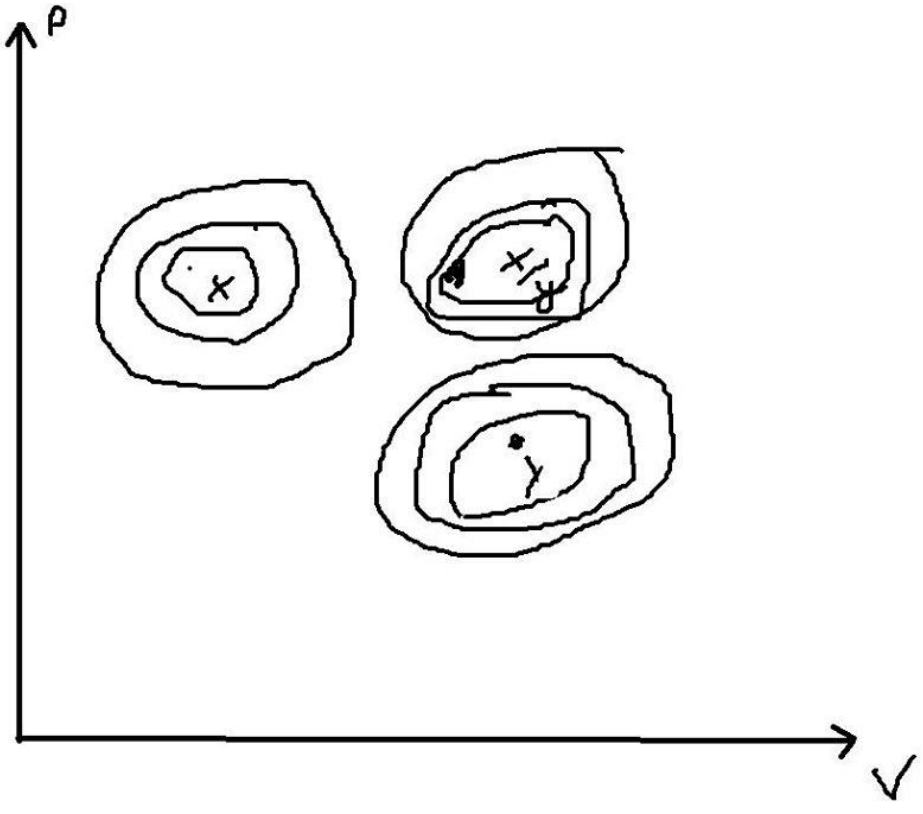


Vậy how to tìm ra x từ y

Câu trả lời đó là luật Bayes :>

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Trong trường hợp này ta cần tìm giá trị vị trí sau dự đoán gọi là x_hat, tức cần tìm xác suất P(xhat/y) = P(y/x_hat)*P(xhat)/P(y)



Giấ trị dự đoán lúc sau là vầy nè

Ta có y = H*x+R y có phân phối xác suất là N(H*x,R), x có phân phối xác suất của nhiễu :>

Sau đó chúng ta cần biểu diễn cái xác suất này qua dạng các phân phối Gauss, nói chung là khá hại não OwO thế nên quăng cái link đây cho ai cần đọc :>

http://web4.cs.ucl.ac.uk/staff/C.Bracegirdle/bayesTheoremForGaussians.pdf

Bạn chỉ cần biết là cách tính xác suất lúc sau là:

- $\mathbf{x}|\mathbf{y} \sim \mathcal{N}\left(\mathbf{R}\left(\mathbf{y} - \mathbf{M}\mu_x - \mu_y\right) + \mu_x, \mathbf{\Sigma}_x - \mathbf{R}\mathbf{M}\mathbf{\Sigma}_x^\top\right)$ where
- $\mathbf{R} = \mathbf{\Sigma}_x M^\top \left(\mathbf{M}\mathbf{\Sigma}_x\mathbf{M}^\top + \mathbf{\Sigma}_y\right)^{-1}$

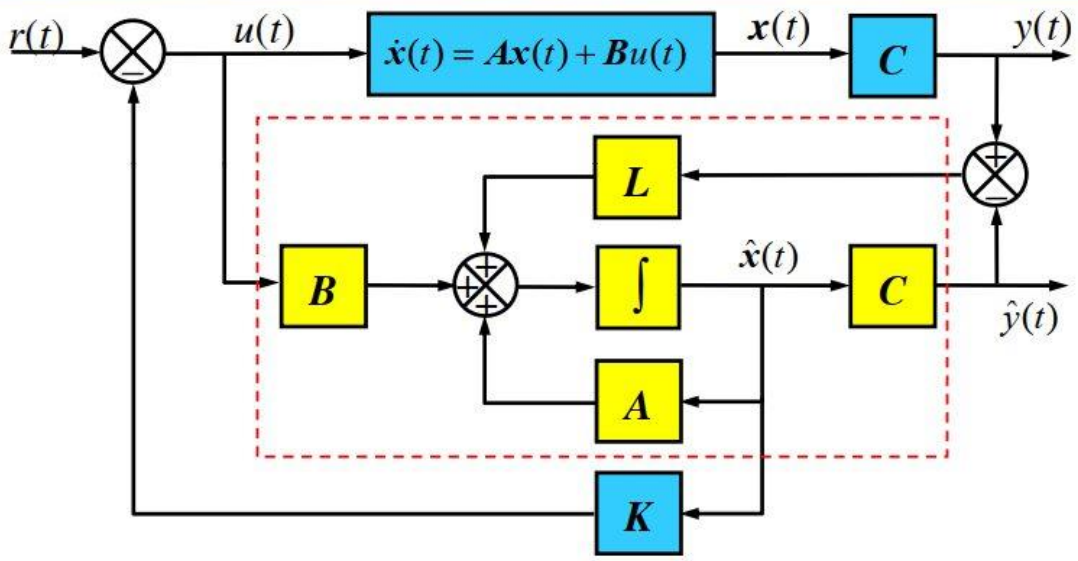
OwO đau não chưa :> chú ý M là ma trận quan sát á, tổng x hay y là phương sai của x và y :>

Nói chung đến đây là hiểu ý nghĩa xác suất roài đúng ko

Đến ý nghĩa tự động :>

Trích slide cơ sở tự động chương 6:

Bộ quan sát trạng thái



✶ Bộ quan sát trạng thái:

$$\begin{cases} \dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(y(t) - \hat{y}(t)) \\ \hat{y}(t) = C\hat{x}(t) \end{cases}$$

trong đó: $L = [l_1 \quad l_2 \quad \dots \quad l_n]^T$

Nói chung là với 1 hệ cần quan sát ta có phương trình quan sát

x_hat(t) = F*x_hat(t) + B*u(t) + w(t) với F là ma trận chuyển trạng thái, w(t) là nhiễu theo thời gian, B là ma trận điều khiển, u(t) là tính hiệu điều khiển

Một ví dụ cho F là trong hệ vị trí và vận tốc x = [p v] thì F = [[1 dt],[0,1]] thử nhân vào sẽ thấy điều kì diệu :>. Ta sẽ cho x ~ N(0, P)

B*u(t) là mức điều khiển ta đưa vào để điều khiển việc đọc, nói chung là không cần do không xài loại có đk

Silde thầy có cái feedback nữa cơ mà mị thì bỏ OwO mà tác giả source mình chôm thì thêm cái B để trừ bias của gyro :>

Tiếp theo, với một quan sát y = H*x+R

Rời rạc hóa theo thời gian, gọi x_hat là x^ cho gọn :> được: x^_k = F*x^k-1 + w_k

y_k = H*x_k-1 + R

Chú ý là ở đây ta muốn dự đoán các kết quả trong tương lai dựa vào quá khứ và dùng ma trận hiệp phương sai P để thể hiện sự sai khác giữa các giá trị trước và sau :> kí hiệu sẽ là Pk/k-1 :>

Theo cái mớ xác suất trên thì ta có:

K = Pk/k-1*H^t*(H*Pk/k-1*H^t + R)^-1 Chú ý là cái y ~(0,R) nhé :> srr vì người ta ghi vậy r

Vậy kì vọng ta sẽ gọi là xk/k cho đơn giản sẽ là

xk/k-1 = K*(yk/k-1 - H*xk/k-1 - R) + xk/k-1 // đoạn này mang ý nghĩa lấy mẫu mới và nhân với và cập nhật trạng thái hiện tại, trong code mẫu có tách cái này làm 2 bước

Hiệp phương sai:

Pk/k-1 = (Pk-1/k-1 - K*H*Pk-1/k-1^t) = (I - K*H)Pk-1/k-1 // hơi kì kì, hình như thiếu Pk/k-1 :>

Vì cuộc sống đã quá khó khăn, ta sẽ đặt S = H*Pk/k-1*H^t + R, K là Kalman Gain

=> K = Pk/k-1*(H^t)*(S^-1)

Vậy là ta đã dự đoán được hiện tại, cơ mà để tiếp tục quá trình này ta sẽ dùng các giá trị hiện tại để dự đoán tương lai:

xk/k = F*xk/k-1 + wk (đoạn này sẽ đổi tùy cách cài)

Pk/k là hiệp phương sai của xk/k = var(F*xk/k-1 + wk) = F*Pk/k-1*F^t +R

Ok vậy là mọi thứ đã make sense :> ta dùng bayes lên cái mớ data và nó trả về cái ta cần, tiếp theo là thủ tục đệ quy:

Bước 1: Cập nhật trạng thái đo:

1.Tính S và K:

S = H*Pk/k-1*H^t + R

K = Pk/k-1*(H^t)*(S^-1)

2. Thực hiện đo và tính các trạng thái hiện tại:

xk/k-1 = K*(yk/k-1 - H*xk/k-1) + xk/k-1 //Ở đây không có -R, :>

Pk/k-1 = (I - K*H)*Pk-1/k-1 // thực hiện cập nhật Pk-1/k-1 dựa vào cái giá trị đã quan sát

Bước 2: Dự đoán tương lai:

xk/k = F*xk/k-1 + wk

Pk/k = F*Pk/k-1*F^t +R

Ok đó là thủ tục cảu thuật toán OwO

Để cài thuật toán , các giá trị cần tìm là:

wk là bộ sai số cảm biến, có dạng: wk = [[e_xx e_xy] [e_yx r_yy]]

H là ma trận quan sát, ví dụ chỉ quan sát vị trí thì H = [1 0]

R là bộ sai số đo đạc ,cái này tự đo :>

P là ma trận quan trọng nhất, thể hiện sự tương quan giữa các giá trị theo thang đo, cũng là phương sai của bộ lọc nó được cập nhật nên có lẽ nên init là 0 hết sạch :>

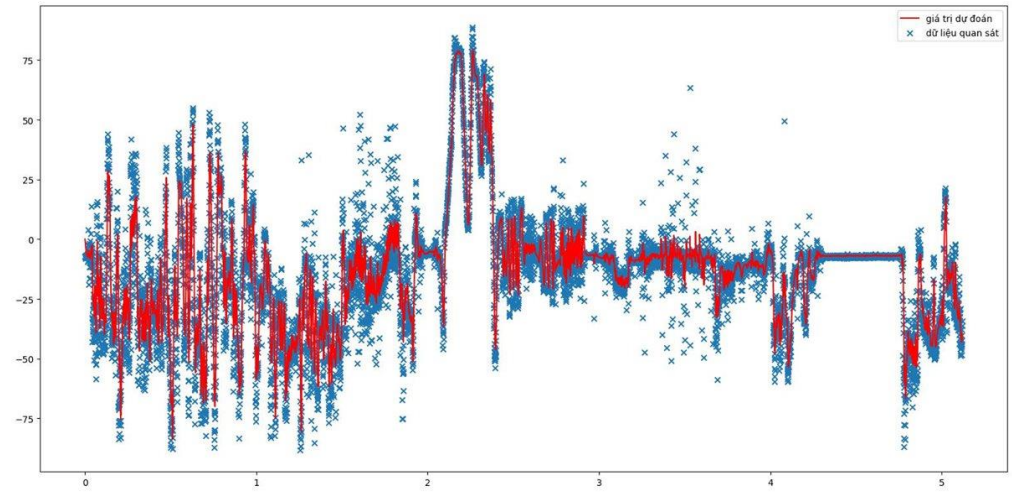
Bạn có thể xây dựng Kalman với bao nhiêu bậc tùy ý, chỉ cần hiểu rõ phương trình trạng thái và phương trình quan sát, tính được Kalman Gain,... là được, có thể dùng matlab để tính đơn giản như link dưới đã đề cập

Về lý thuyết Kalman là bộ lọc tốt nhất, vì nó tính đến phân bố của phép đo, và việc đánh giá độ chuẩn của phép đo cũng được thực hiện khá dễ dàng bằng cách kiểm tra P. Vì vậy kèo này Kalman ngon hơn xD

Một số hình ảnh của bộ Kalman mình mò thông số xD, dựa trên source người ta, nhưng cách cài khác nên chưa trừ bias gyro:

wk = [[0.00007,0],[0,0.0003]] (cái này nên thực hiện đo trên cảm biến tầm 10k lần rồi lấy std)

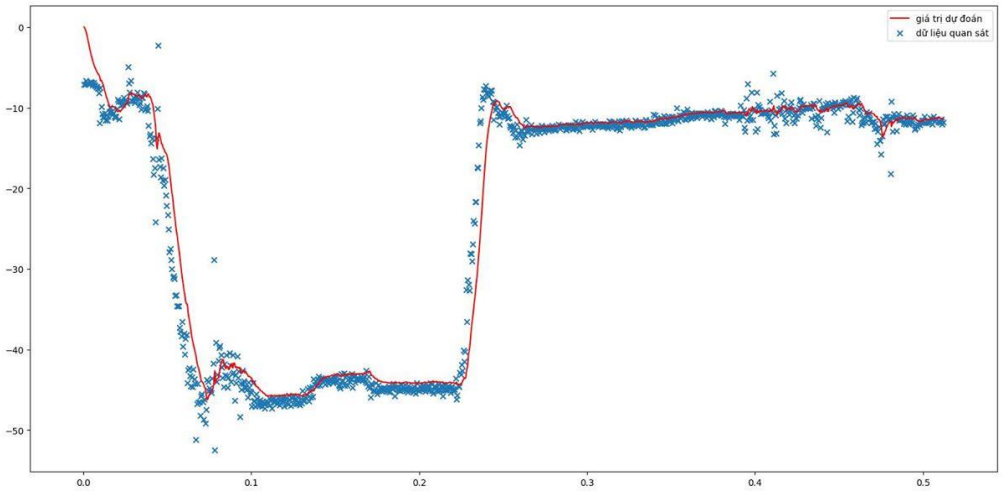
R = 0.003



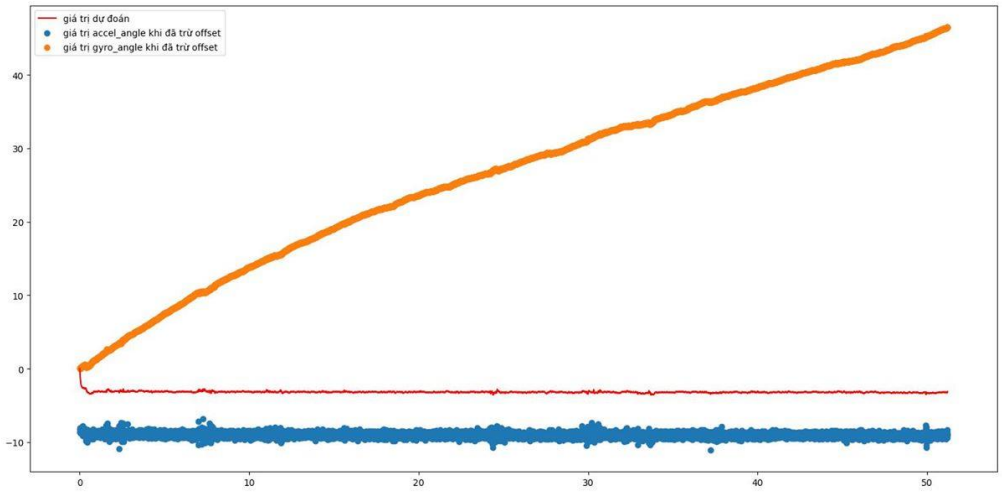
Bộ data đầu tiên

Cảm giác là ít nhiễu hơn, nhưng chưa thực hiện thí nghiệm coi đáp ứng ra sao, góc gyro không đưa vào vì nó trượt quá trời :>

Bộ data thứ 2, chú ý là mình chưa trừ offset



Để cảm biến đứng yên:



Độ lệch chuẩn là 0.24956624460802274

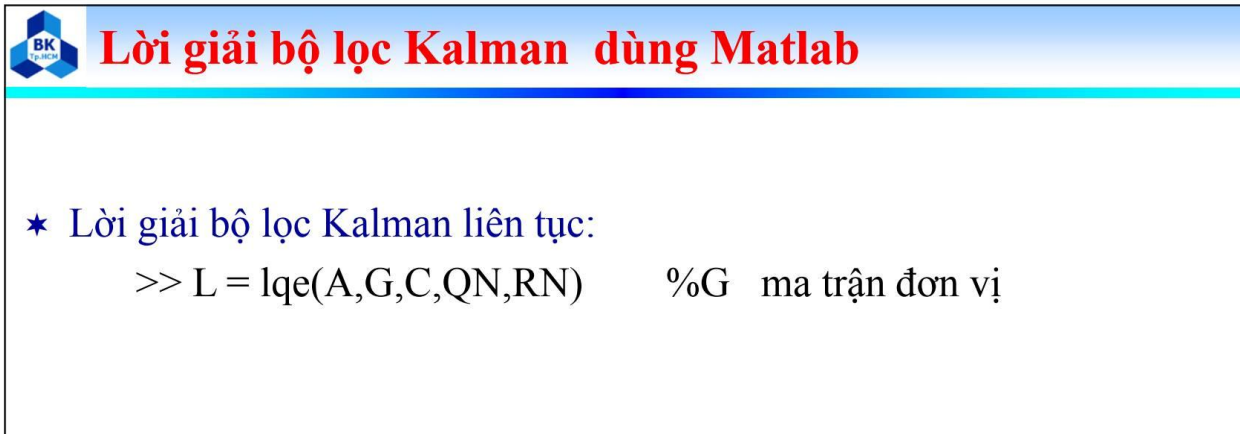
So sánh với lọc bù trên bộ này:

Độ lệch chuẩn là 0.12573595302351506

:> Mức lệch này mình nghĩ vẫn ổn hơn là lần mò cho cái lọc bù

Chú ý là cả hai bộ lọc này đều có thể cài bằng C, và cách cài trên C của họ rất hay nên mình không muốn chép lại chi :>

Vậy là qua được khổ ải :> nói chung về IMU còn nhiều cái để nói lắm, lọc Mahony, Madgwick , Quaternion, ứng dụng của nó quá đã dạng, từ robotics tới AR, VR, máy bay tên lửa, vừa hay mà cũng rất khó :> *thế mà slide lý thuyết đk nâng cao có 3 trang thôi đây :>*



Haizz :"<

Tài liệu tham khảo:

https://thetalog.com/machine-learning/kalman-filter/

http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/

https://en.wikipedia.org/wiki/Kalman_filter

https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

https://en.wikipedia.org/wiki/Multivariate_normal_distribution

http://web4.cs.ucl.ac.uk/staff/C.Bracegirdle/bayesTheoremForGaussians.pdf

https://www.researchgate.net/profile/Abdellatif_Baba/post/Kalman_filter_how_do_I_choose_initial_P_0/attachment/59e124774cde2617ef838763/AS%3A549100454400000%401507927158999/download/Probabilistic.pdf

https://www.pieter-jan.com/node/11

http://www.cs.unc.edu/~welch/kalman/

https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python#what-are-kalman-and-bayesian-filters

https://www.youtube.com/watch?v=mwn8xhgNpFY

https://github.com/TKJElectronics/KalmanFilter //code này hay và kĩ lắm, nếu khó quá thì cop nhé xD

https://github.com/DangLamTung/Kalman-And-Complementary-filter và quan trọng nhất là repo có các kiểu chém gió của tớ xD