Aplicação de métodos numéricos na integração de dados de uma Unidade de Medição Inercial (IMU)

Helena Moyen, NUSP: 13579422 e Gabriel Vilas, NUSP: 13680150

December 6, 2024

Abstract

O documento apresenta uma introdução e modelagem matemática para uma integração numérica das acelerações lineares e velocidades angulares obtidas a partir de uma IMU.

1 Introdução

Os robôs autônomos têm se tornado um tema cada vez mais relevante, na medida em que a humanidade procura automatizar processos e trabalhos em todos os campos da sociedade. Nesse contexto, um sistema de navegação confiável para os robôs é fundamental para uma automação eficiente e que possa, de fato, subsituir um piloto humano. Nesse sentido, um robô precisa ser equipado por um leque de sensores que permitam uma estimativa de seu estado (posição e orientação) a partir da fusão dos dados entre eles. Um desses componentes sensoriais é a Unidade de Medição Inercial (IMU - Inertial Measurement Unit). Esse dispositivo eletrônico mede e relata as componentes de aceleração resultantes de um corpo, a taxa de variação angular em torno de cada eixo dessas coordenadas e, em alguns casos, o campo magnético que circunda o corpo em questão. Todas essas medidas são feitas por meio dos sensores que compõem o IMU, sendo eles os acelerômetros, giroscópios e os magnetômetros. Veja na figura imu um esquema dos eixos e medições de uma IMU.

As IMU's estão presentes em inúmeras aplicações como submarinos nucleares, aeronaves, mísseis, foguetes espaciais, satélites, carros autônomos, drones e veículos aéreos não tripulados (UAV's). Um IMU permite que o sistema de localização funcione quando os sinais de GPS não estão disponíveis no momento, como acontece em túneis, dentro de edifícios ou na presença de interferência eletrônica.

O processo de navegação inercial em si, consiste em determinar o posicionamento linear e as direções

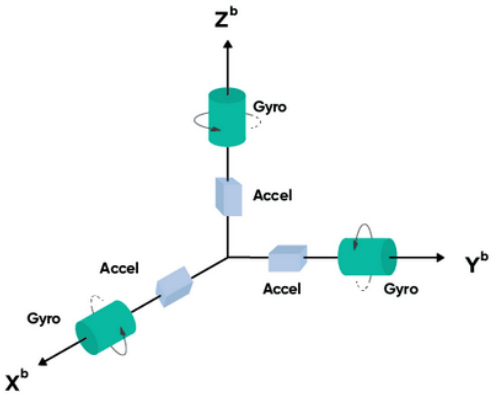

Figure 1: Acelerômetros e giroscópios nos três eixos de movimento em uma IMU. Note a diferença de a 90° entre cada eixo.

angulares utilizando os dados brutos obtidos por esses sensores. A velocidade linear por exemplo, pode ser obtida por meio de uma integração simples das componentes da aceleração. O posicionamento linear por meio da integração dupla dessas mesmas componentes e já o direcionamento, que consta com os três graus de liberdade (pitch, roll e yaw), pode ser obtido por meio da integração das taxas de variação angular lidos pelo giroscópio [3].

Para uma estimativa verossímil do estado do robô, ruídos dos sensores devem ser considerados e uma devida propagação de erros deve ser realizada. Entretanto, como o foco principal do trabalho é a aplicação e análise de métodos númericos, não serão consideradas incertezas na modelagem do problema.

O problema da estimativa da posição do robõ a partir de dados da aceleração e orientação é diferente na medida em que não sabemos o que rege seu movimento. Ou seja, não temos uma equação que defina como a velocidade varia em função da posição ou como a aceleração varia em função da velocidade e do tempo. Tudo isso é mutável, dependendo do ambiente e dos comandos que são dados. Então, esse problema desde o início é numérico: não há resolução analítica, como não temos uma dinâmica do sistema estritamente definida. Sabe-se apenas que a gravidade atua no eixo Z de um robô.

A importância disso é clara: conseguir estimativas do estado, independente das forças atuantes no sistema. Essas estimativas sozinhas, em função das informações da IMU, não são suficientes para ter uma aproximação real do estado do robô, mas em conjunto com dados de outros sensores, a acurácia aumenta. No nosso trabalho, verificaremos o funcionamento do algoritmo numérico definindo uma movimentação específica para o robô, a fim de ter uma solução exata. Assim, poderemos comparar os resultados encontrados com o método número e a partir de uma resolução analítica.

2 Modelagem matemática do problema

Todo objeto que se movimenta no espaço 3D e durante um intervalo de tempo possui algumas variáveis de estado. Primeiramente, temos a posição, velocidade e, potencialmente, uma aceleração linear nos eixos X, Y e Z. Além disso, podemos imaginar que nosso objeto é assimétrico e possui três eixos de rotação também. Ou seja, em cada um dos eixos, ele pode possuir uma direção angular e uma velocidade angular.

Imaginemos que noss robô possua tudo isso. Então, com uma IMU embarcada nele, poderíamos medir suas velocidades angulares e acelerações lineares nos três eixos. Fazendo a relação entre nossas variáveis, podemos pensar que a aceleração linear é a taxa de variação na velocidade linear, assim como a velocidade linear é a taxa de variação da posição. Analogamente, a velocidade angular é a taxa de variação da direção angular. Essa taxa de variação é justamente a definição de uma derivada. Ou seja, temos um sistema matemático com a primeira e segunda derivada da posição, assim como a primeira derivada da direção angular. Isso significa que podemos estimar a posição e os ângulos em cada instante de tempo com os dados da IMU a partir dessas relações. Modelando o sistema como um Problema de Cauchy, obtemos as equações (1) e (2).

$$\begin{cases} \dfrac{dv_1}{dt} = a_1(t), & \dfrac{dv_2}{dt} = a_2(t), & \dfrac{dv_3}{dt} = a_3(t), \\ \dfrac{dx_1}{dt} = v_1(t), & \dfrac{dx_2}{dt} = v_2(t), & \dfrac{dx_3}{dt} = v_3(t), \\ x_1(t_0) = x_{10}, & x_2(t_0) = x_{20}, & x_3(t_0) = x_{30}, \\ v_1(t_0) = v_{10}, & v_2(t_0) = v_{20}, & v_3(t_0) = v_{30}. \end{cases} \tag{1}$$

No primeiro sistema, temos um conjunto de equações diferenciais que descrevem o movimento de um corpo em três dimensões. As variáveis incluem as velocidades $v_1(t)$, $v_2(t)$ e $v_3(t)$, que representam as velocidades nas direções x_1, x_2 e x_3 (ou x, y e z), respectivamente, no tempo t (m/s). As acelerações $a_1(t)$, $a_2(t)$ e $a_3(t)$ descrevem as acelerações nas mesmas direções no tempo t (m/s^2). As posições $x_1(t)$, $x_2(t)$ e $x_3(t)$ representam as posições nas direções x_1, x_2 e x_3 (ou x, y

e z) (m). As posições iniciais x_{10}, x_{20} e x_{30} e as velocidades iniciais v_{10}, v_{20} e v_{30} são especificadas no tempo inicial t_0 (m/s e m, respectivamente).

$$\begin{cases} \dot{\psi}(t) = \omega_2(t)\sin(\phi(t))\sec(\theta(t)) + \omega_3(t)\cos(\phi(t))\sec(\theta(t)), \\ \dot{\theta}(t) = \omega_2(t)\cos(\phi(t)) - \omega_3(t)\sin(\phi(t)), \\ \dot{\phi}(t) = \omega_1(t) + \omega_2(t)\sin(\phi(t))\tan(\theta(t)) + \omega_3(t)\cos(\phi(t))\tan(\theta(t)), \\ \psi(t_0) = \psi_0, \quad \theta(t_0) = \theta_0, \quad \phi(t_0) = \phi_0. \end{cases} \tag{2}$$

No segundo sistema de equações, descreve-se as taxas de mudança dos ângulos de Euler (ψ, θ e ϕ - pitch, roll e yaw) em relação às velocidades angulares ω_1, ω_2 e ω_3 em torno dos eixos x, y e z, respectivamente, em rad/s. As condições iniciais especificadas para os ângulos de Euler (ψ_0, θ_0 e ϕ_0) são fornecidas em radianos no tempo inicial t_0 .

Note que há uma mudança de coordenadas sendo realizada, já que as medições da IMU são feitas com relação ao sistema de coordenadas local do corpo, referentes aos eixos x, y e z do corpo, como visto na Figura 2. Dessa forma, é preciso transformar as velocidades angulares para taxas de mudança dos ângulos de Euler (ϕ, θ e ψ) no sistema de coordenadas global. O mesmo deve ser feito para o primeiro sistema de equação.

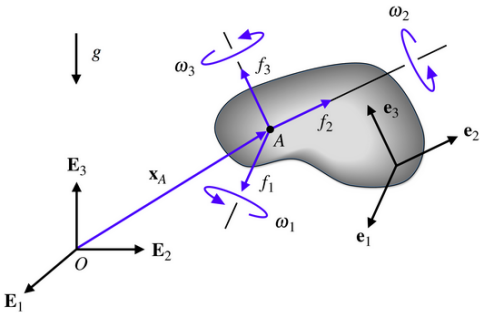

Figure 2: Esquemático de um corpo rígido com um ponto de referência A ao qual um IMU está fixado. Os eixos associados aos acelerômetros e giroscópios de taxa do IMU estão alinhados com a base corotacional e_1, e_2 e e_3 . [5].

Por enquanto, não aplicaremos essa transformação nas acelerações lineares, pois simularemos os dados da IMU já ajustados para as coordenadas globais. Com uma IMU verdadeira, precisaríamos aplicar todas as transformadas de coordenadas, tanto nas velocidades angulares quanto nas acelerações observadas.

Esse equacionamento foi baseado na Revista de Tecnologia de Berkeley [5]. Na Figura 3, é mostrado um esquema resumido do que pretende-se fazer com os dados obtidos na IMU. Veja que a partir dos dados do acelerômetro $f_1(t)$, $f_2(t)$ e $f_3(t)$, obtemos as acelerações lineares no sistema de coordenadas global, após transformá-las e corrigí-las, considerando a gravidade que atua no eixo Z. Por simplificação, não incluímos a gravidade no sistema de equações (1). Isso porque estamos partindo da hipótese que já recebemos as acelerações lineares transformadas e corrigidas com o efeito gravitacional, ou seja, $a_1(t)$, $a_2(t)$ e $a_3(t)$, como visto em (1).

3 Metodologia numérica

Para resolver nossa problemática de estimativa do estado de um objeto a partir dos dados de uma IMU, utilizaremos três técnicas. Uma delas será a Quadratura Numérica, a qual será aplicada na integração numérica das velocidades angulares via a Regra dos Trapézios a fim de estimar a orientação final do objeto em movimento. Nossa segunda técnica escolhida é o Método de Runge Kutta de Terceira Ordem para fazer uma média ponderada dos valores das acelerações lineares. Essa média ponderada será utilizada nas nossas equações de movimento simplificadas

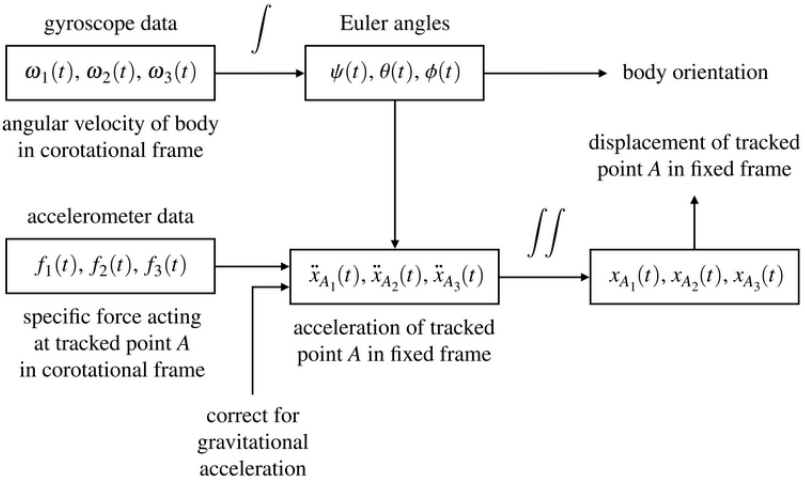

Figure 3: Resumo do processo de integração dos dados de uma IMU [5].

para encontrar, então, uma estimativa da posição, assim como da velocidade linear também. Por fim, utilizaremos a interpolação por splines cúbicos para encontrar uma expressão contínua para a nossa posição, sendo possível ter estimativas no tempo contínua da nossa posição em pontos intermediários e traçar melhor nossa trajetória.

Como nosso sistema é multi-dimensional, vamos descrevê-lo em vetores como visto em 3.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad \boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \tag{3}$$

Assim, temos um vetor $\mathbf{x}$ para a posição, $\mathbf{v}$ para a velocidade linear, $\mathbf{a}$ para a aceleração linear, $\boldsymbol{\omega}$ para a velocidade angular e $\boldsymbol{\alpha}$ para o deslocamento angular em cada eixo (sem aplicação das transformadas de coordenadas). Com a mudança de coordenadas, obteremos as taxas de mudança dos ângulos de Euler (ψ, θ e ϕ).

Portanto, assumindo um modelo de movimento uniformemente variado, ou seja, em que supomos que as acelerações lineares permanecem aproximadamente constantes nos intervalos de medições da IMU, podemos equacionar como descrito em 4, em que discretizamos as medidas de posição e velocidade linear. Isso foi baseado em artigos como [4] que fazem essa simplificação, pois a frequência de medição do IMU é alta suficiente para que possamos assumir um modelo linear sem grande perda de informação. Aqui, $\mathbf{a}_{\text{RK}}$ representa a média ponderada encontrada pela aplicação de Runge Kutta e Δt é o passo de tempo entre as estimativas.

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + \mathbf{v}_n \cdot \Delta t + \frac{\mathbf{a}_{\text{RK}} \cdot \Delta t^2}{2}, \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \mathbf{a}_{\text{RK}} \cdot \Delta t, \\ \boldsymbol{\alpha}_{n+1} &= \boldsymbol{\alpha}_n + \boldsymbol{\omega}_n \cdot \Delta t \end{aligned} \tag{4}$$

3.1 Runge-Kutta

A média ponderada $\mathbf{a}_{\text{RK}}$ será dada pelo equacionamento de Runge Kutta descrito em seguida. O vetor $\mathbf{v}$ é representado por $y(t)$ e $\mathbf{a}$ é sua derivada $f(t, y(t))$. Teremos a formúla diferencial 5.

$$y_{n+1} = y_n + f(t_n, y_n) \cdot \Delta t \tag{5}$$

Na fórmula, $\Delta t = t_{n+1} - t_n$. Selecionaremos três pontos no intervalo $[t_n, t_{n+1}]$, calcularemos as derivadas k_1 a k_4 desses três pontos utilizando a expressão estima pela interpolação, e então

tomaremos a média ponderada dessas derivadas para calcular o próximo estado y_{n+1}. No nosso processo de decisão sobre qual ordem usar no Método de Runge Kutta, analisamos ambas os equacionamentos. A fórmula do algoritmo de Runge-Kutta de Quarta Ordem é a seguinte:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\Delta t \\ &= y_n + k_{\mathrm{RK}}\Delta t \end{aligned} \tag{6}$$

Os pesos são dados por 7.

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{\Delta t}{2}, y_n + k_1\frac{\Delta t}{2}\right) \\ k_3 &= f\left(t_n + \frac{\Delta t}{2}, y_n + k_2\frac{\Delta t}{2}\right) \\ k_4 &= f\left(t_n + \Delta t, y_n + k_3\Delta t\right) \end{aligned} \tag{7}$$

Agora o Runge Kutta de Terceira Ordem 8.

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 4k_2 + k_3)\Delta t \\ &= y_n + k_{\mathrm{RK}}\Delta t \end{aligned} \tag{8}$$

Os pesos são dados por (9).

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{\Delta t}{2}, y_n + k_1\frac{\Delta t}{2}\right) \\ k_3 &= f\left(t_n + \Delta t, y_n - k_1\Delta t + 2k_2\Delta t\right) \end{aligned} \tag{9}$$

A primeira tentativa foi fazer usando Runge Kutta de Quarta Ordem 6, mas observou-se que acabávamos selecionando apenas três pontos. Isso porque como a expressão obtida pela interpolação dependia apenas do tempo. Então, k_2 e k_3 seriam equivalentes no equacionamento 6. Note também que k_2 e k_3 no Runge Kutta de Quarta Ordem seria o equivalente ao k_2 no Runge Kutta de Terceira Ordem. Ou seja, no fim das contas, estaríamos fazendo um Runge Kutta de Terceira Ordem de qualquer forma.

Confirmamos isso quando fizemos uma primeira versão do código utilizando o Runge Kutta de Quarta Ordem, notamos que a ordem de convergência era 3. O que, de fato, é coerente com as explicações acima. Por isso, optamos por usar o Runge Kutta de Terceira Ordem, como o resultado do Runge Kutta de Quarta Ordem não oferecia mais precisão.

Da fórmula (7), podemos ver que o algoritmo integra a informação da derivada dos quatro pontos, e a derivada nos dois pontos médios possui um peso maior. O equacionamento foi tirado do artigo [4], o qual descreve um SLAM (*Simultaneous Localization and Mapping*) aprimorado baseado na fusão de informações visuais e inerciais via o método Runge-Kutta.

O método de Runge-Kutta de 3ª ordem apresenta um erro em cada passo que é dado pelo resto de Lagrange em y_{n+1}, ou seja,

$$e_k = h^4\frac{y^{(4)}(c)}{4!}. \tag{10}$$

Assim sendo, o erro local é de ordem $O(h^4)$. O erro global representa a acumulação dos erros e é dado pelo erro absoluto no último ponto calculado, isto é,

$$E_k = |y_{n+1} - y(t_{n+1})|, \tag{11}$$

e apresenta ordem $O(h^3)$, com solução aproximada igual a y_{n+1} e solução exata igual a $y(t_{n+1})$.

3.2 Quadratura numérica

No nosso trabalho, utilizaremos o método da quadratura numérica para aproximar a integral da função da velocidade angular, a qual só possuímos alguns pontos dados pelas medições da IMU. Dessa forma poderemos calcular o deslocamento angular a cada passo de tempo e estimar a orientação final do objeto. Na equação 12, temos a integral sobre o vetor $\boldsymbol{\omega}$ que resulta nos deslocamentos angulares em cada eixo $\alpha_1, \alpha_2, \alpha_3$. Por enquanto não incluiremos a mudança de coordenadas descritas em 2.

$$\int_{t_0}^{t} \boldsymbol{\omega}(\tau)\, d\tau = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \tag{12}$$

Para solucionar a integral, vamos aplicar a Regra dos Trapézios. Este método segmenta o intervalo de integração em n subintervalos de tamanho uniforme $h = \frac{t-t_0}{n}$. Nos extremos destes subintervalos, conhecemos os valores da função. Em seguida, calculamos a área dos n trapézios formados e este valor aproximado é então considerado como a integral desejada.

$$\int_{t_0}^{t} \boldsymbol{\omega}(\tau)\, d\tau \approx \frac{h}{2}\left(\boldsymbol{\omega}(t_0) + 2\sum_{i=1}^{n-1} \boldsymbol{\omega}(t_i) + \boldsymbol{\omega}(t)\right) \tag{13}$$

O erro desse é dado por 14 e é da ordem $O(h^2)$, onde h é o tamanho do intervalo de integração [2].

$$\text{Erro} = -\frac{(t-t_0)^3}{12n^2}\boldsymbol{\omega}''(\xi), \qquad \xi \in [a, b] \tag{14}$$

Como ξ é um valor desconhecido dentro do intervalo $[t, t_0]$, podemos majorar o erro por 15.

$$\|\text{Erro}\| <= \frac{(t-t_0)^3}{12n^2} \max_{\tau \in [t,t_0]} \boldsymbol{\omega}''(\tau) \tag{15}$$

3.3 Splines cúbicos naturais

Como foi discutido, usaremos a interpolação por splines cúbicos para aproximar a função da posição a partir das estimativas dadas pelo modelo linear usando a média ponderada da aceleração por Runge Kutta. No caso dos splines naturais, assumimos que os polinômios cúbicos são aproximadamente lineares nos extremos do intervalo $[t_0, t_n]$.

$$S''(t_0) = S''(t_n) = 0 \tag{16}$$

A partir dos polinômios encontrados, poderemos depois ilustrar também a trajetória estimada completa em tempo contínuo e compará-la com a exata, simulando, por exemplo, um movimento circular. Na Figura 4, simulamos uma trajetória e fizemos um *dataset* de pontos que seriam, no nosso contexto, as estimativas das posições encontradas pelo equacionamento 4.

Os splines cúbicos se baseiam no cálculo dos coeficientes a_i, b_i, c_i, d_i para cada intervalo de um par de pontos. Para cada dimensão (eixo X, Y e Z) das minhas coordenadas, há um equacionamento de splines distintos para estimar movimentações lineares diferentes.

$$\mathbf{x}(t) = \begin{cases} a_0 + b_0(t-t_0) + c_0(t-t_0)^2 + d_0(t-t_0)^3, & \text{para } t_0 \le t \le t_1 \\ a_1 + b_1(t-t_1) + c_1(t-t_1)^2 + d_1(t-t_1)^3, & \text{para } t_1 \le t \le t_2 \\ \vdots & \\ a_{n-1} + b_{n-1}(t-t_{n-1}) + c_{n-1}(t-t_{n-1})^2 + d_{n-1}(t-t_{n-1})^3, & \text{para } t_{n-1} \le t \le t_n \end{cases} \tag{17}$$

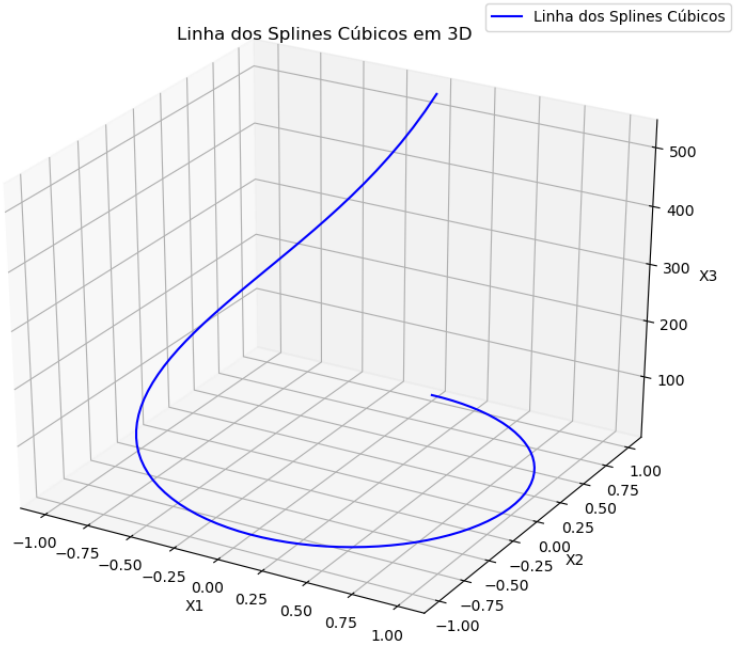

Figure 4: Resultado dos splines cúbicos aproximados para dados simulados das funções $x_1 = sin(t)$, $x_2 = cos(t)$ e $x_3 = e^t$. As expressões dos splines calculados geraram a curva acima.

4 Resultados

Todos os códigos utilizados para obter os resultados estão no Github4.

`https://github.com/hmoyen/numerical-methods/tree/master`

Os resultados apresentados a seguir foram obtidos através dos programas `Quadratura.py` e `RK3_splines.py`, para obter os resultados das posições angulares e das posições lineares, respectivamente.

4.1 Posições e velocidades lineares

A partir do pseudocódigo de splines cúbicas naturais do livro de análise numérica [1], pudemos fazer o código `RK3_splines.py`, o qual combina a interpolação por splines cúbicos, para estimar uma expressão momentânea para as três componenentes da aceleração linear, e o Runge Kutta de Terceira Ordem para aproximar as velocidades e posições lineares. Veja no trecho 4.1 o código em Python do método de calcular coeficientes de splines, baseado no livro [1].

```
1
2     def compute_coefficients(self):
3         # Calcula as diferenças entre os pontos adjacentes em x
4         h = [self.x[i+1] - self.x[i] for i in range(self.n - 1)]
5
6         # Inicializa a lista alpha com zeros
7         alpha = [0] * self.n
8
9         # Calcula os valores de alpha conforme a fórmula do pseudocódigo (Numerical Analysis)
10        for i in range(1, self.n - 1):
11            alpha[i] = 3 * ((self.y[i + 1] - self.y[i]) / h[i] - (self.y[i] - self.y[i - 1])
12            / h[i - 1])
13
```

```
        # Inicializa as listas l, mu e z
        l = [1] + [0] * (self.n - 1)
        mu = [0] * self.n
        z = [0] * self.n

        # Calcula os valores de l, mu e z
        for i in range(1, self.n - 1):
            l[i] = 2 * (self.x[i + 1] - self.x[i - 1]) - h[i - 1] * mu[i - 1]
            mu[i] = h[i] / l[i]
            z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i]

        # Define os valores finais de l e z
        l[self.n - 1] = 1
        z[self.n - 1] = 0

        # Inicializa as listas c, b e d
        c = [0] * self.n
        b = [0] * self.n
        d = [0] * self.n

        # Calcula os valores de c, b e d
        for j in range(self.n - 2, -1, -1):
            c[j] = z[j] - mu[j] * c[j + 1]
            b[j] = (self.y[j + 1] - self.y[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
            d[j] = (c[j + 1] - c[j]) / (3 * h[j])

        # Define os valores de a como os valores de y, exceto o último ponto
        a = self.y[:-1]

        # Retorna os coeficientes calculados
        return a, b, c, d

```

O algoritmo possui uma classe `CubicSpline` a qual é inicializada para cada função de aceleração linear. Quando inicializada, ela recebe uma função `y` e um intervalo de pontos em `x` e calcula os coeficientes dos splines em todos os subintervalos dos pontos dados. Depois, para usar as expressões encontradas no Runge Kutta, chamamos o método `calc_func`, o qual retorna o valor da aceleração estimada naquele ponto a partir das splines dados, como descrito no código 4.1.

```

def runge_kutta(f, x_0, y_0, h):
    """Three step Runge-Kutta method (RK3)
    Solves first order ODEs
    """
    k_0 = f.calc_func(x_0, y_0)
    k_1 = f.calc_func(x_0 + h/2, y_0 + h/2 * k_0)
    k_2 = f.calc_func(x_0 + h, y_0 - h*k_0 + 2*h*k_1)

    k = 1/6 * (k_0 + 4.0*k_1 + k_2)

    x_1 = x_0 + h

    return x_1, k

```

Com a lógica acima, pudemos obter alguns resultados interessantes, manipulando o tamanho do passo `h` tomado.

4.1.1 Análises da solução manufaturada

Para testar o algoritmo, utilizamos a solução manufaturada descrita no equacionamento 19.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -cos(t) \\ -sin(t) \\ e^t \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} sin(t) \\ -cos(t) \\ e^t \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} cos(t) \\ sin(t) \\ e^t \end{bmatrix} \tag{18}$$

Baseado nisso, colocamos as seguintes condições iniciais.

$$\begin{aligned} x_1(0) &= -1, & x_2(0) &= 0, & x_3(0) &= 1, \\ v_1(0) &= 0, & v_2(0) &= -1, & v_3(0) &= 1. \end{aligned} \tag{19}$$

Primeiramente, utilizamos o intervalo de $[0, 3\pi]$ com um passo de $3\pi/50$, o que equivale a uma frequência de aproximadamente 5,3 Hz em uma IMU.

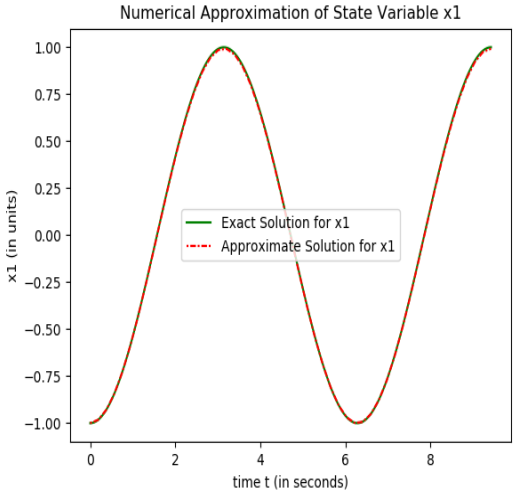

Figure 5: Primeira componente da posição linear

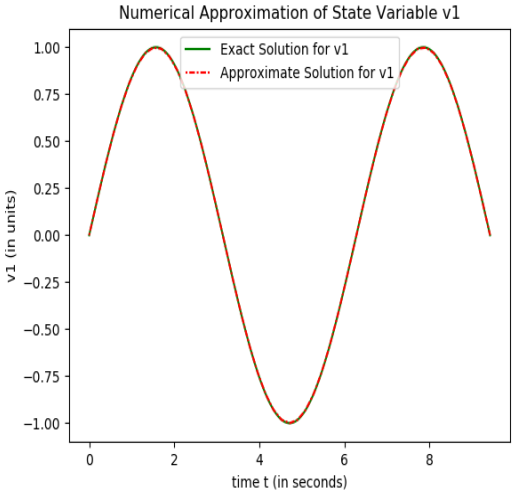

Figure 6: Primeira componente da velocidade linear

Para a primeira componente da posição e da velocidade, os resultados foram precisos para a maior parte do intervalo. Nas tabelas 15 e 16, observam-se o erro de discretização global e em 10 instantes de tempo diferentes. A solução aproximada da posição é dada por x_t, enquanto o valor exato é dado por $x(t)$. O mesmo foi feito para a velocidade.

t	$x_1(t)$	x_{1t}	e
3.769911	0.809017	0.803016	0.006001
3.958407	0.684547	0.678882	0.005665
4.146902	0.535827	0.530570	0.005257
4.335398	0.368125	0.363333	0.004792
4.523893	0.187381	0.183093	0.004289
4.712389	0.000000	-0.003766	0.003766
4.900885	-0.187381	-0.190624	0.003243
5.089380	-0.368125	-0.370864	0.002740
5.277876	-0.535827	-0.538102	0.002275
5.466371	-0.684547	-0.686414	0.001867

Table 1: Erro de discretização global e de x_1

t	$v_1(t)$	v_{1t}	e
3.769911	-0.587785	-0.587957	0.000171
3.958407	-0.728969	-0.729140	0.000171
4.146902	-0.844328	-0.844499	0.000171
4.335398	-0.929776	-0.929948	0.000171
4.523893	-0.982287	-0.982458	0.000171
4.712389	-1.000000	-1.000171	0.000171
4.900885	-0.982287	-0.982458	0.000171
5.089380	-0.929776	-0.929948	0.000171
5.277876	-0.844328	-0.844499	0.000171
5.466371	-0.728969	-0.729140	0.000171

Table 2: Erro de discretização global e de v_1

Resultados análogos foram encontrados para as outras componentes. Todos os gŕaficos de todas as componentes podem ser gerados rodando o código, assim como as tabelas.

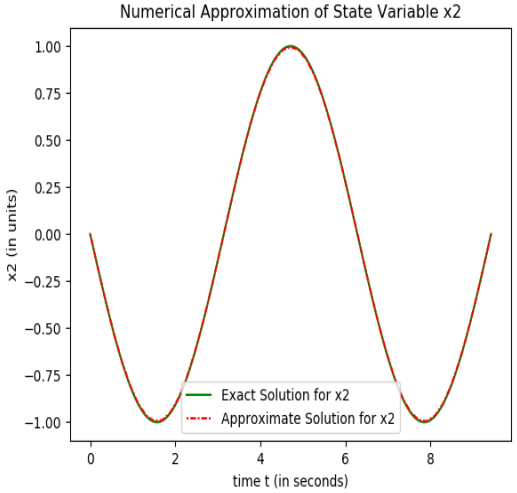

Figure 7: Segunda componente da posição linear

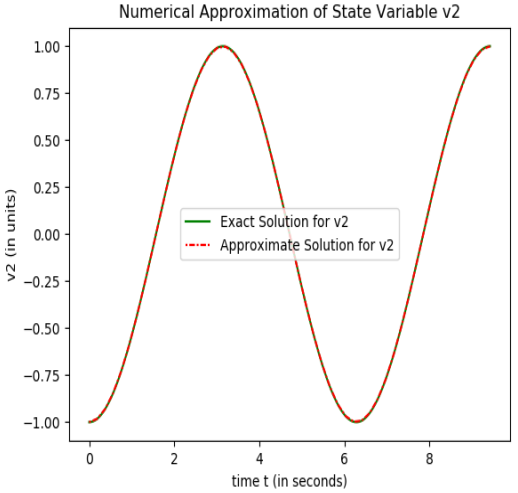

Figure 8: Segunda componente da velocidade linear

t	$x_2(t)$	x_{2t}	e
3.769911	0.587785	0.586036	0.001749
3.958407	0.728969	0.726800	0.002168
4.146902	0.844328	0.841817	0.002511
4.335398	0.929776	0.927012	0.002764
4.523893	0.982287	0.979367	0.002920
4.712389	1.000000	0.997027	0.002973
4.900885	0.982287	0.979366	0.002921
5.089380	0.929776	0.927010	0.002766
5.277876	0.844328	0.841814	0.002514
5.466371	0.728969	0.726796	0.002172

Table 3: Erro de discretização global e de x_2

t	$v_2(t)$	v_{2t}	e
3.769911	0.809017	0.809014	0.000003
3.958407	0.684547	0.684544	0.000003
4.146902	0.535827	0.535824	0.000003
4.335398	0.368125	0.368122	0.000003
4.523893	0.187381	0.187379	0.000003
4.712389	0.000000	-0.000003	0.000003
4.900885	-0.187381	-0.187384	0.000002
5.089380	-0.368125	-0.368127	0.000002
5.277876	-0.535827	-0.535829	0.000002
5.466371	-0.684547	-0.684549	0.000002

Table 4: Erro de discretização global e de v_2

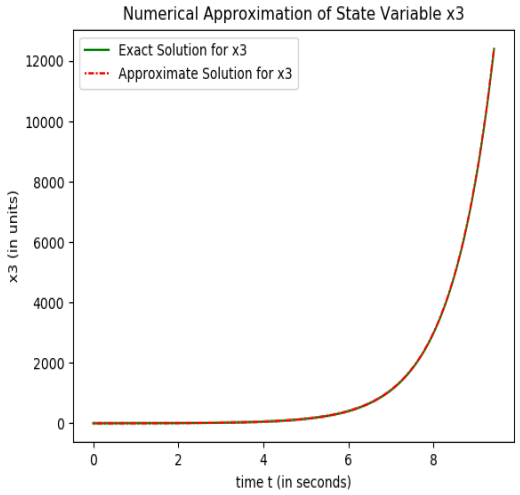

Figure 9: Terceira componente da posição linear

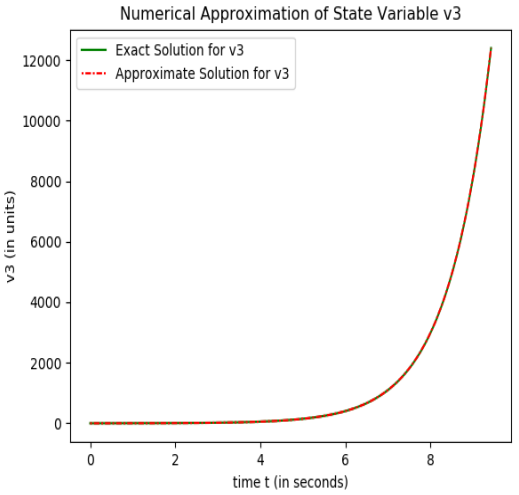

Figure 10: Terceira componente da velocidade linear

t	$x_3(t)$	x_{3t}	e
3.769911	43.37621	43.502180	0.12596
3.958407	52.37381	52.526428	0.15261
4.146902	63.23780	63.422585	0.18478
4.335398	76.35533	76.578945	0.22361
4.523893	92.19385	92.464346	0.27049
4.712389	111.31777	111.644877	0.32709
4.900885	134.40861	134.804053	0.39543
5.089380	162.28922	162.767171	0.47794
5.277876	195.95316	196.530722	0.57756
5.466371	236.60006	237.297898	0.69783

Table 5: Erro de discretização global e de x_3

t	$v_3(t)$	v_{3t}	e
3.769911	43.37621	43.37632	0.000112
3.958407	52.37381	52.37391	0.000104
4.146902	63.23780	63.23789	0.000094
4.335398	76.35533	76.35541	0.000083
4.523893	92.19385	92.19392	0.000071
4.712389	111.31777	111.31783	0.000056
4.900885	134.40861	134.40865	0.000039
5.089380	162.28922	162.28924	0.000018
5.277876	195.95316	195.95315	0.000008
5.466371	236.60006	236.60002	0.000041

Table 6: Erro de discretização global e de v_3

Um ponto a ser melhor analisado é o erro de x_3 em comparação com v_3. Apesar de serem a mesma função, as estimativas se diferenciam e o erro de x_3 é significativamente maior. Isso pode ser explicado pela hipótese de um modelo linear 4.1.1 que assumimos no cálculo da posição final. Ou seja, simplificamos o problema e obtivemos erros maiores por causa disso. Considerando que o custo computacional é menor assumindo o modelo linear, deve-se analisar o custo-benefício dessa abordagem e se valeria a pena aplicar novamente um Runge Kutta para estimar a posição.

Como vimos em artigos [4] que no nosso contexto é comum assumir um modelo linear, mantivemos a abordagem simplificada. Além disso, uma IMU em câmeras de mapeamento, por exemplo, costuma ter frequências que chegam a mais de 100 Hz. Logo, esperemos que com um passo cada vez menor, vamos obter erros menos significativos ao adotar um modelo linear.

```
1
2  def pos_linear_model(p,v,a,delta):
3
4      # p -> posição anterior
5      # v --> velocidade estimada com runge kutta
6      # a --> medida ponderada das acelerações dado por runge kutta
7      # delta --> tempo
8
9      return (p+v*delta+(a*delta**2)*0.5)
10
```

```
11  def vel_linear_model(v,a,delta):
12
13      return (v+a*delta)
14
15
```

A trajetória final estimada no espaço 3D é mostrada no gráfico 11.

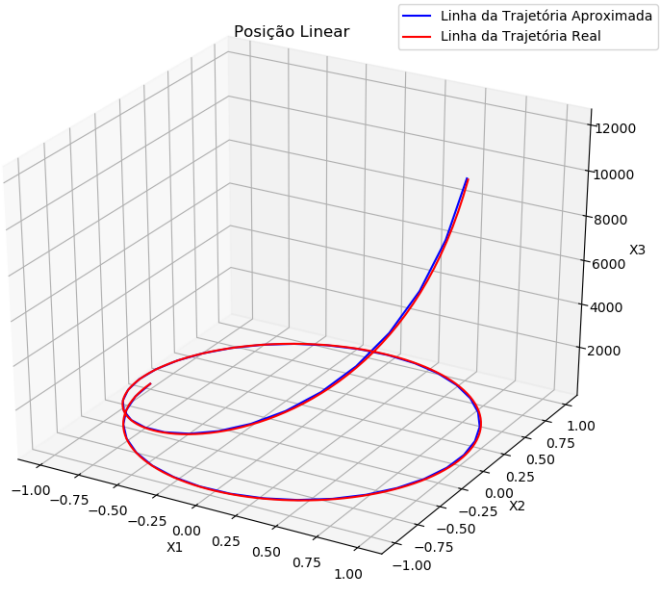

Figure 11: Sobreposição da trajetória estimada e a real para um divisão do intervalo em 50 subintervalos.

Para estimar a convergência do nosso método de Runge Kutta, podemos analisar os resultados aumentando e diminuindo o tamanho do passo. Seja n, o número de divisões de um intervalo que aplicaremos o método e a ordem p do método dada pela equação 20:

$$\text{ordem } p = \log_2 \left(\frac{\text{erro}(2h, t)}{\text{erro}(h, t)}, \right) \tag{20}$$

pudemos montar as tabelas a seguir para todas as posições e velocidades lineares.

n	$h_n = \frac{(\pi/6-0)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
16	3.27249×10^{-2}	3.29916×10^{-6}	——
32	1.63625×10^{-2}	7.90029×10^{-7}	2.05605
64	8.18123×10^{-3}	1.93639×10^{-7}	2.02583
128	4.09062×10^{-3}	4.79534×10^{-8}	2.01298
256	2.04531×10^{-3}	1.19329×10^{-8}	2.00319
512	1.02265×10^{-3}	2.97639×10^{-9}	2.00164
1024	5.11327×10^{-4}	7.43248×10^{-10}	2.00084
2048	2.55663×10^{-4}	1.85706×10^{-10}	2.00042
4096	1.27832×10^{-4}	4.64164×10^{-11}	2.00021
8192	6.39159×10^{-5}	1.15934×10^{-11}	2.00011
16384	3.19579×10^{-5}	2.89557×10^{-12}	2.00005

Table 7: Resultado dos cálculos de erro de x_1 e ordem de convergência para o intervalo de 0 a $\frac{\pi}{6}$.

n	$h_n = \frac{\left(\frac{\pi}{6}-0\right)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
16	3.27249×10^{-2}	2.30983×10^{-5}	——
32	1.63625×10^{-2}	5.77450×10^{-6}	2.00003
64	8.18123×10^{-3}	1.44362×10^{-6}	1.99990
128	4.09062×10^{-3}	3.60905×10^{-7}	1.99977
256	2.04531×10^{-3}	9.02261×10^{-8}	1.99969
512	1.02265×10^{-3}	2.25565×10^{-8}	1.99963
1024	5.11327×10^{-4}	5.63913×10^{-9}	1.99959
2048	2.55663×10^{-4}	1.40979×10^{-9}	1.99957
4096	1.27832×10^{-4}	3.52450×10^{-10}	1.99955
8192	6.39159×10^{-5}	8.80900×10^{-11}	1.99954
16384	3.19579×10^{-5}	2.20023×10^{-11}	1.99953

Table 8: Resultado dos cálculos de erro de x_2 e ordem de convergência para o intervalo de 0 a $\frac{\pi}{6}$.

n	$h_n = \frac{\left(\frac{\pi}{6} - 0\right)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
16	3.27249×10^{-2}	2.69650×10^{-5}	——
32	1.63625×10^{-2}	6.70664×10^{-6}	2.00722
64	8.18123×10^{-3}	1.67280×10^{-6}	2.00448
128	4.09062×10^{-3}	4.17744×10^{-7}	2.00224
256	2.04531×10^{-3}	1.04381×10^{-7}	2.00112
512	1.02265×10^{-3}	2.60883×10^{-8}	2.00056
1024	5.11327×10^{-4}	6.52123×10^{-9}	2.00028
2048	2.55663×10^{-4}	1.63020×10^{-9}	2.00014
4096	1.27832×10^{-4}	4.07530×10^{-10}	2.00007
8192	6.39159×10^{-5}	1.01918×10^{-10}	2.00004
16384	3.19579×10^{-5}	2.54954×10^{-11}	2.00002

Table 9: Resultado dos cálculos de erro de x_3 e ordem de convergência para o intervalo de 0 a $\frac{\pi}{6}$.

n	$h_n = \frac{\left(\frac{\pi}{12} - 0\right)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
16	1.63625×10^{-2}	1.27912×10^{-7}	——
32	8.18123×10^{-3}	1.44901×10^{-8}	3.14201
64	4.09062×10^{-3}	1.72633×10^{-9}	3.06929
128	2.04531×10^{-3}	2.10731×10^{-10}	3.03423
256	1.02265×10^{-3}	2.60329×10^{-11}	3.01699
512	5.11327×10^{-4}	3.23536×10^{-12}	3.00834
1024	2.55663×10^{-4}	4.02345×10^{-13}	3.00742
2048	1.27832×10^{-4}	5.20417×10^{-14}	2.95069

Table 10: Resultado dos cálculos de erro de v_1 e ordem de convergência para o intervalo de 0 a $\frac{\pi}{12}$.

Note que a ordem de convergência de x_1, x_2 e x_3 vale 2. Isso pode ser explicado pelo uso do modelo linear, fazendo com que a ordem seja menor que 3, como seria pelo Runge Kutta de Terceira Ordem utilizado no cálculo das velocidades. Na próxima seção, aprofundaremos melhor a razão disso.

Podemos ver que, para v_1 10 e v_3 12, a ordem de convergência se aproxima de 3 conforme n aumenta, o que está de acordo com o esperado para um método de Runge Kutta de terceira ordem. No entanto, há algumas variações nos valores de p.

Para v_2, ocorreram algumas anomalias na ordem de convergência. O erro de v_2, em geral, é bem menor que o de v_1 e v_3. Observou-se que aumentando muito o n, a ordem p de v_1 e v_3 começava a divergir de 3 também. Por isso, dado que o erro de v_2 caia muito rapidamente, colocamos um intervalo maior para analisar sua ordem de convergência e obtivemos os resultados da tabela 13.

Veja que em 13, a ordem de convergência de v_2 aparenta ser 4, o que é congruente com a reciprocidade do nosso Runge Kutta de Terceira Ordem com um de Quarta Ordem.

Percebemos também que a ordem de convergência de v_3 também pode acabar sendo de Quarta Ordem dependendo do intervalo em análise. Intervalos maiores proporcionam uma ordem de convergência de 4 para v_1 e v_3. Veja, por exemplo, a tabela 14.

n	$h_n = \frac{\left(\frac{\pi}{12} - 0\right)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
16	1.63625×10^{-2}	1.59539×10^{-13}	—
32	8.18123×10^{-3}	4.08562×10^{-14}	1.96528
64	4.09062×10^{-3}	2.77556×10^{-15}	3.8797
128	2.04531×10^{-3}	5.55112×10^{-16}	2.32193

Table 11: Resultado dos cálculos de erro de v_2 e ordem de convergência para o intervalo de 0 a $\frac{\pi}{12}$.

n	$h_n = \frac{\left(\frac{\pi}{12} - 0\right)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
16	1.63625×10^{-2}	1.27877×10^{-7}	—
32	8.18123×10^{-3}	1.44885×10^{-8}	3.14178
64	4.09062×10^{-3}	1.72624×10^{-9}	3.0692
128	2.04531×10^{-3}	2.10725×10^{-10}	3.0342
256	1.02265×10^{-3}	2.60312×10^{-11}	3.01705
512	5.11327×10^{-4}	3.23697×10^{-12}	3.00753
1024	2.55663×10^{-4}	4.03899×10^{-13}	3.00258
2048	1.27832×10^{-4}	4.55191×10^{-14}	3.14945

Table 12: Resultado dos cálculos de erro de v_3 e ordem de convergência para o intervalo de 0 a $\frac{\pi}{12}$.

n	$h_n = \frac{(2\pi - 0)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
32	1.96350×10^{-1}	3.78035×10^{-6}	-
64	9.81748×10^{-2}	2.15971×10^{-7}	4.05848
128	4.90874×10^{-2}	1.29339×10^{-8}	4.05526
256	2.45437×10^{-2}	7.91707×10^{-10}	4.03031
512	1.22718×10^{-2}	4.89694×10^{-11}	4.01522
1024	6.13592×10^{-3}	3.06033×10^{-12}	4.00685
2048	3.06796×10^{-3}	1.96176×10^{-13}	3.96962

Table 13: Resultado dos cálculos de erro de v_2 e ordem de convergência para o intervalo de 0 a 2π.

n	$h_n = \frac{(8\pi - 0)}{n}$	$\lvert e(T, h_n)\rvert$	ordem p
64	3.92699×10^{-1}	4.33174×10^{0}	—
128	1.96350×10^{-1}	2.58563×10^{-1}	4.06636
256	9.81748×10^{-2}	1.56638×10^{-2}	4.04501
512	4.90874×10^{-2}	9.60904×10^{-4}	4.0269
1024	2.45437×10^{-2}	5.93452×10^{-5}	4.01719
2048	1.22718×10^{-2}	3.65754×10^{-6}	4.02019

Table 14: Resultado dos cálculos de erro de v_3 e ordem de convergência para o intervalo de 0 a 8π.

4.2 Posições Angulares

O código Quadraturas.py, utilizado para calcular o valor aproximado da posição angular com base em valores igulamente espaçados no tempo do valor da velocidade angular, fornecidos pela IMU do drone, foi feito de forma simples ao se transformar a equação 14 em linhas de código. Além disso, o progrma permite a um usuário informar o intervalo, o número de amostras e seus valores para que o calculo possa ser realizado. Veja o trecho responsável por calcular o valor final da quadratura 4.2.

```
1
2       # Calcula o valor de h
3       h = (b-a)/n
4
5       # Calcula a somatoria dos valores da função não localizados nos extremos
6       soma = 0
7       for k in range(1, n):
8            soma += (val[k])
9
10      # Multiplica esses valores por dois
11      soma = soma * 2
12
13      # Soma esses valores com os valores nos extremos do intervalo
14      soma = soma +  val[a] + val[n]
15
16      # Multiplica por h/2
17      res = soma * h * 0.5
18
19      print (res)
20      return res
21
```

O erro foi calculado usando o mesmo procedimento da seção 4.1. Abaixo, estão os gráficos gerados da primeira e segunda componentes da posição angular, seguidos por suas tabelas de erro.

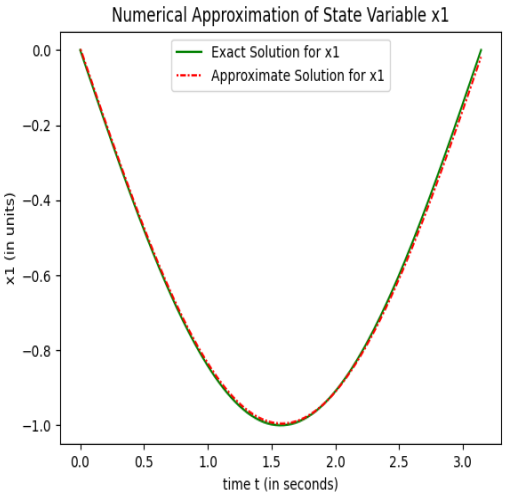

Figure 12: Primeira componente da posição angular

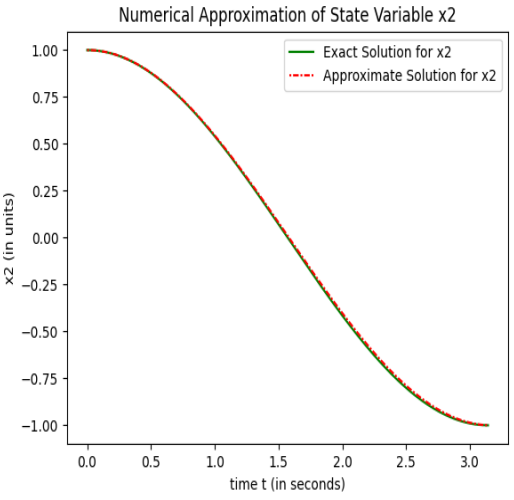

Figure 13: Segunda componente da posição angular

t	$x_1(t)$	x_{1t}	e
1.54011	-0.99952	-0.99459	0.00493
1.54625	-0.99969	-0.99485	0.00487
1.55238	-0.99983	-0.99500	0.00482
1.55852	-0.99992	-0.99515	0.00476
1.56466	-0.99998	-0.99527	0.00470
1.57079	-1.0	-0.99535	0.00464
1.57693	-0.99998	-0.99539	0.00459
1.58306	-0.99969	-0.99539	0.00452
1.58920	-0.99969	-0.99536	0.00446
1.59534	-0.99969	-0.99529	0.00440

Table 15: Erro de discretização global da posição angular 1

t	x_2t	x_{2t}	
1.54011	0.0306	0.0412	0.0103
1.54011	0.0306	0.0412	0.0104
1.54011	0.0306	0.0412	0.0104
1.54011	0.0306	0.0412	0.0106
1.54011	0.0306	0.0412	0.0105
1.54011	0.0306	0.0412	0.0106
1.54011	0.0306	0.0412	0.0107
1.54011	0.0306	0.0412	0.0108
1.54011	0.0306	0.0412	0.0109
1.54011	0.0306	0.0412	0.0107

Table 16: Erro de discretização global da posição angular 2

5 Conclusão

Por fim, pudemos perceber que os resultados obtidos por métodos numéricos são suficientemente eficazes em termos de erro para a apicação proposta, uma vez que conferem resultados com erros pequenos e que não serão acumulados por tempo suficiente para que sejam significativos, tendo em vista a autonomia dos sistemas robóticos atuais. Além disso, percebe-se que dificilmente os métodos aqui citados seriam utilizados em alguma aplicação embarcada real da maneira que foram propostos neste artigo, uma vez que soluções computacionalmente menos demandantes, apesar de apresentarem um menor grau de precisão, são mais vantajosas, considerando a capacidade de processamento limitada de aplicações embarcadas. Em suma, os resultados aqui apresentados poderiam ser aplicados em soluções mais robustas ou utilizados para fins didáticos em laboratórios de robótica embarcada.

References

[1] Richard L. Burden and J. Douglas Faires. Numerical analysis, 2010.

[2] A. F. P. de C. Humes ; I. S. H. de Melo ; L. K. Yoshida ; W. T. Martins. *Noções de Cálculo Numérico*, volume 1, chapter Integração Numérica. McGraw-Hill, 1 edition, 1984.

[3] Vitor Hugo, Romão Machado, Kaio Gonçalves Júnio, Gabriel Dos, and Santos F De Faria. Inertial navigation system based on imu – inertial measurement unit.

[4] Jia shan Cui, Fang rui Zhang, Dong zhu Feng, Cong Li, Fei Li, and Qi chen Tian. An improved slam based on rk-vif: Vision and inertial information fusion via runge-kutta method. *Defence Technology*, 21:133–146, 3 2023.

[5] David Titterton and John Weston. Strapdown inertial navigation technology. *Strapdown Inertial Navigation Technology*, 1 2004.