



UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
COLEGIADO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO

Gabriel Ferrari Batista Martins

**Estudo de acelerômetros para sistemas
embarcados. Uma abordagem de instrumentação
para estimação da velocidade**

Vitória, ES

2022

Gabriel Ferrari Batista Martins

Estudo de acelerômetros para sistemas embarcados. Uma abordagem de instrumentação para estimação da velocidade

Monografia apresentada ao Colegiado do Curso de Engenharia de Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Engenharia de Computação

Orientador: Prof. Camilo Arturo Rodriguez Diaz

Coorientador: Prof. Hans-Jorg Andreas Schneebeli

Vitória, ES

2022

Gabriel Ferrari Batista Martins

Estudo de acelerômetros para sistemas embarcados. Uma abordagem de instrumentação para estimação da velocidade/ Gabriel Ferrari Batista Martins. – Vitória, ES, 2022-

45 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Camilo Arturo Rodriguez Diaz

Coorientador: Prof. Hans-Jorg Andreas Schneebeli

Monografia (PG) – Universidade Federal do Espírito Santo – UFES
Centro Tecnológico

Colegiado do Curso de Engenharia de Computação, 2022.

1. Palavra-chave1. 2. Palavra-chave2. I. Fulano de Tal. II. Universidade Federal do Espírito Santo. IV. Estudo de acelerômetros para sistemas embarcados. Uma abordagem de instrumentação para estimação da velocidade

CDU 02:141:005.7

Gabriel Ferrari Batista Martins

Estudo de acelerômetros para sistemas embarcados. Uma abordagem de instrumentação para estimação da velocidade

Monografia apresentada ao Colegiado do Curso de Engenharia de Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Engenharia de Computação.

Trabalho aprovado. Vitória, ES, 26 de março de 2022:

Prof. Camilo Arturo Rodriguez Diaz
Orientador

Prof. Hans-Jorg Andreas Schneebeli
Coorientador

Prof. Dr. Ricardo Carminati de Mello
Convidado 1

**Prof. Dra. Eliete Maria de Oliveira
Caldeira**
Convidado 2

Vitória, ES
2022

Agradecimentos

Agradeço aos meus pais, que sempre lutaram por mim. A toda a minha família que comemorou minhas vitórias e sempre acreditou que eu tenho potencial de fazer mais.

Agradeço aos amigos e colegas que fiz durante o curso, por serem companheiros de guerra quando o curso apertava. Agradeço aos membros do Vitória Baja, por terem sido sempre amigáveis e terem me dado um impulso que culminou nesse trabalho.

Agradeço aos amigos fora da UFES, por compartilharem suas alegrias comigo e pelas horas de videogames juntos, me trouxeram muitos momentos felizes.

Aos meus coordenadores, que me ajudaram a desenvolver a ideia desse trabalho e foram pacientes ao me guiar.

Resumo

O uso de sensores embarcados hoje em dia é muito comum em diversos tipos de aplicação. Um tipo de sensor normalmente usado é o acelerômetro, que por só medir a aceleração requer integração dos dados para obter velocidade e deslocamento. Junto a isso, é necessário um tratamento das medidas para torna-las mais precisas, reduzindo o ruído, e a escolha de um método de integração menos propenso a erro.

Neste trabalho foi utilizado um acelerômetro MEMS comum, de baixo custo para obter medições em diferentes casos de testes. A escolha de um acelerômetro genérico se dá para que as abordagens utilizadas sejam mais abrangentes, servindo para diferentes modelos de acelerômetro e aplicações além daquelas mostradas aqui. Com posse das medições do acelerômetro será obtida a velocidade e deslocamento, que são mais facilmente comparáveis com o esperado para casos de testes que somente aceleração.

Ao longo desse trabalho serão explicados e comparados algumas abordagens para redução de erro e integração, com o fim de concluir a utilidade do sensor para fins de velocidade e deslocamento e o quanto as abordagens utilizadas ajudam a obter dados mais precisos.

Palavras-chaves: acelerômetro, velocidade, redução de erro, ruído, giroscópio.

Lista de ilustrações

| | |
|--|----|
| Figura 1 – MPU-6050 e seus eixos. | 10 |
| Figura 2 – Demonstração visual da Integração Trapezoidal (FILE. . . , 2009). . . . | 15 |
| Figura 3 – Demonstração visual da integração por triângulos (BROM, 2013). . . . | 15 |
| Figura 4 – Algoritmo do Filtro de Kalman (SHARBAFI et al., 2010). | 16 |
| Figura 5 – MPU-6050. | 19 |
| Figura 6 – Circuito utilizado para coleta de dados. | 20 |
| Figura 7 – Análise do Caso 2. | 24 |
| Figura 8 – Análise do Caso 3. | 25 |
| Figura 9 – Análise do Caso 4. | 26 |
| Figura 10 – Análise do Caso 10. | 27 |

Lista de tabelas

| | |
|--|----|
| Tabela 1 – Formulas Iniciais. | 13 |
| Tabela 2 – Formulas com rotação. | 14 |
| Tabela 3 – Resumo dos casos apresentados na Seção 3.2. | 20 |
| Tabela 4 – Tabela de precisão do sensor. | 20 |
| Tabela 5 – Resumo dos casos apresentados na Seção 3.2. | 23 |
| Tabela 6 – Comparação dos dados antes e depois do Filtro de Kalman para o Acelerômetro. | 28 |
| Tabela 7 – Comparação dos dados antes e depois do Filtro de Kalman para o Giroscópio. | 29 |
| Tabela 8 – Melhora da Variância com Filtro de Kalman. | 29 |
| Tabela 9 – Comparações da Estimação da Distância. | 30 |

Lista de abreviaturas e siglas

| | |
|-------------------|--|
| MEMS | Sistemas micro-eletromecânicos |
| Ax | Leitura do Acelerômetro/Aceleração no eixo X |
| Ay | Leitura do Acelerômetro/Aceleração no eixo Y |
| Gz | Leitura do Giroscópio no eixo Z (deg/s) |
| IMU | Sensor de medição de Inércia |
| DMP | Processador de Movimento Digital |
| I2C | Barramento Serial Multi mestre |
| Matlab | Plataforma de Computação Numérica da MATHWORKS |
| STD | Desvio padrão |
| Var | Variância |
| Trapz./Trapezoide | Método de Integração Trapezoidal |
| Triâng./Triângulo | Método de Integração por Triângulos |
| Kalman | Filtro de Kalman |

Sumário

| | | |
|----------|---|-----------|
| 1 | INTRODUÇÃO | 10 |
| 1.1 | Motivação e Justificativa | 11 |
| 1.2 | Extração de Dados de Sensores | 11 |
| 1.3 | Objetivos | 11 |
| 1.4 | Metodologia | 11 |
| 1.5 | Estrutura do Trabalho | 12 |
| 2 | REFERENCIAL TEÓRICO | 13 |
| 2.1 | Modelo de um Sistema Dinâmico | 13 |
| 2.2 | Obtenção da Posição | 14 |
| 2.3 | Métodos de Integração | 14 |
| 2.3.1 | Integração Trapezoidal | 14 |
| 2.3.2 | Integração por Triângulos | 15 |
| 2.4 | Filtro de Kalman | 16 |
| 3 | IMPLEMENTAÇÃO | 19 |
| 3.1 | Obtenção dos dados do sensor | 19 |
| 3.2 | Modelo | 20 |
| 3.3 | Calculo da posição por integração | 22 |
| 3.4 | Calculo da posição usando filtros de Kalman | 22 |
| 4 | RESULTADOS | 23 |
| 4.1 | Melhores casos | 23 |
| 4.2 | Tabelas de resultados | 28 |
| 5 | CONCLUSÕES E TRABALHOS FUTUROS | 31 |
| | REFERÊNCIAS | 32 |
| | APÊNDICES | 33 |
| | APÊNDICE A – CÓDIGOS UTILIZADOS | 34 |

1 Introdução

Para aplicações de navegação, robótica ou navegação, a utilização de sensores de medição de inércia (IMU) como acelerômetros e giroscópios é cada vez maior. Um tipo de IMU muito popular são os MEMS (Sistemas micro-eletromecânicos) devido ao seu baixo custo, baixo consumo de energia e portabilidade. O MPU 6050 (INVENSENSE; CT, 2020) é um exemplo de IMU MEMS que combina acelerômetro e giroscópio para três dimensões, somando seis eixos de orientação. Isso significa que ele pode medir a aceleração linear e angular em relação aos eixos X, Y e Z estabelecidos pelo dispositivo.

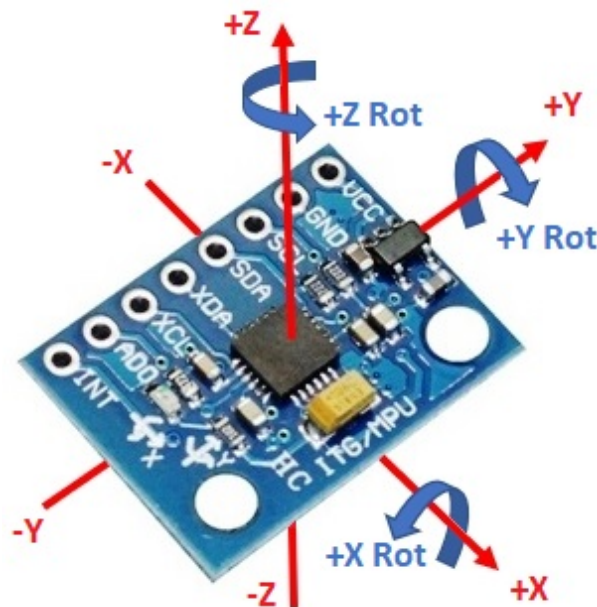


Figura 1 – MPU-6050 e seus eixos.

Utilizando essas medições das acelerações em intervalos conhecidos, podemos integrar para obter uma estimativa das velocidades e integrar novamente para obter a posição e orientação. No entanto, fatores como ruído, falta de calibração e erros de integração levam a dados mais imprecisos, principalmente nos casos de orientação e posição, que requerem mais de uma integração.

Torna-se necessário então a exploração de abordagens que consigam obter acelerações, velocidades e deslocamentos mais precisos a partir dessas medições, levando em conta e reduzindo as distorções e erros que aparecem nos dados obtidos durante o processo. (MA'ARIF et al., 2019)

1.1 Motivação e Justificativa

Esse trabalho foi motivado pela experiência do autor em medição e coleta de dados de estimação de velocidade para sistemas embarcados no projeto BAJA-SAE da UFES, o que levou a um maior interesse no assunto.

A justificativa desse trabalho é a demonstração de métodos de tratamento de dados e posicionamento a partir de sensores IMU que não sejam dependentes da arquitetura utilizada para funcionar, utilizando como exemplo um sensor comum de baixo custo. A generalidade da solução então a tornaria aplicável a um maior numero de casos.

1.2 Extração de Dados de Sensores

A questão é que achar a posição do sistema dado uma medida indireta como aceleração é questão do observador do sistema. O observador deve então escolher referenciais apropriados e métodos de integração e rotação adequados para que as medições obtidas do sensor se traduzam adequadamente ao referencial escolhido. Além disso é necessário lidar com o erro proveniente do método de integração e do ruído do sensor, conforme será demonstrado em capítulos posteriores.

1.3 Objetivos

O objetivo geral desse trabalho é obter informações do estado de um sistema a partir de observações. Como estamos usando um sensor inercial, o objetivo específico torna-se a extração da posição do sistema a partir de leituras do giroscópio e acelerômetro.

1.4 Metodologia

Primeiramente foram levantados os requisitos do projeto, tal como casos de teste, elaboração da arquitetura de registro de dados e algoritmos a serem testados, etc. Esses requisitos foram documentados mais detalhadamente para referência depois.

Com o escopo inicial definido, foi implementado a arquitetura de registro de dados e reunido informações sobre o tratamento dos dados a serem obtidos e algoritmos a serem testados, *datasheets* do sensor ([INVENSENSE; CT, 2020](#)), e referências para os algoritmos ([MA'ARIF et al., 2019](#); [BROM, 2013](#); [MATHWORKS, 2022](#)). A partir dessas informações foi elaborado o o circuito necessário para a coleta de dados e definido o local para testes.

Dado que o sensor tem 4 configurações de sensibilidade para giroscópio e acelerômetro, e que o caso de testes definido inicialmente levava a 8 tipos de rota, o total de testes seria $4 \times 4 \times 8 = 128$ casos. Para simplificar o processo foi definido utilizar apenas

as configurações máximas e mínimas do giroscópio e acelerômetro, deixando 16 casos de teste.

Coletados as medições foram utilizadas ferramentas computacionais como MATLAB para tratar os dados e aplicar os algoritmos propostos. Por fim foi elaborado um exemplo da implementação de um desses algoritmos em C, que poderia ser usado em tempo real com modificações mínimas em qualquer microcontrolador com processador de ponto flutuante e memória o bastante.

1.5 Estrutura do Trabalho

Neste capítulo é introduzida a ideia do trabalho e os passos feitos para realiza lo. Os outros 4 capítulos podem ser descritos como:

- Capítulo 2 - Referencial Teórico: É apresentado as características e formulas que servem de base para o desenvolvimento do projeto, tais como os dados que são obtidos do sensor e como funcionam os algoritmos propostos.
- Capítulo 3 - Implementação: Demonstra como a base teórica estabelecida anteriormente é aplicada no projeto e a caracterização da implementação tais como casos de testes.
- Capítulo 4 - Resultados: Os resultados obtidos são demonstrados, contextualizados e analisados.
- Capítulo 5 - Comentários e conclusões: É feito um levantamento geral da analise dos resultados, como o que era esperado, o que não era esperado, explicações para o inesperado, o que foi feito e o que ainda pode ser elaborado por outros trabalhos.

2 Referencial Teórico

Aqui serão apresentados as tecnologias utilizadas no desenvolvimento do projeto.

2.1 Modelo de um Sistema Dinâmico

A cinemática é o estudo do movimento de pontos corpos ou sistemas sem se preocupar com sua origem (BOTTEMA; ROTH, 1990). Para o fim desse trabalho serão demonstrados as formulas base usadas para modelar o sistema.

t, k : Tempo contínuo e discreto respectivamente.

A : Aceleração Linear / Medição do Acelerômetro (m/s^2).

V : Velocidade Linear (m/s).

P : Posição do corpo (m, Km, etc).

G : Velocidade Angular / Medição do Giroscópio ($deg/s, rad/s$).

θ : Angulo da rotação do corpo em relação a origem do plano deg, rad .

Δt : Intervalo desde a ultima medição / Intervalo de amostragem.

$\int x dx$: Integração em tempo contínuo.

$\sum_{i=0}^n a_i$: Soma de a_i de 0 a n / Integração em tempo discreto.

E : Plano de referência do sensor conforme a Figura 1.

G : Plano de referência global.

R_1^2 : Rotação do plano de referência 1 para 2.

Me_t^E : Medição feita no eixo e do plano P no instante t .

Inicialmente, podemos então dizer que:

Tabela 1 – Formulas Iniciais.

| Tempo contínuo | Tempo discreto |
|--|---|
| $V_{(t)} = V_{t=0} + \int_0^t A_{(t)} dt$ $P_{(t)} = P_{t=0} + \int_0^t V_{(t)} dt + \int_0^t \int_0^t A_{(t)} dt^2$ $\theta_{(t)} = \theta_{t=0} + \int_0^t G_{(t)} dt$ | $V_{[k]} = V_{[k-1]} + \Delta t * A_{[k]}$ $P_{[k]} = P_{[k-1]} + \Delta t * V_{[k]} + \frac{\Delta t^2}{2} * A_{[k]}$ $\theta_{[k]} = \theta_{[k-1]} + \Delta t * G_{[k]}$ |

Mas ao levar em conta as sucessivas rotações do sistema, temos que: Para $E_0 = G$,

Tabela 2 – Formulas com rotação.

| Tempo contínuo | Tempo discreto |
|--|---|
| $R(t)_E^G = R_{-\theta z(t)} * R_{-\theta y(t)} * R_{-\theta x(t)}$ | $R(t)_E^G = R_{-\theta z[k]} * R_{-\theta y[k]} * R_{-\theta x[k]}$ |
| $Vx, y, z_{(t)}^G = Vx, y, z_{t=0}^G + \int_0^t R_E^G * Ax, y, z_{(t)} dt$ | $Vx, y, z_{[k]}^G = V_{[k-1]} + \Delta t * R_E^G * Ax, y, z_{[i]}$ |
| $Px, y, z_{(t)} = Px, y, z_{t=0} + \int_0^t Vx, y, z_{(t)} dt + \int_0^t \int_0^t * Ax, y, z_{(t)} dt^2$ | $Px, y, z_{[k]} = Px, y, z_{[k-1]} + \Delta t * Vx, y, z_{[i]} + \frac{\Delta t^2}{2} * Ax, y, z_{[i]}$ |
| $\theta x, y, z_{(t)} = \theta x, y, z_{t=0} + \int_0^t R_E^G * Gx, y, z_{(t)} dt$ | $\theta x, y, z_{[k]} = \theta x, y, z_{[k-1]} + \Delta t * R_E^G * Gx, y, z_{[i]}$ |

2.2 Obtenção da Posição

Conforme visto na Seção 2.1, já existem formulas para modelar um sistema que estime a posição a partir da aceleração. O processo pode ser descrito simplesmente como integrar a aceleração para obter a velocidade e integrar a velocidade para obter a posição. No entanto há duas fontes principais de erro nesse processo.

A primeira fonte é o processo de integração. Como podemos ver nas Figuras 2 e 3 a escolha do método de integração e dos intervalos leva a áreas do gráfico da função não serem representadas corretamente. Isso leva ao acúmulo do erro da integral, que piora com integrações sucessivas (LAURIE, 1985). Outra fonte de erro é o ruído de medição, que tende a aumentar junto com a sensibilidade do sensor e pode ter diversas causas.

Uma solução apresentada para ambos os problemas é o Filtro de Kalman (LAGES, 2005), um filtro recursivo preditivo para reduzir o ruído, que pode ser aplicado em um sistema que realize a integração e fusão das medições para reduzir o ruído das leituras originais e o erro do processo de integração. O Filtro de Kalman é apresentado com mais detalhe na Seção 2.4.

2.3 Métodos de Integração

Aqui são apresentados o funcionamento de dois métodos de integração utilizados a fim de comparação na sua precisão em obter os estados do sistema a partir das leituras do sensor.

2.3.1 Integração Trapezoidal

A Integração Trapezoidal é um método muito utilizado de aproximação da integral que consiste em dividir a área abaixo do gráfico em trapezoides, calculando a área de cada trapezoide e somando. Para a Figura 2 por exemplo tem se que a área do primeiro

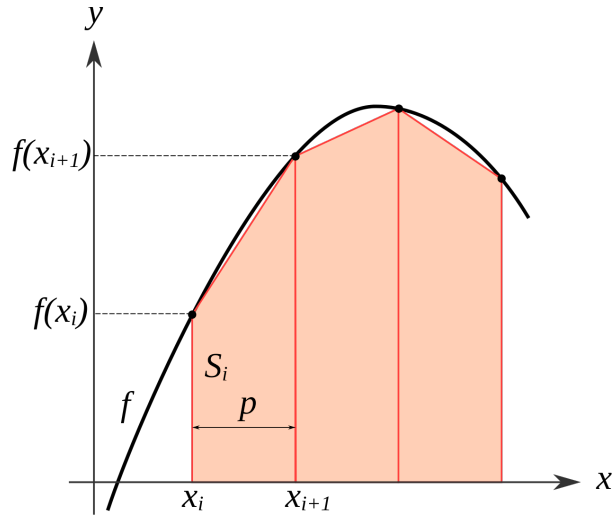


Figura 2 – Demonstração visual da Integração Trapezoidal (FILE. . . , 2009).

trapézio é aproximadamente $p * \frac{1}{2} * (f(x_{i+1}) - f(x_i))$ e repetindo esse calculo e somando consecutivamente as áreas entre pontos a e b temos a aproximação de $\int_a^b f(x)dx$. Apesar de fácil de implementar, o erro tende a aumentar quanto maior o intervalo entre os pontos conhecidos e para funções que alteram bruscamente dentro do intervalo p pode não obter resultados muito precisos.

2.3.2 Integração por Triângulos

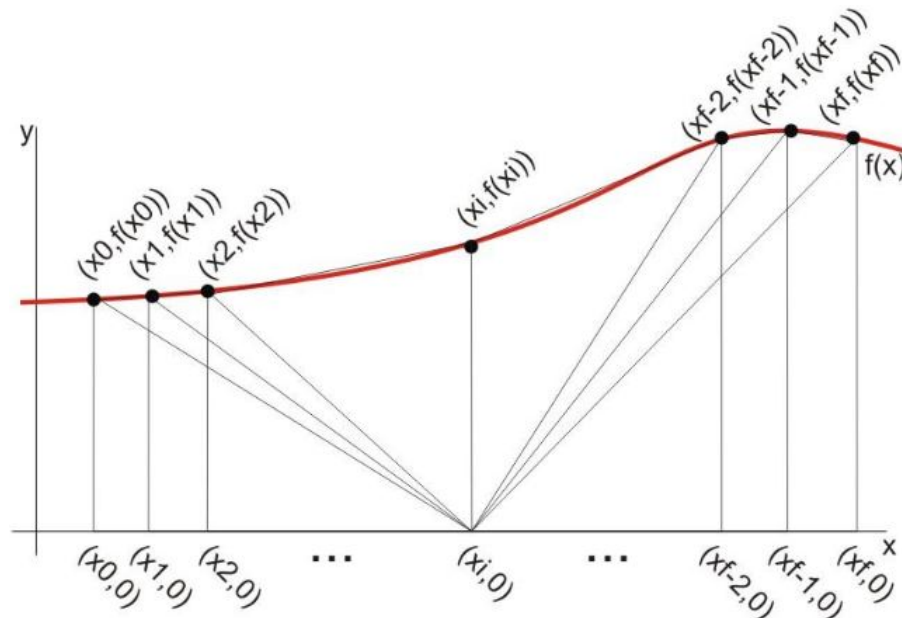


Figura 3 – Demonstração visual da integração por triângulos (BROM, 2013).

A Integração por Triângulos (BROM, 2013) é um método de integração que escolhe um ponto x_i no intervalo de integração e soma as áreas dos triângulos formados pelos pontos $(x_i, f(x_i))$, $(x_j, 0)$, $(x_j, f(x_j))$ para j de 0 a f . Ele apresenta uma precisão similar a

outros métodos mais conhecidos e converge mais rapidamente, porém não faz nada para reduzir o erro de integração.

2.4 Filtro de Kalman

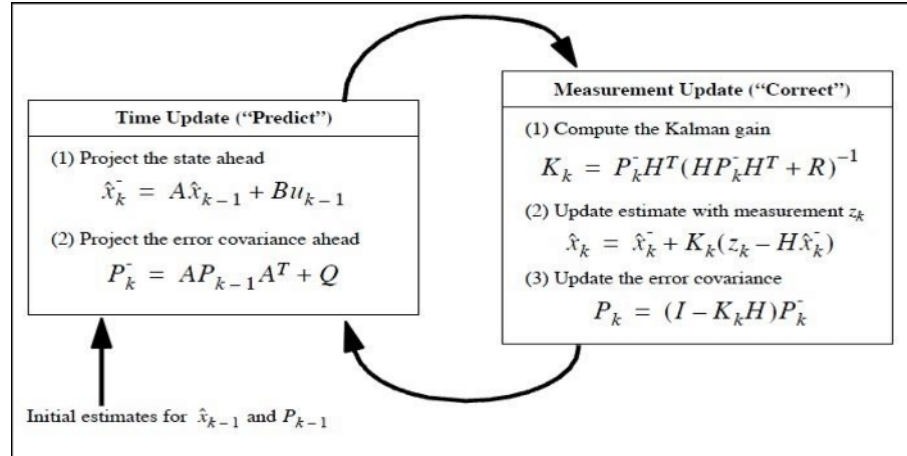


Figura 4 – Algoritmo do Filtro de Kalman (SHARBAFI et al., 2010).

O Filtro de Kalman é uma solução recursiva para filtragem de dados discretos (LAGES, 2005). Como mostrado na Figura 4 o algoritmo do Filtro é um ciclo composto de predição e atualização, onde na predição ele faz uma previsão dos estados do sistema e da matriz de covariância de erro, e na atualização ele computa o ganho de Kalman para atualizar as predições com os valores reais recebidos. Dado um sistema discreto cujo estado $x[k]$ pode ser descrito por

$$x[k] = Ax[k-1] + Bu[k-1] + w[k-1] \quad z[k] = Hx[k] + v[k]$$

Onde:

$x[k]$: Estados do sistema no momento k

$z[k]$: Estados de observação do sistema em k / medições do sensor

$u[k]$: Entradas do sistema em k

A : Matriz de transição de estados

B : Matriz de transformação da entrada

H : Matriz de transformação dos estados do sistema para os estados de observação

$w[k]$: Ruído de medição com distribuição normal e matriz de covariância Q

$v[k]$: Ruído de observação com distribuição normal e matriz de covariância R

Q : Matriz de covariância do erro dos estados de $x[k]$

R : Matriz de covariância do ruído dos estados de $z[k]$

Considerando o sistema com que estamos trabalhando não tem entradas, apenas

estados de observação, podemos fazer:

$$x_{[k]} = Ax_{[k-1]} + w_{[k-1]}$$

Logo as equações de Kalman para o sistema se tornam:

Predição:

1. $x_{[k|k-1]} = Ax_{[k-1|k-1]} + w_{[k-1]}$
2. $P_{[k|k-1]} = AP_{[k-1|k-1]}A^T + Q$

Atualização:

1. $K_{[k]} = P_{[k|k-1]}H^T(H P_{[k|k-1]}H^T + R)^{-1}$
2. $x_{[k|k]} = x_{[k|k-1]} + K_{[k]}(z_{[k]} - Hx_{[k|k-1]} - v_{[k]})$
3. $P_{[k|k]} = (I - K_{[k]}H)P_{[k|k-1]}$

Onde:

$Valor_{[k1|k2]}$: Predição do valor em $k1$ dado o valor em $k2$, se $k1 \leq k2$ então é apenas o valor da variável já conhecido

$Matriz^T$: Matriz Transposta

$Matriz^{-1}$: Inversa da Matriz

I : Matriz Identidade

K : Ganho de Kalman

P : Matriz de covariância do erro

Covariância: A covariância é uma medida do grau de dependência de duas variáveis. No contexto atual, a covariância entre variáveis X e Y é o desvio padrão de uma multiplicado pela outra, ou seja $COV(X, Y) = \sigma_X * \sigma_Y$. O desvio padrão é a média das diferenças entre os valores e a média do todo, e aqui representa uma expectativa do ruído a ser encontrado.

Para $x = [x_1, x_2, \dots, x_n]^T$ temos $w = [r_1 * \sigma_1, r_2 * \sigma_2, \dots, r_n * \sigma_n]^T$ onde r_n é um numero aleatório de distribuição normal e σ_n é o desvio padrão de x_n . Podemos definir então $Q = [\sigma_1, \dots, \sigma_n]^T * [\sigma_1, \dots, \sigma_n]$, $v = Hw$ e $R = (H * [\sigma_1, \dots, \sigma_n])^T * (H * [\sigma_1, \dots, \sigma_n])$. Para os sensores utilizados os valores de σ para acelerômetro e giroscópio são os valores padrões de ruído encontrados no datasheet (INVENSENSE; CT, 2020), que são $\sigma_{accel} = \frac{400 * 10^{-6} * g}{\sqrt{f_{amostragem}}}$ e $\sigma_{gyro} = 0.05$. Podemos também aplicar as formulas da Seção 2.1 para derivar os outros desvios padrões como $\sigma_{vel} = \Delta t * \sigma_{[accel]}$.

Para que o Filtro de Kalman possa ser usado de forma adequada, o ruído do sistema deve ser Gaussiano, a frequência de amostragem deve ser regular e bem menor que a

frequencia do sistema, e o processo deve poder ser modelado como as equações descritas no início dessa sessão. A forma normal dele não funciona em sistemas não-lineares, mas há variações como o Filtro de Kalman Extendido que pode ser usado no lugar([LAGES, 2005](#)).

3 Implementação

Aqui será especificado o circuito de coleta de dados, os códigos utilizados para coleta e análise de dados e casos de teste.

3.1 Obtenção dos dados do sensor

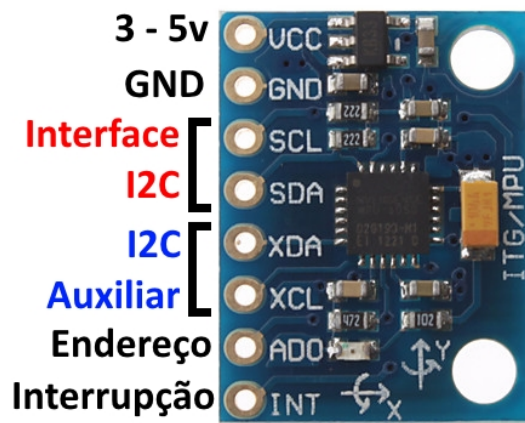


Figura 5 – MPU-6050.

Foi utilizado o MPU-6050 ([INVENSENSE; CT, 2020](#)), um IMU MEMS, que conta com giroscópio e acelerômetro para 3 eixos de 16 bits. Isso significa que ele pode medir a aceleração angular e linear nos eixos X,Y e Z referentes a si mesmo, conforme descrito na Figura 1. Ele foi escolhido por ser um modelo padrão de baixo custo mas pode facilmente ser substituído por similares dependendo da aplicação.

Ele conta com quatro configurações de sensibilidade, conforme descrito na Tabela 4, onde o máximo e mínimo da resolução é distribuída em incrementos iguais pelos 16 bits do sensor como um numero inteiro com complemento de 2. A configuração e leitura dos dados pode ser feita através da manipulação de registradores por I2C ([MANKAR et al., 2014](#)) com suporte a frequências de até 400 kHz e também é possível configurar o sensor para acumular as medições em um registrador FIFO de 1024 bytes, que pode enviar uma interrupção assim que encher, automatizando o processo de coleta de medidas.

Um fator interessante do MPU-6050 é a presença do Processador Digital de Movimento (DMP), um processador embutido no sensor capaz de realizar cálculos sobre as medições, tais como combinar os dados dos acelerômetros e giroscópios e entregar medições que fatoram a rotação do sistema enquanto ativo e o escorregamento das medições do giroscópio, além de converter valores brutos dos registradores em valores reais. Em C++,

Tabela 3 – Resumo dos casos apresentados na Seção 3.2.

| Sensor | Precisão | Resolução | Sensitividade |
|---------------------|----------|------------------|----------------|
| Giroscópio | 16 bits | ± 250 °/seg | 131 LSB/(°/s) |
| | | ± 500 °/seg | 65,5 LSB/(°/s) |
| | | ± 1000 °/seg | 32,8 LSB/(°/s) |
| | | ± 2000 °/seg | 16,4 LSB/(°/s) |
| Acelerômetro | 16 bits | $\pm 2g$ | 16384 LSB/g |
| | | $\pm 4g$ | 8192 LSB/g |
| | | $\pm 8g$ | 4096 LSB/g |
| | | $\pm 16g$ | 2048 LSB/g |

Tabela 4 – Tabela de precisão do sensor.

já existem bibliotecas para o MPU-6050 que utilizam o DMP na coleta de dados tais como a I2CDevLib (ROWBERG, 2014) facilitando a coleta de dados.

3.2 Modelo

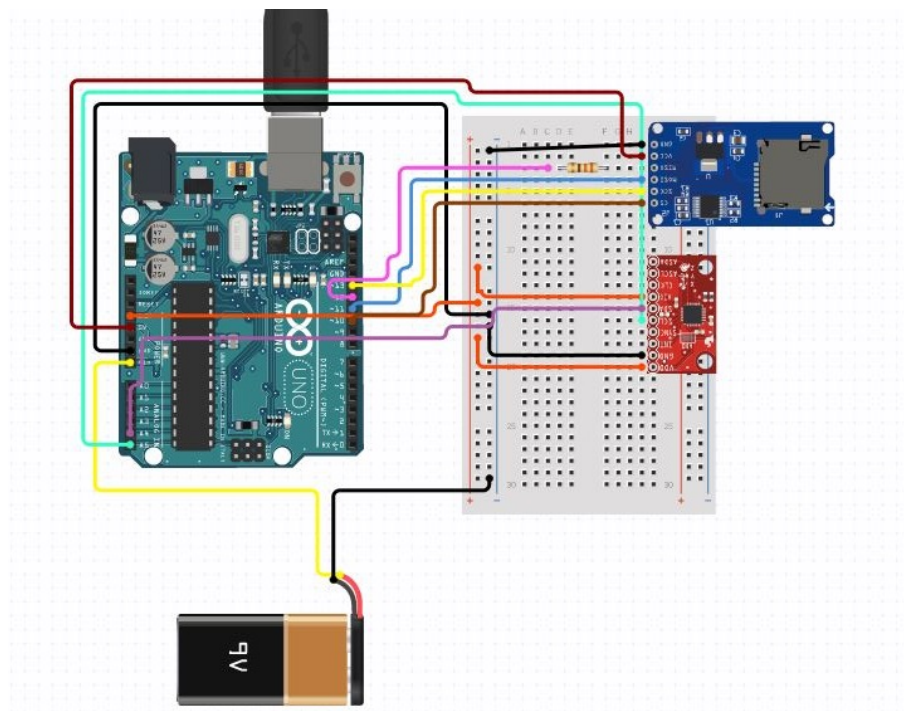


Figura 6 – Circuito utilizado para coleta de dados.

O circuito utilizado para a coleta de dados foi o da Figura 6, composto de:

- Uma placa Arduino UNO;
- Um leitor de cartão SD para armazenamento;
- Uma pilha 9V como fonte;

- O sensor MPU-6050;
- Cartão SD;

É importante notar que o Arduino foi escolhido apenas por conveniência, e que pode ser substituído por qualquer microcontrolador com ponto flutuante que cumpra os requisitos elétricos do projeto. O código utilizado nesse circuito estão na Listagem A.1 do Apêndice A.

O circuito foi utilizado para coletar dados nos seguintes casos de teste:

1. Reta no asfalto devagar com baixa precisão do acelerômetro e giroscópio;
2. Reta no asfalto rápido com baixa precisão do acelerômetro e giroscópio;
3. Curva no asfalto devagar com baixa precisão do acelerômetro e giroscópio;
4. Curva no asfalto rápido com baixa precisão do acelerômetro e giroscópio;
5. Reta no chão devagar com baixa precisão do acelerômetro e giroscópio;
6. Reta no chão rápido com baixa precisão do acelerômetro e giroscópio;
7. Curva no chão devagar com baixa precisão do acelerômetro e giroscópio;
8. Curva no chão rápido com baixa precisão do acelerômetro e giroscópio;
9. Reta no asfalto devagar com alta precisão do acelerômetro e giroscópio;
10. Reta no asfalto rápido com alta precisão do acelerômetro e giroscópio;
11. Curva no asfalto devagar com alta precisão do acelerômetro e giroscópio;
12. Curva no asfalto rápido com alta precisão do acelerômetro e giroscópio;
13. Reta no chão devagar com alta precisão do acelerômetro e giroscópio;
14. Reta no chão rápido com alta precisão do acelerômetro e giroscópio;
15. Curva no chão devagar com alta precisão do acelerômetro e giroscópio;
16. Curva no chão rápido com alta precisão do acelerômetro e giroscópio;

As medições obtidas são armazenadas em arquivos separados para cada caso de teste no cartão de memória SD. Graças ao DMP do sensor, descrito na Seção 3.1, podemos obter aproximadamente as medidas já contabilizando com a rotação do sistema durante o processo, simplificando os cálculos. É esperado que as medidas necessitem de alguma calibração e sofram com desvio/erro de medição.

3.3 Cálculo da posição por integração

Em ambos os casos de integração será integrada a aceleração em cada eixo duas vezes para obter o deslocamento do sistema em relação a sua posição inicial, e integrado as leituras do giroscópio para obter a orientação do objeto em relação a sua orientação inicial, conforme descrevemos na Seção 2.1. É esperado que essas integrações acumulem mais erros por não ter nenhum filtro ou medida para reduzir o erro da medida ou de integração.

A integração trapezoidal descrita na Seção 2.3.1 será realizada através da função do Matlab *cumtrapz* (MATHWORKS, 2022), enquanto a integração por triângulos mostrado na Seção 2.3.2 será feita no Matlab através do código mostrado na Listagem A.4 do Anexo A.

3.4 Cálculo da posição usando filtros de Kalman

Será utilizado o Filtro de Kalman explicado na Seção 2.4 para integrar as medidas e reduzir o erro, utilizando as formulas da Seção 2.1 de forma matricial. O Filtro foi modelado de forma a prever as medições e suas integrais e comparar as medições previstas com as reais, utilizando o ruído de observação para que após a comparação, o estado registrado tenha um erro menor que o erro esperado na medição real.

O código utilizado para o Filtro de Kalman Unidimensional no Matlab está na Listagem A.2 do Anexo A. Uma conversão do código de Matlab para C na Listagem A.4 do Anexo A como referência de como esse filtro pode ser aplicado em tempo real em um microcontrolador com ponto flutuante.

4 Resultados

Aqui será mostrado e discutido os resultados obtidos a cada passo do projeto.

Tabela 5 – Resumo dos casos apresentados na Seção 3.2.

| Caso | Descrição |
|------|---|
| 1 | Reta no asfalto devagar com baixa precisão |
| 2 | Reta no asfalto rápido com baixa precisão |
| 3 | Curva no asfalto devagar com baixa precisão |
| 4 | Curva no asfalto rápido com baixa precisão |
| 5 | Reta no chão devagar com baixa precisão |
| 6 | Reta no chão rápido com baixa precisão |
| 7 | Curva no chão devagar com baixa precisão |
| 8 | Curva no chão rápido com baixa precisão |
| 9 | Reta no asfalto devagar com alta precisão |
| 10 | Reta no asfalto rápido com alta precisão |
| 11 | Curva no asfalto devagar com alta precisão |
| 12 | Curva no asfalto rápido com alta precisão |
| 13 | Reta no chão devagar com alta precisão |
| 14 | Reta no chão rápido com alta precisão |
| 15 | Curva no chão devagar com alta precisão |
| 16 | Curva no chão rápido com alta precisão |

4.1 Melhores casos

Nas Figuras 7, 8, 9 e 10 é possível ver o comportamento esperado do Filtro de Kalman, principalmente como ele age para reduzir o ruído, reduzindo picos destoantes e harmonizando as retas dos gráficos, especialmente na Figura 10.

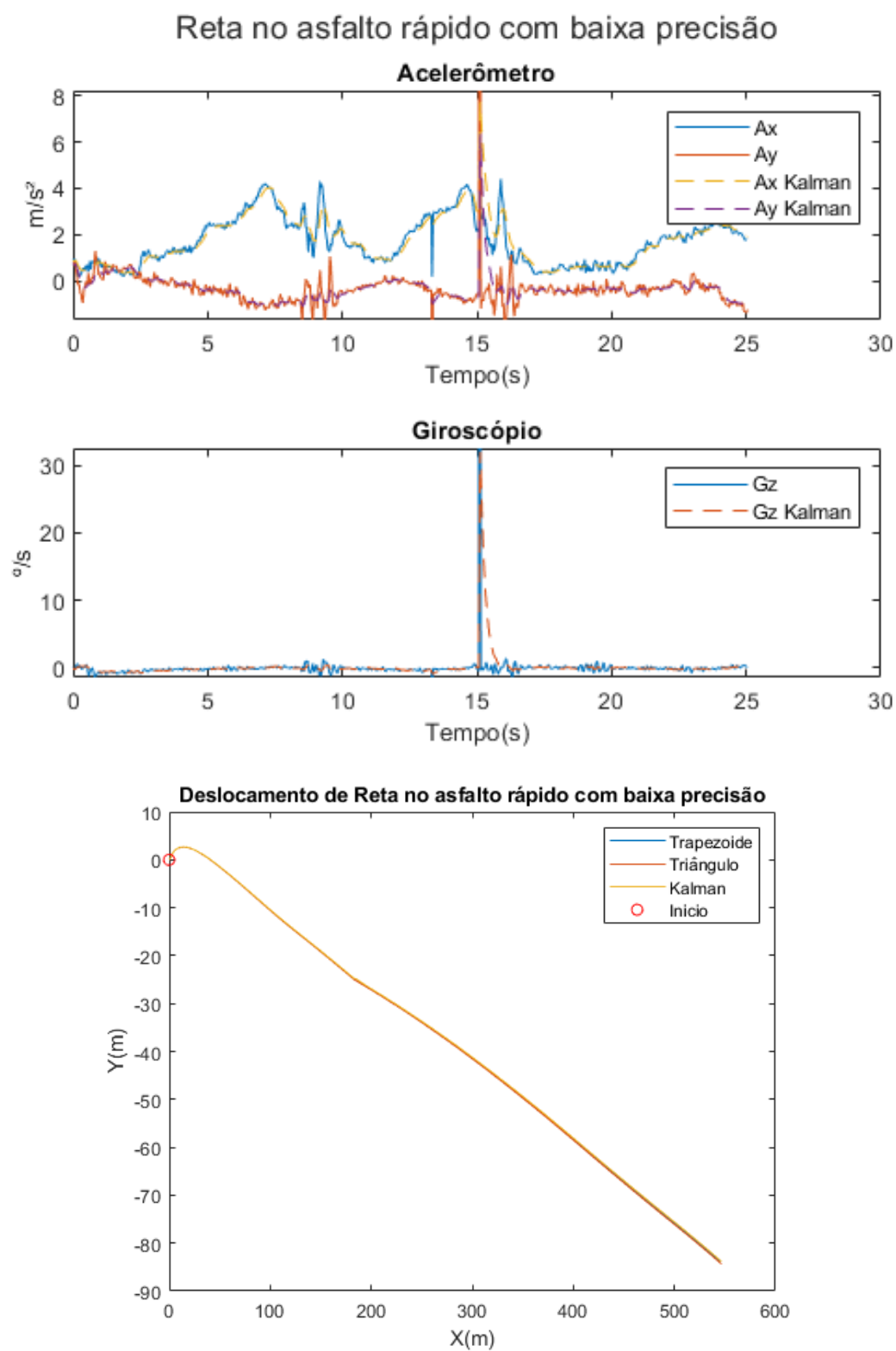


Figura 7 – Análise do Caso 2.

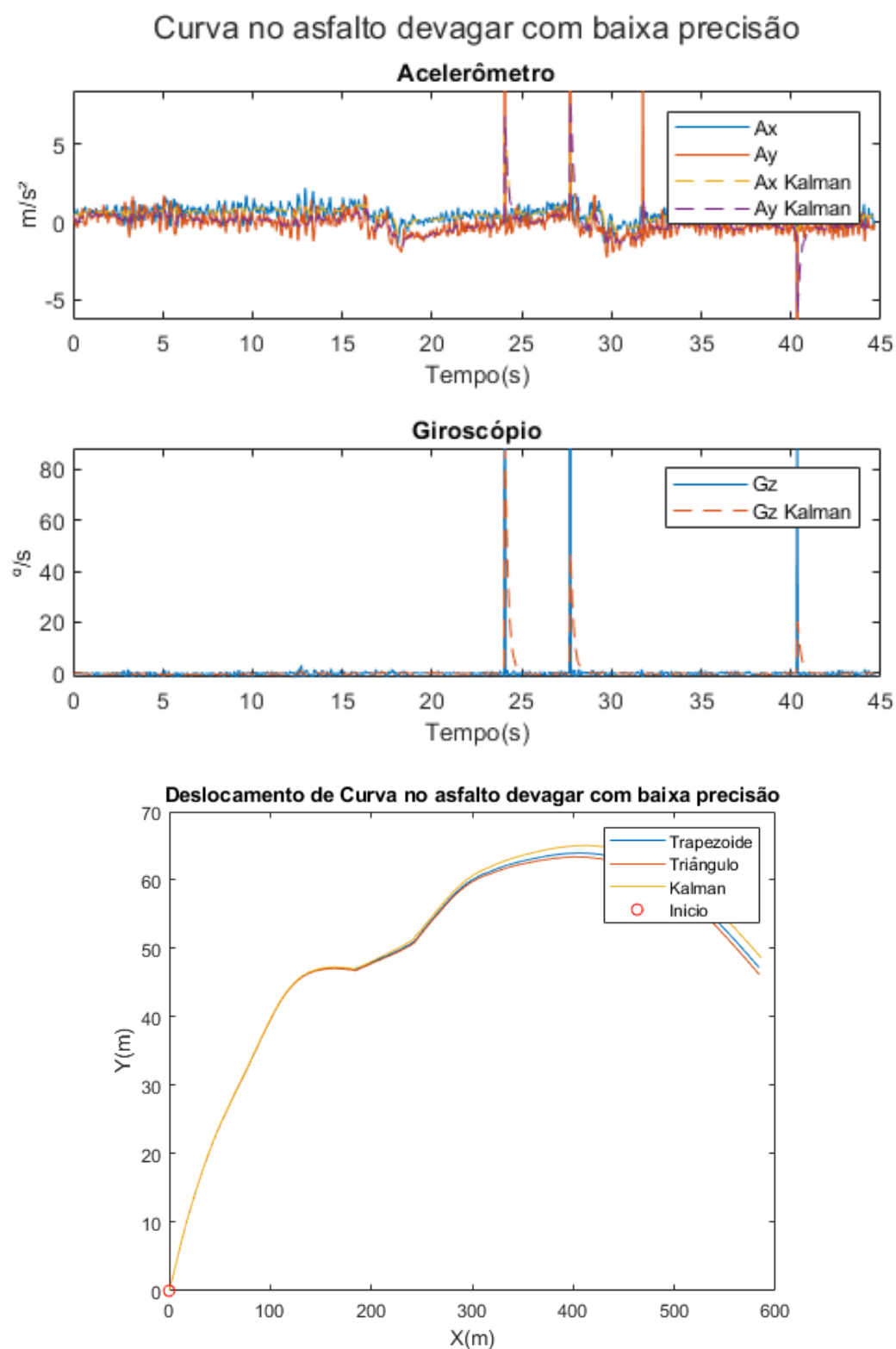


Figura 8 – Análise do Caso 3.

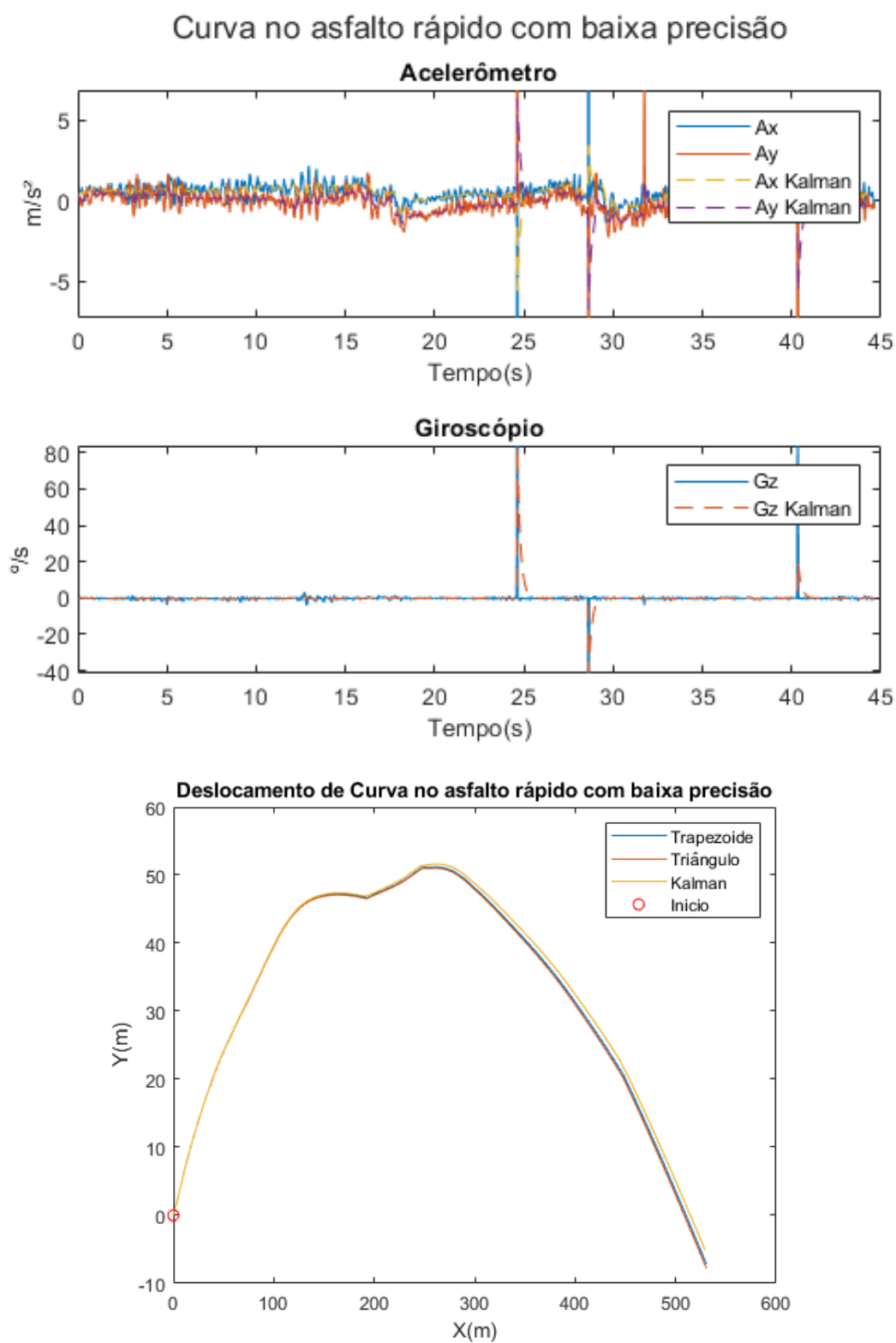


Figura 9 – Análise do Caso 4.

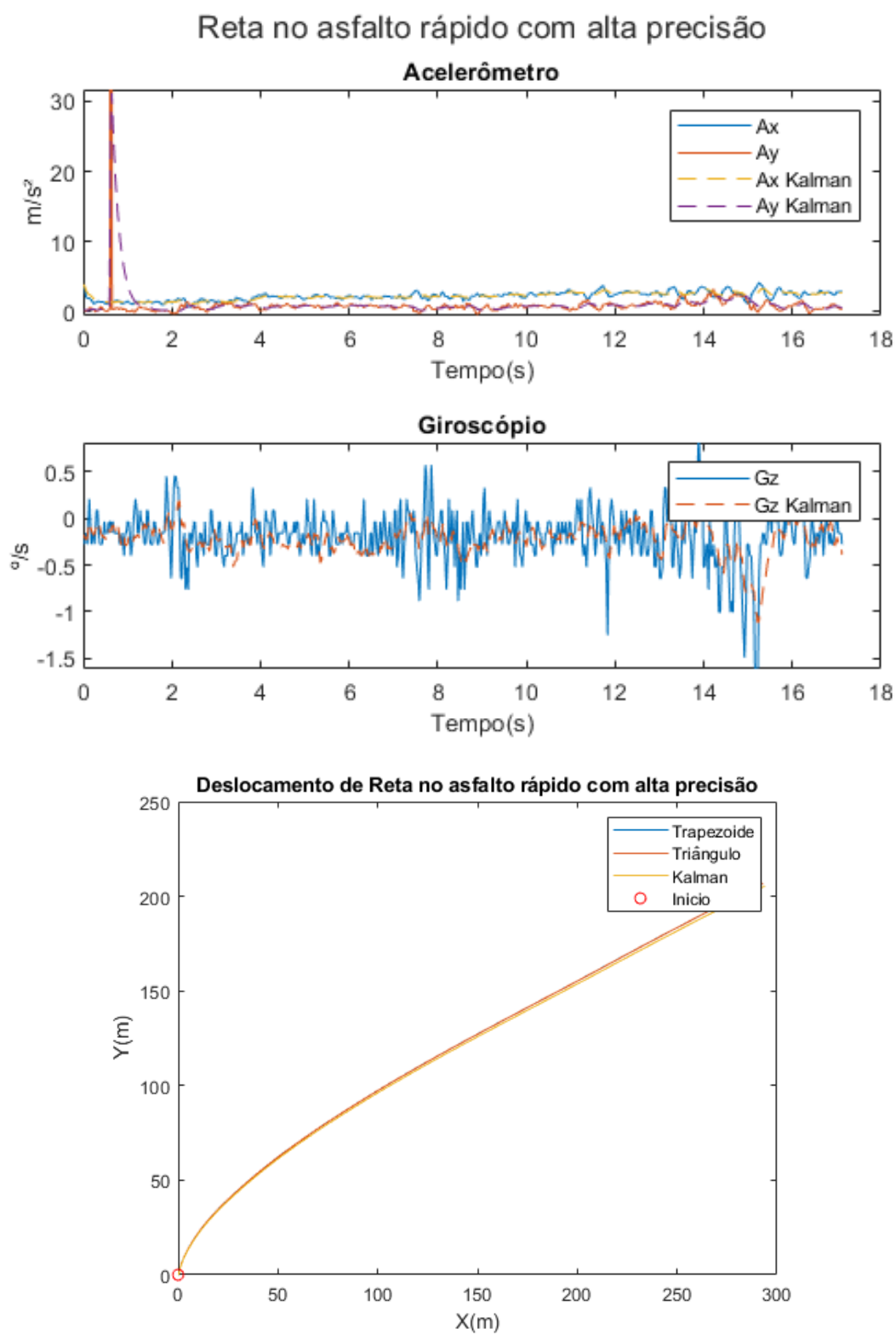


Figura 10 – Análise do Caso 10.

4.2 Tabelas de resultados

Como podemos ver nas Tabelas 6, 7 e 8 a aplicação do filtro de Kalman teve grande efeito em diminuir o desvio padrão (STD) e a variância, sem mudar muito a média dos dados originais. Com isso o Filtro consegue amortecer os erros de medição do sensor, como fica especialmente claro na Figura 10.

A estimação da distância na Tabela 9 deu valores maiores do que era esperado (300 a 500 metros para cada caso), mas como os três métodos resultaram em valores similares, pode se atribuir essa variação a erros durante a medição, não a erros dos métodos de integração. Diferente dos outros, no entanto o Filtro de Kalman conseguiu além de integrar o deslocamento, reduzir o ruído presente na medição original, como é possível ver na Seção 4.1, sendo então a melhor opção em geral.

Tabela 6 – Comparação dos dados antes e depois do Filtro de Kalman para o Acelerômetro.

| Caso | Média Ax (m/s ²) | Média Ax Kalman | STD Ax | STD Ax Kalman | Média Ay (m/s ²) | Média Ay Kalman | STD Ay | STD Ay Kalman |
|------|---------------------------------|--------------------|--------|------------------|---------------------------------|--------------------|---------|------------------|
| 1 | 2.1220 | 2.1276 | 0.6173 | 0.5876 | -0.2675 | -0.2731 | 0.3456 | 0.3154 |
| 2 | 1.7329 | 1.7538 | 1.4605 | 1.0754 | -0.3298 | -0.3149 | 1.4972 | 0.5943 |
| 3 | 0.5053 | 0.5050 | 1.2127 | 0.5202 | -0.0846 | -0.0840 | 1.8213 | 0.7873 |
| 4 | 0.4536 | 0.4433 | 1.2141 | 0.4708 | -0.1415 | -0.1534 | 1.8156 | 0.710 |
| 5 | 2.3722 | 2.3720 | 0.6365 | 0.4049 | -1.2775 | -1.2936 | 1.3900 | 0.9821 |
| 6 | 1.6261 | 1.6345 | 1.7559 | 1.0071 | -2.4145 | -2.4043 | 1.8289 | 1.1011 |
| 7 | -0.3991 | -0.4261 | 1.4414 | 1.0208 | -1.5334 | -1.5778 | 2.6035 | 2.3767 |
| 8 | -0.1539 | -0.1746 | 1.6648 | 1.3102 | -1.9500 | -1.9603 | 1.9854 | 1.3090 |
| 9 | -1.7845 | -1.7861 | 0.4547 | 0.3985 | -2.7311 | -2.7426 | 10.1704 | 3.2708 |
| 10 | 2.2372 | 2.2505 | 0.5962 | 0.4980 | 1.2328 | 1.1801 | 10.7364 | 2.4977 |
| 11 | -1.0951 | -1.0883 | 1.0409 | 1.0307 | -2.6819 | -2.6772 | 3.2063 | 1.3852 |
| 12 | -0.1695 | -0.1002 | 9.1457 | 3.1769 | -3.3820 | -3.4299 | 4.0216 | 1.5292 |
| 13 | -0.4485 | -0.4051 | 9.3274 | 3.5731 | -0.8162 | -0.6864 | 11.3755 | 4.20581 |
| 14 | -2.2519 | -2.2814 | 0.6026 | 0.4785 | -1.4136 | -1.4316 | 0.5144 | 0.3583 |
| 15 | 0.0366 | 0.0148 | 0.7664 | 0.7014 | -1.6327 | -1.6631 | 3.8447 | 1.3840 |
| 16 | -0.7503 | -0.7682 | 1.0150 | 0.9722 | -1.5765 | -1.5633 | 7.8562 | 2.8546 |

Tabela 7 – Comparação dos dados antes e depois do Filtro de Kalman para o Giroscópio.

| Caso | Média Gz (deg/s) | Média Gz Kalman | STD Gz | STD Gz Kalman |
|------|------------------------|-----------------------|-----------|---------------------|
| 1 | 0.0222 | 0.0161 | 0.2296 | 0.1592 |
| 2 | 0.0522 | 0.0972 | 6.4951 | 2.2962 |
| 3 | 0.5019 | 0.6467 | 15.2720 | 5.3810 |
| 4 | 0.1406 | 0.1465 | 15.2914 | 4.7181 |
| 5 | 0.6425 | 0.6397 | 5.3792 | 1.8924 |
| 6 | 0.7668 | 0.8881 | 21.3074 | 6.4665 |
| 7 | -0.1058 | -0.1453 | 13.9473 | 4.7637 |
| 8 | 1.3198 | 1.3089 | 8.4777 | 3.1773 |
| 9 | -0.1813 | -0.1938 | 1.4583 | 0.4500 |
| 10 | -0.1841 | -0.1918 | 0.2846 | 0.1501 |
| 11 | 0.8129 | 0.8279 | 1.3809 | 0.8236 |
| 12 | 4.9322 | 4.2200 | 106.5994 | 21.9780 |
| 13 | 5.1015 | 1.6765 | 146.0136 | 18.7571 |
| 14 | -0.5928 | -0.5941 | 0.3509 | 0.1511 |
| 15 | 0.7765 | 0.7625 | 1.3409 | 0.8702 |
| 16 | 0.8489 | 0.8472 | 1.7699 | 1.5266 |

Tabela 8 – Melhora da Variância com Filtro de Kalman.

| Caso | Variância Ax | Redução da Var. Ax com Kalman(%) | Variância Ay | Redução da Var. Ay com Kalman(%) | Variância Gz | Redução da Var. Gz com Kalman(%) |
|------|-----------------|---|-----------------|---|-----------------|---|
| 1 | 0.3810 | 9.4079 | 0.1195 | 16.6556 | 0.0527 | 51.4081 |
| 2 | 2.1332 | 45.7553 | 2.2415 | 84.2350 | 42.1866 | 87.5607 |
| 3 | 1.4707 | 81.5954 | 3.3172 | 81.3201 | 233.2333 | 87.5776 |
| 4 | 1.4740 | 84.9797 | 3.2964 | 84.6926 | 233.8278 | 90.4231 |
| 5 | 0.4051 | 59.5163 | 1.9321 | 50.0875 | 28.9354 | 87.5317 |
| 6 | 3.0830 | 67.0887 | 3.3449 | 63.7465 | 454.0057 | 90.7860 |
| 7 | 2.0775 | 49.8614 | 6.7784 | 16.6602 | 194.5279 | 88.3450 |
| 8 | 2.7716 | 38.0528 | 3.9420 | 56.5372 | 71.8712 | 85.8900 |
| 9 | 0.2068 | 23.2441 | 103.4379 | 89.6569 | 2.1266 | 90.6041 |
| 10 | 0.3554 | 30.0729 | 115.2713 | 94.5879 | 0.0810 | 66.6492 |
| 11 | 1.0834 | 1.9352 | 10.2806 | 81.3366 | 1.9068 | 64.0807 |
| 12 | 83.6439 | 87.9337 | 16.1734 | 85.5427 | 1.1363e+04 | 95.7517 |
| 13 | 87.0002 | 85.3244 | 129.4012 | 86.3311 | 2.1320e+04 | 98.3516 |
| 14 | 0.3631 | 37.0158 | 0.2646 | 51.4830 | 0.1231 | 86.2514 |
| 15 | 0.5873 | 16.2317 | 14.7818 | 87.0437 | 1.7980 | 58.9155 |
| 16 | 1.0301 | 8.2190 | 61.7200 | 86.7968 | 3.1325 | 27.0880 |

Tabela 9 – Comparações da Estimação da Distância.

| Caso | Distância por Trapz. (m) | Distância por Triâng. (m) | Distância por Kalman (m) |
|------|-----------------------------|------------------------------|-----------------------------|
| 1 | 1.4529e+03 | 1.4557e+03 | 1.4550e+03 |
| 2 | 552.9186 | 554.3824 | 553.9364 |
| 3 | 594.7302 | 594.9029 | 596.8336 |
| 4 | 546.8125 | 546.8800 | 545.2621 |
| 5 | 3.7090e+03 | 3.7174e+03 | 3.7148e+03 |
| 6 | 2.1868e+03 | 2.1954e+03 | 2.1905e+03 |
| 7 | 1.1168e+03 | 1.1190e+03 | 1.1200e+03 |
| 8 | 2.7520e+03 | 2.7589e+03 | 2.7646e+03 |
| 9 | 1.5803e+03 | 1.5831e+03 | 1.5716e+03 |
| 10 | 363.8710 | 364.6418 | 364.3848 |
| 11 | 3.3769e+03 | 3.3833e+03 | 3.3852e+03 |
| 12 | 2.7306e+03 | 2.7349e+03 | 2.7342e+03 |
| 13 | 897.1738 | 899.0061 | 899.3049 |
| 14 | 66.0393 | 66.5385 | 66.6447 |
| 15 | 1.1610e+03 | 1.1628e+03 | 1.1625e+03 |
| 16 | 2.1402e+03 | 2.1433e+03 | 2.1424e+03 |

5 Conclusões e Trabalhos Futuros

Foi possível concluir que o Filtro de Kalman é uma ótima opção para reduzir os ruídos nas medições e realizar integrações com resultados muito similares aos esperados de outros métodos em tempo real, desde que seja possível amostrar regularmente a uma taxa muito menor que a taxa de mudança do sistema e que o ruído possa ser assumido Gaussiano(MA'ARIF et al., 2019). Sobre o sensor MPU-6050 (INVENSENSE; CT, 2020) é um sensor comum de baixo custo interessante, principalmente com o DMP para auxiliar nos cálculos, mas a configuração e utilização desse recurso não é tão clara quanto deveria e a leitura e conversão dos dados para medidas reais poderia ser mais prática.

Como trabalho futuro é proposto esclarecer os motivos pelos quais as medidas de alguns dos casos não saíram como esperado e a aplicação do Filtro de Kalman multidimensional, sem ter a rotação realizada pelo próprio sensor.

Referências

- ANMOLIO. *Anmolio/Mpu6050-Datalogging*. 2016. Disponível em: <<https://github.com/anmolio/mpu6050-datalogging>>. Citado na página 34.
- BOTTEMA, O.; ROTH, B. *Theoretical kinematics*. [S.l.]: Courier Corporation, 1990. v. 24. Citado na página 13.
- BROM, P. C. Integração numérica por soma de áreas de triângulos: Regra dos triângulos repetidos. *REVISTA EIXO*, v. 2, n. 1, p. 53–68, 2013. Citado 3 vezes nas páginas 6, 11 e 15.
- FILE:INTEGRATION num trapezes notation.svg. 2009. Disponível em: <https://commons.wikimedia.org/wiki/File:Integration_num_trapezes_notation.svg>. Citado 2 vezes nas páginas 6 e 15.
- INVENSENSE, T.; CT, P. C. T. Mpu-6050. *TDX Invensense*, 2020. Citado 5 vezes nas páginas 10, 11, 17, 19 e 31.
- LAGES, W. F. Filtro de kalman. *Apostila Curso de Controle Digital*, v. 7, 2005. Citado 3 vezes nas páginas 14, 16 e 18.
- LAURIE, D. P. Practical error estimation in numerical integration. *Journal of Computational and Applied mathematics*, Elsevier, v. 12, p. 425–431, 1985. Citado na página 14.
- MA'ARIF, A. et al. Kalman filter for noise reducer on sensor readings. *Signal and Image Processing Letters*, v. 1, n. 2, p. 50–61, 2019. Citado 4 vezes nas páginas 10, 11, 31 e 39.
- MANKAR, J. et al. Review of i2c protocol. *International Journal of Research in Advent Technology*, Citeseer, v. 2, n. 1, 2014. Citado na página 19.
- MATHWORKS. *cumtrapz: Cumulative trapezoidal numerical integration*. MATHWORKS, 2022. Disponível em: <<https://www.mathworks.com/help/matlab/ref/cumtrapz.html>>. Citado na página 22.
- MATHWORKS. *Estimate position and orientation of a ground vehicle*. 2022. Disponível em: <<https://www.mathworks.com/help/fusion/ug/estimate-position-and-orientation-of-a-ground-vehicle.html>>. Citado na página 11.
- ROWBERG, J. I2cdevlib. mpu-6050 6-axis accelerometer/gyroscope. *Publicación electrónica: http://www.i2cdevlib.com/devices/mpu6050 Consultada*, v. 12, n. 01, 2014. Citado na página 20.
- SHARBAFI, M. et al. *MRL Team Description 2010*. [S.l.]: Islamic Azad University of Qazvin, 2010. Citado 2 vezes nas páginas 6 e 16.

Apêndices

APÊNDICE A – Códigos Utilizados

Nesse apêndice são detalhados os códigos utilizados no projeto.

Listagem A.1 – Código de armazenamento das medições do sensor ([ANMOLIO, 2016](#)).

```

1 #include "I2Cdev.h"
2 #include <SPI.h>
3 #include <SD.h>
4 #include "MPU6050_6Axis_MotionApps20.h"
5 // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
6 // is used in I2Cdev.h
7 #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
8 #include "Wire.h"
9 #endif
10
11
12 // class default I2C address is 0x68
13 // specific I2C addresses may be passed as a parameter here
14 // AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
15 // AD0 high = 0x69
16 MPU6050 mpu;
17
18 #define INTERRUPT_PIN 2 // use pin 2 on Arduino Uno & most boards
19 #define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
20 #define CS_PIN 4 // cs pin from sd card
21 #define pausepin 8 // button de pause
22 #define configpin 9 // button de config gyro e accel
23 #define rotapin 7 // button de rota
24
25
26 uint8_t gyro_config = 0x00;
27 uint8_t accel_config = 0x00;
28 uint8_t rota = 0;
29 char nome_arquivo[20]; //nome do arquivo de texto
30
31 // MPU control/status vars
32 bool dmpReady = false; // set true if DMP init was successful
33 uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
34 uint8_t devStatus; // return status after each device operation (0 = success
    , !0 = error)
35 uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
36 uint16_t fifoCount; // count of all bytes currently in FIFO
37 uint8_t fifoBuffer[64]; // FIFO storage buffer
38
39 // orientation/motion vars
40 VectorInt16 aa; // [x, y, z] accel sensor measurements
41 VectorInt16 gg; // [x, y, z] gyro sensor measurements
42 int bpress[6]; // status atual do button e status anterior button
43 unsigned long miliseconds, start;
44 bool blinkState = false;
45 bool pause = true, novo=true;
46 String dataString;

```

```

47 File dataFile;
48
49 // =====
50 // ===== INTERRUPT DETECTION ROUTINE =====
51 // =====
52
53 volatile bool mpuInterrupt = false;    // indicates whether MPU interrupt pin
    has gone high
54 void dmpDataReady() {
55     mpuInterrupt = true;
56 }
57
58 // =====
59 // ===== INITIAL SETUP =====
60 // =====
61
62
63 void refresh(){ //atualiza o nome do arquivo
64     mpu.setFullScaleGyroRange(gyro_config);
65     mpu.setFullScaleAccelRange(accel_config);
66     sprintf(nome_arquivo, "G%uA%uR%u.txt", mpu.getFullScaleGyroRange(), mpu.
        getFullScaleAccelRange(), rota);
67     Serial.println(nome_arquivo);
68 }
69
70
71
72 void buttons(){ //le os buttons
73     //atualiza estados
74     bool mudou = false;
75     bpress[1]=bpress[0];
76     bpress[3]=bpress[2];
77     bpress[5]=bpress[4];
78     bpress[0]=digitalRead(pausepin);
79     bpress[2]=digitalRead(configpin);
80     bpress[4]=digitalRead(rotapin);
81     if( !bpress[1] && bpress[0]){
82         pause=!pause;
83         if(!pause){ //comecou de novo
84             Serial.print("Recomecou ");
85             novo=true;
86             refresh();
87         }else Serial.println("Parou.");
88     }
89     if(pause){ //le os outros quando pausado
90         if( !bpress[3] && bpress[2] ){
91             if(gyro_config <3){
92                 gyro_config=0x03;
93             }else {
94                 gyro_config=0x00;
95             }
96             if(accel_config <3){
97                 accel_config=0x03;
98             }else accel_config= 0x00;
99
100             refresh();
101         }

```

```

102     if( !bpress[5] && bpress[4] ){
103         if(rota <7){
104             rota++;
105         }else rota=0;
106         refresh();
107     }
108 }
109 }
110
111 void setup() {
112
113     // configure LED for output
114     pinMode(LED_PIN, OUTPUT);
115     pinMode(pausepin, INPUT);
116     pinMode(configpin, INPUT);
117     pinMode(rotapin, INPUT);
118     digitalWrite(LED_PIN,HIGH);
119
120     // join I2C bus (I2Cdev library doesn't do this automatically)
121     #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
122         Wire.begin();
123         Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having
            compilation difficulties
124     #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
125         Fastwire::setup(400, true);
126     #endif
127
128     // initialize serial communication
129     // (115200 chosen because it is required for Teapot Demo output, but it's
130     // really up to you depending on your project)
131     Serial.begin(9600);
132     while (!Serial); // wait for Leonardo enumeration, others continue
            immediately
133
134     // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
135     // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
136     // the baud timing being too misaligned with processor ticks. You must use
137     // 38400 or slower in these cases, or use some kind of external separate
138     // crystal solution for the UART timer.
139
140     // initialize device
141     Serial.println(F("Initializing I2C devices..."));
142     mpu.initialize();
143     pinMode(INTERRUPT_PIN, INPUT);
144
145     // verify connection
146     Serial.println(F("Testing device connections..."));
147     Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F(
        "MPU6050 connection failed"));
148
149     // wait for ready
150     Serial.println(F("\n Press leftmost button to start: "));
151     while(pause){
152         buttons();
153     };
154     // load and configure the DMP
155     Serial.println(F("Initializing DMP..."));

```

```

156     devStatus = mpu.dmpInitialize();
157
158
159
160
161     // see if the card is present and can be initialized:
162     if (!SD.begin(CS_PIN)) {
163         Serial.println("Card failed, or not present");
164         // don't do anything more:
165         return;
166     }
167     Serial.println("card initialized.");
168     if (SD.exists(nome_arquivo)) {
169         SD.remove(nome_arquivo);
170     }
171
172     // supply your own gyro offsets here, scaled for min sensitivity
173     mpu.setXAccelOffset(-1604);
174     mpu.setYAccelOffset(-1057);
175     mpu.setZAccelOffset(358); // 1688 factory default for my test chip
176     mpu.setXGyroOffset(111);
177     mpu.setYGyroOffset(98);
178     mpu.setZGyroOffset(-64);
179     // make sure it worked (returns 0 if so)
180     if (devStatus == 0) {
181         // Calibration Time: generate offsets and calibrate our MPU6050
182         mpu.CalibrateAccel(6);
183         mpu.CalibrateGyro(6);
184         mpu.PrintActiveOffsets();
185         // turn on the DMP, now that it's ready
186         Serial.println(F("Enabling DMP..."));
187         mpu.setDMPEnabled(true);
188
189         // enable Arduino interrupt detection
190         Serial.print(F("Enabling interrupt detection (Arduino external interrupt
191             "));
192         Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
193         Serial.println(F(")..."));
194         attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady,
195             RISING);
196         mpuIntStatus = mpu.getIntStatus();
197
198         // set our DMP Ready flag so the main loop() function knows it's okay to
199         use it
200         Serial.println(F("DMP ready! Waiting for first interrupt..."));
201         dmpReady = true;
202
203         // get expected DMP packet size for later comparison
204         packetSize = mpu.dmpGetFIFOPacketSize();
205     } else {
206         // ERROR!
207         // 1 = initial memory load failed
208         // 2 = DMP configuration updates failed
209         // (if it's going to break, usually the code will be 1)
210         Serial.print(F("DMP Initialization failed (code "));
211         Serial.print(devStatus);
212         Serial.println(F(")"));

```

```

210     }
211     refresh();
212 }
213
214 // =====
215 // ===== MAIN PROGRAM LOOP =====
216 // =====
217
218 void loop() {
219     buttons();
220     if(pause){ //parado esperando button
221         return;
222     }
223     if(novo){ //novo comeco
224         novo=!novo;
225         start=millis();
226         refresh();
227     }
228     // if programming failed, don't try to do anything
229     if (!dmpReady) return;
230     // read a packet from FIFO
231     if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
232         //RAW READINGS for config accuracy
233         mpu.dmpGetAccel(&aa, fifoBuffer);
234         mpu.dmpGetGyro(&gg, fifoBuffer);
235
236         // blink LED to indicate activity
237         blinkState = !blinkState;
238         digitalWrite(LED_PIN, blinkState);
239         ////////////////////////////////////
240         // make a string for assembling the data to log:
241         dataString = "";
242
243         dataString+=String( aa.x )+" ";
244         dataString+=String( aa.y )+" ";
245         dataString+=String( aa.z )+" ";
246         dataString+=String( gg.x )+" ";
247         dataString+=String( gg.y )+" ";
248         dataString+=String( gg.z )+" ";
249         dataString += String(millis()-start);
250
251         // open the file. note that only one file can be open at a time,
252         // so you have to close this one before opening another.
253
254         dataFile = SD.open(nome_arquivo, FILE_WRITE);
255         Serial.println(dataString);
256         // if the file is available, write to it:
257         if (dataFile) {
258             dataFile.println(dataString);
259             dataFile.close();
260
261         } else {// if the file isn't open, pop up an error:
262             Serial.println("error opening datalog.txt");
263         }
264     }
265 }
266 }

```

Listagem A.2 – Código do Filtro de Kalman unidimensional no Matlab (MA'ARIF et al., 2019).

```

1 function dados=KalmanAccelC(dT,tamanho,f)
2 %kalman 1d com integração
3   %inicializa
4   %xk=[a;v;p]
5   Xk=zeros(3,tamanho);
6   Xk_prev=zeros(3,1);
7   X_pred =Xk_prev;
8   H=[1,0,0];
9   A=[1,0,0;dT,1,0;dT^2/2, dT,1];
10  Pk_prev=eye(3);
11  I=eye(3);
12  vvar=400*9.81*1e-6 *sqrt(dT);
13  wvar=[vvar;vvar*dT;vvar*dT^2/2];
14  n=3;
15  Q=wvar*wvar.' * dT;
16  R=vvar^2;
17  for i=1:tamanho
18      z=f(i);
19      v=randn(1) * vvar;
20      w=randn(n,1) .* wvar;
21      %predicao
22      X_pred=A*Xk_prev+w;
23      P_pred=A*Pk_prev*(A.')+Q;
24      %atualização
25      Kk=P_pred*(H.)/(H*P_pred*(H.')+R);
26      Xk(:,i)=X_pred+Kk*(z-H*X_pred - v);
27      Pk=(I-Kk*H)*P_pred*((I-Kk*H).')+Kk*R*(Kk. ');
28      Xk_prev=Xk(:,i);
29      %anteriores
30      Pk_prev=Pk;
31  end
32  dados=array2table(Xk.', 'VariableNames',{ 'Accel', 'Velocidade', 'Deslocamento' })
33  end

```

Listagem A.3 – Código da Listagem A.2 convertido para C.

```

1 /*
2  * File: KalmanAccelC.c
3  *
4  * MATLAB Coder version      : 5.2
5  * C/C++ source code generated on : 10-Mar-2022 13:26:23
6  */
7
8 /* Include Files */
9 #include "KalmanAccelC.h"
10 #include "KalmanAccelC_data.h"
11 #include "KalmanAccelC_emxutil.h"
12 #include "KalmanAccelC_initialize.h"
13 #include "KalmanAccelC_types.h"
14 #include "randn.h"
15 #include <math.h>
16 #include <string.h>

```



```

17
18 /* Function Definitions */
19 /*
20  * kalman 1d com integração
21  * incializa
22  * xk=[a;v;p]
23  *
24  * Arguments      : double dT
25  *                  double tamanho
26  *                  const emxArray_real_T *f
27  *                  table *dados
28  * Return Type    : void
29  */
30 void KalmanAccelC(double dT, double tamanho, const emxArray_real_T *f,
31                  table *dados)
32 {
33     static const signed char b_I[9] = {1, 0, 0, 0, 1, 0, 0, 0, 1};
34     static const signed char b[3] = {1, 0, 0};
35     static const signed char iv[3] = {1, 0, 0};
36     emxArray_real_T *Xk;
37     emxArray_real_T *x;
38     double A[9];
39     double Pk[9];
40     double Q[9];
41     double b_A[9];
42     double dv[9];
43     double X_pred[3];
44     double Xk_prev[3];
45     double w[3];
46     double wvar[3];
47     double A_tmp;
48     double R;
49     double d;
50     double v;
51     double vvar;
52     double y;
53     int b_A_tmp;
54     int b_i;
55     int i;
56     int il;
57     int k;
58     if (!isInitialized_KalmanAccelC) {
59         KalmanAccelC_initialize();
60     }
61     emxInit_real_T(&Xk, 2);
62     i = Xk->size[0] * Xk->size[1];
63     Xk->size[0] = 3;
64     il = (int)tamanho;
65     Xk->size[1] = (int)tamanho;
66     emxEnsureCapacity_real_T(Xk, i);
67     Xk_prev[0] = 0.0;
68     A[0] = 1.0;
69     Xk_prev[1] = 0.0;
70     A[3] = 0.0;
71     Xk_prev[2] = 0.0;
72     A[6] = 0.0;
73     A[1] = dT;

```

```

74  A[4] = 1.0;
75  A[7] = 0.0;
76  A_tmp = dT * dT;
77  A[2] = A_tmp / 2.0;
78  A[5] = dT;
79  A[8] = 1.0;
80  memset(&Pk[0], 0, 9U * sizeof(double));
81  vvar = 0.003923999999999995 * sqrt(dT);
82  wvar[0] = vvar;
83  wvar[1] = vvar * dT;
84  wvar[2] = vvar * A_tmp / 2.0;
85  for (k = 0; k < 3; k++) {
86      Pk[k + 3 * k] = 1.0;
87      Q[3 * k] = vvar * wvar[k];
88      Q[3 * k + 1] = wvar[1] * wvar[k];
89      Q[3 * k + 2] = wvar[2] * wvar[k];
90  }
91  for (i = 0; i < 9; i++) {
92      Q[i] *= dT;
93  }
94  R = vvar * vvar;
95  for (b_i = 0; b_i < il; b_i++) {
96      v = randn() * vvar;
97      b_randn(w);
98      /* predicao */
99      for (i = 0; i < 3; i++) {
100         A_tmp = w[i] * wvar[i];
101         w[i] = A_tmp;
102         y = 0.0;
103         for (b_A_tmp = 0; b_A_tmp < 3; b_A_tmp++) {
104             k = i + 3 * b_A_tmp;
105             y += A[k] * Xk_prev[b_A_tmp];
106             b_A[k] = (A[i] * Pk[3 * b_A_tmp] + A[i + 3] * Pk[3 * b_A_tmp + 1]) +
107                     A[i + 6] * Pk[3 * b_A_tmp + 2];
108         }
109         X_pred[i] = y + A_tmp;
110     }
111     for (i = 0; i < 3; i++) {
112         A_tmp = b_A[i];
113         y = b_A[i + 3];
114         d = b_A[i + 6];
115         for (b_A_tmp = 0; b_A_tmp < 3; b_A_tmp++) {
116             k = i + 3 * b_A_tmp;
117             Pk[k] =
118                 ((A_tmp * A[b_A_tmp] + y * A[b_A_tmp + 3]) + d * A[b_A_tmp + 6]) +
119                 Q[k];
120         }
121     }
122     /* atualizaçao */
123     A_tmp = 0.0;
124     for (i = 0; i < 3; i++) {
125         A_tmp += ((Pk[3 * i] + 0.0 * Pk[3 * i + 1]) + 0.0 * Pk[3 * i + 2]) *
126                 (double)iv[i];
127     }
128     y = A_tmp + R;
129     A_tmp = 0.0;
130     for (i = 0; i < 3; i++) {

```

```

131     Xk_prev[i] = ((Pk[i] + Pk[i + 3] * 0.0) + Pk[i + 6] * 0.0) / y;
132     A_tmp += (double)b[i] * X_pred[i];
133 }
134 A_tmp = (f->data[b_i] - A_tmp) - v;
135 for (i = 0; i < 3; i++) {
136     Xk->data[i + 3 * b_i] = X_pred[i] + Xk_prev[i] * A_tmp;
137     k = b[i];
138     b_A[3 * i] = (double)b_I[3 * i] - Xk_prev[0] * (double)k;
139     b_A_tmp = 3 * i + 1;
140     b_A[b_A_tmp] = (double)b_I[b_A_tmp] - Xk_prev[1] * (double)k;
141     b_A_tmp = 3 * i + 2;
142     b_A[b_A_tmp] = (double)b_I[b_A_tmp] - Xk_prev[2] * (double)k;
143 }
144 for (i = 0; i < 3; i++) {
145     A_tmp = b_A[i];
146     y = b_A[i + 3];
147     d = b_A[i + 6];
148     for (b_A_tmp = 0; b_A_tmp < 3; b_A_tmp++) {
149         dv[i + 3 * b_A_tmp] =
150             (A_tmp * Pk[3 * b_A_tmp] + y * Pk[3 * b_A_tmp + 1]) +
151             d * Pk[3 * b_A_tmp + 2];
152     }
153 }
154 for (i = 0; i < 3; i++) {
155     A_tmp = dv[i];
156     y = dv[i + 3];
157     d = dv[i + 6];
158     for (b_A_tmp = 0; b_A_tmp < 3; b_A_tmp++) {
159         Pk[i + 3 * b_A_tmp] = (A_tmp * b_A[b_A_tmp] + y * b_A[b_A_tmp + 3]) +
160             d * b_A[b_A_tmp + 6];
161     }
162 }
163 for (i = 0; i < 3; i++) {
164     b_A[3 * i] = Xk_prev[0] * R * Xk_prev[i];
165     b_A[3 * i + 1] = Xk_prev[1] * R * Xk_prev[i];
166     b_A[3 * i + 2] = Xk_prev[2] * R * Xk_prev[i];
167 }
168 for (i = 0; i < 9; i++) {
169     Pk[i] += b_A[i];
170 }
171 Xk_prev[0] = Xk->data[3 * b_i];
172 Xk_prev[1] = Xk->data[3 * b_i + 1];
173 Xk_prev[2] = Xk->data[3 * b_i + 2];
174 /* anteriores */
175 }
176 emxInit_real_T(&x, 2);
177 i = x->size[0] * x->size[1];
178 x->size[0] = Xk->size[1];
179 x->size[1] = 3;
180 emxEnsureCapacity_real_T(x, i);
181 k = Xk->size[1];
182 for (i = 0; i < 3; i++) {
183     for (i1 = 0; i1 < k; i1++) {
184         x->data[i1 + x->size[0] * i] = Xk->data[i + 3 * i1];
185     }
186 }
187 emxFree_real_T(&Xk);

```

```

188 k = x->size [0];
189 i = dados->data [0].f1->size [0];
190 dados->data [0].f1->size [0] = x->size [0];
191 emxEnsureCapacity_real_T(dados->data [0].f1 , i);
192 for (i = 0; i < k; i++) {
193     dados->data [0].f1->data [i] = x->data [i];
194 }
195 k = x->size [0];
196 i = dados->data [1].f1->size [0];
197 dados->data [1].f1->size [0] = x->size [0];
198 emxEnsureCapacity_real_T(dados->data [1].f1 , i);
199 for (i = 0; i < k; i++) {
200     dados->data [1].f1->data [i] = x->data [i + x->size [0]];
201 }
202 k = x->size [0];
203 i = dados->data [2].f1->size [0];
204 dados->data [2].f1->size [0] = x->size [0];
205 emxEnsureCapacity_real_T(dados->data [2].f1 , i);
206 for (i = 0; i < k; i++) {
207     dados->data [2].f1->data [i] = x->data [i + x->size [0] * 2];
208 }
209 dados->rowDim.length = x->size [0];
210 dados->varDim.length = 3.0;
211 dados->varDim.hasUnits = false;
212 dados->varDim.units [0].f1.size [0] = 1;
213 dados->varDim.units [0].f1.size [1] = 0;
214 dados->varDim.units [1].f1.size [0] = 1;
215 dados->varDim.units [1].f1.size [1] = 0;
216 dados->varDim.units [2].f1.size [0] = 1;
217 dados->varDim.units [2].f1.size [1] = 0;
218 dados->varDim.hasDescrs = false;
219 dados->varDim.descrs [0].f1.size [0] = 1;
220 dados->varDim.descrs [0].f1.size [1] = 0;
221 dados->varDim.descrs [1].f1.size [0] = 1;
222 dados->varDim.descrs [1].f1.size [1] = 0;
223 dados->varDim.descrs [2].f1.size [0] = 1;
224 dados->varDim.descrs [2].f1.size [1] = 0;
225 dados->varDim.hasContinuity = false;
226 dados->varDim.continuity [0] = unset;
227 dados->varDim.continuity [1] = unset;
228 dados->varDim.continuity [2] = unset;
229 dados->varDim.hasCustomProps = false;
230 dados->metaDim.length = 2.0;
231 dados->arrayProps.Description.size [0] = 1;
232 dados->arrayProps.Description.size [1] = 0;
233 emxFree_real_T(&x);
234 }
235
236 /*
237  * File trailer for KalmanAccelC.c
238  *
239  * [EOF]
240  */

```

```

1 function AT =IntTriang(x,y)
2     %AT = matriz [x2;fx2] da integral triangular acumulativa de
3     %vetores de ponto x e y(x)
4     %como e usado com medidas reais , x deve ser em segundos
5     arguments
6         x (1,:) {mustBeNumeric,mustBeReal}
7         y (1,:) {mustBeNumeric,mustBeReal}
8     end
9     tamanho = length(x);
10    if(tamanho ~= length(y))
11        error("X e Y devem ter o mesmo tamanho!");
12    elseif(tamanho<3)
13        error(" Vetor X muito pequeno!");
14    end
15    n=10; %numero de pontos usado para cada intervalo , 10 tem maior convergencia
16    integral_y = zeros(1,tamanho-1); % integral de cada intervalo
17    integral_x = integral_y; %pontos médios de cada intervalo
18    for i=1:tamanho-1
19        xi=x(i:i+1); %pontos nesse intervalo
20        yi=y(i:i+1);
21        integral_x(i) = mean(xi);
22        integral_y(i) = IterIntTriang(xi,yi,n);
23        if i > 1 %acumulativa
24            integral_y(i) = integral_y(i)+integral_y(i-1);
25        end
26
27    end
28    % como a integração retorna a media entre pontos, vamos ajustar para o
29    % intervalo dado onde x2 sao os pontos médios do intervalo
30    AT = interp1(integral_x,integral_y,x,'pchip','extrap');
31 end
32
33 function AT = IterIntTriang (xn,yn,n)
34 % uma iteracao da integração por triangulos
35 % onde xo =[x0 xf] , fxo = [fx0 fxf] e n > 0
36 arguments
37     xn (1,:) {mustBeNumeric,mustBeReal}
38     yn (1,:) {mustBeNumeric,mustBeReal}
39     n (1,1) {mustBeScalarOrEmpty,mustBePositive}
40 end
41 x=interp1 ([1 n+2],xn,[1:n+2]);
42 y=interp1(xn,yn,x);
43 i = round((n+2)/2); % elemento xi entre x0 e xf escolhido o do meio
44 x0 = [x(1) x(i) x(1)];
45 y0 = [0 0 y(1)];
46 xf = [x(end) x(i) x(end)];
47 yf = [0 0 y(end)];
48 A0 = AreaTriangulo(x0,y0) ; %area incial
49 Af = AreaTriangulo(xf,yf); %area final
50 Ak = 0; %area intermediaria
51 %soma de 1 a n+1 se comeca
52 % com x0, como index matlab x(1) = x0 , x1= x(2)
53 %adicionar triangulos inferiores
54 for k = 1:n+1
55     xk = [x(k) x(i) x(k+1)];
56     yk = [y(k) 0 y(k+1)];
57     Ak = Ak + AreaTriangulo(xk,yk);

```

```
58     end
59     AT = A0 + Ak + Af;
60 end
```