

ĐH QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA: ĐIỆN – ĐIỆN TỬ
BỘ MÔN: ĐIỀU KHIỂN TỰ ĐỘNG

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

-----o0o-----

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

HỌ VÀ TÊN: ĐẶNG ANH TÙNG

MSSV: 40802528

NGÀNH: ĐIỀU KHIỂN TỰ ĐỘNG

LỚP: DD08KSTD

1. Đề tài luận văn:

XÂY DỰNG BỘ UỚC LƯỢNG GÓC QUAY BA TRỤC

2. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu) :

- Tìm hiểu và đánh giá các giải thuật khác nhau để xây dựng bộ ước lượng ba góc quay.
- Hiển thực giải thuật trên hệ thống nhúng dùng vi điều khiển với tần số cập nhật giá trị ba góc quay là 100Hz.
- Đánh giá sai số của hệ thống IMU và cải tiến để sai số RMS ba góc nhỏ hơn 3°
- Xây dựng phần mềm đồ họa 3D hiển thị mô hình IMU trên máy tính.

3. Ngày giao nhiệm vụ luận văn : 01/09/2012

4. Ngày hoàn thành nhiệm vụ : 17/12/2012

5. Người hướng dẫn : TS. NGUYỄN VĨNH HÀO

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Ngày.....tháng.....năm.....

CHỦ NHIỆM BỘ MÔN

(Ký và ghi rõ họ tên)

NGƯỜI HƯỚNG DẪN CHÍNH

(Ký và ghi rõ họ tên)

PHẦN DÀNH CHO KHOA, BỘ MÔN :

Người duyệt (châm sơ bộ) :

Đơn vị :

Ngày bảo vệ :

Điểm tổng kết :

Nơi lưu trữ luận văn :

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC QUỐC GIA TP.HCM
Trường Đại Học Bách Khoa
Khoa: Điện – Điện tử
Bộ môn: Điều khiển tự động

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

Ngày....tháng....năm 2012

PHIẾU CHẤM BẢO VỆ LVTN

(Dành cho người hướng dẫn)

1. Họ và tên SV:
MSSV: Chuyên ngành: Điều Khiển Tự Động
2. Đề tài :
3. Họ tên người hướng dẫn: TS. Nguyễn Vĩnh Hảo
4. Tổng quát về bản thuyết minh:
Số trang : Só chương :
Bảng số liệu : Số hình vẽ :
Số tài liệu tham khảo : Phần mềm tính toán :
Hiện vật (*sản phẩm*) :
5. Tổng quát về các bản vẽ :
- Số bản vẽ : Bản A0 Bản A1 Khô khác
- Số bản vẽ vẽ tay : Số bản vẽ trên máy tính :
6. Những ưu điểm chính của LVTN :
.....
.....
.....
.....
7. Những thiếu sót chính của LVTN :
.....
.....
.....
.....
8. Đề nghị : Được bảo vệ Bổ sung thêm để bảo vệ Không được bảo vệ
9. 3 câu hỏi SV phải trả lời trước Hội đồng :
 - 1)
 - 2)
 - 3)
10. Đánh giá chung (*bảng chữ : giỏi, khá, TB*) : Điểm _____ /10
Ký tên (ghi rõ họ tên)

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC QUỐC GIA TP.HCM
Trường Đại Học Bách Khoa
Khoa: Điện – Điện tử
Bộ môn: Điều khiển tự động

CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập – Tự do – Hạnh phúc

Ngày.... tháng.... năm 2012

PHIẾU CHẤM BẢO VỆ LVTN

(*Dành cho người phản biện*)

1. Họ và tên SV:
MSSV: Chuyên ngành: Điều Khiển Tự Động
2. Đề tài :
3. Họ tên người phản biện:
4. Tổng quát về bản thuyết minh:
Số trang : Só chương :
Bảng số liệu : Số hình vẽ :
Số tài liệu tham khảo : Phần mềm tính toán :
Hiện vật (*sản phẩm*) :
5. Tổng quát về các bản vẽ :
- Số bản vẽ : Bản A0 Bản A1 Khô khác
- Số bản vẽ vẽ tay : Số bản vẽ trên máy tính :
6. Những ưu điểm chính của LVTN :
.....
.....
.....
.....
7. Những thiếu sót chính của LVTN :
.....
.....
.....
.....
8. Đề nghị : Được bảo vệ Bổ sung thêm để bảo vệ Không được bảo vệ
9. 3 câu hỏi SV phải trả lời trước Hội đồng :
 - 1)
 - 2)
 - 3)
10. Đánh giá chung (*bảng chữ : giỏi, khá, TB*) : Điểm _____ /10
Ký tên (*ghi rõ họ tên*)

Nhận xét của giáo viên hướng dẫn

Tp. Hồ Chí Minh, Ngày Thángnăm 2012

GVHD

Nhận xét của giáo viên phản biện

Tp. Hồ Chí Minh, Ngày Thángnăm 2012

GVPB

LỜI CẢM ƠN

Xin gửi lời cảm ơn sâu sắc đến thầy hướng dẫn TS. Nguyễn Vĩnh Hảo đã gợi ý đề tài cũng như hướng dẫn tận tình để giúp tôi hoàn thành luận văn. Cảm ơn các thầy cô trong Bộ môn Tự Động nói riêng và Khoa Điện-Điện tử nói chung đã tạo điều kiện giúp đỡ và giảng dạy nhiệt tình trong suốt các năm học qua.

Xin gửi lời cảm ơn đến các anh chị, các bạn và các em trong Câu lạc bộ Nghiên Cứu Khoa Học Khoa Điện-Điện tử, Đại học Bách Khoa TPHCM *Pay It Forward* đã đồng hành, giúp đỡ và động viên tôi trong suốt quá trình học tập cũng như thực hiện luận văn.

Đặc biệt xin cảm ơn đến gia đình đã luôn hỗ trợ và động viên giúp tôi có động lực để phấn đấu trong học tập và cuộc sống.

Tp HCM, Tháng 12 năm 2012

Đặng Anh Tùng

TÓM TẮT

Hướng nghiên cứu về thuật toán các bộ ước lượng góc quay ba trục hay xây dựng hệ thống đo quán tính ba trục IMU (Inertial Measurement Unit) đã xuất hiện từ lâu do nhu cầu cần thông tin về góc nghiêng hay góc quay của vật thể trong không gian để giải quyết các bài toán về điều khiển cân bằng, hệ thống định vị dẫn đường, hệ thống mô phỏng cử động người... Trên thế giới đã có rất nhiều sản phẩm thương mại với độ chính xác rất cao ($<0.001^\circ$) dùng trong hàng không, tên lửa hay các sản phẩm với độ chính xác thấp hơn ($<1^\circ$) dùng cho dẫn đường. Các sản phẩm thương mại độ chính xác cao này rất đắt tiền, tuy nhiên đối với các ứng dụng trong điều khiển và dẫn đường không cần đòi hỏi quá khắt khe thì thông thường chỉ yêu cầu sai số nhỏ hơn 2° . Do đó bài toán đặt ra là nắm bắt các công nghệ để tạo ra IMU phù hợp có thể tích hợp sâu vào các ứng dụng này. Vì nhu cầu này nên rất nhiều phòng nghiên cứu đã đưa ra các thuật toán ước lượng tuy không đạt kết quả cao như các sản phẩm thương mại nhưng vẫn đảm bảo trong tầm sai số phù hợp cho các ứng dụng này. Cùng với sự phát triển của công nghệ MEMS và các dòng vi điều khiển hỗ trợ tính toán số thực, các cảm biến có sai số càng nhỏ và dễ dàng tích hợp với vi điều khiển tạo nên IMU nhỏ gọn.

Từ các ý trên luận văn sẽ tập trung tìm hiểu các thuật toán để xây dựng bộ ước lượng góc quay ba trục, đưa ra lựa chọn giải thuật phù hợp để nhúng trên IMU. Sau đó luận văn sẽ đánh giá sai số của IMU khi chạy giải thuật này và cải tiến để giảm thiểu sai số nếu được.

Trong luận văn sử dụng IMU 9-DOF được chế tạo tại phòng thí nghiệm Điều Khiển Tự Động - Bộ môn Tự Động, Đại học Bách Khoa TPHCM. Thuật toán được dùng là bộ lọc Kalman mở rộng cải tiến dùng DCM ước lượng chín số hạng trong ma trận xoay và ba giá trị gia tốc ngoài. Kết quả đánh giá cho thấy sai số ba góc quay RMS đạt được nhỏ hơn 3° . Tần số cập nhật dữ liệu ba góc quay là 100Hz đảm bảo yêu cầu của các bài toán điều khiển. Luận văn cũng đã phát triển chương trình đồ họa mô hình 3D của IMU hiển thị thời gian thực.

Hướng phát triển sau luận văn là cải tiến thuật toán để IMU thích nghi tốt hơn nữa với nhiễu từ trường bằng cách đưa thêm vào thuật toán ba giá trị ước lượng nhiễu từ trường, giảm thiểu sai số các góc quay và tăng tần số cập nhật của IMU dùng bộ lọc Kalman gián tiếp (Indirect-Kalman Filter).

MỤC LỤC

LỜI CẢM ƠN	ii
TÓM TẮT	ii
MỤC LỤC	iii
DANH SÁCH BẢNG BIÊU	v
DANH SÁCH HÌNH VẼ	vi
DANH SÁCH VIẾT TẮT	viii
CHƯƠNG 1. TỔNG QUAN	1
1.1 Các khái niệm chung về bộ ước lượng góc quay ba trục	1
1.2 Các ứng dụng, tình hình nghiên cứu trong và ngoài nước	1
1.3 Nhiệm vụ của luận văn.....	3
1.4 Nội dung luận văn.....	3
CHƯƠNG 2. THUẬT TOÁN ƯỚC LUỢNG	4
2.1 Tổng quan về các giải thuật ước lượng ba góc nghiêng	4
2.1.1 Ma trận xoay.....	4
2.1.2 Ma trận xoay dựa trên ba góc Euler.....	5
2.1.3 Ma trận xoay dựa trên Quaternion.....	6
2.1.4 Các phương trình động học	10
2.1.5 Cách tính các góc Euler từ quaternion và ma trận DCM	11
2.2 Bộ lọc bù	11
2.2.1 Tổng quan các bộ lọc bù	11
2.2.2 Các phương pháp ước lượng bình phương tối thiểu	11
2.2.3 Bộ lọc bù dùng phương pháp Gauss-Newton.....	13
2.2.4 Bộ lọc bù dùng phương pháp Gradient Descent.....	15
2.3 Bộ lọc Kalman	16
2.3.1 Tổng quan về bộ lọc Kalman	16

2.3.2	Bộ lọc Kalman mở rộng dùng DCM.....	20
2.3.3	Bộ lọc Kalman mở rộng dùng Quaternion	22
2.4	Mô phỏng và đánh giá các bộ ước lượng.....	24
2.4.1	Phương pháp mô phỏng và đánh giá.....	24
2.4.2	Kết quả và nhận xét	25
CHƯƠNG 3. XÂY DỰNG BỘ UỚC LƯỢNG		29
3.1	Hệ thống IMU tích hợp ARM Cortex-M4 STM32F40x và cảm biến ADIS16405...	29
3.1.1	Vì điều khiển ARM Cortex-M4F STM32F40x.....	29
3.1.2	Cảm biến ADIS16405 và phương pháp calib.....	46
3.2	Xây dựng mô hình MATLAB Simulink giải thuật Kalman mở rộng cải tiến dùng DCM	49
3.3	Lập trình cho IMU kết hợp MATLAB Simulink Embedded Coder và trình biên dịch KeilC	53
3.4	Lập trình đồ họa mô hình 3D cho IMU dùng ngôn ngữ Python	55
3.4.1	Tổng quan cấu trúc và cú pháp của một chương trình viết bằng Python.....	55
3.4.2	Các thư viện.....	57
3.4.3	Phần mềm đồ họa mô hình 3D cho IMU	61
3.5	Hệ thống bàn xoay kiểm tra IMU	62
CHƯƠNG 4. THỰC HIỆN ĐÁNH GIÁ & KẾT QUẢ ĐẠT ĐƯỢC		64
4.1	Phương pháp đánh giá.....	64
4.2	Thực hiện đánh giá	65
4.3	Kết quả	66
CHƯƠNG 5. KẾT LUẬN.....		76
5.1	Kết luận	76
5.2	Hướng phát triển của đề tài	76
TÀI LIỆU THAM KHẢO		77

DANH SÁCH BẢNG BIỂU

Bảng 1.1 Bảng so sánh độ chính xác các IMU thương mại	2
Bảng 2.1 Kết quả mô phỏng đối với bộ lọc bù	265
Bảng 2.2 Kết quả mô phỏng đối với bộ lọc Kalman.....	26
Bảng 3.1 Tóm tắt số chu kì máy khi thực hiện lệnh số học dùng FPU.....	4832
Bảng 3.2 Bảng hệ số tỉ lệ cho các giá trị cảm biến.....	48
Bảng 4.1 Kết quả thí nghiệm 1	67
Bảng 4.2 Kết quả thí nghiệm 2	67

DANH SÁCH HÌNH VẼ

Hình 1.1 Hệ tọa độ chuẩn NED	1
Hình 1.2 Ba góc nghiêng trong không gian.....	1
Hình 1.3 Một số ứng dụng của IMU.....	2
Hình 1.4 Một số IMU thương mại	2
Hình 2.1 Đồ thị xoay góc yaw	5
Hình 2.2 Vector và điểm, Quaternion và vector.....	7
Hình 2.3 Bộ lọc bù QUEST.....	14
Hình 2.4 Bộ lọc bù Madgwick.....	16
Hình 2.5 Sơ đồ hệ thống ước lượng	17
Hình 2.7 Kết quả mô phỏng Kalman cho hệ tuyến tính.....	19
Hình 2.8. Giải thuật bộ lọc Kalman mở rộng	20
Hình 2.9 Mô hình Kalman dùng DCM	20
Hình 2.10. Mô hình Kalman dùng Quaternion	22
Hình 2.11 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Gauss-Newton	26
Hình 2.12 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Gradient Descent.....	27
Hình 2.13 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Kalman DCM	27
Hình 2.14 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Kalman Quaternion	27
Hình 3.1 IMU-9DOF được chế tạo tại PTN Bộ môn Tự Động ĐHBK TPHCM.....	29
Hình 3.2 Sơ đồ khối lõi ARM Cortex-M4	31
Hình 3.3. Kiến trúc ma trận bus trong STM32F407	34
Hình 3.4 Chương trình tạo file thiết lập clock cho STM32F407.....	36
Hình 3.5 Giản đồ hoạt động Timer cơ bản	38
Hình 3.6 Frame truyền của USART	39
Hình 3.7 Giản đồ thứ tự truyền SPI	42
Hình 3.8 Sơ đồ khối ADIS16405 và hệ trục của các cảm biến	46
Hình 3.9 Mô hình kết nối phần cứng giữa VĐK và ADIS16405	47
Hình 3.10 Giản đồ xung trao đổi dữ liệu trên SPI và chế độ đọc burst-mode	47
Hình 3.11 Giản đồ từ trường trên các trục trước khi calib	49
Hình 3.12 Giản đồ từ trường trên các trục sau khi calib	49
Hình 3.13 Mô hình EKF-DCM cài tiến	50
Hình 3.14 Mô hình Simulink của phương pháp EKF-DCM	53

Hình 3.15 Mô hình Compiler kết hợp MATLAB Simulink và Keil C.....	53
Hình 3.16 Kiến trúc phần mềm trong IMU.....	54
Hình 3.17 Lưu đồ giải thuật xử lí trong IMU.....	54
Hình 3.18 Mô hình của một trình biên dịch (interpreted language)	55
Hình 3.19 Hệ tọa độ và vật thể trong VPython	58
Hình 3.20 Hình cầu trong VPython	59
Hình 3.21 Hình hộp trong VPython.....	59
Hình 3.22 Hệ tọa độ NED và Hệ tọa độ trong VPython	60
Hình 3.23 Kiến trúc phần mềm hiển thị mô hình 3D của IMU.....	61
Hình 3.24 Lưu đồ giải thuật phần mềm đồ họa cho IMU	61
Hình 3.25 Giao diện chương trình đồ họa 3D dùng Python.....	62
Hình 3.26 Bàn xoay kiểm tra IMU	63
Hình 3.27 Mạch đọc xung encoder giao tiếp máy tính tại PTN Bộ môn Tự Động, ĐHBK TPHCM	63
 Hình 4.1 File text lưu dữ liệu ước lượng.....	66
Hình 4.2 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ1 (1).....	68
Hình 4.3 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ1 (2).....	68
Hình 4.4 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ2 (1).....	69
Hình 4.5 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ2 (2).....	69
Hình 4.6 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ3 (1).....	70
Hình 4.7 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ3 (2).....	70
Hình 4.8 Biểu đồ ba góc quay và sai số của mẫu khi có nhiễu từ trường (1)	71
Hình 4.9 Biểu đồ ba góc quay và sai số của mẫu khi có nhiễu từ trường (2)	71
Hình 4.10 Biểu đồ ba góc quay và sai số của mẫu TN2-S.....	72
Hình 4.11 Biểu đồ ba góc quay và sai số của mẫu TN2-X	72
Hình 4.12 Biểu đồ ba góc quay và sai số của mẫu TN2-Y	72
Hình 4.13 Biểu đồ ba góc quay và sai số của mẫu TN2-Z.....	73
Hình 4.14 Biểu đồ ba góc quay và sai số của mẫu TN2-XYZ	73
Hình 4.15 Biểu đồ ba góc quay và sai số của mẫu TN2-AX	74
Hình 4.16 Biểu đồ ba góc quay và sai số của mẫu TN2-AY	74
Hình 4.17 Biểu đồ ba góc quay và sai số của mẫu TN2-AZ.....	74
Hình 4.18 Biểu đồ ba góc quay và sai số của mẫu TN2-AXYZ	75
Hình 4.19 Biểu đồ ba góc quay và sai số của mẫu TN2-M (1)	75

DANH SÁCH VIẾT TẮT

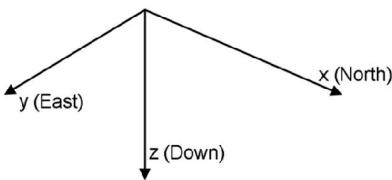
- IMU : Inertial Measurement Unit - Đơn vị đo quán tính
- INS : Inertial Navigation System – Hệ thống quán tính dẫn đường
- DCM : Direction Cosine Matrix – Ma trận cosine trực tiếp
- KF : Kalman Filter – Bộ lọc Kalman
- EKF : Extended Kalman Filter – Bộ lọc Kalman mở rộng
- NED : North-East-Down – Hệ tọa độ Bắc-Đông-Hướng xuống
- ENU : East-North-Up – Hệ tọa độ Đông-Bắc-Hướng lên
- MEMS: Microelectromechanical systems – Hệ thống vi cơ điện tử
- VĐK : Vi điều khiển

Chương 1. Tổng quan

1.1 Các khái niệm chung về bộ ước lượng góc quay ba trục

Bộ ước lượng góc quay ba trục (hay còn gọi là IMU-Inertial Measurement Unit-Bộ đo lường quán tính) là hệ thống ước lượng ba góc nghiêng Euler trong không gian của vật thể. Ba góc Euler được định nghĩa là góc lệch giữa ba trục trên vật thể so với hệ trục chuẩn, chiều dương góc quay được xác định theo qui tắc bàn tay phải. Hệ trục chuẩn được dùng trong luận văn là NED (Bắc-Đông-Hướng Xuống) gắn liền mặt đất, hệ trục gắn với vật thể là hệ trục Descartes với ba trục x, y, z đọc theo hệ trục của các cảm biến trên IMU. Kí hiệu ba góc quay lần lượt là:

- Roll: góc quay theo trục x, kí hiệu ϕ ($-\pi \leq \phi \leq \pi$)
- Pitch: góc quay theo trục y, kí hiệu θ ($-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$)
- Yaw: góc quay theo trục z, kí hiệu ψ ($-\pi < \psi < \pi$)



Hình 1.1 Hệ tọa độ chuẩn NED



Hình 1.2 Ba góc nghiêng trong không gian

Một hệ thống IMU thông thường sẽ gồm ba loại cảm biến Gia tốc (Accelerometer), Vận tốc góc quay (Gyroscope) và Từ trường (Magnetometer). Trên IMU có thể có thêm cảm biến nhiệt độ để bù nhiệt cho các cảm biến hoặc cảm biến áp suất đo độ cao... Nhờ sự phát triển của công nghệ MEMS các cảm biến này đều được chế tạo ngày càng nhỏ gọn và sai số nhỏ, vì vậy công việc xây dựng bộ ước lượng góc quay ba trục chủ yếu là xây dựng thuật toán để kết hợp ba loại cảm biến này và ước lượng ra ba góc quay với sai số nhỏ nhất.

1.2 Các ứng dụng, tình hình nghiên cứu trong và ngoài nước

Các sản phẩm IMU ước lượng ba góc quay được sử dụng rộng rãi trong các lĩnh vực như: robot tự cân bằng, điều khiển máy bay, điều khiển quadrotor, mô phỏng chuyển động người, tương tác người-máy, hệ thống dẫn đường INS kết hợp IMU và GPS...



Hình 1.3 Một số ứng dụng của IMU

Trên thế giới lĩnh vực nghiên cứu này đã có từ lâu và được nghiên cứu rộng rãi cũng như cho ra đời các sản phẩm thương mại đạt được độ chính xác rất cao. Ví dụ một số sản phẩm thương mại:



Hình 1.4 Một số IMU thương mại

Bảng 1.1 Bảng so sánh độ chính xác các IMU thương mại

Sản phẩm	Độ chính xác	Ứng dụng
HG9900-Honeywell	- Gyroscope bias: <0.003°/hr - Accelerometer bias: <25 µg	Hàng không, tên lửa
MTi-Xsens Technology	- Tĩnh: roll, pitch < 0.5° ; yaw < 1°; - Động: 2° RMS - Gyroscope bias: 1°/s - Accelerometer bias: 2mg - Magnetometer bias: 0.1 mGauss	Điều khiển, mô phỏng chuyển động người
3DM GX3-MicroStrain	- Tĩnh: <0.5° - Động: <2° - Gyroscope bias: 0.2°/s - Accelerometer bias: 5mg - Magnetometer bias: 10mGauss	Điều khiển

Các sản phẩm IMU cũng đã được dùng rộng rãi ở nước ta cho các ứng dụng như: điều khiển robot, mô hình quadrotor, hệ thống INS... tuy nhiên vẫn thiếu các nghiên cứu đầy đủ về IMU (phải gồm hai phần xây dựng thuật toán và đánh giá sai số). Các hướng nghiên cứu về

IMU hiện nay tập trung phát triển và hoàn thiện hai loại thuật toán cho các bộ IMU là bộ lọc bù (Complementary Filter) ở [5], [6] và lọc Kalman (Kalman Filter) ở [7], [8], [9], [10] hay kết hợp cả hai như ở [11], [12]. Trong đó bộ lọc bù tuy có ưu thế hơn về tính đơn giản và khối lượng tính toán ít nhưng vẫn không hiệu quả bằng bộ lọc Kalman. Với sự phát triển của các dòng vi điều khiển hỗ trợ tính toán số thực thì khối lượng tính toán không phải là vấn đề lớn với bộ IMU. Sự chính xác của các IMU phần lớn được quyết định ở chất lượng của thuật toán đã được phát triển, do đó mục tiêu chính của luận văn này là xây dựng và đánh giá các thuật toán hiện nay để chọn ra thuật toán với sai số nhỏ nhất, từ đó đưa ra các cải tiến để hoàn thiện bộ lọc.

1.3 Nhiệm vụ của luận văn

Nhiệm vụ của luận văn:

1. Tìm hiểu và đánh giá các giải thuật khác nhau để xây dựng bộ ước lượng ba góc quay.
2. Hiện thực giải thuật trên hệ thống nhúng dùng vi điều khiển với tần số cập nhật giá trị ba góc quay là 100Hz.
3. Đánh giá sai số của hệ thống IMU và cải tiến để sai số RMS ba góc nhỏ hơn 3° .
4. Xây dựng phần mềm đồ họa 3D hiển thị mô hình IMU trên máy tính.

1.4 Nội dung luận văn

Luận văn gồm năm phần:

Chương 1. Tổng quan

Chương 2. Thuật toán ước lượng

Chương 3. Xây dựng bộ ước lượng.

Chương 4. Thực hiện đánh giá & Kết quả đạt được

Chương 5. Kết luận

Chương 2. Thuật toán ước lượng

2.1 Tổng quan về các giải thuật ước lượng ba góc nghiêng

2.1.1 Ma trận xoay

Ma trận xoay R_{BE} chuyển giá trị vector từ hệ tọa độ gắn liền với vật (B-body) sang hệ tọa độ NED (E-Earth)

$$R_{BE} = (X_{BE} \ Y_{BE} \ Z_{BE}) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Giả sử ${}^Bv, {}^Ev$ là 2 vector trong tọa độ (B) & (E) thì ta có quan hệ:

$${}^Ev = R_{BE} \cdot {}^Bv$$

Hoặc:

$${}^Bv = R_{BE}^{-1} \cdot {}^Ev = R_{BE}^T \cdot {}^Ev$$

Ma trận xoay R_{BE} là ma trận 3x3 với các tính chất sau:

- Tính chất 1:** Ma trận xoay phải được chuẩn hóa (normalize) tức là tổng bình phương các giá trị trên hàng và cột bằng 1.

$$\|X\| = \|Y\| = \|Z\| = 1$$

- Tính chất 2:** Ma trận xoay phải là ma trận trực giao (orthogonal) tức là tích của bất cứ một cặp vector hàng hay một cặp vector cột đều bằng 0.

$$X \cdot Y^T = 0$$

$$Y \cdot Z^T = 0$$

$$Z \cdot X^T = 0$$

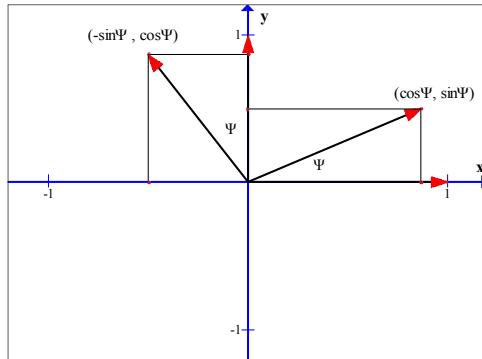
- Tính chất 3:** Tích ma trận xoay và chuyển vị của nó là ma trận đơn vị.

$$R \cdot R^T = I$$

$$\text{hay } R^T = R^{-1}$$

2.1.2 Ma trận xoay dựa trên ba góc Euler

Ta sẽ tính ma trận xoay dựa trên 3 góc euler.



Hình 2.1 Đồ thị xoay góc yaw

Giả sử ban đầu giữ vật sao cho hệ trục $0x'y'z'$ gắn liền với vật (B) trùng với hệ trục $0xyz$ (E hay N-E-D) như trên. Xét vector trong hệ trục (B) ${}^Bv = (x_0, y_0, z_0)$ thì hình chiếu của nó trong hệ trục (E) là vector ${}^Ev = (x_1, y_1, z_1)$

Giữ nguyên trục z quay hệ trục (B) một góc ψ trên mặt Oxy.

Trong hệ trục (B) vector vẫn có giá trị là ${}^Bv = (x_0, y_0, z_0)$, tuy nhiên lúc này hình chiếu của nó xuống (E) là ${}^Ev = (x_1, y_1, z_1)$. Với:

$$\begin{cases} x_1 = x_0 \cdot \cos\psi - y_0 \cdot \sin\psi \\ y_1 = x_0 \cdot \sin\psi + y_0 \cdot \cos\psi \\ z_1 = z_0 \end{cases}$$

Sắp xếp lại hệ phương trình trên ta được:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$$

Hay:

$${}^Ev = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} {}^Bv$$

Đặt $R_{BE,Z}$ là ma trận xoay quanh trục z từ hệ tọa độ B đến hệ tọa độ E thì:

$$R_{BE,Z} = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Tương tự:

$$R_{BE,X} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}$$

$$R_{BE,Y} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Vậy nếu xoay hệ trục (B) theo 3 góc ϕ, θ, ψ so với hệ trục (E) thì một vector Bv trong hệ trục (B) sẽ có hình chiếu Ev trên hệ trục (E) như sau:

$${}^Ev = R_{BE,Z} \cdot R_{BE,Y} \cdot R_{BE,X} \cdot {}^Bv = R_{BE} \cdot {}^Bv$$

Ở đây chú ý chiều dương góc xoay theo quy tắc bàn tay phải và 3 góc Euler có giới hạn lần lượt là:

$$-\pi \leq \phi \leq \pi, -\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}, -\pi < \psi < \pi$$

Khi đó ma trận xoay tính được như sau:

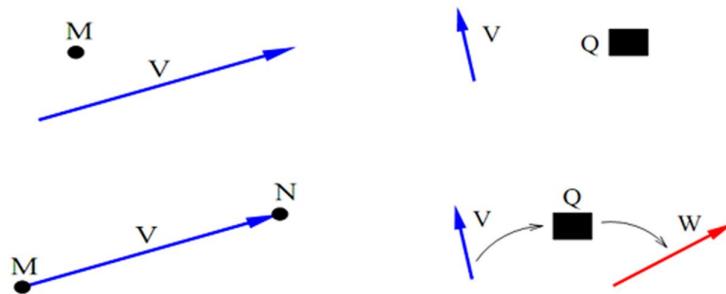
$$R_{BE} = \begin{pmatrix} \cos\psi \cdot \cos\theta & \sin\phi \cdot \sin\theta \cdot \cos\psi - \cos\phi \cdot \sin\psi & \sin\phi \cdot \sin\psi + \cos\phi \cdot \cos\psi \cdot \sin\theta \\ \sin\psi \cdot \cos\theta & \sin\phi \cdot \sin\theta \cdot \sin\psi + \cos\phi \cdot \cos\psi & \cos\phi \cdot \sin\theta \cdot \sin\psi - \cos\psi \cdot \sin\phi \\ -\sin\theta & \cos\theta \cdot \sin\phi & \cos\phi \cdot \cos\theta \end{pmatrix}$$

2.1.3 Ma trận xoay dựa trên Quaternion

a. Đại số Quaternion

Toán tử hình học quaternion và đại số quaternion được giới thiệu bởi nhà toán học William Hamilton. Khái niệm quaternion dùng để biểu diễn mối quan hệ giữa hai vector được sáng tạo từ khái niệm vector biểu diễn mối quan hệ giữa hai điểm.

Trong đại số vector ta có từ một điểm M toán tử vector v sẽ tạo ra duy nhất một điểm N với chiều dài MN là độ lớn của vector và hướng của MN là hướng của vector v . Tương tự từ vector v đại số quaternion q sẽ tạo ra duy nhất vector w .



Hình 2.2 Vector và điểm, Quaternion và vector

Toán tử quaternion q gồm hai phần “có hướng” (hay phần ảo) biểu diễn dạng vector trong không gian \vec{q} và phần “vô hướng” (hay phần thực) q_4 được biểu diễn như sau:

$$q \stackrel{\text{def}}{=} [\vec{q}, q_4] = \vec{q} + q_4 = q_1i + q_2j + q_3k + q_4 \quad (\vec{q} \in \mathbf{R}^3, q_4 \in \mathbf{R})$$

Với:

$$\begin{aligned} i^2 &= j^2 = k^2 = ijk = -1 \\ ij &= k = -ji \\ jk &= i = -kj \\ ki &= j = -ik \end{aligned}$$

Các vector $\vec{a} = xi + yj + zk$ trong không gian ba chiều sẽ được biểu diễn dạng quaternion như sau: $q_a = xi + yj + zk + 0$. Còn các giá trị thực a được biểu diễn dạng quaternion như sau $q_a = 0i + 0j + 0k + a$.

Một số phép toán trong đại số quaternion như sau:

Đặt q, p là hai quaternion như sau: $\begin{cases} q = q_1i + q_2j + q_3k + q_4 \\ p = p_1i + p_2j + p_3k + p_4 \end{cases}$

i. Phép cộng:

$$q + p = [\vec{q}, q_4] + [\vec{p}, p_4] = [\vec{q} + \vec{p}, q_4 + p_4]$$

$$\rightarrow q + p = (q_1 + p_1)i + (q_2 + p_2)j + (q_3 + p_3)k + (q_4 + p_4)$$

ii. Phép nhân:

$$q \otimes p = [\vec{q}, q_4] \otimes [\vec{p}, p_4] = [\vec{q} \otimes \vec{p} + q_4 \cdot \vec{p} + p_4 \cdot \vec{q}, q_4 \cdot p_4 - \vec{q} \cdot \vec{p}^T]$$

$$\begin{aligned}
 \rightarrow q \otimes p &= (q_1 i + q_2 j + q_3 k + q_4) \otimes (p_1 i + p_2 j + p_3 k + p_4) \\
 &= (-q_1 p_1 - q_2 p_2 - q_3 p_3 + q_4 p_4) + i(q_1 p_4 + q_4 p_1 + q_2 p_3 - q_3 p_2) + j(q_2 p_4 \\
 &\quad + q_4 p_2 + q_3 p_1 - q_1 p_3) + k(q_3 p_4 + q_4 p_3 + q_1 p_2 - q_2 p_1) \\
 \rightarrow q \otimes p &= \begin{pmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = L(q).p \\
 \text{hoặc: } q \otimes p &= \begin{pmatrix} p_4 & p_3 & -p_2 & p_1 \\ -p_3 & p_4 & p_1 & p_2 \\ p_2 & -p_1 & p_4 & p_3 \\ -p_1 & -p_2 & -p_3 & p_4 \end{pmatrix} \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} = R(p).q
 \end{aligned}$$

iii. Lượng liên hợp:

$$q^* = [\vec{q}, q_4]^* \stackrel{\text{def}}{=} [-\vec{q}, q_4] \rightarrow q^* = -q_1 i - q_2 j - q_3 k + q_4$$

iv. Dạng chuẩn hóa:

$$N(q) \stackrel{\text{def}}{=} q \otimes q^* = q_4^2 - \vec{q} \cdot \vec{q}^T \rightarrow N(q) = q_1^2 + q_2^2 + q_3^2 + q_4^2$$

v. Quaternion nghịch đảo: $(q \otimes q^{-1} = 1)$

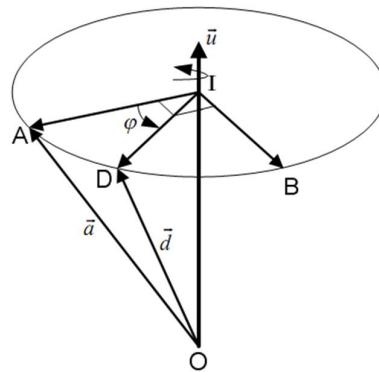
$$q^{-1} = \frac{q^*}{N(q)} \rightarrow q^{-1} = \frac{1}{q_1^2 + q_2^2 + q_3^2 + q_4^2} (-q_1 i - q_2 j - q_3 k + q_4)$$

Trong đó chú ý tích có hướng $\vec{q} \otimes \vec{p}$ và vô hướng $\vec{q} \cdot \vec{p}^T$ của hai vector có công thức sau:

$$\vec{q} \otimes \vec{p} = (q_2 p_3 - q_3 p_2) i + (q_3 p_1 - q_1 p_3) j + (q_1 p_2 - q_2 p_1) k$$

$$\vec{q} \cdot \vec{p}^T = q_1 p_1 + q_2 p_2 + q_3 p_3$$

b. Áp dụng quaternion vào các phép quay



Ở đây khi quay vector \vec{a} xung quanh vector \vec{u} một góc φ thì ta được vector \vec{d} được biểu diễn dạng đại số quaternion như sau:

$$q_d = q^* \otimes q_a \otimes q$$

Với:

$$q_d = [\vec{d}, 0], q_a = [\vec{a}, 0]$$

$$\text{Khi đó: } q_4 = \cos\left(\frac{\varphi}{2}\right), \vec{q} = \sin\left(\frac{\varphi}{2}\right) \cdot \vec{u}$$

Và q^* là quaternion liên hợp của q :

$$q^* = -q_1 i - q_2 j - q_3 k + q_4$$

Từ đó ta có được cách quay vector dựa vào quaternion với chú ý quaternion để quay phải là quaternion đơn vị tức là:

$$N(q) = 1$$

c. Ma trận xoay

+ Tính ma trận xoay từ A sang A' theo hai hướng (vì quaternion là đơn vị nên $q^{-1} = -q_1 i - q_2 j - q_3 k + q_4$)

$$\begin{aligned} A' &= q \otimes A \otimes q^{-1} = (q \otimes A) \otimes q^{-1} = L(q).A \otimes q^{-1} = L(q).R(q^{-1}).A \\ &= \begin{pmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{pmatrix} \begin{pmatrix} q_4 & -q_3 & q_2 & -q_1 \\ q_3 & q_4 & -q_1 & -q_2 \\ -q_2 & q_1 & q_4 & -q_3 \\ q_1 & q_2 & q_3 & q_4 \end{pmatrix}.A \\ &= \begin{pmatrix} 2.q_1^2 + 2q_4^2 - 1 & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) & 0 \\ 2(q_1q_2 + q_3q_4) & 2q_2^2 + 2q_4^2 - 1 & 2(q_2q_3 - q_1q_4) & 0 \\ 2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & 2q_3^2 + 2q_4^2 - 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.A \end{aligned}$$

$$A' = q^{-1} \otimes A \otimes q = \begin{pmatrix} 2.q_1^2 + 2q_4^2 - 1 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) & 0 \\ 2(q_1q_2 - q_3q_4) & 2q_2^2 + 2q_4^2 - 1 & 2(q_2q_3 + q_1q_4) & 0 \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & 2q_3^2 + 2q_4^2 - 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.A$$

Ma trận xoay dựa trên Quaternion (với q_4 phần thực)

$$R_{EB} = \begin{pmatrix} 2.q_1^2 + 2q_4^2 - 1 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & 2q_2^2 + 2q_4^2 - 1 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & 2q_3^2 + 2q_4^2 - 1 \end{pmatrix}$$

2.1.4 Các phương trình động học

a. Phương trình động học biểu diễn bởi ba góc Euler

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \tan\theta \sin\phi & \tan\theta \cos\phi \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sec\theta \sin\phi & \sec\theta \cos\phi \end{bmatrix} \begin{bmatrix} \omega_x^B \\ \omega_y^B \\ \omega_z^B \end{bmatrix}$$

Trong đó:

$$\sec\theta = \frac{1}{\cos\theta}$$

và $\omega_x^B, \omega_y^B, \omega_z^B$ là vận tốc góc các trục x, y, z

b. Phương trình động học biểu diễn dạng DCM

$$\dot{R_{BE}} = R_{BE} \cdot [\omega \times]$$

Với:

$$\begin{aligned} \text{■ } [\omega \times] &= \begin{bmatrix} 0 & -\omega_z^B & \omega_y^B \\ \omega_z^B & 0 & -\omega_x^B \\ -\omega_y^B & \omega_x^B & 0 \end{bmatrix} \\ \text{■ } R_{BE} &= \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} = \\ &\begin{pmatrix} \cos\psi \cdot \cos\theta & \sin\phi \cdot \sin\theta \cdot \cos\psi - \cos\phi \cdot \sin\psi & \sin\phi \cdot \sin\psi + \cos\phi \cdot \cos\psi \cdot \sin\theta \\ \sin\psi \cdot \cos\theta & \sin\phi \cdot \sin\theta \cdot \sin\psi + \cos\phi \cdot \cos\psi & \cos\phi \cdot \sin\theta \cdot \sin\psi - \cos\psi \cdot \sin\phi \\ -\sin\theta & \cos\theta \cdot \sin\phi & \cos\phi \cdot \cos\theta \end{pmatrix} \end{aligned}$$

c. Phương trình động học biểu diễn dạng Quaternion

$$\dot{q_{BE}} = \frac{1}{2} \cdot q_{BE} \otimes \omega^B$$

Khi p là vector trong không gian ba chiều ta được công thức rút gọn như sau:

$$\begin{cases} q = q_1 i + q_2 j + q_3 k + q_4 \\ \omega^B = \omega_x i + \omega_y j + \omega_z k + 0 \end{cases}$$

$$\rightarrow \dot{q} = \frac{1}{2} q \otimes \omega = \frac{1}{2} L(q) \cdot \omega = \frac{1}{2} \begin{pmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \\ 0 \end{pmatrix}$$

$$\rightarrow \dot{q} = \frac{1}{2} \begin{pmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

2.1.5 Cách tính các góc Euler từ quaternion và ma trận DCM

Từ các ma trận xoay trên ta tính được các góc Euler dựa vào các hệ số của ma trận DCM hoặc dựa vào quaternion như sau:

Góc Euler	Tính từ DCM	Tính từ quaternion
ϕ	$\text{atan2}(R_{32}, R_{33})$	$\text{atan2}(2q_2q_3 + 2q_1q_4, 2q_4^2 + 2q_3^2 - 1)$
θ	$-\text{asin}(R_{31})$	$\text{asin}(-2q_1q_3 + 2q_2q_4)$
ψ	$\text{atan2}(R_{21}, R_{11})$	$\text{atan2}(2q_1q_2 + 2q_3q_4, 2q_4^2 + 2q_1^2 - 1)$

2.2 Bộ lọc bù

2.2.1 Tổng quan các bộ lọc bù

Các bộ lọc bù đều hoạt động dựa trên nguyên lý: giá trị ước lượng sẽ là giá trị phù hợp nhất sao cho tối thiểu hóa tổng các bình phương các sai khác từ các nguồn dữ liệu khác nhau (phương pháp “*Least-mean-square*” – *Ước lượng bình phương tối thiểu*).

Đối với IMU trong các thuật toán ở đây có ba nguồn dữ liệu để ước lượng góc là: Gia tốc ba trục, Vận tốc góc quay ba trục và Từ trường ba trục. Để thực hiện điều này ta sẽ dùng các thuật toán tìm giá trị nhỏ nhất của **hàm mục tiêu** là:

$$f(x) = \epsilon(x)^T \cdot \epsilon(x)$$

Bài toán này còn gọi là “*nonlinear least-square*”, để giải thì có nhiều thuật toán, ở đây trình bày hai thuật toán được dùng rộng rãi là: **Gauss-Newton** và **Gradient Descent**.

Ưu điểm của các bộ lọc bù là thuật toán đơn giản, khối lượng tính toán ít rất dễ nhúng xuống các dòng vi điều khiển. Tuy nhiên vì quá đơn giản nên việc tối ưu và cải tiến giải thuật rất khó. Các vấn đề cần giải quyết khi giải bài toán này là:

- Tính hội tụ của giải thuật
- Điều kiện đảm bảo giá trị nhỏ nhất

2.2.2 Các phương pháp ước lượng bình phương tối thiểu

a. Phương pháp Newton

Đầu tiên ta xem xét phương pháp Newton tìm giá trị nhỏ nhất của hàm $f(x)$ tổng quát sau đó ứng dụng tìm giá trị nhỏ nhất của các hàm bình phương (nonlinear least-squares)

Ta bắt đầu từ khai triển Taylor quen thuộc:

$$f(x) = f(x^k) + (x - x^k)^T \cdot g^k + \frac{1}{2} (x - x^k)^T \cdot F(x^k) \cdot (x - x^k) \triangleq q(x)$$

Trong đó $g^k = \nabla f(x^k)$, $F(x^k) = \nabla^2 f(x^k)$ và x^k là giá trị biến x tại thời điểm (hay bước thứ) k . Thực hiện phép đạo hàm (hay gradient) ta được:

$$0 = \nabla q(x) = g^k + F(x^k) \cdot (x - x^k)$$

Nếu $F(x^k) > 0$ (còn gọi là Hessian) (đạo hàm bậc một bằng 0, đạo hàm bậc hai lớn hơn không tức là tìm giá trị min) ta có biểu thức đệ qui như sau:

$$x^{k+1} = x^k - (F(x^k))^{-1} \cdot g^k$$

Từ đó ta có thể tìm giá trị nhỏ nhất của hàm $f(x)$ bằng phép lặp qua 2 bước:

- Bước 1: tính Δ^k từ phương trình $F(x^k) \cdot \Delta^k = -g^k$
- Bước 2: tính $x^{k+1} = x^k + \Delta^k$

Đối với bài toán “nonlinear least-squares”: cho m giá trị hàm $r = [r_1, \dots, r_m]^T$ của n biến $x = [x_1, \dots, x_n]^T$ với $m \geq n$, phương pháp Newton sẽ tìm giá trị nhỏ nhất của tổng tất cả các bình phương sau:

$$\sum_{i=1}^m r_i^2(x)$$

Để dùng phương pháp Newton ta sẽ xét hàm $f(x) = r(x)^T \cdot r(x)$ (còn gọi là hàm mục tiêu).

Bắt đầu tính gradient và Hessian của f :

$$(\nabla f(x))_j = \frac{\partial f(x)}{\partial x_j} = 2 \sum_{i=1}^m r_i(x) \cdot \frac{\partial r_i(x)}{\partial x_j}$$

Chú ý ma trận Jacobian biến x của hàm r :

$$J_r(x) = \frac{\partial r_i}{\partial x_j} = \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix}$$

Do đó: $\nabla f(x) = 2J_r(x)^T \cdot r(x)$

Tiếp tục tính ma trận Hessian của f . Thành phần thứ (k, j) là:

$$\frac{\partial^2 f}{\partial x_k \partial x_j}(x) = \frac{\partial f}{\partial x_k} \left(\frac{\partial f}{\partial x_j}(x) \right) = 2 \sum_{i=1}^m \left(\frac{\partial r_i}{\partial x_k}(x) \cdot \frac{\partial r_i}{\partial x_j}(x) + r_i(x) \cdot \frac{\partial^2 r_i}{\partial x_k \partial x_j}(x) \right)$$

Đặt:

$$S(x) = r_i(x) \cdot \frac{\partial^2 r_i}{\partial x_k \partial x_j}(x)$$

$$\rightarrow F(x) = 2(J_r(x)^T \cdot J_r(x) + S(x))$$

Như vậy phương trình đệ qui cho trường hợp này là:

$$x^{n+1} = x^n - (J_r(x)^T \cdot J_r(x) + S(x))^{-1} \cdot J_r(x)^T \cdot r(x)$$

Trong trường hợp thành phần đạo hàm bậc hai $S(x)$ quá nhỏ có thể bỏ qua khi đó giải thuật này được gọi là **phương pháp Gauss-Newton** với phương trình đệ qui như sau:

$$x^{n+1} = x^n - (J_r(x)^T \cdot J_r(x))^{-1} \cdot J_r(x)^T \cdot r(x)$$

Giải thuật này hội tụ khi lặp đủ số lần cần thiết và đạo hàm bậc hai (Hessian $F(x)$) phải luôn lớn hơn không để đảm bảo về giá trị nhỏ nhất, ngoài ra nếu giá trị ban đầu càng gần với giá trị nhỏ nhất thì hội tụ càng nhanh. Để xem xét tất cả các điều kiện này sẽ rất phức tạp và không phù hợp nhúng xuống vì điều khiển nên ta sẽ kiểm chứng tính hội tụ dựa trên thực tế mô phỏng để chọn số lần lặp cho hợp lý.

b. Phương pháp Gradient

Bài toán đặt ra là tìm giá trị nhỏ nhất của hàm số $f(x)$ bất kì. Quá trình tìm giá trị này sẽ cho giá trị ban đầu chạy theo hướng $-\nabla f(x)$ qua phương trình đệ quy sau:

$$x^{n+1} = x^n - \alpha \cdot \nabla f(x^n)$$

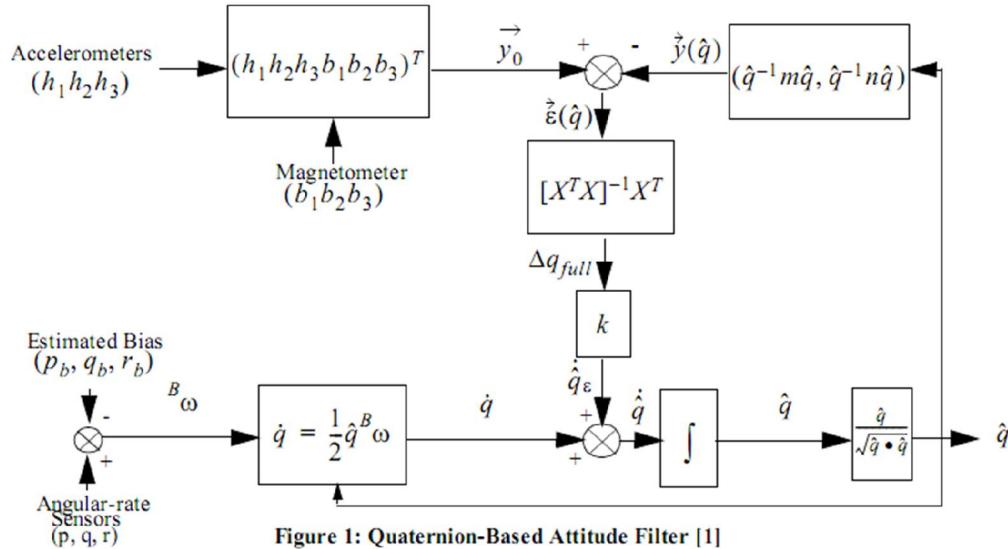
Trong đó $\alpha > 0$ gọi là độ lớn bước nhảy (step size). Giải thuật dừng lại khi $\nabla f \approx \mathbf{0}$ hay $\nabla f \leq \epsilon$

Với trường hợp “nonlinear least-squares”: $\nabla f(x) = 2J_r(x)^T \cdot r(x)$ thì phương trình trở thành:

$$x^{n+1} = x^n - \alpha \cdot 2 \cdot J_r(x)^T \cdot r(x)$$

Cũng tương tự như Gauss-Newton độ lớn bước nhảy α và giá trị ban đầu sẽ ảnh hưởng đến thời gian và tính hội tụ của giải thuật do đó ta sẽ xem xét trong phần mô phỏng. Đặc biệt phương pháp này chỉ cần tính gradient (không có nhân ma trận và nghịch đảo như phương pháp Newton) nên rất đơn giản để nhúng vào vi điều khiển.

2.2.3 Bộ lọc bù dùng phương pháp Gauss-Newton



Hình 2.3 Bộ lọc bù QUEST

Ở giải thuật này ta sẽ dùng phương pháp Gauss-Newton để ước lượng giá trị của quaternion q từ hai nguồn là {Gyrometer} và {Accelerometer, Magnetometer}.

Ta có phương trình động học của vận tốc góc quay: $\dot{q} = \frac{1}{2} q^B \cdot \omega$

Với giá trị \dot{q} này ta có thể tính được quaternion bằng tích phân đơn giản: $q^{n+1} = q^n + \dot{q} \cdot T$ (với T là thời gian giữa các lần chạy giải thuật). Tuy nhiên đặc tính vật lí của Gyrometer là không chính xác ở tần số thấp (chỉ chính xác ở tần số cao) do đó ta phải kết hợp thêm nguồn thông tin từ {Accelerometer, Magnetometer} để rút ra giá trị hợp lí (Accelerometer chính xác ở tần số thấp). Do đó giải thuật Gauss-Newton được dùng ở đây để tối thiểu hóa hàm mục tiêu

$$f(q) = \epsilon(q)^T \cdot \epsilon(q)$$

Trong đó:

$$\epsilon(q) = M^E y - {}^B y$$

- ${}^B y$ chứa sáu giá trị đo từ cảm biến Accelerometer, Magnetometer (hệ gắn với IMU B)
- ${}^E y$ chứa sáu giá trị cố định của gia tốc và từ trường (hệ gắn liền mặt đất E). Trong đó thành phần gia tốc luôn luôn là [0,0,1] còn thành phần từ trường thay đổi theo vị trí địa lý.
- M là ma trận xoay từ hệ E sang B (R_{EB})

Từ $\epsilon(q)$ như trên tính ma trận Jacobian:

$$\begin{aligned} J_\epsilon(q) &= \left[\frac{\partial \epsilon}{\partial q_1} q, \frac{\partial \epsilon}{\partial q_2} q, \frac{\partial \epsilon}{\partial q_3} q, \frac{\partial \epsilon}{\partial q_4} q \right] \\ &= \left[\left(\frac{\partial M}{\partial q_1} q \right) \cdot {}^E y, \left(\frac{\partial M}{\partial q_2} q \right) \cdot {}^E y, \left(\frac{\partial M}{\partial q_3} q \right) \cdot {}^E y, \left(\frac{\partial M}{\partial q_4} q \right) \cdot {}^E y \right] \end{aligned}$$

Khi đó ta có phương trình đệ quy:

$$q^{n+1} = q^n - (J_\epsilon(q)^T \cdot J_\epsilon(q))^{-1} \cdot J_\epsilon(q)^T \cdot \epsilon(q)$$

Lúc này:

$$\dot{\hat{q}} = \dot{q} + k \cdot \dot{q}_\epsilon \text{ (với } k \text{ là hệ số độ lợi)}$$

Cuối cùng ta sẽ tính được:

$$q^{n+1} = q^n + \dot{\hat{q}} \cdot T$$

Phép chuẩn hóa cuối $q = \frac{q}{\sqrt{q \cdot q^T}}$ để đảm bảo tính đơn vị của quaternion

2.2.4 Bộ lọc bù dùng phương pháp Gradient Descent

Tương tự như giải thuật QUEST ở trên, ở giải thuật này ta sẽ dùng phương pháp Gradient Descent để ước lượng. Đặt hàm mục tiêu như sau:

$$f(q) = \epsilon(q)^T \cdot \epsilon(q)$$

Trong đó:

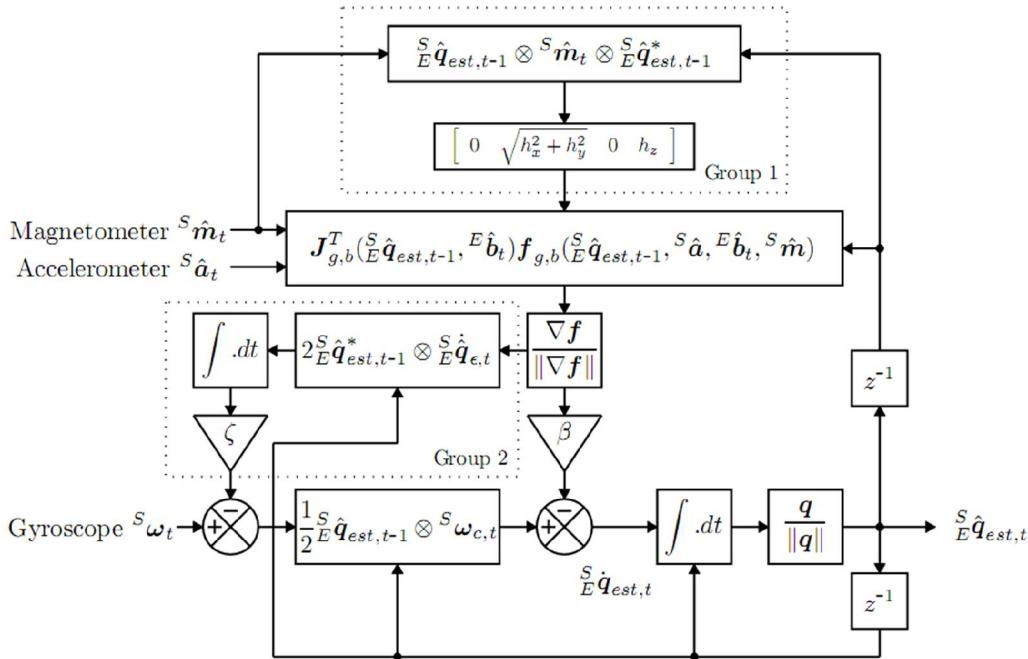
$$\epsilon(q) = M {}^E y - {}^B y$$

$$\nabla f(q) = 2 \cdot J_\epsilon(q)^T \cdot \epsilon(q)$$

Phương trình đệ quy:

$$q^{n+1} = q^n - \beta \cdot \nabla f(q)$$

Các bước tính tương tự mô hình QUEST ở trên



Hình 2.4 Bộ lọc bù Madgwick

2.3 Bộ lọc Kalman

2.3.1 Tổng quan về bộ lọc Kalman

Các bộ lọc Kalman dựa vào giải thuật ước lượng Kalman bao gồm hai phần: đầu tiên là dự báo giá trị (prediction) dựa vào dữ liệu trước đó, sau đó là hiệu chỉnh lại (correction) nhờ vào các giá trị thực tế vừa đo. Bộ lọc này được phát triển ban đầu từ nhu cầu của việc theo dõi quỹ đạo và dẫn đường cho các hệ thống trong không gian của NASA. Sau này nó được áp dụng rộng rãi vào các hệ thống khác như xử lý ảnh, điều khiển...

Các vấn đề của việc hiện thực một bộ lọc Kalman gồm:

- Chỉ ra mô hình toán của hệ thống.
- Ước lượng được các covariance của nhiễu: do mô hình & đo lường.

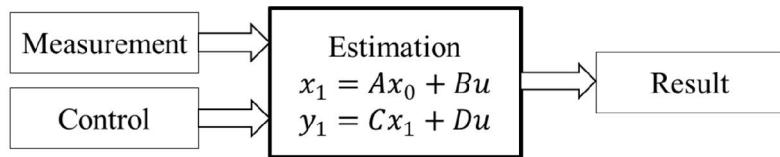
Xét hệ thống *tuyến tính* tổng quát được biểu diễn bởi hệ phương trình trạng thái sau:

$$\begin{cases} x_{k+1} = A \cdot x_k + B \cdot u_{k+1} + w_k \\ y_k = C \cdot x_k + D \cdot u_k + v_k \end{cases}$$

Trong đó:

- A, B, C, D : các ma trận **hằng** (phân biệt với hệ phi tuyến là các hàm số)
- x_k : giá trị trạng thái của hệ thống tại thời điểm kT
- y_k : ngõ ra của hệ thống tại thời điểm kT

- w_k : nhiễu do mô hình hay nhiễu quá trình (process noise)
- v_k : nhiễu do đo lường (measurement noise)

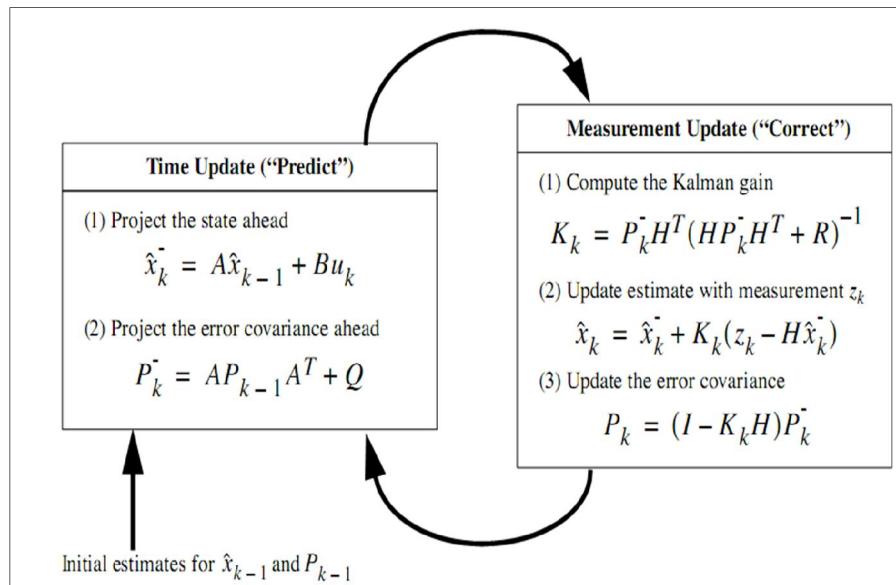


Hình 2.5 Sơ đồ hệ thống ước lượng

Để tìm ngõ ra y của hệ thống đối với các ngõ vào u khác nhau và các trạng thái x khác nhau của hệ thống ta phải thiết lập được hệ phương trình trên. Tuy nhiên việc tính toán có thể dẫn ra kết quả sai với thực tế do:

1. Mô hình toán sai vì tuyến tính hóa bỏ qua các thành phần bậc cao, bỏ qua các đại lượng như nhiệt độ, ma sát...
2. Nhiễu do đo đạc từ cảm biến hay phương pháp đo...

Do đó yêu cầu đặt ra là cần có các bộ lọc hay giải thuật ước lượng để tối ưu hóa giá trị tính toán. Các bước tính toán trong giải thuật Kalman như sau:



Hình 2.6. Giải thuật Kalman

Các kí hiệu dùng trong này là:

- A, B : các hệ số của phương trình vi phân tuyến tính.
- P : ma trận covariance của nhiễu ước lượng được (P^- & P^+ là giá trị trước & sau).
- Q : ma trận covariance của nhiễu do quá trình (hiệp phương sai).
- R : ma trận covariance của nhiễu do đo lường.

- K : ma trận độ lợi.
- H : ma trận độ nhạy xác định quan hệ tuyến tính giữa tín hiệu của hệ thống và tín hiệu ước lượng.

Để xem xét giải thuật ước lượng Kalman ta sẽ tìm hiểu ví dụ sau:

Một chiếc xe được đặt vào gia tốc $a = 1m/s^2$ không đổi, tính toán vị trí của nó sau mỗi $T = 0.1s$. Với nhiễu của phép đo vị trí là $10m$ (độ lệch chuẩn), nhiễu của gia tốc là $0.2m/s^2$ (độ lệch chuẩn)

Giải:

Ta có các phương trình động học đơn giản sau: (bỏ qua các yếu tố khác)

- Vận tốc xe: $v_{k+1} = v_k + aT$
- Vị trí xe: $p_{k+1} = p_k + v_k T + \frac{1}{2} aT^2$

Mô hình hóa hệ thống như sau:

- Tín hiệu ngõ vào (điều khiển): $u = a$
- Ngõ ra: $y = p$
- Các biến trạng thái: $x = [p, v]^T$
- Nhiễu quá trình: $w = u^\sim$
- Nhiễu đo lường: $v = p^\sim$

Phương trình hệ thống như sau:

$$\begin{cases} x_{k+1} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} \frac{T^2}{2} \\ T \end{bmatrix} u_k + w_k \\ y_{k+1} = [1 \ 0] x_k + v_k \end{cases}$$

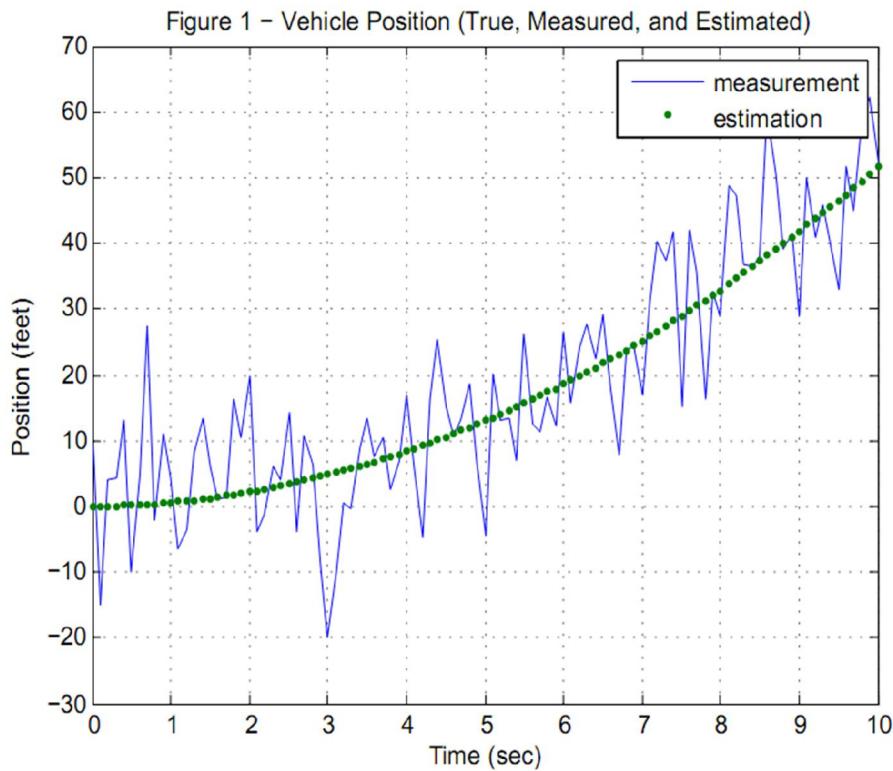
Nhận xét:

- Covariance nhiễu quá trình:

$$\begin{aligned} Q &= E(x \cdot x^T) = E\left(\begin{bmatrix} p \\ v \end{bmatrix} \begin{bmatrix} p & v \end{bmatrix}\right) = E\left(\begin{bmatrix} p^2 & qv \\ pv & v^2 \end{bmatrix}\right) = \begin{bmatrix} E(p^2) & E(pv) \\ E(vp) & E(v^2) \end{bmatrix} \\ &= a_{noise}^2 \cdot \begin{bmatrix} T^4/4 & T^3/2 \\ T^3/2 & T^2 \end{bmatrix} = \begin{bmatrix} 10^{-6} & 2 \cdot 10^{-5} \\ 2 \cdot 10^{-5} & 4 \cdot 10^{-4} \end{bmatrix} \end{aligned}$$

- Covariance nhiễu đo lường $R = E(y \cdot y^T) = E([p] \cdot [p]) = E(p^2) = 10^2 = 100$

Từ các ma trận trên ta mô phỏng được kết quả như sau:



Hình 2.7 Kết quả mô phỏng Kalman cho hệ tuyến tính

Ở đây đường liền nét là giá trị đo “raw” chưa được xử lí, đường chấm và đứt là giá trị ước lượng “estimate”. Ta thấy sau khi qua lọc Kalman thì tín hiệu tốt hơn rất nhiều so với giá trị “raw”.

Một thực tế là rất nhiều các hệ thống thực đều là phi tuyến do đó khi áp dụng Kalman cho các đối tượng này ta phải dùng bộ lọc **Kalman mở rộng (Extended-Kalman Filter)**

Ta bắt đầu với mô hình tổng quát của hệ thống phi tuyến:

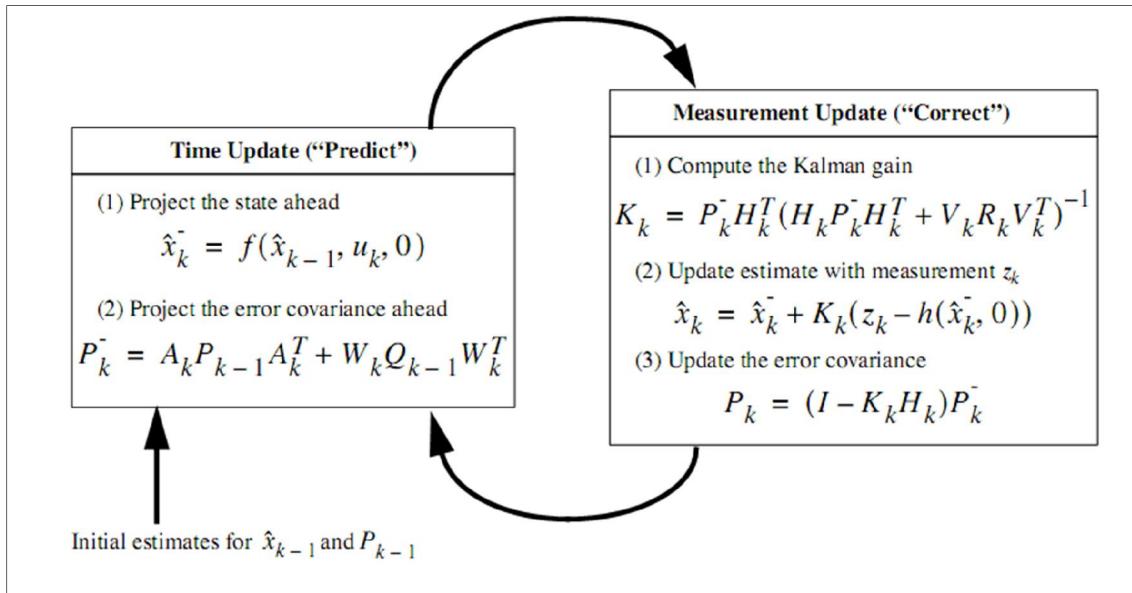
$$\begin{cases} x_{k+1} = f(x_k, u_{k+1}, w_k) \\ y_k = h(x_k, u_k, v_k) \end{cases}$$

Đặt các ma trận Jacobian:

$$A = \frac{\partial f_i}{\partial x_j}(x_k, u_{k+1}, 0) \rightarrow A_k = I + A \cdot T$$

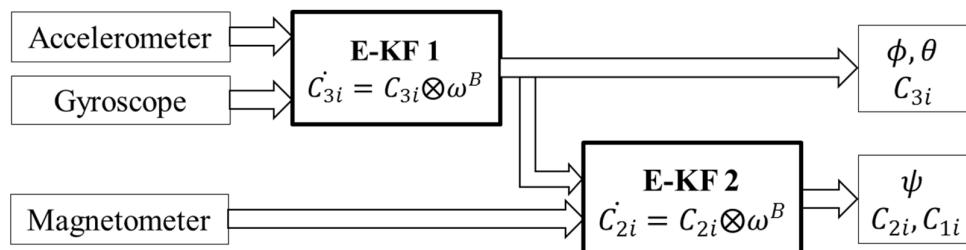
$$H_k = \frac{\partial h_i}{\partial x_j}(x_k, u_{k+1}, 0)$$

Các bước tính toán như sau:



Hình 2.8. Giải thuật bộ lọc Kalman mở rộng

2.3.2 Bộ lọc Kalman mở rộng dùng DCM



Hình 2.9 Mô hình Kalman dùng DCM

Xây dựng KF1:

Phương trình động học dựa vào DCM:

$$C_b^n = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}$$

$$\dot{C}_b^n = C_b^n \cdot \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

Rút riêng hàng thứ 3 của ma trận DCM ta được phương trình:

$$\begin{pmatrix} \dot{C}_{31}^n \\ \dot{C}_{32}^n \\ \dot{C}_{33}^n \end{pmatrix} = \begin{pmatrix} 0 & -C_{33} & C_{32} \\ C_{33} & 0 & -C_{31} \\ -C_{32} & C_{31} & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

Phương trình đo lường cho Accelerometer:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = {}^n_b C^T \cdot \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} C_{31} \\ C_{32} \\ C_{33} \end{pmatrix} \cdot g$$

Từ đó ta xây dựng được mô hình toán cho khói lọc Kalman 1 như sau:

- $x = [C_{31}, C_{32}, C_{33}, \omega_x, \omega_y, \omega_z]^T$
- $y = [a_x, a_y, a_z, \omega_x, \omega_y, \omega_z]^T$
- $\dot{x} = \phi \cdot x + w \rightarrow \phi = \begin{bmatrix} 0_{3x3} & C_{3i} \\ 0_{3x3} & 0_{3x3} \end{bmatrix}$
- $y = H \cdot x + v \rightarrow H = \begin{bmatrix} g \cdot I_{3x3} & 0_{3x3} \\ 0_{3x3} & I_{3x3} \end{bmatrix}$

Giả sử nhiễu từ Gyrometer 3 trục không tương quan nhau ta sẽ có biểu thức covariance của nhiễu quá trình:

$$Q = E(w \cdot w^T) = \begin{bmatrix} 0_{3x3} & 0_{3x3} \\ 0_{3x3} & q \cdot I_{3x3} \end{bmatrix}$$

Và biểu thức covariance của nhiễu đo lường:

$$R = E(v \cdot v^T) = \begin{bmatrix} r_a I_{3x3} & 0_{3x3} \\ 0_{3x3} & r_\omega \cdot I_{3x3} \end{bmatrix}$$

Rời rạc hóa các covariance:

$$\begin{aligned} \phi_k &= \begin{pmatrix} I_{3x3} & \widehat{C}_3 \cdot dT \\ 0_{3x3} & I_{3x3} \end{pmatrix} \\ Q_k &= q \begin{pmatrix} dT^3 \cdot \widehat{C}_3 \cdot \widehat{C}_3^T & dT^2 \cdot \widehat{C}_3 \\ dT^2 \cdot \widehat{C}_3^T & dT \cdot I_{3x3} \end{pmatrix} \end{aligned}$$

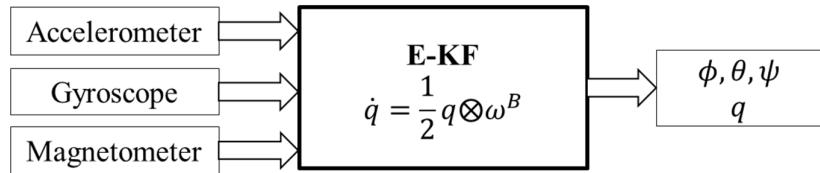
Trong đó r_a, r_ω là variance của Accelerometer và Gyrometer dựa vào điều kiện

$$|a_x^2 + a_y^2 + a_z^2 - 1| > \sigma$$

Xây dựng mô hình cho khói Kalman 2:

- $x = [C_{21}, C_{22}, C_{23}, \omega_x, \omega_y, \omega_z]^T$
- $y = [C_{21}, C_{22}, C_{23}, \omega_x, \omega_y, \omega_z]^T$
- $\dot{x} = \phi \cdot x + w \rightarrow \phi = \begin{bmatrix} 0_{3x3} & C_{2i} \\ 0_{3x3} & 0_{3x3} \end{bmatrix}, \dot{C}_{2i} = \begin{pmatrix} 0 & -C_{23} & C_{22} \\ C_{23} & 0 & -C_{21} \\ -C_{22} & C_{21} & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$
- $y = H \cdot x + v \rightarrow H = [I_{6x6}]$
- $Q = \begin{bmatrix} 0_{3x3} & 0_{3x3} \\ 0_{3x3} & q \cdot I_{3x3} \end{bmatrix}$
- $R = \begin{bmatrix} \mu \cdot I_{3x3} & 0_{3x3} \\ 0_{3x3} & r \cdot I_{3x3} \end{bmatrix}$

2.3.3 Bộ lọc Kalman mở rộng dùng Quaternion



Hình 2.10. Mô hình Kalman dùng Quaternion

Đầu tiên ta có hệ phương trình động học của hệ thống như sau:

$$\left\{ \begin{array}{l} \dot{\omega}_x = -\frac{1}{\tau_x} \omega_x + w_x \\ \dot{\omega}_y = -\frac{1}{\tau_y} \omega_y + w_y \\ \dot{\omega}_z = -\frac{1}{\tau_z} \omega_z + w_z \\ \dot{q}_1 = \frac{1}{2} (\omega_x \cdot q_4 - \omega_y \cdot q_3 + \omega_z \cdot q_2) \\ \dot{q}_2 = \frac{1}{2} (\omega_x \cdot q_3 + \omega_y \cdot q_4 - \omega_z \cdot q_1) \\ \dot{q}_3 = \frac{1}{2} (-\omega_x \cdot q_2 + \omega_y \cdot q_1 + \omega_z \cdot q_4) \\ \dot{q}_4 = \frac{1}{2} (-\omega_x \cdot q_1 - \omega_y \cdot q_2 - \omega_z \cdot q_3) \end{array} \right.$$

Đặt $x = [x_1, \dots, x_7] = [\omega_x, \omega_y, \omega_z, q_1, q_2, q_3, q_4]$ (Với q_4 : phần thực của quaternion)

Tính ma trận Jacobian:

$$A = \frac{\partial f_i}{\partial x_j} = \begin{bmatrix} -\frac{1}{\tau_x} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{\tau_y} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{\tau_z} & 0 & 0 & 0 & 0 \\ 0.5x_7 & -0.5x_6 & 0.5x_5 & 0 & 0.5x_3 & -0.5x_2 & 0.5x_1 \\ 0.5x_6 & 0.5x_7 & -0.5x_4 & -0.5x_3 & 0 & 0.5x_1 & 0.5x_2 \\ -0.5x_5 & 0.5x_4 & 0.5x_7 & 0.5x_2 & -0.5x_1 & 0 & 0.5x_3 \\ -0.5x_4 & -0.5x_5 & -0.5x_6 & -0.5x_1 & -0.5x_2 & -0.5x_3 & 0 \end{bmatrix}$$

$$\rightarrow A_k = \phi \approx I + A.T$$

$$= \begin{bmatrix} 1 - \frac{T}{\tau_x} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 - \frac{T}{\tau_y} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 - \frac{T}{\tau_z} & 0 & 0 & 0 & 0 \\ 0.5Tx_7 & -0.5Tx_6 & 0.5Tx_5 & 1 & 0.5Tx_3 & -0.5Tx_2 & 0.5Tx_1 \\ 0.5Tx_6 & 0.5Tx_7 & -0.5Tx_4 & -0.5Tx_3 & 1 & 0.5Tx_1 & 0.5Tx_2 \\ -0.5Tx_5 & 0.5Tx_4 & 0.5Tx_7 & 0.5Tx_2 & -0.5Tx_1 & 1 & 0.5Tx_3 \\ -0.5Tx_4 & -0.5Tx_5 & -0.5Tx_6 & -0.5Tx_1 & -0.5Tx_2 & -0.5Tx_3 & 1 \end{bmatrix}$$

Thêm nữa đặt $y = [\omega_x, \omega_y, \omega_z, a_x, a_y, a_z, m_x, m_y, m_z]$ thì:

$$y = h(x) = \begin{bmatrix} \omega_{x,y,z} \\ R_{3x3} \cdot g_0 \\ R_{3x3} \cdot m_0 \end{bmatrix}$$

Với R_{3x3} là ma trận xoay:

$$R_{3x3} = \begin{pmatrix} 2q_1^2 + 2q_4^2 - 1 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & 2q_2^2 + 2q_4^2 - 1 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & 2q_3^2 + 2q_4^2 - 1 \end{pmatrix}$$

$$H_k = \frac{\partial h}{\partial x} = \begin{bmatrix} I_{3x3} \\ \frac{\partial R_{3x3} \cdot g_0}{\partial x} \\ \frac{\partial R_{3x3} \cdot m_0}{\partial x} \end{bmatrix}$$

$$\frac{\partial R}{\partial q_1} = \begin{pmatrix} 4q_1 & 2q_2 & 2q_3 \\ 2q_2 & 0 & 2q_4 \\ 2q_3 & -2q_4 & 0 \end{pmatrix}$$

$$\frac{\partial R}{\partial q_2} = \begin{pmatrix} 0 & 2q_1 & -2q_4 \\ 2q_1 & 4q_2 & 2q_3 \\ 2q_4 & 2q_3 & 0 \end{pmatrix}$$

$$\frac{\partial R}{\partial q_3} = \begin{pmatrix} 0 & 2q_4 & 2q_1 \\ -2q_4 & 0 & 2q_2 \\ 2q_1 & 2q_2 & 4q_3 \end{pmatrix}$$

$$\frac{\partial R}{\partial q_4} = \begin{pmatrix} 4q_4 & 2q_3 & -2q_2 \\ -2q_3 & 4q_4 & 2q_1 \\ 2q_2 & -2q_1 & 4q_4 \end{pmatrix}$$

$$\rightarrow H = \begin{bmatrix} I_{3x3} & 0 & 0 & 0 & 0 \\ 0_{3x3} & \frac{\partial R}{\partial q_1} g_0 & \frac{\partial R}{\partial q_2} g_0 & \frac{\partial R}{\partial q_3} g_0 & \frac{\partial R}{\partial q_4} g_0 \\ 0_{3x3} & \frac{\partial R}{\partial q_1} m_0 & \frac{\partial R}{\partial q_2} m_0 & \frac{\partial R}{\partial q_3} m_0 & \frac{\partial R}{\partial q_4} m_0 \end{bmatrix} x$$

Các ma trận covariance:

- $Q_k = \begin{bmatrix} q_{3x3} & 0_{4x3} \\ 0_{4x3} & 0_{4x4} \end{bmatrix}$ với $q_{3x3} = \begin{bmatrix} q_{11} & 0 & 0 \\ 0 & q_{22} & 0 \\ 0 & 0 & q_{33} \end{bmatrix}$
- $R_k = diag([r_{11}, r_{22}, r_{33}, r_{44}, r_{55}, r_{66}, r_{77}, r_{88}, r_{99}])$ Với $diag()$: ma trận đường chéo

2.4 Mô phỏng và đánh giá các bộ ước lượng

2.4.1 Phương pháp mô phỏng và đánh giá

Ta sẽ dùng mẫu dữ liệu thu được từ IMU thương mại có độ chính xác rất cao của hãng Xsens bao gồm các giá trị:

- Ba góc ước lượng
- Ba giá trị vận tốc góc ba trục
- Ba giá trị gia tốc ba trục
- Ba giá trị từ trường ba trục

Ta sẽ dùng MATLAB Simulink để mô phỏng lần lượt các mẫu thử bằng bốn phương pháp lọc đã liệt kê ở trên:

- Bộ lọc bù bằng Gauss-Newton
- Bộ lọc bù bằng Gradient Descent
- Bộ lọc Kalman mở rộng bằng DCM
- Bộ lọc Kalman mở rộng bằng quaternion

Dữ liệu thô là các giá trị thu được từ IMU (x^{raw}) sẽ được dùng để ước lượng ra ba góc nghiêng (x^{est}) sau đó so sánh với giá trị chính xác từ IMU của (x^{ref}).

Với mỗi bộ lọc ta sẽ thử 5 mẫu dữ liệu với các kiểu quay khác nhau được kí hiệu:

- Xoay cảm biến tự do: EX-F2
- Xoay cảm biến tự do: EX-F1
- Đặt cảm biến đứng im: EX-S
- Di chuyển cảm biến dọc trục x: EX-XMOVE
- Xoay cảm biến quanh trục z: EX-Z

Để đánh giá sai số của phép ước lượng, ta dùng giá trị RMS của sai số giữa giá trị ước lượng và giá trị chính xác:

$$e_{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^N e_i^2}$$

Với:

$$e_i = x_i^{est} - x_i^{ref}$$

Chú ý: ở đây đối với các sai số do chuyển từ các góc ± 180 độ ta sẽ cho $e = 0$

2.4.2 Kết quả và nhận xét

Các kí hiệu:

- G-N: Bộ lọc bù bằng Gauss-Newton
- GD: Bộ lọc bù bằng Gradient Descent
- E-KF: Bộ lọc Kalman mở rộng bằng quaternion
- DCM: Bộ lọc Kalman mở rộng bằng DCM

Bảng 2.1 Kết quả mô phỏng đối với bộ lọc bù

STT	Mẫu thử	Bộ lọc bù	Sai số RMS (độ)		
			ϕ	θ	ψ
1	EX-F2	G-N	0.8784	0.3733	0.9637
		GD ($\beta = 0.1$)	2.0067	0.5009	3.1664
2	EX-F1	G-N	3.7005	2.489	4.945
		GD ($\beta = 0.4$)	9.1692	7.0878	21.935
3	EX-S	G-N	0.0457	0.0533	0.1271
		GD ($\beta = 0.07$)	0.0564	0.0664	0.0703
4	EX-XMOVE	G-N	0.1724	0.7104	1.1006
		GD ($\beta = 0.1$)	0.597	0.5292	1.2589
5	EX-Z	G-N	0.4185	0.2655	10.7354
		GD ($\beta = 0.5$)	0.9263	0.7639	5.0267

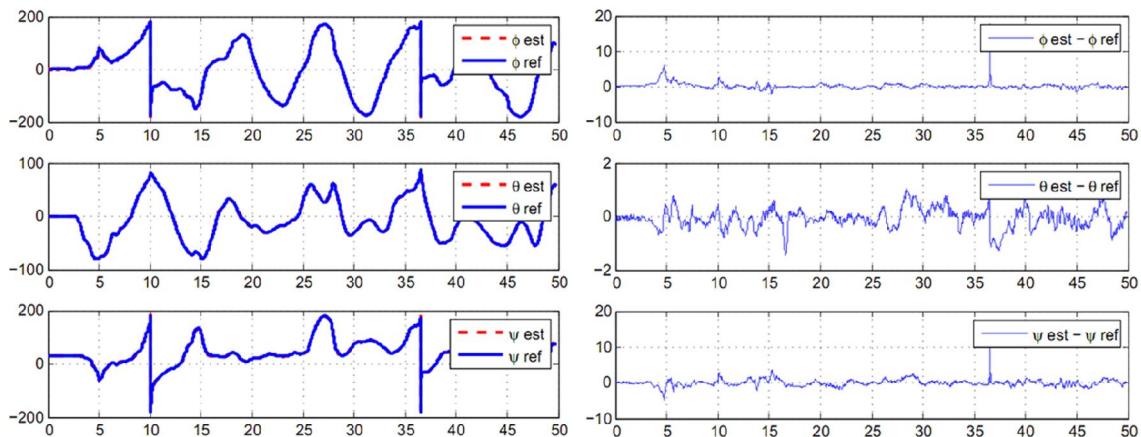
Bảng 2.2 Kết quả mô phỏng đối với bộ lọc Kalman

STT	Mẫu thử	Bộ lọc Kalman	Sai số RMS (độ)		
			ϕ	θ	ψ
1	EX-F2	E-KF	1.2193	0.419	4.3721
		DCM	1.4836	0.7122	1.5629
2	EX-F2	E-KF (***)	2.0442/9.0854	1.2628/11.2862	45.2105/8.0918
		DCM	2.0167	1.0612	2.3396
3	EX-S	E-KF	0.1924	0.6838	1.062
		DCM	0.1086	0.1115	0.3944
4	EX-XMOVE	E-KF	2.3144	2.0432	3.7116
		DCM	0.3357	0.5617	1.9229
5	EX-Z	E-KF	0.8362	0.5016	31.1851
		DCM	0.1762	0.1239	3.5559

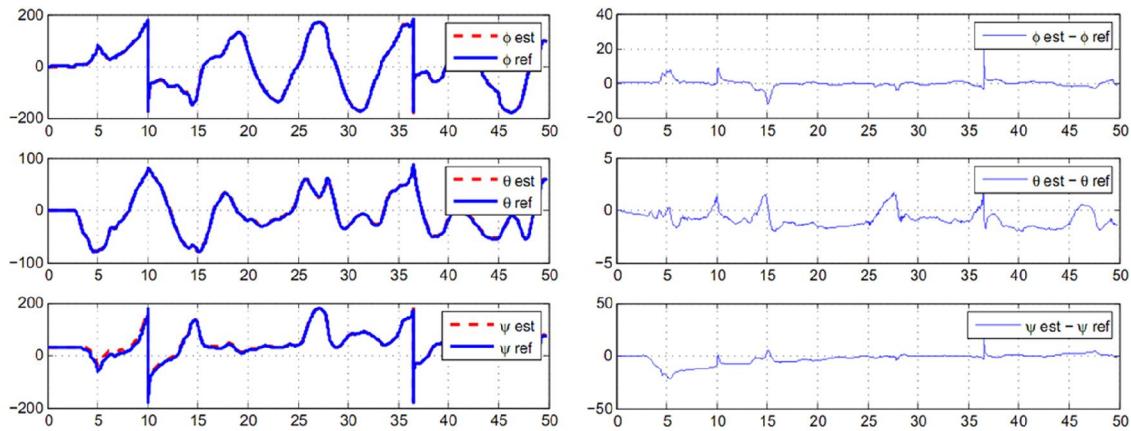
Chú ý: trường hợp (***) cho thấy sự đánh đổi khi lựa chọn tối ưu góc ϕ, θ, ψ trong hai trường hợp.

Nhận xét:

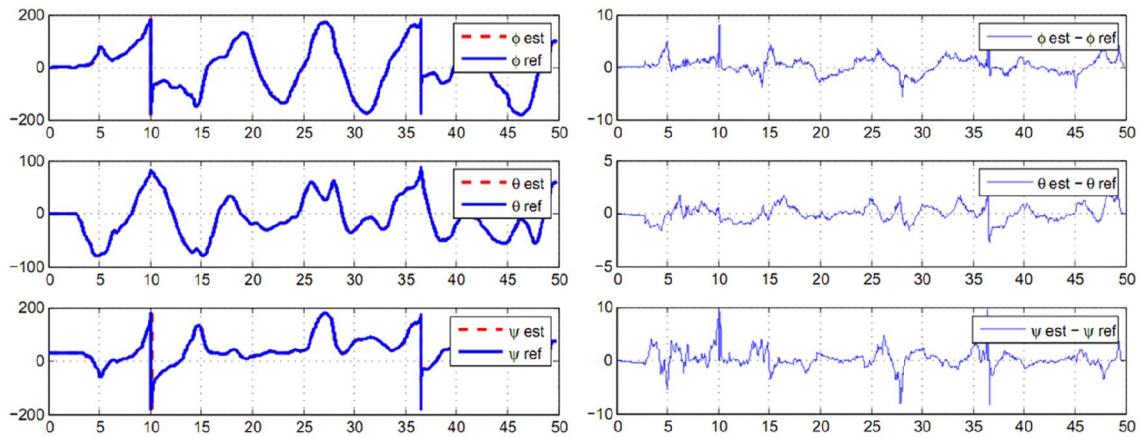
- Các phương pháp dùng bộ lọc bù tuy đơn giản nhưng các hệ số không “thích nghi” được các trường hợp chuyển động khác nhau của IMU.
- Các bộ lọc dùng Kalman cho kết quả khá tốt khi thay đổi các điều kiện khác nhau và bộ lọc dùng DCM thì tốt hơn Kalman mở rộng vì đã tách được ảnh hưởng của giá trị từ trường ra khỏi ước lượng ϕ và θ .
- Trong điều kiện không có gia tốc ngoài thì Gyrometer & Accelerometer ước lượng góc ϕ, θ khá tốt còn việc ước lượng góc ψ không được tốt.



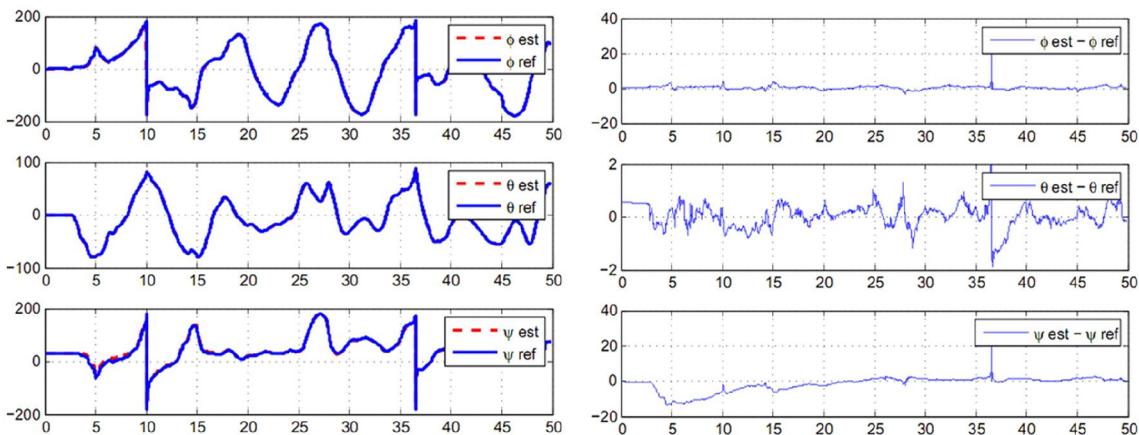
Hình 2.11 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Gauss-Newton



Hình 2.12 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Gradient Descent



Hình 2.13 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Kalman DCM



Hình 2.14 Biểu đồ ba góc và sai số của mẫu EX-F2 dùng Kalman Quaternion

Có ba vấn đề (được sắp xếp theo mức độ phức tạp dần) cần phải giải quyết đối với một thuật toán ước lượng cho IMU đó là:

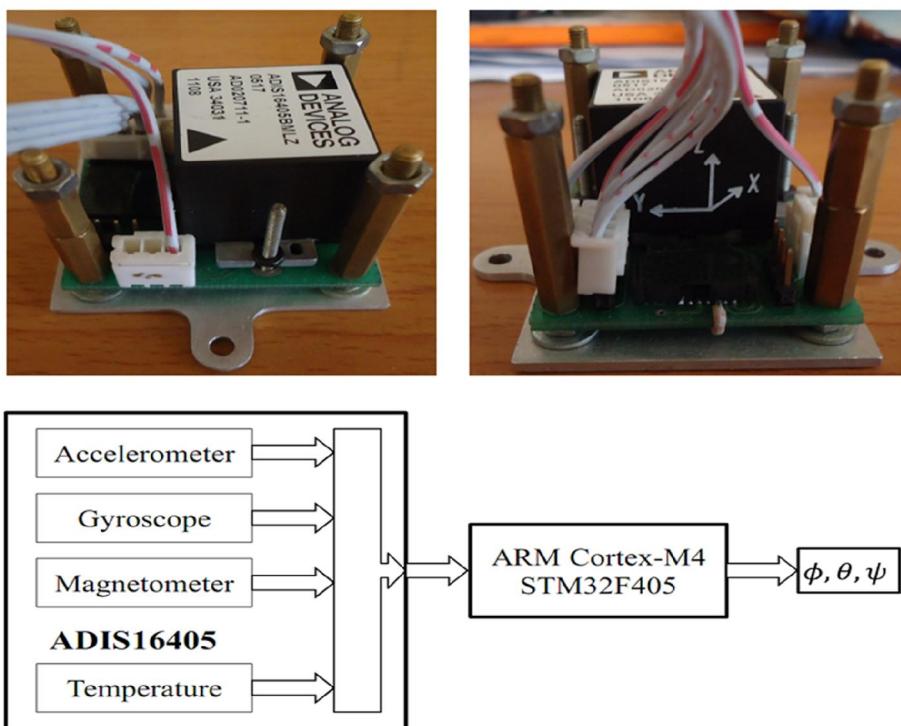
1. Đáp ứng tĩnh tốt ở điều kiện thông thường (gia tốc ngoài nhỏ, ít nhiễu từ trường)
2. Đáp ứng động tốt ngay cả khi có gia tốc ngoài
3. Đáp ứng tốt khi có nhiễu từ trường can thiệp

Hai phương pháp bù và Kalman ở trên cơ bản là giống nhau khi giải quyết vấn đề [1] tuy nhiên mô hình Kalman mở rộng sẽ có ưu điểm hơn khi giải quyết vấn đề [2] và [3] vì Kalman cho phép chỉnh định các covariance của từng thành phần giá trị dùng trong mô hình. Đặc biệt mô hình DCM cho phép tách ảnh hưởng của từ trường lên hai góc roll và pitch nên ở phần tiếp theo sẽ tập trung phát triển và cải tiến **giải thuật Kalman mở rộng dùng DCM** cho IMU.

Chương 3. Xây dựng bộ ước lượng

3.1 Hệ thống IMU tích hợp ARM Cortex-M4 STM32F40x và cảm biến ADIS16405

Hệ thống IMU 9-DOF bao gồm vi điều khiển ARM-Cortex M4 STM32F405 hỗ trợ tính toán dấu chấm động và cảm biến 9-DOF ADIS16405 (gồm ba loại cảm biến Gia tốc ba trục, Vận tốc góc ba trục, Từ trường ba trục) được chế tạo tại phòng thí nghiệm Điều Khiển Tự Động - Bộ môn Tự Động, Đại học Bách Khoa TPHCM. Sau đây là hình ảnh và mô hình của hệ thống này:



Hình 3.1 IMU-9DOF được chế tạo tại PTN Bộ môn Tự Động ĐHBK TPHCM

Trong luận văn này ta sẽ sử dụng phần cứng IMU ở trên để kiểm tra giải thuật ước lượng ba góc quay. Sau đây ta sẽ tìm hiểu về dòng vi điều khiển ARM Cortex-M4 và cảm biến ADIS16405 cũng như cách calib loại cảm biến này.

3.1.1 Vi điều khiển ARM Cortex-M4F STM32F40x

a. Giới thiệu vi xử lí ARM Cortex-M4F

Các vi điều khiển sử dụng lõi ARM ngày càng phổ biến trên các sản phẩm như: di động, các thiết bị điện tử cầm tay, các thiết bị điện tử, ứng dụng trong xe hơi... Các đặc điểm của vi điều khiển dùng lõi ARM là giá thành rẻ, xử lý tốc độ cao, khả năng tiết kiệm năng lượng và một lợi thế rất lớn là có rất nhiều dòng vi điều khiển cung cấp đa dạng các ngoại vi bởi các nhà sản xuất chip hàng đầu như TI (với dòng LM3Sx), ST (với dòng STM32x) và NXP (với dòng LPCx). Cùng với một cộng đồng dùng ARM rộng lớn và sự hỗ trợ thư viện lập trình cũng như thiết kế mạch mẫu từ các nhà sản xuất lớn này việc ứng dụng các dòng vi điều khiển này vào sản phẩm trở nên dễ dàng và nhanh chóng. Do đó các vi điều khiển lõi ARM là lựa chọn của nhiều nhà sản xuất điện tử hiện nay.

Trong các dòng ARM, dòng Cortex-M (M0, M1, M3, M4) được sản xuất nhằm đến ứng dụng nhúng yêu cầu tốc độ vừa phải, nhiều nguồn ngắn và đáp ứng ngắn nhanh. Các ngoại vi đi kèm do các nhà sản xuất vi điều khiển thêm vào cũng rất đa dạng ví dụ: LPC1769 của NXP sử dụng lõi ARM Cortex-M3 có các ngoại vi như: UART, Ethernet, USB, CAN là các chuẩn truyền thông dùng nhiều trong các sản phẩm điện tử. Sử dụng các vi điều khiển này để tiếp cận với các ứng dụng nhúng hay điều khiển tự động trong phạm vi các đồ án và luận văn là lựa chọn phổ biến và phù hợp với xu hướng hiện nay.

Vi xử lí Cortex- M4F là bản “nâng cấp” của Cortex-M4 với khả năng tính toán số thực FPU (Floating Point Unit) và vẫn giữ nguyên các khả năng khác nên ta gọi gọn là vi xử lí Cortex-M4 để đề cập tới vi xử lí này. Các ưu điểm của vi xử lí Cortex-M4:

- Cơ chế xử lí ngắn lồng nhau NVIC (Nested Vector Interrupt Controller) cho phép xử lí nhiều ngắn xảy ra đồng thời với thời gian trễ thấp (low latency).
- Bộ xử lí số thực FPU (Floating Point Unit).

Ở đây hai đặc điểm quan trọng cần tìm hiểu để việc xây dựng phần mềm trên các vi điều khiển dùng vi xử lí này linh hoạt và an toàn hơn là: cơ chế xử lí ngắn NVIC và bộ xử lí số thực FPU.

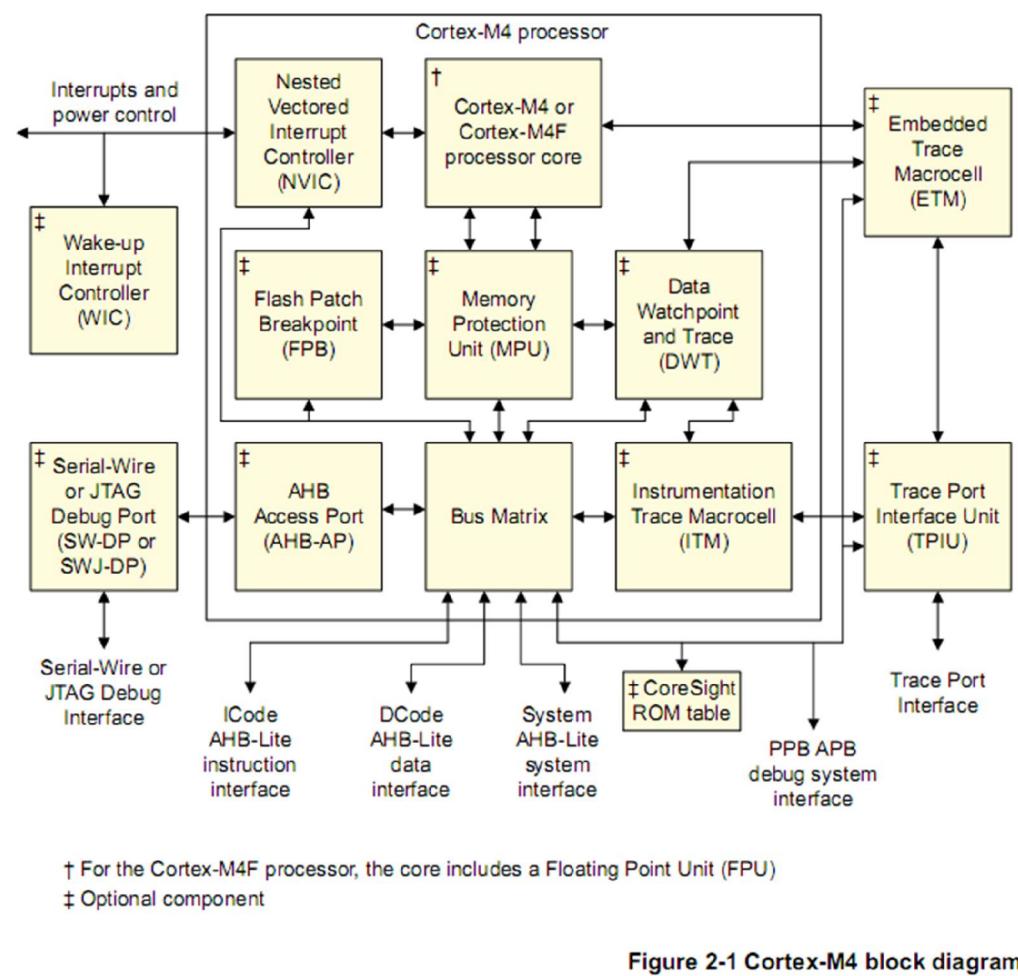
Bộ điều khiển ngắn lồng nhau NVIC: cho phép cài đặt các khả năng

- Xử lí nhiều ngắn (240 ngắn) với nhiều mức ưu tiên (256 mức)
- Thời gian trễ thấp (low-latency)
- Tiết kiệm năng lượng

Lõi ARM cung cấp 2 loại ngắn:

- Ngắt mức (level interrupt): sẽ được giữ đến khi ngắt bị xóa. Nếu không bị xóa thì ngắt này sẽ được kích hoạt lại ngay khi thoát ra khỏi chương trình phục vụ ngắt của nó. Việc này đảm bảo không có thời gian trễ một khi muốn vi xử lí luôn trong ngắt khi có ngắt mức xảy ra.
- Ngắt xung (pulse interrupt): nếu có thêm các xung nữa cùng đến trong trạng thái xử lí thì ngắt cũng chỉ xử lí một lần.

Figure 2-1 shows the structure of the Cortex-M4 processor.



Hình 3.2 Sơ đồ khái lõi ARM Cortex-M4

Bộ xử lí số thực FPU: FPU trong Cortex – M4 dựa trên tiêu chuẩn IEEE 754 (IEEE Standard 754 for Binary Floating-Point Arithmetic) qui định cách xử lí số thực dưới dạng nhị phân. FPU cung cấp ba chế độ cài đặt:

- Full-compliance mode: Hoàn toàn theo chuẩn IEEE-754 về phần cứng

- Flush-to-zero mode: các số subnormal trong các phép toán sẽ qui về 0 trước khi tính hoặc các kết quả phép toán cũng được qui về 0
- Default NaN mode: các số hạng là NaN sẽ tạo ra các kết quả cũng là NaN. Các kiểu phép tính gây ra NaN là: căn bậc hai của số âm, nhân 0 và vô cùng, chia cho 0, chia cho vô cùng, phần dư của phép chia 0, phần dư của phép chia vô cùng cho bất cứ số nào, cộng trừ hai số vô cùng.

FPU hỗ trợ đầy đủ các phép toán với độ chính xác đơn (single-precision) là: cộng, trừ, nhân, chia, nhân cộng dồn, căn bậc hai. Tuy nhiên không hỗ trợ các phép tính sau: tính lấy dư, làm tròn số thực sang số nguyên giữ nguyên dạng dấu chấm động (floating point number to integer-valued floating-point number), chuyển nhị phân sang thập phân, thập phân sang nhị phân, so sánh trực tiếp giữa hai số dạng chính xác đơn (single-precision) và chính xác kép (double-precision). Các phép tính này phải được hỗ trợ bằng các hàm thư viện.

Bảng thống kê số chu kỳ máy tốn để thực hiện các phép toán phổ biến với số dạng dấu chấm động trên FPU:

Bảng 3.1 Tóm tắt số chu kỳ máy khi thực hiện lệnh số học dùng FPU

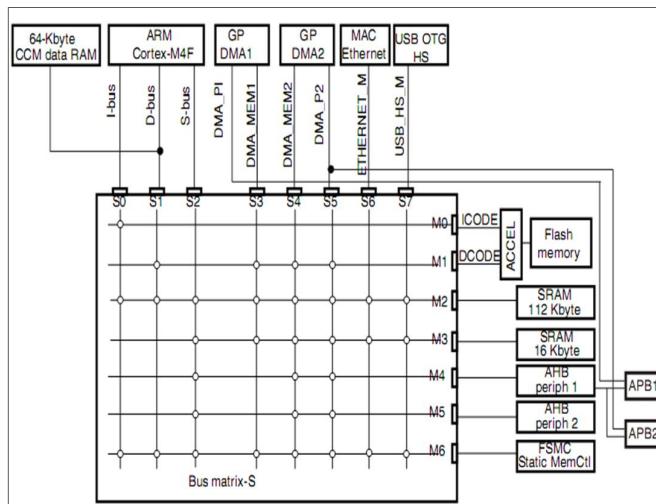
Toán hạng	Số chu kỳ	Mô tả
Trị tuyệt đối	1	
Cộng	1	
Trừ	1	
Nhân	1	
Nhân xong cộng/trừ dồn	3	
Nhân xong cộng/trừ dồn rồi đảo dấu	3	
Chia	14	
Căn bậc hai	14	
So sánh	1	Với 0
Chuyển đổi	1	Giữa 2 số nguyên, 2 số thực.
Gọi N số (Load)	Double 64 bit: 1+2*N Float 32 bit: 1+N	Double hoặc float
Chuyển thanh ghi (Move)	Hầu hết là 1 chu kỳ	

b. Giới thiệu vi điều khiển STM32F40x

Vi điều khiển STM32F40x là dòng vi điều khiển 32 bit của STMicroelectronics sử dụng lõi vi xử lí ARM Cortex-M4F. Các ngoại vi hỗ trợ bởi STM32F40x gồm:

- Bộ nhớ: 1 Mbyte bộ nhớ flash, 192 Kbyte SRAM + 4 Kbyte SRAM dự trữ dùng cho RTC
- Điện áp, nguồn clock: 1.8 V – 3.6V, 4 Mhz – 26Mhz thạch anh ngoài
- Chế độ standby tiết kiệm năng lượng
- GPIO: 82 chân (đối với STM32F407)
- ADC: 12 bit, 10 bit, 8 bit hoặc 6 bit
- DAC: 12 bit
- 16 bộ DMA, 8 kênh trên mỗi DMA
- Timer: 16 bit, 32 bit
- Hỗ trợ các chuẩn debug:
 - JTAG, SWD (Serial Wire Debug)
 - Cortex-M4F Embedded Trace Macrocell
- Hỗ trợ các giao thức truyền thông :
 - Giao tiếp nối tiếp: I2C, USART, SPI, I2S, CAN (2.0B), SDIO, USB 2.0, Ethernet
 - Giao tiếp song song FSMC: Flash, SRAM, PC Card...
- 8 đến 14 bit giao tiếp camera đến 54Mbyte/s
- Bộ tạo giá trị ngẫu nhiên (analog random number generator)
- RTC

Kiến trúc bộ nhớ và bus: hệ thống bus gồm có ma trận bus giao tiếp gồm nhiều lớp AHB (Advanced High-Performance Bus) 32 bit nối với nhau và các bus APB (Advanced Peripheral Bus).



Hình 3.3. Kiến trúc ma trận bus trong STM32F407

Đầu tiên ta thấy 3 đường bus S0, S1, S2 từ lõi Cortex-M4 ứng với:

- I-bus : đường nối truy cập (fetch) các dòng lệnh (Instruction) từ bộ nhớ Flash, SRAM hoặc bộ nhớ ngoài thông qua FSMC (flexible static memory controller)
- D-bus: kết nối bus dữ liệu (Data) của lõi và 64Kbyte dữ liệu của CCM RAM (Coupled Core Memory) với các bộ nhớ chứa lệnh hoặc dữ liệu (Flash, SRAM, FSMC)
- S-bus: kết nối bus hệ thống (System) của lõi đến các bộ nhớ chứa lệnh hoặc dữ liệu (SRAM, AHB của ngoại vi)

Tiếp theo là các bus S3, S4, S5 dùng cho hai bộ DMA1 và DMA2. Mỗi DMA gồm có 1 đường bộ nhớ nối trực tiếp vào ma trận bus và 1 đường nối trực tiếp ra ngoại vi không thông qua ma trận bus. Nhờ có đặc điểm này mà các DMA có thể dùng để trao đổi trực tiếp giá trị các thanh ghi của ngoại vi vào SRAM để thực thi thay vì thông qua lõi chuyển sang SRAM. Do đó DMA dùng cho các ứng dụng cần tốc độ trao đổi nhanh để tiết kiệm thời gian tính toán.

Cuối cùng là 2 bus S6, S7 là DMA dùng cho Ethernet và USB. Ma trận bus sẽ làm nhiệm vụ quản lý các truy cập từ các master S0, S1...S7. Giải thuật để quản lý đường truyền là round-robin tức là xử lý quay vòng trên nguyên tắc không ưu tiên, các truy cập phải chia sẻ đường truyền trong một thời gian cố định nếu hoàn thành thì bị xóa ra còn không sẽ tiếp tục thực thi ở vòng lặp sau. Các bus APB là các bus giao tiếp ngoại vi nên cần có các cầu (AHB/APB Bridge) đồng bộ APB và AHB.

Bộ nhớ bao gồm Flash, SRAM được chia ra thành các phần cho ngoại vi, chương trình. Các bộ nhớ này dùng 4 loại bus AHB1, AHB2, APB1, APB2. Các bus này có thể thiết lập ở

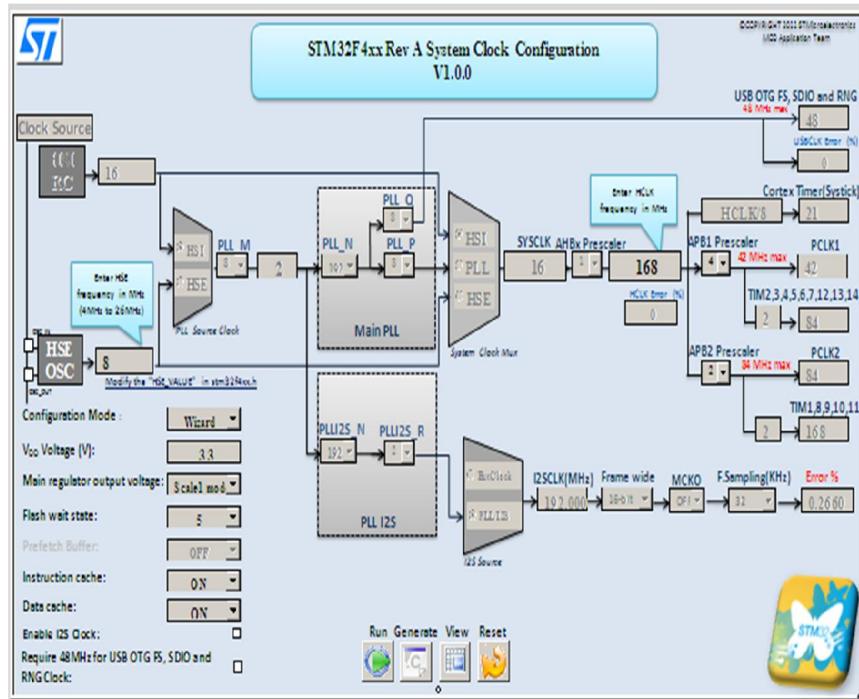
các tốc độ clock khác nhau nên dẫn đến tốc độ các ngoại vi khác nhau. Phân chia các vùng nhớ quan trọng như sau:

- SRAM gồm 3 khối 112 KByte, 64 Kbyte và 12 KByte bắt đầu ở 0x20000000. Việc truy cập SRAM sẽ không tốn thời gian trễ hay trạng thái chờ (latency or wait state).
- Bộ nhớ Flash cho lập trình gồm 12 sector từ 0x08000000 đến 0x080FFFFF (1 MByte). Việc truy cập đến Flash sẽ tốn một khoảng thời gian chờ (thời gian trễ này có thể lập trình được) tùy thuộc vào tốc độ truy cập và điện áp cung cấp cho vi điều khiển.
- Bộ nhớ chương trình Bootloader bắt đầu tại 0x00000000 có thể dùng với USART1, USART3, CAN2 hoặc USB ở chế độ là DFU (Device Firmware Update).

Nguồn xung clock: có 3 nguồn cung cấp clock cho vi điều khiển

- HIS clock: dao động nội 16Mhz có thể dùng trực tiếp hoặc qua bộ PLL.
- HSE clock: dao động ngoài 4Mhz đến 26Mhz.
- PLL clock: được cung cấp clock từ HIS hoặc HSE gồm 2 phần
 - Main PLL: cung cấp clock cho lõi và các ngoại vi.
 - PLL I2S: cung cấp clock chính xác cao cho bộ I2S (dùng cho xử lý âm thanh).

Để đơn giản lập trình nhà sản xuất ST cung cấp một phần mềm đi kèm để tạo file thiết lập xung clock cho STM32F407. Người dùng chỉ cần nhập các thông số cần thiết sau đó sử dụng file tạo ra để đưa vào chương trình của mình. Giao diện chương trình:



Hình 3.4 Chương trình tạo file thiết lập clock cho STM32F407

GPIO (General Purpose In/Out)

Dòng STM32F407 có 100 chân trong đó có khoảng 80 chân có thể dùng chức năng GPIO thông thường. Các chân này chia làm 5 port PA, PB, PC, PD, PE; mỗi port có 16 chân. Các chân xuất có thể thiết lập chế độ push-pull, open-drain hoặc có điện trở pull-up/pull-down. Ngoài ra có thể thiết lập ở chế độ floating, pull-up/pull-down hoặc analog. Tất cả các chân này đều có thể cài đặt làm ngắt ngoài. Đặc biệt phần lớn các chân có tính chất là “5V tolerant” do đó có thể dùng giao tiếp trực tiếp với các IC ngoài có mức logic 0-5V. Ví dụ chương trình cài đặt chân PA0 chế độ input để đọc nút nhấn và chân PD12 ở chế độ output để xuất LED, khi có nút nhấn thì chuyển trạng thái LED:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

```

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIO_InitStructure);

while (!GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0))
{
    GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
}

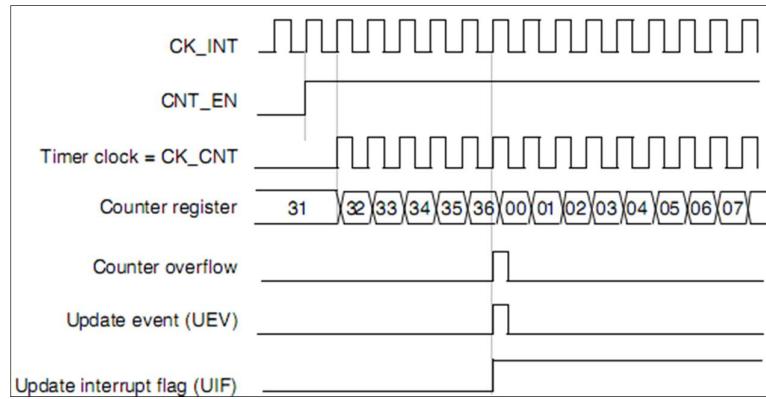
```

Timer

Nếu không tính bộ SysTickTimer của lõi thì ngoại vi của STM32F407 có 14 timer. Các timer này đều có thể dùng để định thời, tạo delay hay PWM...Nhưng trong STM32F407 chia thành 3 nhóm với một số khả năng riêng chuyên dùng cho các ứng dụng cụ thể.

- Timer điều khiển cao cấp: (Advanced-control timers) TIM1 & TIM8
 - Timer 16 bit với bộ đếm counter tự lấy lại giá trị đặt (auto-reload) và bộ chia trước 16 bit (prescaler) lập trình được.
 - Có thể dùng cho capture, PWM có dead-time...
- Timer dùng cho mục đích chung chung: (General-purpose timers)
 - TIM2 đến TIM5: 16 hoặc 32 bit.
 - TIM9 đến TIM14: 16 bit.
 - Các timer này đều có khả năng “auto-reload” và bộ chia trước 16 bit lập trình được.
 - Dùng timer này cho capture, PWM...
- Timer cơ bản: (Basic Timer) TIM6 & TIM7
 - 16 bit với bộ đếm lên, bộ chia 16 bit.
 - Dùng cho tạo khoảng thời gian chuẩn để xử lý các tác vụ.
 - Dùng clock cho bộ DAC.

Ở trong phần này dùng TIM6 trong nhóm timer cơ bản (các timer khác nếu chỉ dùng với chức năng định thời thì cũng tương tự). Như đã tóm tắt ở trên nhóm timer cơ bản là timer với bộ đếm lên 16 bit, bộ chia trước 16 bit.



Hình 3.5 Giản đồ hoạt động Timer cơ bản

Ví dụ chương trình thiết lập TIM6 ở chế độ ngắt 1s và SysTick Timer ngắt 1ms:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
prescaleValue = (uint16_t) 8400;
TIM_TimeBaseInitStruct.TIM_Prescaler = prescaleValue - 1;
TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInitStruct.TIM_Period = 10000;
TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE);
TIM_TimeBaseInit(TIM6, &TIM_TimeBaseInitStruct);
TIM_Cmd(TIM6, ENABLE);
SysTick_Config(21000);
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
```

Đoạn chương trình ngắt đảo trạng thái bit trong ngắt TIM6 và SysTick Timer

```
void SysTick_Handler(void)
{
    static uint32_t SysTick_counter;
    SysTick_counter++;
    if(SysTick_counter == 1000)
    {
        SysTick_counter = 0;
        GPIO_ToggleBits(GPIOB, GPIO_Pin_13);
    }
}
void TIM6_DAC_IRQHandler(void)
{
    if(TIM_GetFlagStatus(TIM6, TIM_FLAG_Update) == SET)
```

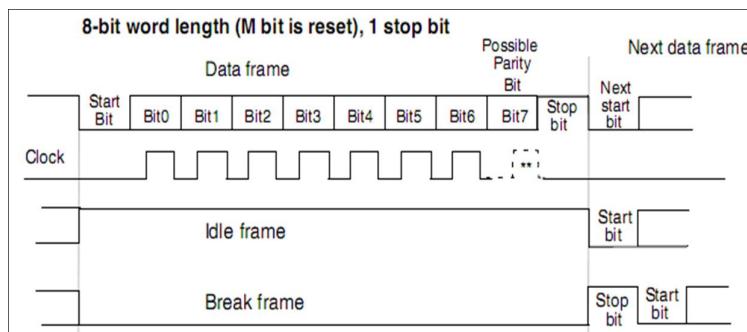
```

    {
        TIM_ClearFlag(TIM6, TIM_FLAG_Update);
        GPIO_ToggleBits(GPIOB, GPIO_Pin_12);
    }
}

```

USART

Giao thức truyền thông nối tiếp UART đã quá phổ biến trong các thiết bị hiện nay. Trong luận văn này khói UART sẽ được dùng để giao tiếp truyền dữ liệu từ IMU lên máy tính. STM32F407 có 4 bộ USART (1,2,3,6) và 2 bộ UART (4,5) truyền thông song công, bắt đồng bộ hoặc đồng bộ. Ở đây em quan tâm đến truyền thông bắt đồng bộ để trao đổi dữ liệu với IMU. Dữ liệu được cài đặt 8 bit, 1 stop bit và không dùng parity. Đặc biệt các bộ USART này hỗ trợ DMA nên rất thuận lợi cho việc nhận và gửi lượng lớn dữ liệu mà không làm tốn nhiều thời gian của lõi vi xử lí.



Hình 3.6 Frame truyền của USART

Các bước cài đặt để gửi và nhận qua USART trong phần mềm:

- Bật clock cho bus APB của ngoại vi USART.
- Bật clock cho các Port của các chân liên quan TX, RX.
- Cài đặt chức năng USART cho các chân liên quan TX, RX.
- Cài đặt các thông số cho bộ USART. Ví dụ: baud rate 115200, dữ liệu 8 bit, 1 stop bit, không dùng Parity, cho phép bộ thu và bộ phát, cho phép cờ ngắt báo nhận xong và gửi xong, cho phép sử dụng DMA ở bộ thu và bộ phát.
- Cài đặt DMA: (có chi tiết trong phần *DMA*)
 - Trường hợp gửi:
 - Nguồn gửi là buffer chứa dữ liệu muốn gửi.
 - Đích đến là thanh ghi dữ liệu của USART USART_DR.

- Trường hợp nhận: thì ngược lại.
- Cài đặt ngắt cho bộ thu và bộ phát (phải xóa các cờ trong ngắt).

Ví dụ chương trình echo dùng USART1:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE) ;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
//enable alternate function at TX,RX pins
GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);
// Init GPIO
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
//TX PIN
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
GPIO_Init(GPIOB, &GPIO_InitStructure);
//RX PIN
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
GPIO_Init(GPIOB, &GPIO_InitStructure);
//Init Uart properties
USART_InitStructureUSART_BaudRate = 115200;
USART_InitStructureUSART_WordLength = USART_WordLength_8b;
USART_InitStructureUSART_StopBits = USART_StopBits_1;
USART_InitStructureUSART_Parity = USART_Parity_No;
USART_InitStructureUSART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_InitStructureUSART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_Init(USART1, &USART_InitStructure);
USART_Cmd(USART1, ENABLE);
while(1)
{
    while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == RESET);
    getChar = USART_ReceiveData(USART1);
    USART_SendData(USART1, getChar);
}
```

SPI

Giao thức nối tiếp đồng bộ SPI dùng nhiều trong giao tiếp với các loại cảm biến. Trong giao thức này một chip đóng vai trò Master và nhiều chip đóng vai trò Slave. Giao thức SPI yêu cầu 4 dây:

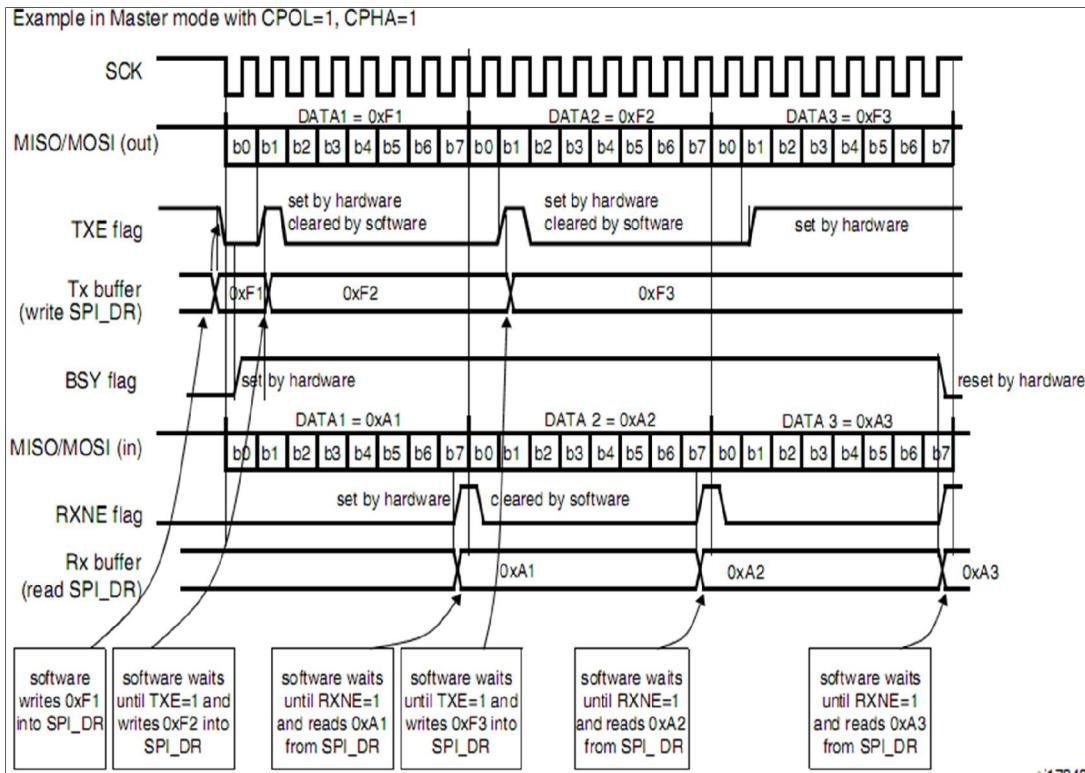
- CS: (Chip Select) chọn Slave giao tiếp với Master.
- TX: xuất dữ liệu (còn gọi là MOSI: Master Out Slave In).
- RX: nhận dữ liệu (còn gọi là MISO: Master In Slave Out).
- CLK: chân clock đồng bộ dữ liệu được điều khiển bởi Master.

Quá trình truyền nhận trên đường truyền SPI tóm tắt như sau:

- Quá trình truyền:
 - Master sẽ kéo chân CS xuống mức thấp để chọn Slave.
 - Sau đó dữ liệu 8 bit được dịch từ song song sang nối tiếp trên đường TX. Tùy vào loại cảm biến thì ban đầu cần thiết lập bit Read/Write và địa chỉ thanh ghi trong Slave.
- Quá trình nhận:
 - Master phải kéo chân CS xuống mức thấp để chọn Slave.
 - Sau đó bắt đầu truyền dữ liệu 8 bit báo cho Slave địa chỉ thanh ghi cần đọc về.
 - Các chu kỳ sau Master phải tiếp tục giữ clock trên đường CLK bằng cách ghi giá trị giả không có ý nghĩa ra TX (dummy write). Và bộ thu sẽ đọc giá trị gửi về trên RX từ Slave.

Các bước cài đặt để gửi và nhận bằng SPI trong phần mềm:

- Bật clock cho khối ngoại vi SPI, cho các port ứng với các chân có liên quan đến SPI.
- Cài đặt chức năng chân CS là dạng digital, output còn các chân CLK, TX, RX là chức năng SPI.
- Thiết lập các thông số cho SPI: truyền 2 hướng, dữ liệu 8 bit hay 16 bit, bit LSB đi trước và đóng vai trò là Master.
- Cho phép SPI và DMA trên SPI.
- Cài đặt DMA cho SPI: địa chỉ nguồn, địa chỉ đích, độ dài khối dữ liệu...
- Cài đặt các ngắt truyền và nhận hay DMA.



Hình 3.7 Giản đồ thứ tự truyền SPI

Ví dụ chương trình cài đặt SPI:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1); //MOSI
GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1); //MISO
GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1); //SCK
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;
// MOSI
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
GPIO_Init(GPIOA, &GPIO_InitStructure);
// MISO
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
GPIO_Init(GPIOA, &GPIO_InitStructure);
// SCK
```

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
GPIO_Init(GPIOA, &GPIO_InitStructure);
// SPI Configuration
SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4;
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_Init(SPI1, &SPI_InitStructure);
SPI_Cmd(SPI1, ENABLE);
// Chip Select
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 ;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOE, &GPIO_InitStructure);
```

Chương trình trao đổi dữ liệu trên bus SPI:

```
uint8_t mySpiExchangeByte(SPI_TypeDef* SPIx, uint8_t byte)
{
    while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_TXE) == RESET);
    SPI_I2S_SendData(SPIx, byte);
    while (SPI_I2S_GetFlagStatus(SPIx, SPI_I2S_FLAG_RXNE) == RESET);
    return SPI_I2S_ReceiveData(SPIx);
}
```

DMA

DMA (Direct Memory Access) cho phép truyền dữ liệu tốc độ cao giữa ngoại vi và các vùng nhớ, giữa các vùng nhớ với các vùng nhớ. Dữ liệu được chuyển đi trên bus riêng của DMA nên không cần có sự can thiệp của lõi vi xử lí, trong lúc đó lõi vẫn thực hiện các tác vụ khác nên tiết kiệm nhiều thời gian cho tính toán. Các đặc điểm của DMA trong STM32F407:

- Có 2 bộ DMA (DMA1, DMA2) mỗi bộ điều khiển 8 stream, trên mỗi stream có bộ ghép kênh xử lí 8 yêu cầu (request).

- Mỗi stream có thể cài đặt dùng cho chuyển dữ liệu 3 nhóm từ:
 - Ngoại vi tới vùng nhớ (peripheral-to-memory).
 - Vùng nhớ tới ngoại vi.
 - Vùng nhớ tới vùng nhớ.
- Có 4 bộ FIFO dùng cho:
 - FIFO mode: phần mềm chọn các mức dùng FIFO $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ hoặc đầy FIFO.
 - Direct mode: chuyển dữ liệu trực tiếp không qua FIFO.
- Các stream có thể cài đặt 4 cấp độ ưu tiên: rất cao, cao, trung bình, thấp (very high, high, medium, low).
- Số lượng khối dữ liệu truyền trên DMA có thể lên tới 65535.
- Có 5 cờ đi kèm quá trình truyền DMA để kiểm soát là:
 - Truyền được một nửa (Half-transfer reached).
 - Truyền xong hoàn toàn (Transfer complete).
 - Truyền có lỗi (Transfer error).
 - FIFO có lỗi (FIFO error).
 - Truyền ở chế độ trực tiếp có lỗi (Direct mode error).

Các bước cài đặt sử dụng DMA:

- Cài đặt trước ngoại vi sử dụng (như USART và SPI trên) trong đó cho phép ngắt DMA.
- Thiết lập DMA:
 - Chọn kênh DMA.
 - Chọn địa chỉ gửi và nhận cho nguồn và đích. Địa chỉ có thể là thanh ghi trong phần cứng hoặc địa chỉ biến lưu trữ tự đặt.
 - Chọn dùng FIFO hay không và 4 mức tương ứng.
 - Một số tùy chọn khác nếu cần.
- Thiết lập các thông số như trên DMA trên kênh này.
- Cho phép ngắt hoặc không ngắt 5 cờ báo DMA ở trên.
- Mỗi khi muốn ta sẽ cho phép DMA.

Ví dụ chương trình cài đặt sử dụng DMA trong truyền dữ liệu trên USART1_TX:

```
// When using rx_dma do not enable receive interrupt
USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
USART_Init(USART1, &USART_InitStructure);
// Enable usart1
USART_Cmd(USART1, ENABLE);
// Enable TX-RX DMA request
USART_DMACmd(USART1, USART_DMAReq_Tx|USART_DMAReq_Rx, ENABLE);

// Enable DMA2 Stream7-Channel4
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);
DMA_Usart1TxStruct.DMA_Channel = DMA_Channel_4;
DMA_Usart1TxStruct.DMA_DIR = DMA_DIR_MemoryToPeripheral;
DMA_Usart1TxStruct.DMA_Memory0BaseAddr = (uint32_t)usart1TxBuffer;
DMA_Usart1TxStruct.DMA_PeripheralBaseAddr = (uint32_t)USART1 + 0x04;
DMA_Usart1TxStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_Usart1TxStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
DMA_Usart1TxStruct.DMA_Priority = DMA_Priority_High;
DMA_Usart1TxStruct.DMA_FIFOMode = DMA_FIFOMode_Enable;
DMA_Usart1TxStruct.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
DMA_Usart1TxStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_Usart1TxStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_Usart1TxStruct.DMA_Mode = DMA_Mode_Normal;
DMA_Usart1TxStruct.DMA_BufferSize = (uint32_t)10 ;
DMA_Usart1TxStruct.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_Usart1TxStruct.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
//Enable DMA Stream Transfer Complete interrupt
DMA_ITConfig(DMA_USART1TX_STREAM, DMA_IT_TC, ENABLE);
DMA_Init(DMA_USART1TX_STREAM, &DMA_Usart1TxStruct);

// Enable DMA Interrupt DMA2_STREAM7
NVIC_InitStruct.NVIC_IRQChannel = DMA2_Stream7_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

Chương trình nhận dữ liệu trên ngắt DMA USART1_TX DMA2-Stream7

```

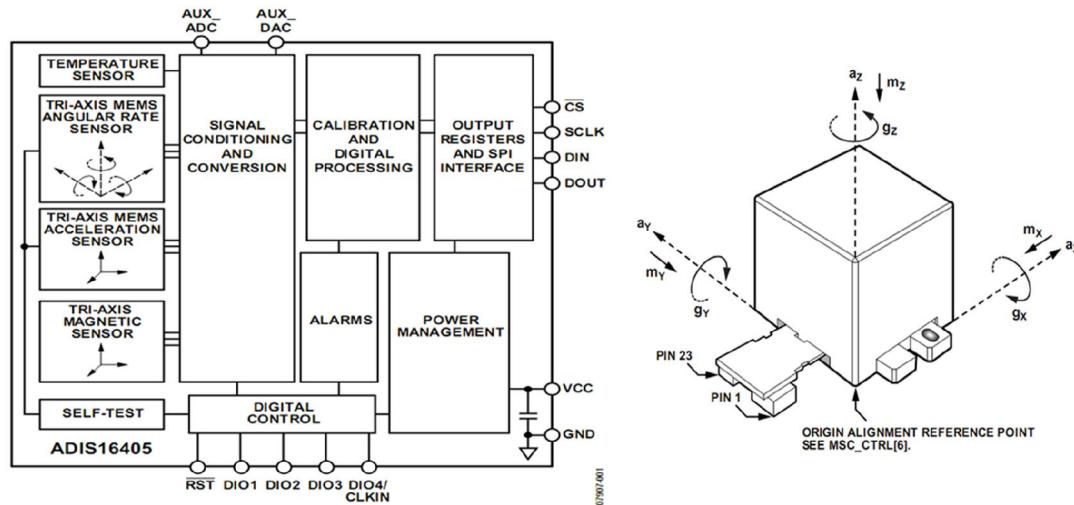
void DMA2_Stream7_IRQHandler(void)
{
    if (DMA_GetFlagStatus(DMA_USART1TX_STREAM, DMA_FLAG_TCIF7) == SET)
    {
        DMA_ClearFlag(DMA_USART1TX_STREAM, DMA_FLAG_TCIF7);
        GPIO_ToggleBits(GPIOB, GPIO_Pin_12);
    }
}

```

3.1.2 Cảm biến ADIS16405 và phương pháp calib

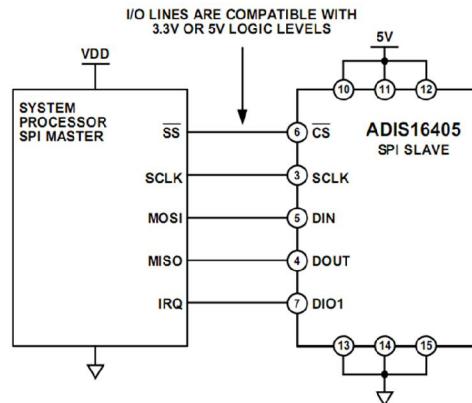
a. Giới thiệu cảm biến ADIS16405

Cảm biến ADIS16405 là hệ thống đo quán tính đầy đủ bao gồm ba loại Gia tốc, Vận tốc góc và Từ trường. Giá trị của các loại cảm biến đã được calib khi sản xuất và đặc biệt đã được bù theo nhiệt độ nên rất thuận lợi cho việc kiểm tra giải thuật (không bị ảnh hưởng bởi sai số do nhiệt độ). Cảm biến ADIS16405 cung cấp giao thức SPI tốc độ cao để giao tiếp nhanh chóng tiện cho việc tích hợp IMU nhỏ gọn.



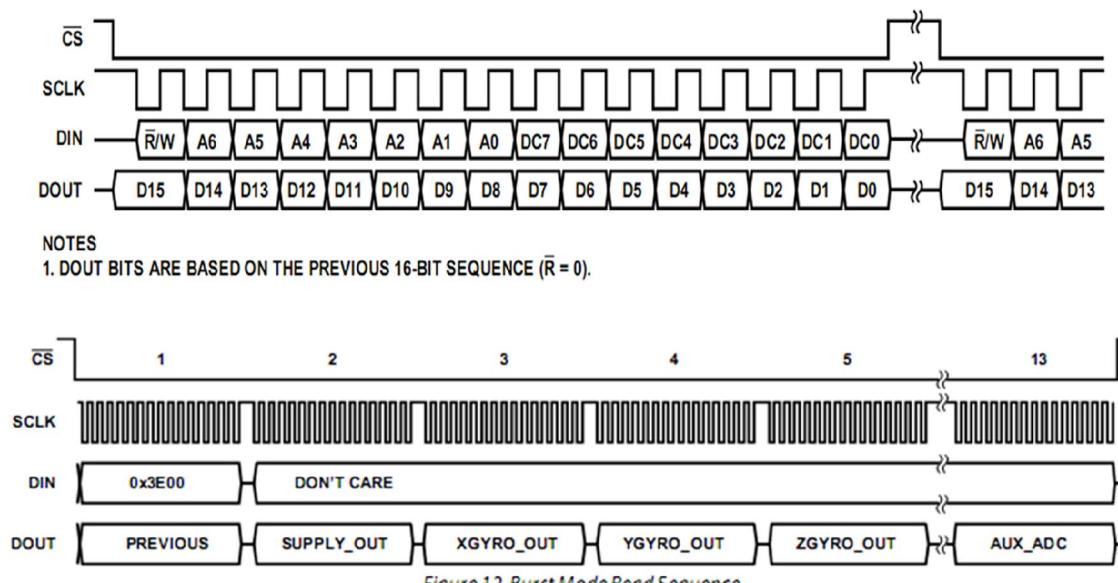
Hình 3.8 Sơ đồ khối ADIS16405 và hệ trục của các cảm biến

b. Giao tiếp giữa VDK STM32F405 và cảm biến ADIS16405



Hình 3.9 Mô hình kết nối phần cứng giữa VDK và ADIS16405

Giao tiếp giữa STM32F405 và ADIS16405 là đường giao tiếp nối tiếp SPI 16-bit. Mỗi giá trị của cảm biến sẽ chứa trong các thanh ghi 16-bit có địa chỉ 7-bit.



Hình 3.10 Giản đồ xung trao đổi dữ liệu trên SPI và chế độ đọc burst-mode

Một frame truyền của SPI từ VDK sẽ gồm 16 bit, trong đó bit thứ 16 sẽ quy định đọc hay ghi (bit 0: đọc, bit 1: ghi), 7 bit tiếp theo chứa địa chỉ thanh ghi muốn giao tiếp và 8 bit cuối là giá trị ghi vào hoặc không quan tâm nếu là đọc. Giá trị đọc về sẽ được nhận trong chu kỳ tiếp theo.

\bar{R}/W	A[6:0]	DC[7:0]
-------------	--------	---------

Ví dụ: muốn đọc giá trị thanh ghi XACCL_OUT ở địa chỉ **0x0A** ta sẽ ghi dữ liệu ra SPI **0x0A00** và đọc về 16-bit dữ liệu tiếp theo bằng cách ghi ra giá trị “dummy byte” 0x0000.

ADIS16405 hỗ trợ chế độ đọc Burst-mode để tiện cho việc đọc liên tiếp nhiều byte. Ban đầu sẽ phải ghi ra đường SPI giá trị **0x3E00** sau đó lần lượt ghi ra “dummy byte” **0x0000** và đọc về các giá trị từ thanh ghi **0x02** (SUPPLY_OUT) đến thanh ghi **0x18** (AUX_ADC). Giá trị 16-bit sau khi đọc về ở dạng số bù 2 14-bit (bit dấu là bit số 14, 13 bit thấp là số) nên phải chuyển qua số nguyên trước khi nhân với hệ số tỉ lệ ứng với riêng từng thanh ghi.

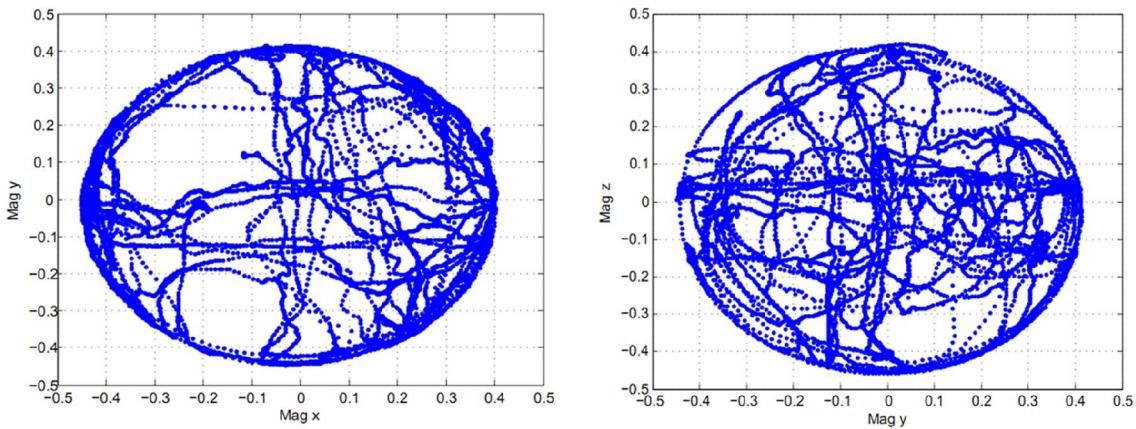
Table 9. Output Data Register Formats

Register	Bits	Format	Scale
SUPPLY_OUT	14	Binary, 5 V = 0x0814	2.42 mV
XGYRO_OUT ¹	14	Twos complement	0.05°/sec
YGYRO_OUT ¹	14	Twos complement	0.05°/sec
ZGYRO_OUT ¹	14	Twos complement	0.05°/sec
XACCL_OUT	14	Twos complement	10 mg
YACCL_OUT	14	Twos complement	10 mg
ZACCL_OUT	14	Twos complement	10 mg
XMAGN_OUT	14	Twos complement	0.5 mgauss
YMAGN_OUT	14	Twos complement	0.5 mgauss
ZMAGN_OUT	14	Twos complement	0.5 mgauss
TEMP_OUT ²	12	Twos complement	0.14°C
AUX_ADC	12	Binary, 1 V = 0x04D9	0.81 mV

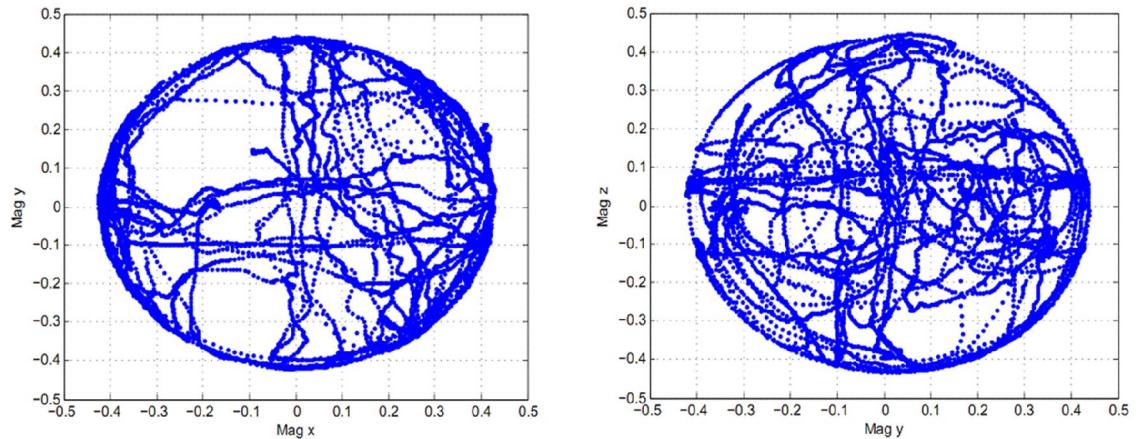
Bảng 3.2 Bảng hệ số tỉ lệ cho các giá trị cảm biến

c. Calib cảm biến

Các giá trị của các loại cảm biến Accelerometer và Gyroscope đã được calib trong khi sản xuất nên không cần phải calib lại. Tuy nhiên do từ trường thay đổi theo các vùng địa lý khác nhau, bị nhiễu bởi các thiết bị điện tử, hay kim loại để gần... nên phải calib lại cảm biến Từ trường. Các giá trị offset sẽ được thêm vào sau khi thu giá trị từ cảm biến để sao cho từ trường trên ba trục x,y,z tạo thành hình cầu. Sau đây là giản đồ từ trường trên các trục trước và sau khi calib.



Hình 3.11 Giản đồ từ trường trên các trục trước khi calib



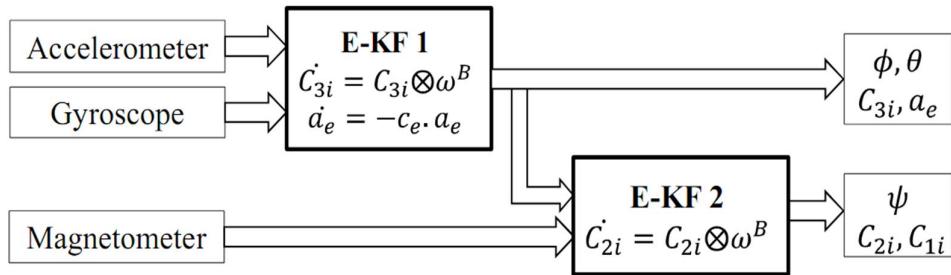
Hình 3.12 Giản đồ từ trường trên các trục sau khi calib

Như trên hình vẽ lần lượt các offset của từ trường trên ba trục là:

$$\begin{cases} mx_{offset} = 0.025 \\ my_{offset} = 0.025 \\ mz_{offset} = 0.01 \end{cases}$$

3.2 Xây dựng mô hình MATLAB Simulink giải thuật Kalman mở rộng cài tiến dùng DCM

Tương tự như mô hình DCM phần hai, mô hình này thêm ba thông số ước lượng ba giá trị gia tốc ngoài a_{ex} , a_{ey} , a_{ez} để tăng khả năng thích nghi cho bộ ước lượng IMU khi có gia tốc ngoài.



Hình 3.13 Mô hình EKF-DCM cải tiến

Mô hình KFI:

Phương trình động học dựa vào DCM: (C_b^n : ma trận xoay chuyển tọa độ từ hệ B-IMU sang N-Earth)

$$C_b^n = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}$$

$$\dot{C}_b^n = C_b^n \cdot \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

Rút riêng hàng thứ 3 của ma trận DCM ta được phương trình:

$$\begin{pmatrix} \dot{C}_{31} \\ \dot{C}_{32} \\ \dot{C}_{33} \end{pmatrix} = \begin{pmatrix} 0 & -C_{33} & C_{32} \\ C_{33} & 0 & -C_{31} \\ -C_{32} & C_{31} & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

Để cải tiến giải thuật ở đây ta thêm khâu ước lượng gia tốc ngoài với phương trình động học như sau:

$$\begin{pmatrix} \dot{a}_{ex}^B \\ \dot{a}_{ey}^B \\ \dot{a}_{ez}^B \end{pmatrix} = -c_e \cdot \begin{pmatrix} a_{ex}^B \\ a_{ey}^B \\ a_{ez}^B \end{pmatrix} \text{ Với } c_e \in [0,1]$$

Phương trình đo lường cho Accelerometer:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = {}_b^n C^T \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} a_{ex}^B \\ a_{ey}^B \\ a_{ez}^B \end{pmatrix} = \begin{pmatrix} C_{31} \\ C_{32} \\ C_{33} \end{pmatrix} + \begin{pmatrix} a_{ex}^B \\ a_{ey}^B \\ a_{ez}^B \end{pmatrix}$$

Mô hình KF1 sẽ dùng các biến trạng thái ở hàng số ba của ma trận DCM. Từ đó ta xây dựng được mô hình toán cho khối lọc Kalman 1 như sau:

- $x = [C_{31}, C_{32}, C_{33}, \omega_x^B, \omega_y^B, \omega_z^B, a_{ex}^B, a_{ey}^B, a_{ez}^B]^T$
- $y = [a_x^B, a_y^B, a_z^B, \omega_x^B, \omega_y^B, \omega_z^B]^T$

- $\dot{x} = \phi \cdot x + w \rightarrow \phi = \begin{bmatrix} 0_{3x3} & C_{3i} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & (1 - c_e T)I_{3x3} \end{bmatrix} \rightarrow \phi_k = I + \phi T$

Với $C_{3i} = \begin{pmatrix} 0 & -C_{33} & C_{32} \\ C_{33} & 0 & -C_{31} \\ -C_{32} & C_{31} & 0 \end{pmatrix}$

- $y = H \cdot x + v \rightarrow H = \begin{bmatrix} I_{3x3} & 0_{3x3} & I_{3x3} \\ 0_{3x3} & I_{3x3} & 0_{3x3} \end{bmatrix}$

Giả sử nhiễu từ Gyrometer 3 trục không tương quan nhau ta sẽ có biểu thức covariance và dạng rời rạc của nhiễu quá trình:

$$Q = E(w \cdot w^T) = \begin{bmatrix} 0_{3x3} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & q_\omega \cdot I_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & q_{ae} I_{3x3} \end{bmatrix} \rightarrow Q_k = \begin{bmatrix} C_{3i}^2 \cdot T^2 \cdot q_\omega^2 & C_{3i} \cdot T \cdot q_\omega^2 & 0_{3x3} \\ C_{3i} \cdot T \cdot q_\omega^2 & q_\omega^2 \cdot I_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & q_{ae}^2 \cdot I_{3x3} \end{bmatrix}$$

Và biểu thức covariance của nhiễu đo lường:

$$R = E(v \cdot v^T) = \begin{bmatrix} r_a I_{3x3} & 0_{3x3} \\ 0_{3x3} & r_\omega \cdot I_{3x3} \end{bmatrix}$$

Trong đó r_a, r_ω là variance tĩnh của Accelerometer và Gyrometer.

Từ ba giá trị ước lượng C_{31}, C_{32}, C_{33} tính được hai góc ϕ, θ như sau:

$$\phi = -\text{asin}(C_{31})$$

$$\theta = \text{atan2}(C_{32}, C_{33})$$

Mô hình KF2:

Ở mô hình KF2 ta dùng các biến trạng thái là hàng hai của ma trận DCM:

$$\begin{pmatrix} \dot{C}_{21} \\ \dot{C}_{22} \\ \dot{C}_{23} \end{pmatrix} = \begin{pmatrix} 0 & -C_{23} & C_{22} \\ C_{23} & 0 & -C_{21} \\ -C_{22} & C_{21} & 0 \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$$

Đặt:

$$C_{2i} = \begin{pmatrix} 0 & -C_{23} & C_{22} \\ C_{23} & 0 & -C_{21} \\ -C_{22} & C_{21} & 0 \end{pmatrix}$$

Mặt khác C_{21}, C_{22}, C_{23} được tính từ góc ϕ, θ và ba giá trị từ Magnetometer theo các công thức sau:

$$X_h = m_x \cdot \cos\theta + m_y \cdot \sin\theta \cdot \sin\phi + m_z \cdot \sin\theta \cdot \cos\phi$$

$$Y_h = m_y \cos\phi - m_z \sin\phi$$

$$\sin\psi = -\frac{Y_h}{X_h^2 + Y_h^2}, \cos\psi = \frac{X_h}{X_h^2 + Y_h^2}$$

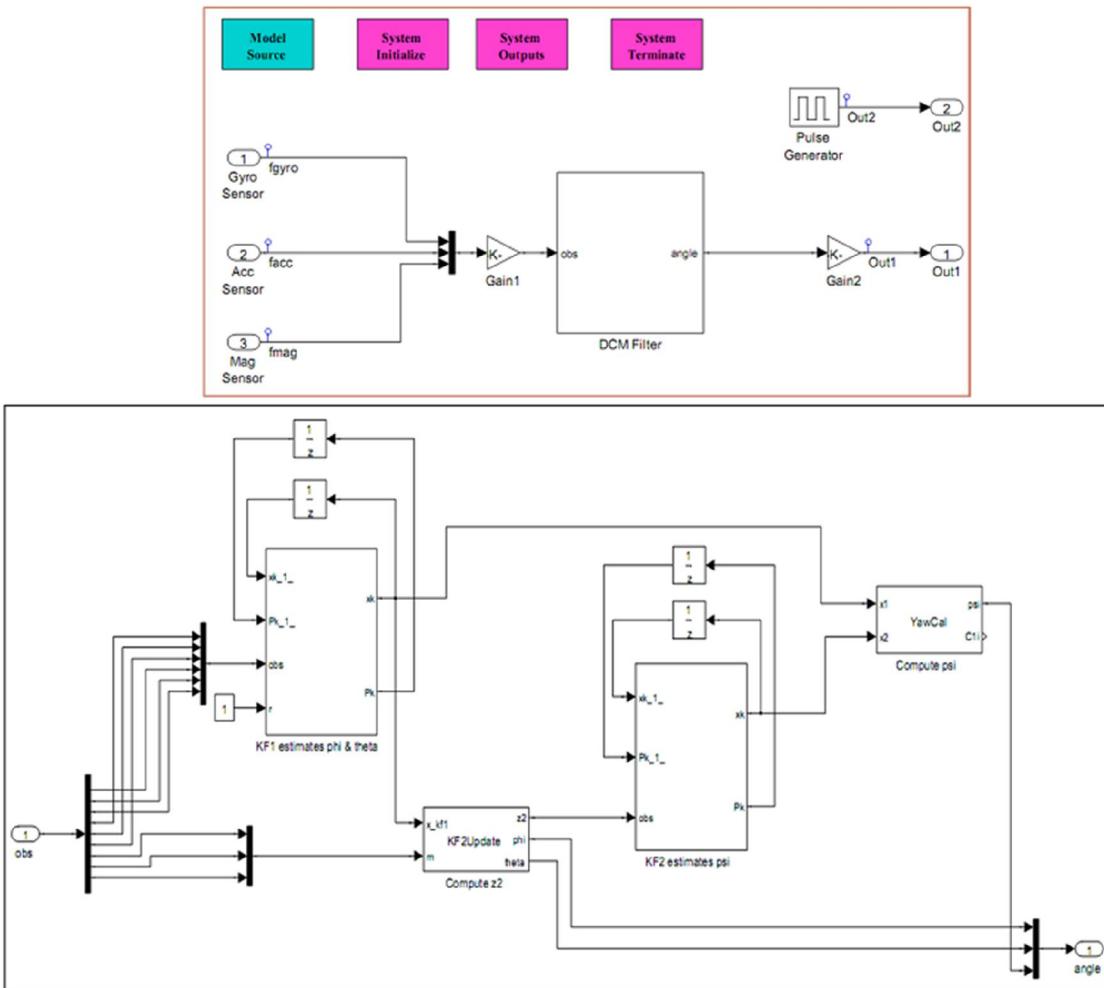
$$\begin{aligned} \rightarrow C_{21} &= \cos\theta \cdot \sin\psi, C_{22} = \cos\phi \cdot \cos\psi + \sin\phi \cdot \sin\theta \cdot \sin\psi, C_{23} \\ &= -\sin\phi \cdot \cos\psi + \cos\phi \cdot \sin\theta \cdot \sin\psi \end{aligned}$$

- $x = [C_{21}, C_{22}, C_{23}, \omega_x^B, \omega_y^B, \omega_z^B]^T$
- $y = [C_{21}, C_{22}, C_{23}, \omega_x^B, \omega_y^B, \omega_z^B]^T$
- $\dot{x} = \phi \cdot x + w \rightarrow \phi = \begin{bmatrix} 0_{3 \times 3} & C_{2i} \\ 0_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \rightarrow \phi_k = I + \phi T = \begin{bmatrix} I_{3 \times 3} & C_{2i} \cdot T \\ 0_{3 \times 3} & I_{3 \times 3} \end{bmatrix}$
- $y = H \cdot x + v \rightarrow H = [I_{6 \times 6}]$
- $Q = \begin{bmatrix} 0_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & q_\omega \cdot I_{3 \times 3} \end{bmatrix} \rightarrow Q_k = \begin{bmatrix} T^3 \cdot C_{2i} \cdot C_{2i}^T \cdot q_\omega & T^2 \cdot C_{2i} \cdot q_\omega \\ T^2 \cdot C_{2i}^T \cdot q_\omega & T \cdot q_\omega \cdot I_{3 \times 3} \end{bmatrix}$
- $R = \begin{bmatrix} \mu \cdot I_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & r \cdot I_{3 \times 3} \end{bmatrix}$

Từ ba giá trị C_{21}, C_{22}, C_{23} tính được các số hạng còn lại của ma trận xoay dựa vào tích hưu hướng: $C_1 = C_2 \otimes C_3$

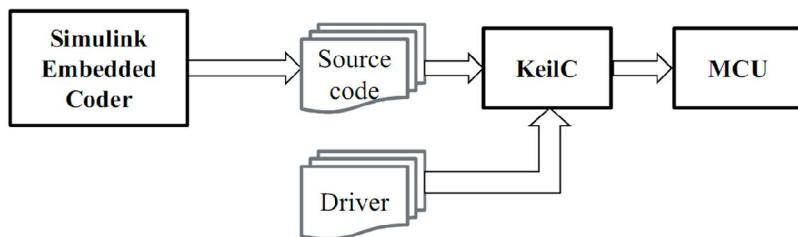
$$\begin{aligned} C_{11} &= C_{22} \cdot C_{33} - C_{23} \cdot C_{32} \\ C_{12} &= C_{23} \cdot C_{31} - C_{21} \cdot C_{33} \\ C_{13} &= C_{21} \cdot C_{32} - C_{22} \cdot C_{31} \\ \rightarrow \psi &= \text{atan2}(C_{21}, C_{11}) \end{aligned}$$

Trong bộ **KF1** các hệ số covariance của nhiễu đo lường được chọn là giá trị variance của các cảm biến khi tính, các hệ số covariance của nhiễu quá trình được chọn bằng phương pháp thử nhiều lần sao cho sai số của IMU là tốt nhất. Trong bộ **KF2** các hệ số covariance của nhiễu đo lường và nhiễu quá trình cũng được thử nhiều lần sao cho sai số nhỏ nhất.



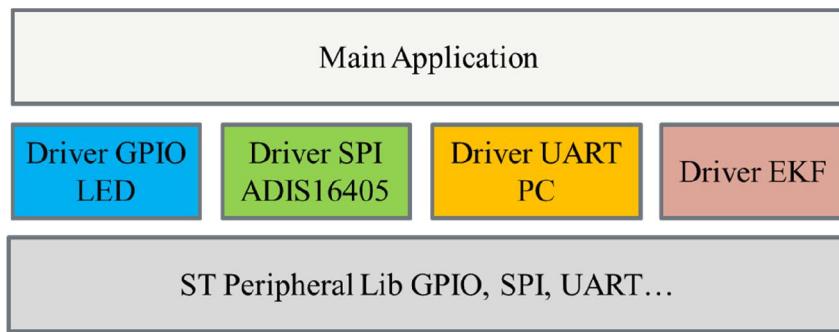
Hình 3.14 Mô hình Simulink của phương pháp EKF-DCM

3.3 Lập trình cho IMU kết hợp MATLAB Simulink Embedded Coder và trình biên dịch KeilC



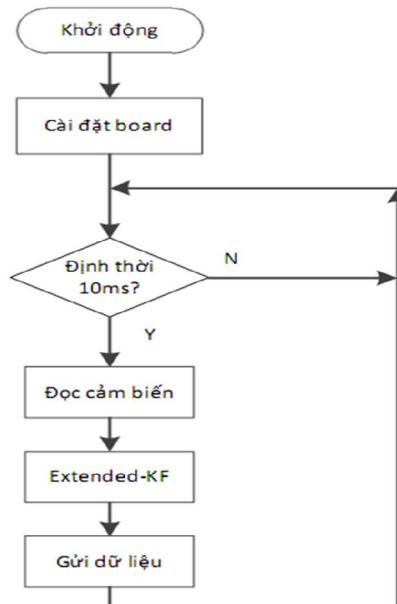
Hình 3.15 Mô hình Compiler kết hợp MATLAB Simulink và Keil C

Một trong những khó khăn lớn của việc hiện thực bộ lọc Kalman trên hệ thống nhúng là mô hình sau khi mô phỏng trên MATLAB phải được chuyển sang ngôn ngữ C. Tuy nhiên MATLAB đã cung cấp một công cụ rất mạnh để hỗ trợ người dùng là **MATLAB Embedded Coder**. Công cụ này cho phép chuyển từ mô hình MATLAB Simulink sang ngôn ngữ C (hoặc C++). Từ các file được biên dịch (dạng .h và .c) kết hợp với các file driver viết cho STM32F40x ta sẽ có bộ source code bằng ngôn ngữ C và có thể biên dịch ra file .hex dùng cho VDK. Kiến trúc của phần mềm trên board IMU như sau:



Hình 3.16 Kiến trúc phần mềm trong IMU

Giải thuật cho IMU được chạy với tần số 100Hz theo lưu đồ giải thuật sau:



Hình 3.17 Lưu đồ giải thuật xử lí trong IMU

Trong đó thời gian chạy các tác vụ trong vòng lặp 10ms là **8.5ms**

3.4 Lập trình đồ họa mô hình 3D cho IMU dùng ngôn ngữ Python

Python là trình biên dịch cấp cao (interpreted language), chương trình khi viết xong sẽ được chạy trực tiếp bởi trình biên dịch mà không cần dịch (compile) trước. Các cú pháp và tính chất của ngôn ngữ Python giống các ngôn ngữ lập trình hướng đối tượng khác như: C#, C++, Java... Tuy nhiên một ưu điểm rất lớn của Python là mã nguồn mở nên được sử dụng ngày càng rộng rãi trên thế giới. Ngôn ngữ Python ít được sử dụng trong các đại học ở Việt Nam, tuy nhiên theo khuynh hướng trên thế giới thì các phần mềm mã nguồn mở sẽ ngày càng phát triển. Vì các lí do trên mà trong luận văn này em sử dụng Python (version 2.7) để làm phần mềm đồ họa mô phỏng 3D cho IMU, đồng thời cũng tìm hiểu cách sử dụng và một số thư viện kèm theo của loại ngôn ngữ “mới mẻ” này.



Hình 3.18 Mô hình của một trình biên dịch (interpreted language)

3.4.1 Tổng quan cấu trúc và cú pháp của một chương trình viết bằng Python

a. Biến, biểu thức, biểu thức điều kiện, và lệnh

Tương tự như các ngôn ngữ khác Python cũng cung cấp các kiểu biến, các toán tử và các biểu thức điều kiện như `int`, `float`, `string`, `object`...`cộng`, `trừ`, `nhân`, `chia`, `mũ`...`if`, `for`, `while`... Sau đây là ví dụ một số chương trình nhỏ viết bằng Python để làm rõ hơn cấu trúc của chương trình phần mềm viết bằng Python

Chương trình 1: *Hello World* trong Python

```
print "Hello World!"
```

Chương trình 2: người dùng nhập hai kích thước rộng, dài tính chu vi và diện tích hình chữ nhật

```
def circle(width, height):
    return 2 * (width + height)
def area(width, height):
    return (width * height)
```

```
w = input("Width : ")
h = input("Height : ")
print 'Cirle: ', circle(w,h)
print 'Area : ', area(w,h)
```

Chương trình 3: viết hàm tính đệ quy dãy số Fibonacci, sau đó tính trong trường hợp n=6

```
def fibonacci(n):
    """ f(n) = f(n-1) + f(n-2) """
    if not isinstance(n, int):
        print "Inavalid value"
        return -1
    if n == 0 or n == 1:
        return 1
    else:
        res = fibonacci(n-1) + fibonacci(n-2)
        return res
print fibonacci(6)
```

Ở chương trình 1 lệnh *print* sẽ in chuỗi “Hello World!” ra màn hình. Ở chương trình 2 hàm *input* sẽ xuất ra tên các kích thước và người dùng phải nhập số vào để tính. Có hai chương trình con tính diện tích và chu vi là hàm *circle* và *area*. Ở chương trình 3 là hàm đệ quy (recursion) để tính chuỗi số Fibonacci. Trong hàm có dùng biểu thức điều kiện *if* để loại trừ trường hợp *n* không phải là số nguyên.

b. Class& Thread

Tương tự như các ngôn ngữ lập trình hướng đối tượng khác Python cũng cung cấp cách xây dựng các *class* của riêng mình. Trong chương trình này ta sẽ xây dựng ba *class* kế thừa *class Thread* có sẵn của Python để chạy song song với nhau đó là:

- *Class* quản lý giao tiếp cổng nối tiếp với board IMU (*CommThread*): đọc dữ liệu từ IMU thông qua UART, kiểm tra dữ liệu hợp lệ hay không và gửi dữ liệu góc qua hai class khác thông qua *Queue* để hiển thị đồ họa.
- *Class* quản lý đồ họa mô hình 3D của IMU (*Graph3DThread*): class này sẽ nhận dữ liệu chuyển qua từ *CommThread* sau đó hiển thị lên các cửa sổ đồ họa 3D.
- *Class* quản lý đồ họa giản đồ theo thời gian của IMU (*Graph2DThread*): class này sẽ nhận dữ liệu chuyển qua từ *CommThread* sau đó hiển thị lên các cửa sổ vẽ đồ thị theo thời gian ba góc Euler.

Ví dụ chương trình tạo class *CommThread*, chương trình sẽ mở port nối tiếp sau đó lặp vòng chờ khi nhận được dữ liệu từ IMU sẽ gửi dữ liệu này qua hai *Thread* còn lại thông qua *queue* để xử lý :

```
class CommThread(threading.Thread):
    def __init__(self, queue3D, queue2D):
        threading.Thread.__init__(self)
        self.queue3D = queue3D
        self.queue2D = queue2D
        self.com = serial.Serial(port='COM1', baudrate=115200)
    def run(self):
        while True:
            line = self.com.readline()
            words = string.split(line)
            if len(words) >= 12:
                self.queue3D.put(words)
                self.queue2D.put(words)
    def close(self):
        self.com.close()
```

3.4.2 Các thư viện

Bản thân ngôn ngữ Python không hỗ trợ đầy đủ các giao tiếp phần cứng hay đồ họa 3D nên các lập trình viên trên thế giới đã tạo ra các mã nguồn mở để hỗ trợ các ứng dụng này. Trong luận văn này em dùng hai bộ thư viện:

- *PySerial*: hỗ trợ giao tiếp cổng nối tiếp
- *VPython*: hỗ trợ hiển thị các mô hình 3D và đồ thị theo thời gian

a. *PySerial*

Thư viện *PySerial* được phát triển bởi Chris Liechti nhằm mục đích tạo ra một *interface* đơn giản để giao tiếp giữa Python và phần cứng thông qua cổng nối tiếp. Các đặc điểm của thư viện này là:

- Hỗ trợ các giao tiếp nối tiếp bao gồm: UART, TCP/IP
- Cùng chung một *interface* cho tất cả hai kiểu giao tiếp trên
- Hỗ trợ các kiểu truy cập khác nhau như: đọc, viết từng byte, chuỗi...
- API đơn giản dễ sử dụng

Một số *interface* hỗ trợ bởi thư viện là:

Interface	Chức năng
<code>__init__()</code>	khởi động các giá trị cho cổng như tên cổng, baudrate, số bit, thời gian timeout...
<code>open()</code>	mở cổng
<code>close()</code>	đóng cổng
<code>read(size)</code>	đọc số byte
<code>write(data)</code>	viết ra dữ liệu
<code>readline()</code>	đọc cho tới khi nhận kí tự '\n'
<code>writeline(string)</code>	viết string ra cổng

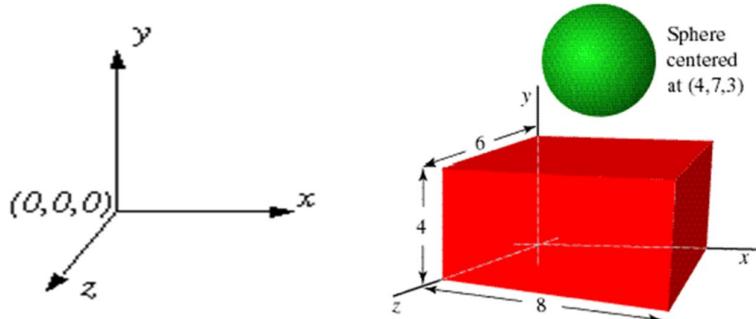
Ví dụ chương trình mở port nối tiếp, đọc dữ liệu và xuất ra màn hình dùng *PySerial*:

```
import serial
com = serial.Serial(port="COM1", baudrate=115200, timeout=1)
data = com.readline()
print data
com.close()
```

b. VPython

Thư viện *VPython* được phát triển bởi nhóm sinh viên trường Carnegie Mellon, Mĩ nhằm mục đích hỗ trợ các mô phỏng 3D, giúp người dùng Python tốn ít thời gian hơn trong việc phát triển chương trình đồ họa. Thư viện cung cấp các hình khối 3D như: hình cầu, hình hộp, mũi tên, hình elip, text... và các biến vị trí vật trong không gian cũng như hệ trực để quay vật thể...

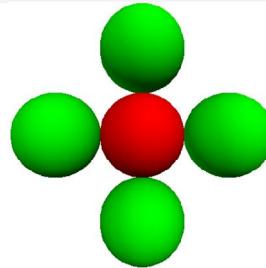
Hệ tọa độ trong Python chia làm ba trục: x hướng sang phải, y hướng lên, z vuông góc hướng ra màn hình. Vị trí cũng như kích cỡ của các vật thể được chương trình tự động scale sao cho phù hợp với màn hình. Trong màn hình hiển thị có thể phóng to, nhỏ bằng cách dùng chuột phải.



Hình 3.19 Hệ tọa độ và vật thể trong VPython

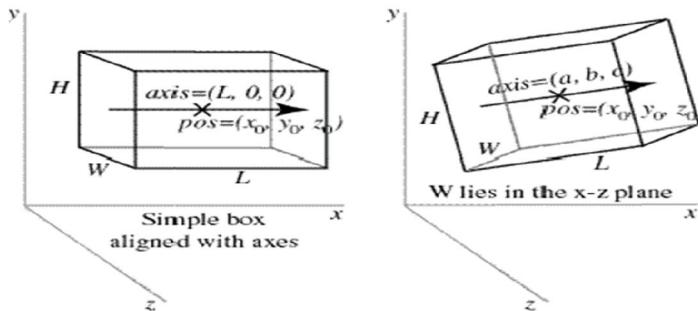
Các vật thể trong VPython được xác định bởi tên, và tất cả đều có thuộc tính vị trí cũng như kích thước. Ví dụ chương trình tạo các hình cầu tiếp xúc nhau:

```
from visual import*
ball = sphere(pos=(0,0,0), radius=2, color=color.red)
ball1 = sphere(pos=(0,4,0), radius=2, color=color.green)
ball2 = sphere(pos=(0,-4,0), radius=2, color=color.green)
ball3 = sphere(pos=(4,0,0), radius=2, color=color.green)
ball4 = sphere(pos=(-4,0,0), radius=2, color=color.green)
```



Hình 3.20 Hình cầu trong VPython

Ở đây ta sẽ tìm hiểu đại diện một vật thể là hình hộp để rõ hơn cách dùng thư viện này.



Hình 3.21 Hình hộp trong VPython

Các thuộc tính quan trọng của hình hộp trong VPython:

- Kích cỡ: *Length*-Chiều dài, *Width*-Chiều rộng, *Height*-Chiều Cao
- Hướng: *axis*-vector song song chiều dài của hình hộp, *up*-vector song song với chiều cao của hình hộp
- Vị trí: *pos*-vector vị trí của tâm hình hộp
- Màu sắc: *color*-màu sắc của hình hộp (*green*, *red*, *blue*, ...)

Để hiển thị mô hình 3D của IMU trong VPython ta sẽ tạo trước đối tượng hình hộp sau đó thay đổi hướng quay của hình hộp giống như hướng quay của IMU bằng cách tính ra hai

vector thuộc tính hướng *up, axis* của hình hộp từ dữ liệu đầu vào là ba góc xoay nhận được từ IMU. Cách tính hai vector *up, axis* như sau:

Đầu tiên ta có ma trận xoay chuyển vector từ hệ trục IMU sang hệ trục chuẩn NED:

$$C_{BE} = \begin{pmatrix} \cos\psi \cdot \cos\theta & \sin\phi \cdot \sin\theta \cdot \cos\psi - \cos\phi \cdot \sin\psi & \sin\phi \cdot \sin\psi + \cos\phi \cdot \cos\psi \cdot \sin\theta \\ \sin\psi \cdot \cos\theta & \sin\phi \cdot \sin\theta \cdot \sin\psi + \cos\phi \cdot \cos\psi & \cos\phi \cdot \sin\theta \cdot \sin\psi - \cos\psi \cdot \sin\phi \\ -\sin\theta & \cos\theta \cdot \sin\phi & \cos\phi \cdot \cos\theta \end{pmatrix}$$

Hay viết gọn lại:

$$C_{BE} = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix}$$

Vector *up* trong hệ trục IMU sẽ là: $up^B = (0,0,1)^T$

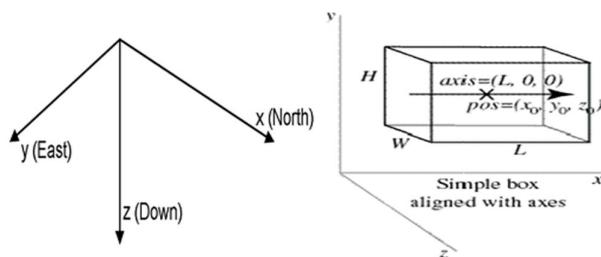
Vector *axis* trong hệ trục IMU sẽ là: $axis^B = (1,0,0)^T$

Từ đó tính được tọa độ hai vector này trong NED là:

$$up^E = C_{BE} \cdot up^B = (C_{13}, C_{23}, C_{33})^T$$

$$axis^E = C_{BE} \cdot axis^B = (C_{11}, C_{21}, C_{31})^T$$

Tuy nhiên hai hệ trục NED và hệ trục trong Vpython (VP) khác nhau do đó ta phải chuyển đổi tọa độ của hai vector *axis, up* như sau:



Hình 3.22 Hệ tọa độ NED và Hệ tọa độ trong VPython

$$\begin{cases} x_{NED} \rightarrow -z_{VP} \\ y_{NED} \rightarrow +x_{VP} \\ z_{NED} \rightarrow -y_{VP} \end{cases}$$

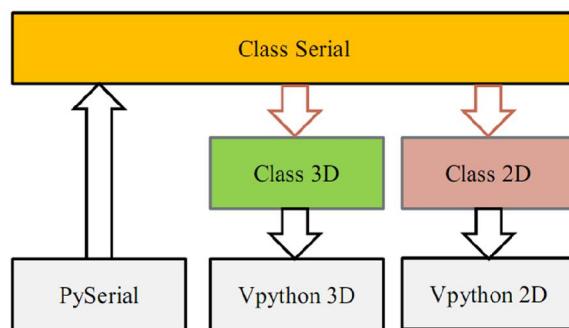
$$\rightarrow \begin{cases} up^{VP} = (C_{23}, -C_{33}, -C_{13})^T \\ axis^{VP} = (C_{21}, -C_{31}, -C_{11})^T \end{cases}$$

3.4.3 Phần mềm đồ họa mô hình 3D cho IMU

Chương trình sẽ chia làm ba *class* quản lý riêng ba tác vụ khác nhau là:

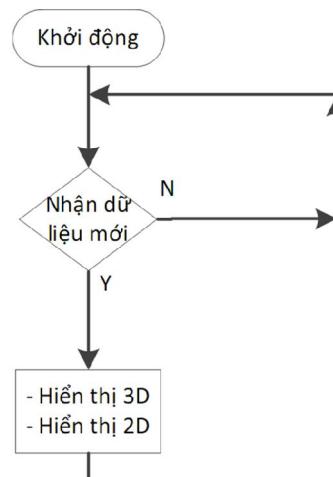
- Giao tiếp với IMU thông qua cổng nối tiếp
- Hiển thị hình ảnh 3D của IMU
- Hiển thị đồ thị theo thời gian của ba góc quay

Kiến trúc phần mềm đồ họa 3D cho IMU như sau:

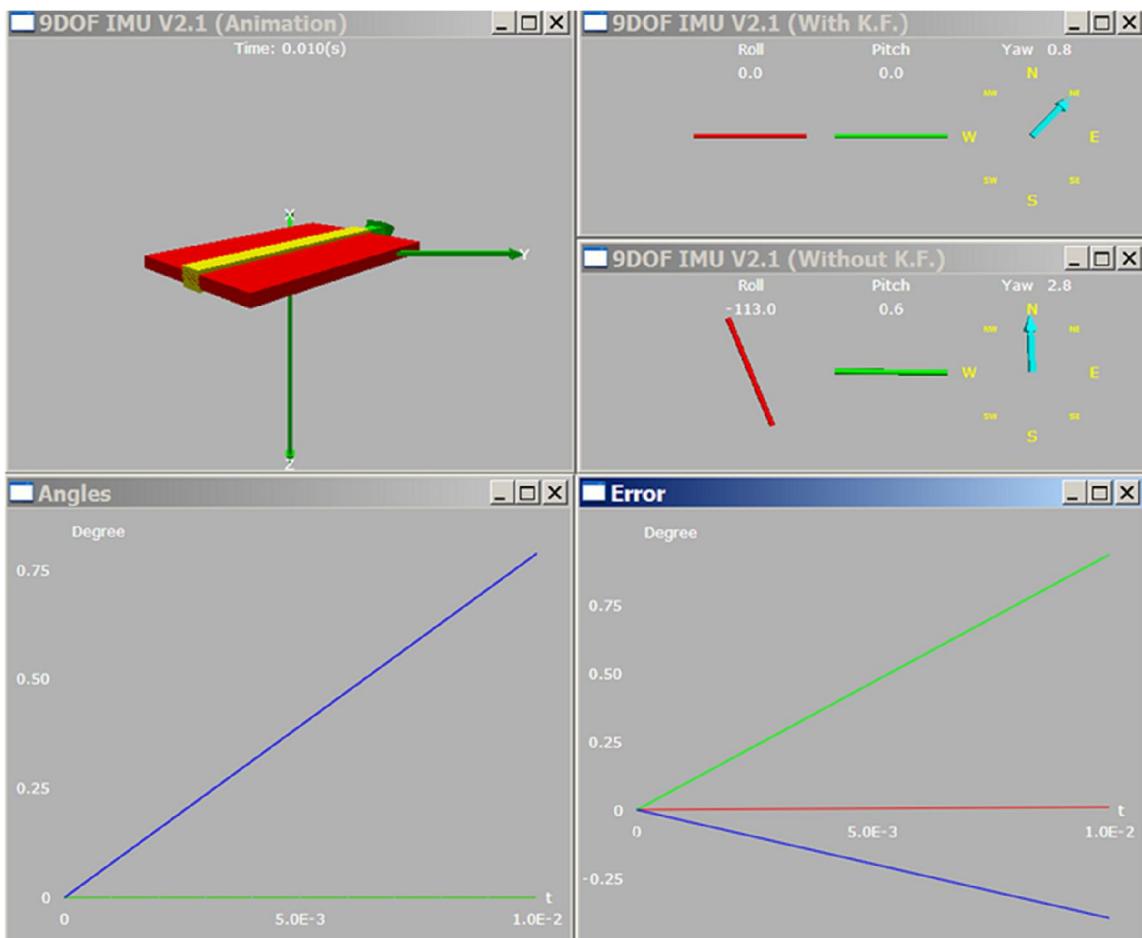


Hình 3.23 Kiến trúc phần mềm hiển thị mô hình 3D của IMU

Lưu đồ giải thuật cho phần mềm:



Hình 3.24 Lưu đồ giải thuật phần mềm đồ họa cho IMU

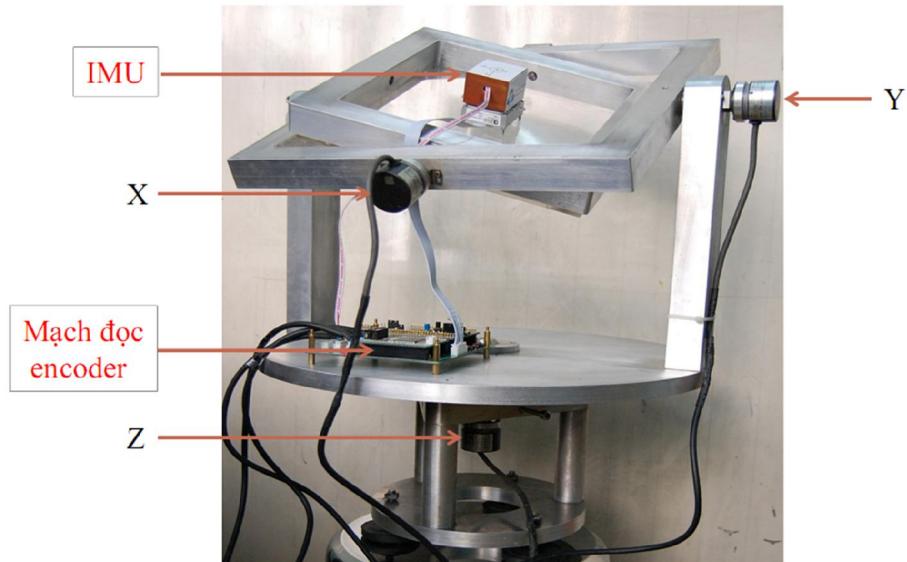


Hình 3.25 Giao diện chương trình đồ họa 3D dùng Python

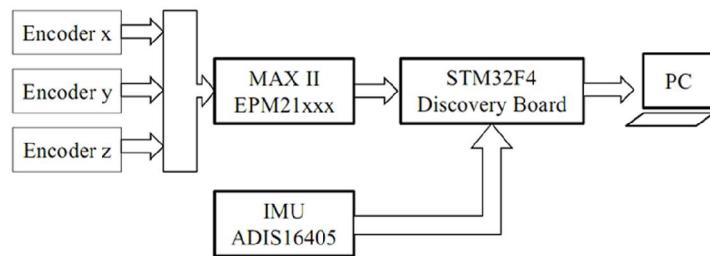
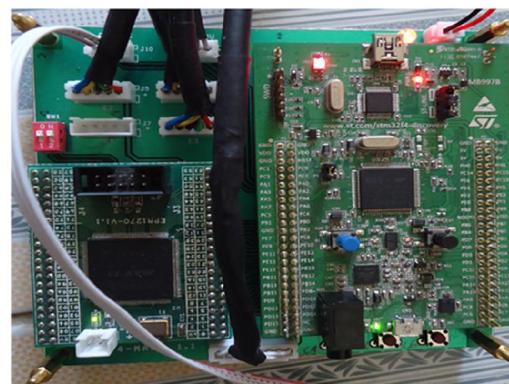
3.5 Hệ thống bàn xoay kiểm tra IMU

Để kiểm tra độ chính xác của IMU, luận văn sử dụng bàn xoay có thể quay tự do ba trục x, y, z sau đó so sánh góc quay tính từ encoder và góc quay ước lượng từ IMU. Mỗi trục quay trên bàn xoay được trang bị một encoder đo xung 2500 xung/vòng. Bàn xoay sử dụng mạch tích hợp đo số xung ba kênh encoder và đọc dữ liệu từ IMU sau đó gửi lên máy tính. Mạch tích hợp gồm hai board phát triển STM32F4 Discovery Board và FPGA MAXII được chế tạo tại phòng thí nghiệm Điều Khiển Tự Động-Bộ môn Tự Động, Đại học Bách Khoa TPHCM. Board MAXII được lập trình chế độ nhân bốn dùng để đọc ba kênh encoder ($2500 \times 4 = 10000$ xung/vòng), board STM32F4 dùng để đọc các góc quay từ IMU thông qua UART và đọc số xung encoder từ MAXII thông

qua chuẩn giao tiếp song song FSMC. Sau đó toàn bộ bảng tin sẽ được gửi lên máy tính.



Hình 3.26 Bàn xoay kiểm tra IMU



Hình 3.27 Mạch đọc xung encoder giao tiếp máy tính tại PTN Bộ môn Tự Động, ĐHBK
TPHCM

Chương 4. Thực hiện đánh giá & Kết quả đạt được

4.1 Phương pháp đánh giá

Trong phần này IMU sử dụng thuật toán đã phát triển sẽ được kiểm tra so sánh với một IMU thương mại có độ chính xác cao là IMU ADIS16480 (độ chính xác tĩnh là 0.1° với góc roll và pitch, 0.3° với góc yaw; độ chính xác động là 0.3° với roll và pitch, 0.5° với góc yaw) và bàn xoay đã giới thiệu phần 3.5.

Một vấn đề khó khăn đối với việc kiểm chứng IMU là việc xây dựng hệ thống thí nghiệm sao cho ba trục của IMU phải trùng với ba trục của bàn xoay hoặc trùng với ba trục của IMU thương mại. Do đó ở đây luận văn sử dụng hai bước thí nghiệm để kiểm chứng thuật toán ước lượng:

Thí nghiệm 1: Đặt IMU thương mại ADIS16480 lên bàn xoay ba trục x,y,z, sau đó ta sẽ so sánh ba giá trị góc từ IMU thương mại ADIS16480 với giá trị ước lượng từ thuật toán đã phát triển dùng chung các giá trị “raw” thu được từ IMU ADIS16480 và giá trị từ ba encoder trên bàn xoay. Trong thí nghiệm này ta lần lượt lấy các mẫu thử xoay quanh các trục x,y,z của bàn xoay và chọn giá trị từ bàn xoay làm chuẩn sau đó sẽ so sánh sai số hai bộ IMU với nó. Đồng thời sau khi xác nhận sai số giữa IMU thương mại và bàn xoay nhỏ ta chỉ cần dùng bàn xoay để kiểm tra IMU.

Thí nghiệm 2: Đặt IMU lên bàn xoay và tính các giá trị sai số giữa góc từ ba encoder trên bàn xoay với giá trị góc ước lượng từ IMU được gắn trực tiếp lên bàn xoay này. Trong thí nghiệm này ta lần lượt đánh giá sai số ở các trường hợp khác nhau:

- Kiểm tra đáp ứng tĩnh:
 - Đứng yên
 - Quay quanh trục x
 - Quay quanh trục y
 - Quay quanh trục z
 - Quay tự do
- Kiểm tra đáp ứng động:

- Gia tốc ngoài trên trục x
- Gia tốc ngoài trên trục y
- Gia tốc ngoài trên trục z
- Gia tốc ngoài tự do trên 3 trục
- Từ trường ngoài

Thí nghiệm 1 sẽ cho phép kiểm chứng tính chính xác của thuật toán mà không bị ảnh hưởng bởi quá trình thiết lập hệ thống thí nghiệm (bị lệch trục hay bị nghiêng...), đồng thời cũng xác nhận tính chính xác của bàn xoay. Thí nghiệm 2 sẽ kiểm chứng đầy đủ đáp ứng thực tế của IMU khi áp dụng thuật toán đã phát triển. Đồng thời thế mạnh của bàn xoay là hoàn toàn không bị ảnh hưởng bởi các điều kiện như gia tốc ngoài, nhiễu từ trường do đó thích hợp kiểm tra tính chính xác của IMU.

Ta sẽ dùng hai giá trị sai số là RMS trong từng trường hợp. Với giá trị RMS của sai số giữa giá trị ước lượng và giá trị chính xác được tính như sau:

$$e_{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^N e_i^2}$$

Với:

$$e_i = x_i^{est} - x_i^{ref}$$

Chú ý: ở đây đối với các sai số do chuyển từ các góc ± 180 độ ta sẽ cho $e = 0$

4.2 Thực hiện đánh giá

Tiến hành calib lại giá trị offset cho cảm biến từ trường trên IMU theo phần 3.1.2. Gắn chặt IMU lên bàn xoay và tiến hành lấy các mẫu thí nghiệm. Các giá trị ước lượng từ bộ lọc, các giá trị encoder sẽ được gửi lên máy tính và lưu dạng file text, sau đó dùng MATLAB vẽ lại các đồ thị góc quay ba trục, sai số ba góc và tính trị RMS. Khung giá trị được lưu như sau:

EST	GYR	ACC	MAG	ENC	REF
-----	-----	-----	-----	-----	-----

Với:

- EST: ba giá trị góc ước lượng độ phân giải 0.1°
- GYR: ba giá trị vận tốc góc đọc từ cảm biến độ phân giải 1 mrad/s
- ACC: ba giá trị gia tốc đọc từ cảm biến độ phân giải 1mg

- MAG: ba giá trị từ trường đọc từ cảm biến độ phân giải 1mGauss
- ENC: ba giá trị encoder với công thức chuyển góc $\phi = (ENC - 32000) * \frac{360}{10000}$
- REF: ba giá trị góc từ IMU thường mại nếu có độ phân giải 0.1°

Hình 4.1 File text lưu dữ liệu ước lượng

Thí nghiệm 1 sẽ lấy ba mẫu giá trị quay ba trục bàn xoay tự do: TN1-XYZ1, TN1-XYZ2, TN1-XYZ3.

Thí nghiệm 2 sẽ lấy tất cả là 10 mẫu gồm:

- Kiểm tra đáp ứng tĩnh:
 - Đứng yên: TN2-S
 - Chỉ quay một trục x, y, z: TN2-X, TN2-Y, TN2-Z
 - Quay tự do ba trục: TN2-XYZ
- Kiểm tra đáp ứng động:
 - Quay mạnh từng trục x, y, z: TN2-AX, TN2-AY, TN2-AZ
 - Quay mạnh cả ba trục: TN2-AXYZ
 - Đứng yên chịu tác động nhiễu từ trường: TN2-M

4.3 Kết quả

Thí nghiệm 1:

Bảng 4.1 Kết quả thí nghiệm 1

STT	Mẫu	Sai số RMS (độ)					
		ϕ		θ		ψ	
		Est	Ref	Est	Ref	Est	Ref
1	TN1-XYZ1	0.1792	0.1860	0.0968	0.1000	0.2629	0.0504
2	TN1-XYZ2	0.2125	0.3437	0.1990	0.1230	0.3887	0.0889
3	TN1-XYZ3	0.1624	0.2697	0.3299	0.2887	0.3246	0.1418

Chú ý ở đây sử dụng các kí hiệu:

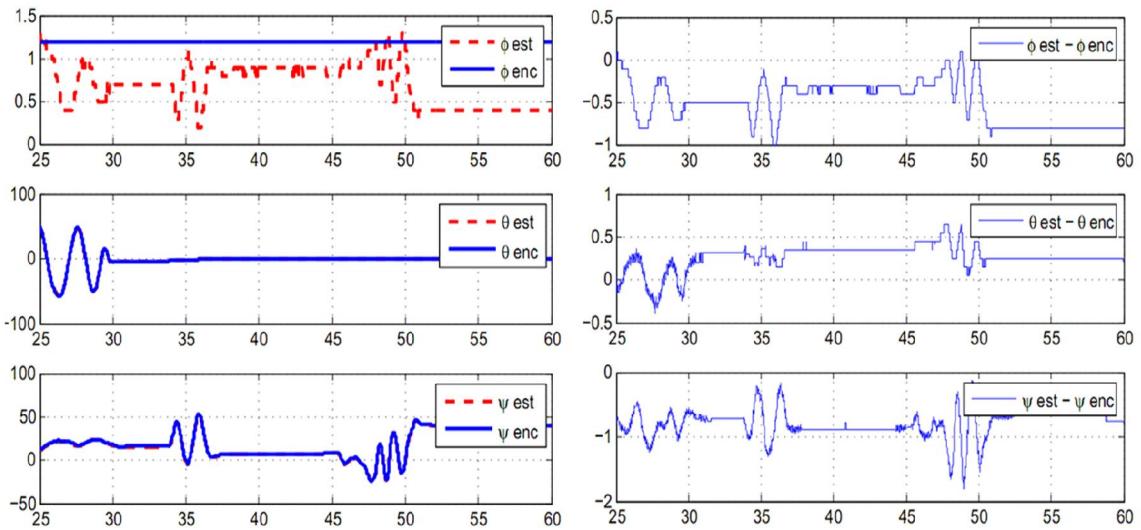
- Est: giá trị ước lượng
- Ref: giá trị tham khảo (từ IMU thương mại)
- Enc: giá trị tính từ encoder
- Raw: giá trị tính từ dữ liệu cảm biến chưa qua bộ lọc

Thí nghiệm 2:

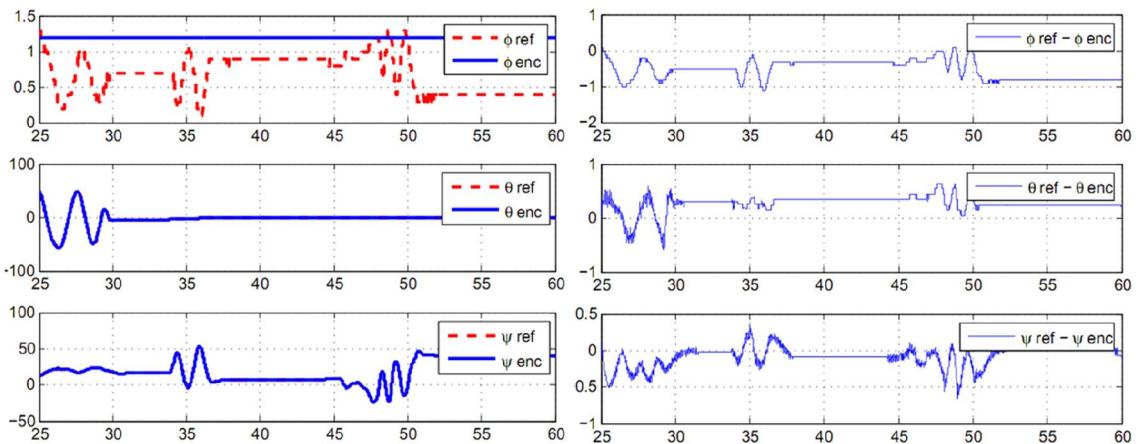
Bảng 4.2 Kết quả thí nghiệm 2

STT	Mẫu	Sai số RMS (độ)		
		ϕ	θ	ψ
1	TN2-S	0.0895	0.0500	0.2075
2	TN2-X	0.8712	0.8546	0.4884
3	TN2-Y	0.6934	0.6567	0.2355
4	TN2-Z	0.2405	0.4736	0.4537
5	TN2-XYZ	0.6826	0.6028	0.4162
6	TN2-AX	2.0132	0.5197	0.3845
7	TN2-AY	0.6767	1.3537	0.2759
8	TN2-AZ	0.3768	0.5203	1.0024
9	TN2-AXYZ	1.0792	1.1794	0.4829
10	TN2-M	0.2622	0.2998	1.8557

Từ hai bảng kết quả trên cho thấy sai số RMS của IMU đạt được nhỏ hơn 3° . Sau đây là các biểu đồ kết quả cho hai thí nghiệm và nhận xét kèm theo:

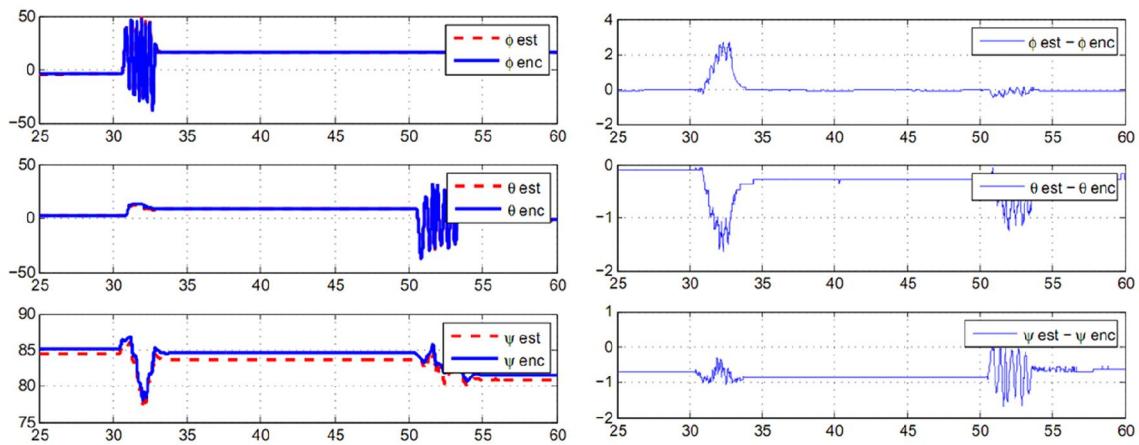


Hình 4.2 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ1 (1)

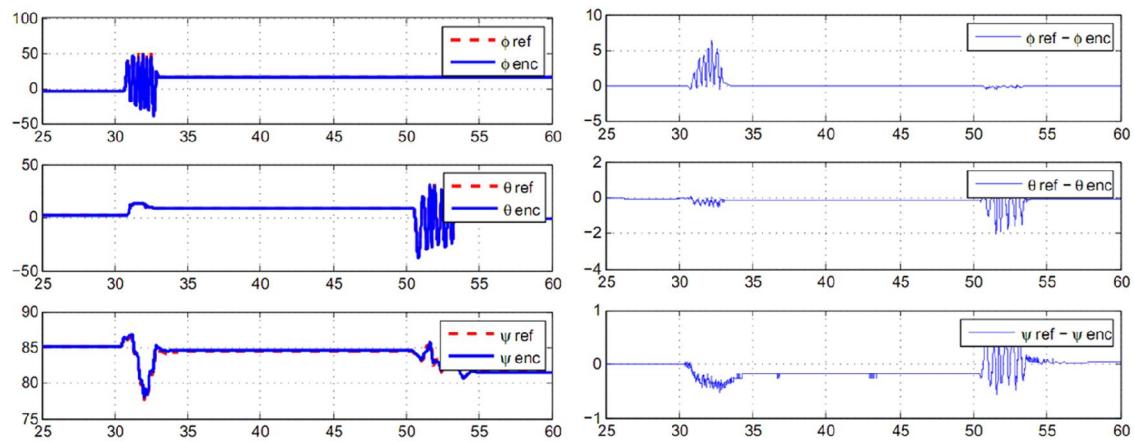


Hình 4.3 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ1 (2)

Nhận xét: đáp ứng sai số giữa thuật toán đã ước lượng và thuật toán trong IMU thương mại gần như nhau

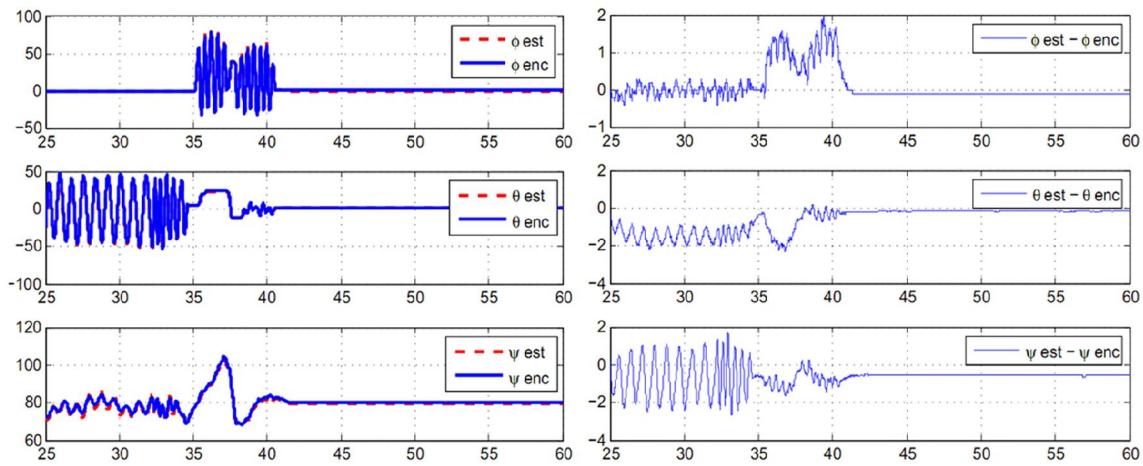


Hình 4.4 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ2 (1)

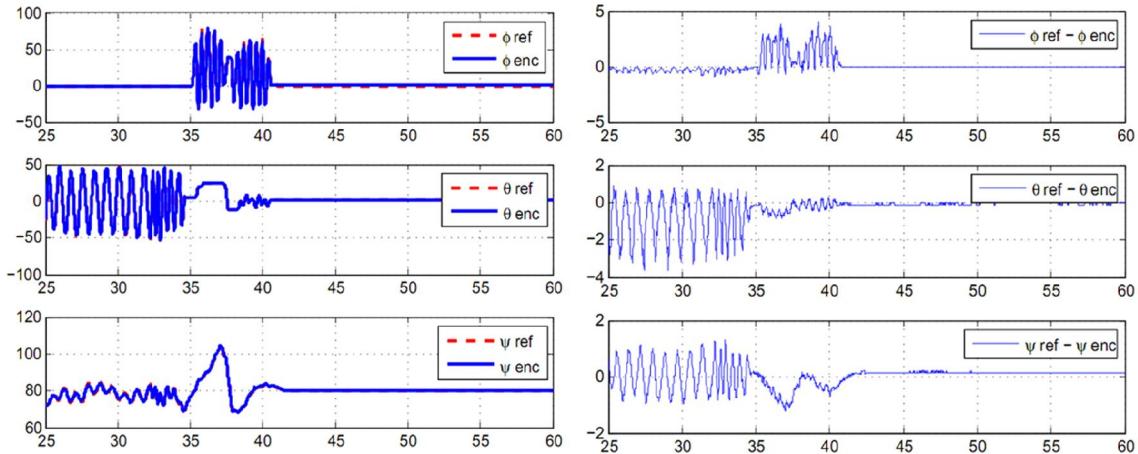


Hình 4.5 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ2 (2)

Nhận xét: đáp ứng sai số của thuật toán trong IMU thường mại tốt hơn so với thuật toán đã phát triển

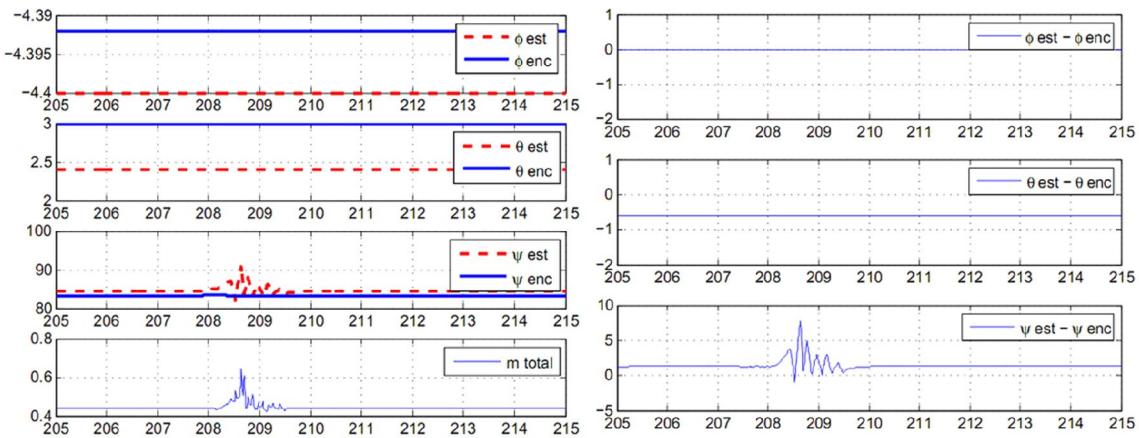


Hình 4.6 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ3 (1)

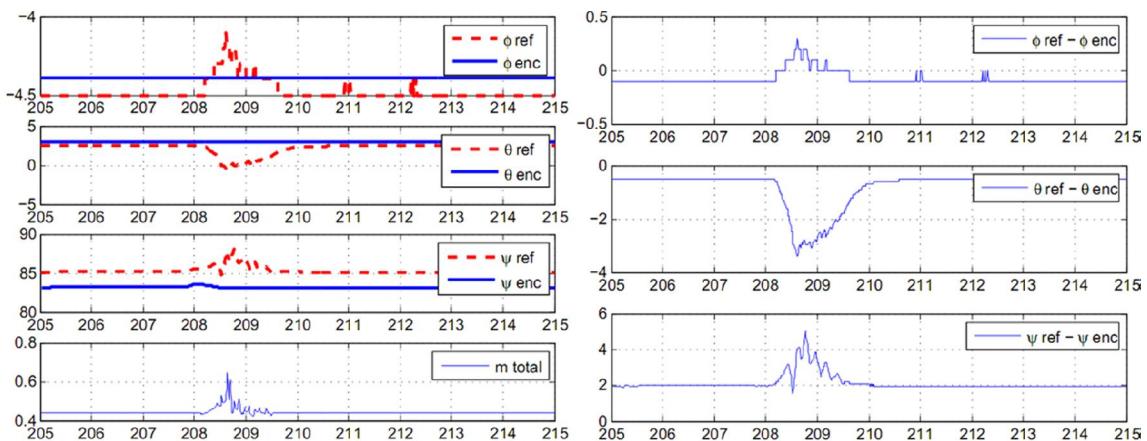


Hình 4.7 Biểu đồ ba góc quay và sai số của mẫu TN1-XYZ3 (2)

Nhận xét: trong trường hợp này đáp ứng sai số giữa thuật toán đã ước lượng cho góc roll và pitch tốt hơn thuật toán trong IMU thương mại, nhưng góc yaw có sai số lớn hơn



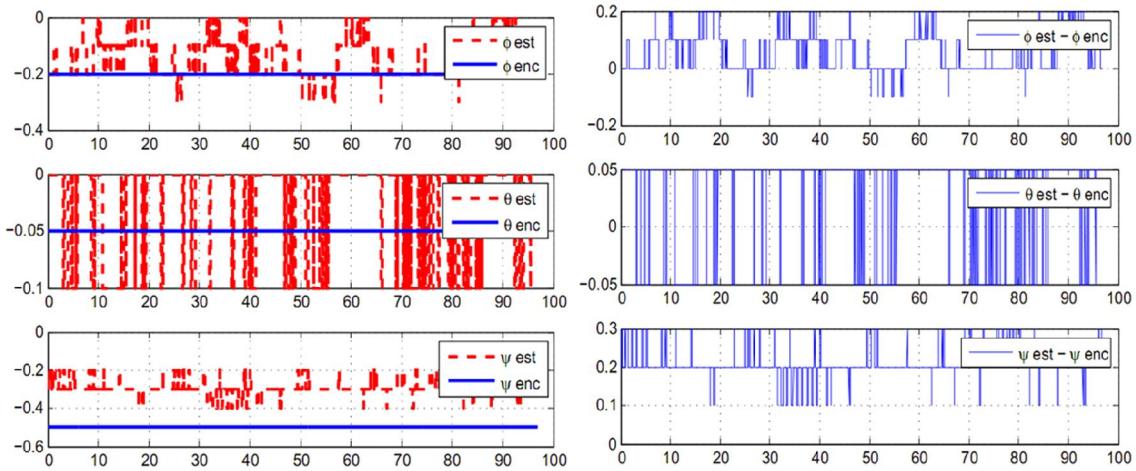
Hình 4.8 Biểu đồ ba góc quay và sai số của mẫu khi có nhiễu từ trường (1)



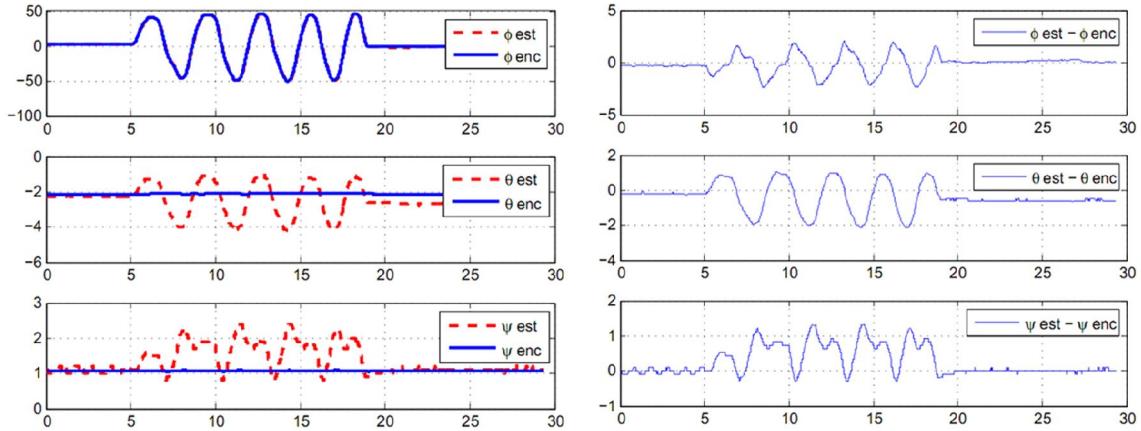
Hình 4.9 Biểu đồ ba góc quay và sai số của mẫu khi có nhiễu từ trường (2)

Nhận xét: trong trường hợp có nhiễu từ trường ngoài đáp ứng sai số góc yaw của IMU thương mại rất tốt tuy nhiên dẫn đến hai góc roll và pitch cũng có sai số đi kèm; đối với thuật toán đã phát triển đã tách ảnh hưởng từ trường lên hai góc roll và pitch nên mặc dù góc yaw sai nhưng vẫn không ảnh hưởng đến hai góc roll, pitch. Đây có thể nói là ưu điểm của mô hình Kalman dùng DCM so với mô hình Kalman dùng quaternion.

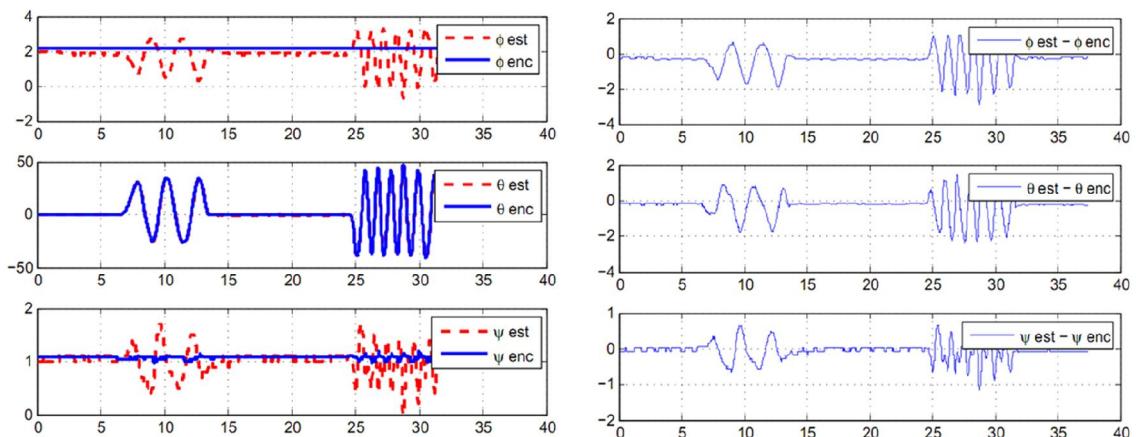
Tóm lại TN1 đã chứng minh tính đúng đắn của thuật toán Kalman dùng DCM, tuy đáp ứng của sai số khá tốt nhưng vẫn không đạt bằng kết quả ước lượng từ IMU thương mại do đó thuật toán còn phải cải tiến nhiều đặc biệt trong trường hợp có nhiễu từ trường. Qua kết quả này cũng xác nhận có thể tin tưởng vào việc sử dụng bàn xoay để kiểm tra độ chính xác IMU không cần thiết phải dùng IMU thương mại.



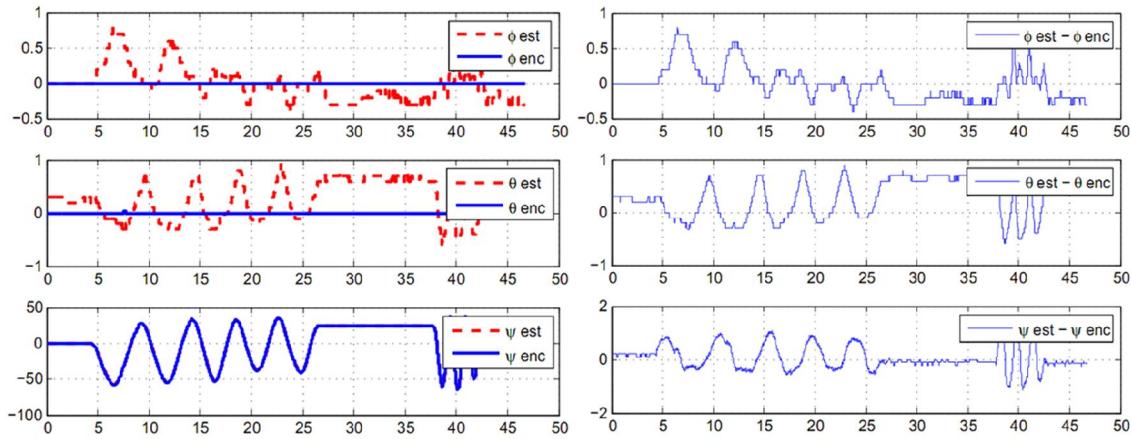
Hình 4.10 Biểu đồ ba góc quay và sai số của mẫu TN2-S



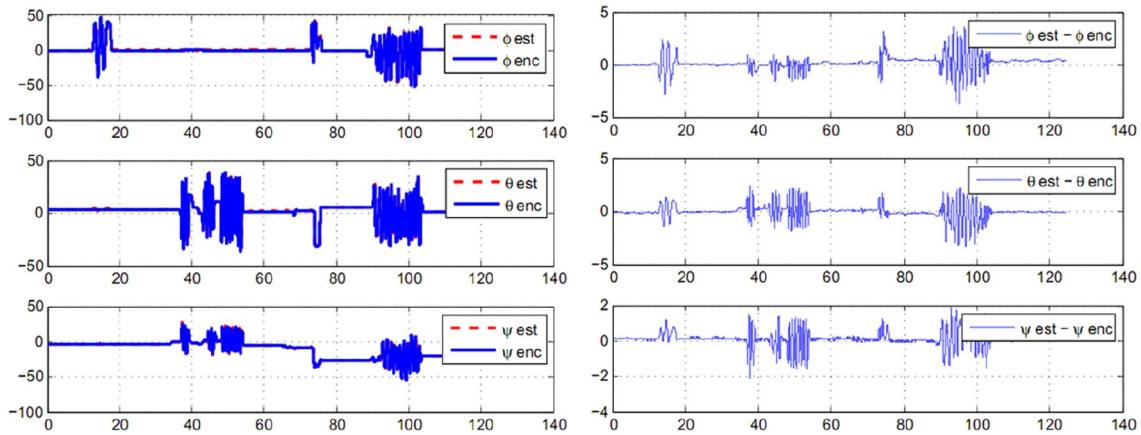
Hình 4.11 Biểu đồ ba góc quay và sai số của mẫu TN2-X



Hình 4.12 Biểu đồ ba góc quay và sai số của mẫu TN2-Y

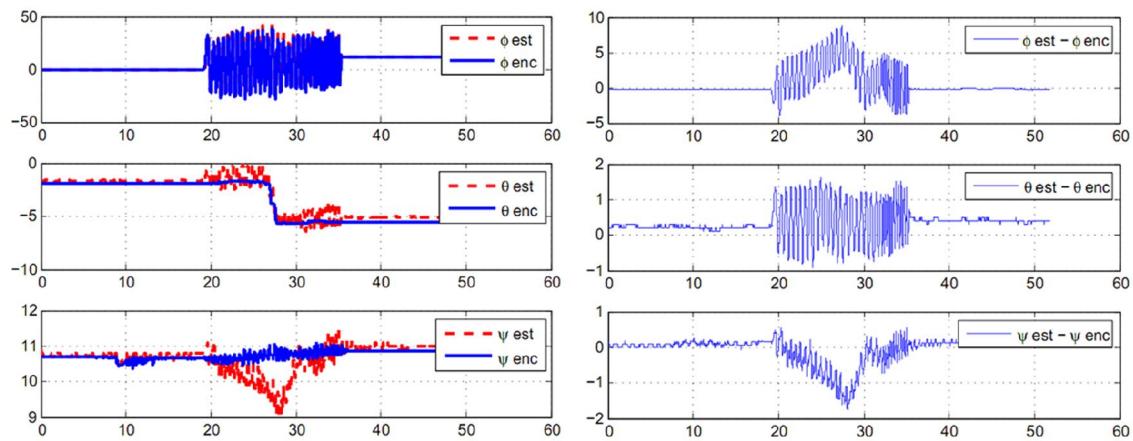


Hình 4.13 Biểu đồ ba góc quay và sai số của mẫu TN2-Z

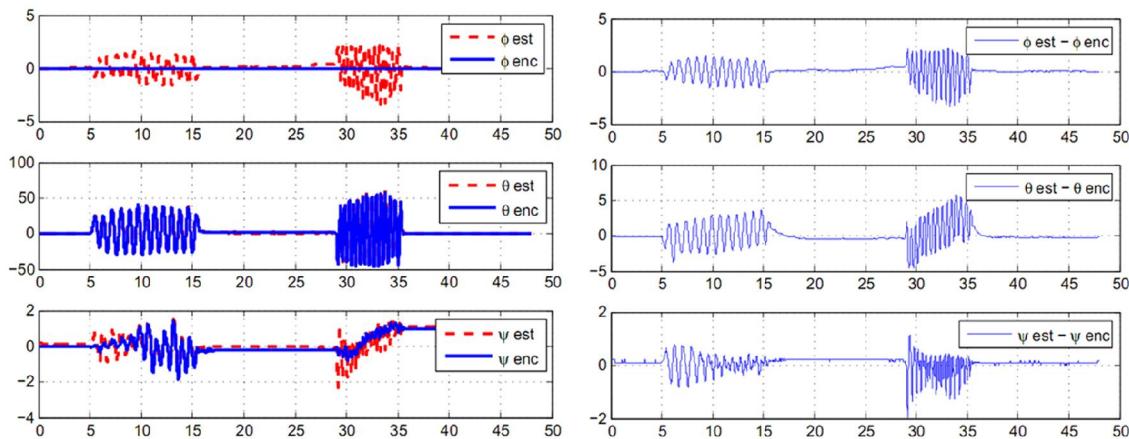


Hình 4.14 Biểu đồ ba góc quay và sai số của mẫu TN2-XYZ

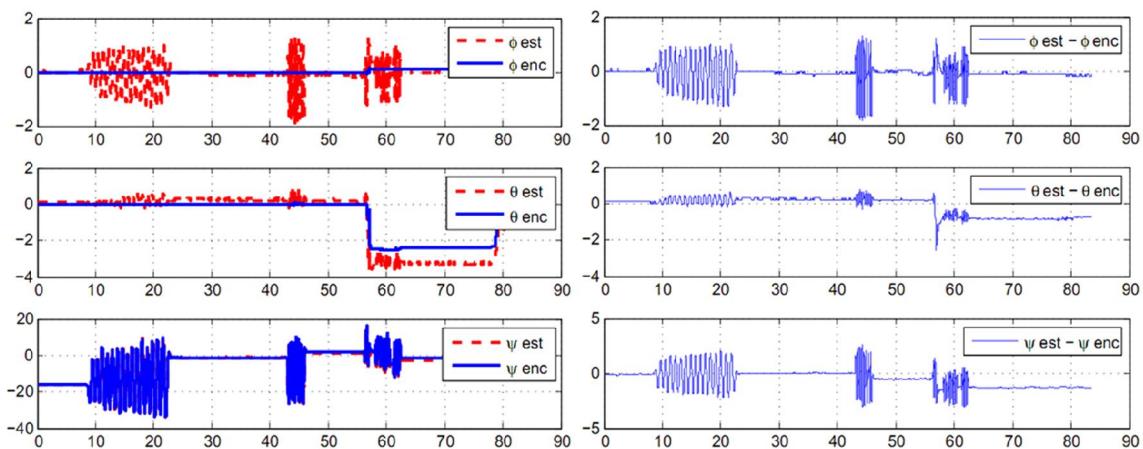
Nhận xét: kết quả sai số đối với kiểm tra tĩnh khá tốt, nhưng trong trường hợp quay từng trục các góc còn lại vẫn chạy theo sai số khá lớn (2° - 3°). Góc yaw đáp ứng khá tốt sai số nhỏ hơn 2° .



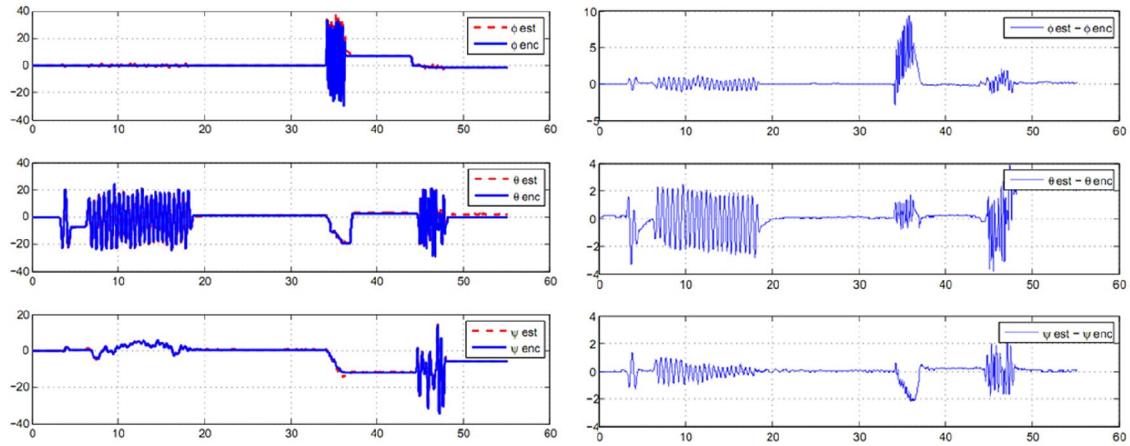
Hình 4.15 Biểu đồ ba góc quay và sai số của mẫu TN2-AX



Hình 4.16 Biểu đồ ba góc quay và sai số của mẫu TN2-AY

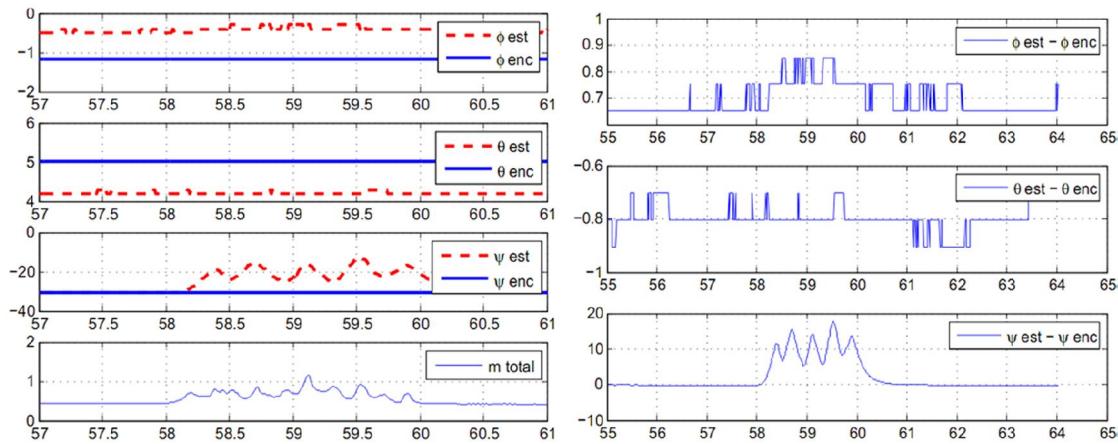


Hình 4.17 Biểu đồ ba góc quay và sai số của mẫu TN2-AZ



Hình 4.18 Biểu đồ ba góc quay và sai số của mẫu TN2-AXYZ

Nhận xét: đáp ứng khá tốt với gia tốc ngoài, sai số tối đa có thể lên 5° nhưng giá trị RMS vẫn đảm bảo dưới 3°



Hình 4.19 Biểu đồ ba góc quay và sai số của mẫu TN2-M (1)

Nhận xét: Thuật toán vẫn chưa thích nghi được trường hợp từ trường ngoài nên sai số góc yaw khá lớn, tuy nhiên góc roll và pitch vẫn không bị ảnh hưởng.

TN2 đã kiểm chứng các đáp ứng thực của IMU khi sử dụng thuật toán. Đáp ứng sai số tĩnh và động khi có gia tốc ngoài khá tốt, tuy nhiên không thích nghi được với trường hợp có nhiều từ trường.

Chương 5. Kết luận

5.1 Kết luận

Luận văn đã thực hiện được các nhiệm vụ đề ra:

- Tìm hiểu và đánh giá các giải thuật khác nhau để xây dựng bộ ước lượng ba góc quay.
- Hiển thực giải thuật trên hệ thống nhúng dùng vi điều khiển với tần số cập nhật giá trị ba góc quay là 100Hz.
- Đánh giá sai số của hệ thống IMU và cải tiến để sai số RMS ba góc nhỏ hơn 3° .
- Xây dựng phần mềm đồ họa 3D hiển thị mô hình IMU trên máy tính.

Tuy nhiên thuật toán ước lượng hiện vẫn chưa giải quyết hết bài toán ước lượng trong trường hợp có tốc độ ngoài mạnh hoặc nhiễu từ trường ngoài. Hướng cải tiến là thêm vào bộ lọc ba giá trị ước lượng cho từ trường ngoài, đồng thời chuyển sang mô hình Kalman gián tiếp (Indirect-Kalman Filter) để giảm thời gian chạy giải thuật Kalman trên IMU.

5.2 Hướng phát triển của đề tài

Đề tài có thể được ứng dụng vào các bài toán điều khiển cân bằng như robot một bánh, quadrotor... Có thể phát triển thành hướng nghiên cứu kết hợp IMU và GPS thành hệ thống INS dùng cho định vị và dẫn đường, hoặc kết hợp nhiều IMU thành hệ thống mô phỏng cử động người dùng trong y tế, điện ảnh...

TÀI LIỆU THAM KHẢO

- [1] Dan Simon (2006). *Optimal State Estimation Kalman, H_∞, and Nonlinear Approaches*. John Wiley & Sons.
- [2] Edwin K.P. Chong, Stanislaw H.Zak (2008) *An Introduction to Optimization*. John Wiley & Sons.
- [3] Greg Welch, and Gary Bishop (2006). *An Introduction to the Kalman Filter*
- [4] Lê Mạnh Thắng (2010). Luận văn cao học về IMU, Đại học Bách Khoa Tp HCM.
- [5] Sebastian O.H. Madgwick (2010). *Estimation of IMU and MARG orientation using a gradient descent algorithm*. <http://www.x-io.co.uk/>.
- [6] E. R. Bachmann et al (1999). “Orientation Tracking for Humans and Robots Using Inertial Sensors”. In *1999 International Symposium on Computational Intelligence in Robotics & Automation (CIRA99)*.
- [7] João Luís Marins et al. “An Extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors”. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- [8] Xiaoping Yun et al (2005). “Implementation and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking”. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation Spain 2005*.
- [9] Xiaoping Yun et al (2006). “Design, Implementation, and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking”. In *IEEE Transactions On Robotics, 2006*.
- [10] Nguyen Ho Quoc Phuong et al. “Study On Orientation Estimation With Three Different Representations”. In *Proceedings of the International Symposium on Electrical & Electronics Engineering 2007, Vietnam*.
- [11] Daniel Roetenberg et al. “Compensation of Magnetic Disturbances Improves Inertial and Magnetic Sensing of Human Body Segment Orientation”. In *IEEE Transactions On Neural Systems And Rehabilitation Engineering, 2005*.
- [12] Daniel Roetenberg (2006). *Inertial and Magnetic Sensing of Human Motion*. PhD Thesis. University of Twente.
- [13] 9-DOF Razor IMU: <https://www.sparkfun.com/products/9623>
- [14] Tài liệu hướng dẫn lập trình Python: <http://www.python.org/>

- [15] Allen Downey (2002). *How to Think Like a Computer Scientist, Learning with Python*. Green Tea Press
- [16] Tài liệu hướng dẫn lập trình *Vpython*: <http://www.vpython.org/>
- [17] Tài liệu hướng dẫn lập trình *PySerial*: <http://pyserial.sourceforge.net/>
- [18] Tài liệu ARM-Cortex M4 STM32F40x: <http://www.st.com/>