UFC

Trabalho de Sinais e Sistemas

Vinicius Alexandre Gomes do Nascimento	568594
Matheus Parente Reis	571954
Diego Mendes de Freitas	565585

Conteúdo

1 Introdução **2**
1.1 Processamento do Sinal . . . 2

2 Decomposição **3**
2.1 Fundamentação Teórica e Decomposição . . . 3
2.1.1 Desenvolvimento da Componente Real ($y_R[n]$) . . . 3
2.1.2 Desenvolvimento da Componente Imaginária ($y_I[n]$) . . . 3
2.1.3 Análise de Resultados . . . 4
2.2 Aplicação em Código Python . . . 4

3 Filtragem **5**
3.1 Análise Matemática . . . 5
3.2 Análise Visual e Validação . . . 7
3.2.1 Resposta em Frequência (Domínio da Frequência) . . . 7
3.2.2 Filtragem Espectral . . . 8
3.3 Aplicação em Código Python . . . 8

4 Amostragem e Estimação **9**
4.1 Processo de Amostragem . . . 9
4.1.1 Aplicação em Código Python . . . 9
4.2 Critério de Decisão Euclidiana . . . 9
4.2.1 Aplicação em Código Python . . . 9

5 Resultados **11**
5.1 Diagrama de Blocos . . . 11
5.2 Constelação . . . 11
5.3 Trajetória . . . 13

6 Anexo 1 - Desenvolvimento das Identidades Trigonométricas **14**

7 Anexo 2 - Código Completo **15**
7.1 Código Principal . . . 15
7.2 Outros Gráficos . . . 19

1 Introdução

Este documento descreve o procedimento de demodulação implementado para a identificação de quatro símbolos distintos $\{S_1, S_2, S_3, S_4\}$. Estes símbolos consistem em valores complexos que codificam os vetores de movimentação do drone no plano xy.

Cada m-ésimo símbolo X_m é transmitido no tempo discreto conforme a expressão:

$$x_m[n] = \sqrt{2}\left(\text{Re}\{X_m\}\cos\left(\frac{2\pi n}{N}\right) - \text{Im}\{X_m\}\sin\left(\frac{2\pi n}{N}\right)\right)$$

para o intervalo $N_0 \leq n < N_0 + N$, onde N representa o número de amostras por símbolo e N_0 o instante inicial do intervalo.

O sinal recebido pelo demodulador, $y[n]$, é modelado pela composição do sinal transmitido $x[n]$ (contendo uma sequência de N_s símbolos) e um ruído aditivo $z[n]$:

$$y[n] = x[n] + z[n]$$

1.1 Processamento do Sinal

A extração dos N_s símbolos é realizada através de um processo dividido em três etapas:

- **Decomposição:** O sinal de entrada $y[n]$ é decomposto nas componentes reais e imaginárias dos símbolos. Este processo gera os sinais $y_R[n]$ e $y_I[n]$ através da correlação com as portadoras ortonormais, que carregam majoritariamente a parte real e imaginária dos símbolos.
- **Filtragem:** Aplicação de filtros para a mitigação do ruído $z[n]$ e eliminação de componentes de alta frequência resultantes da demodulação.
- **Amostragem e Estimação:** A partir dos sinais filtrados, o demodulador realiza a amostragem no instante ideal para identificar o símbolo transmitido dentro do espaço de estados. Os símbolos recuperados são então convertidos em coordenadas de comando para o mapeamento da rota do drone.

2 Decomposição

2.1 Fundamentação Teórica e Decomposição

A etapa inicial da demodulação consiste na extração das componentes reais $y_R[n]$ e imaginarias $y_I[n]$ dos símbolos a partir do sinal $y[n]$. Este processo é fundamentado na ortogonalidade das funções seno e cosseno, sendo definido pelas operações:

$$y_R[n] = y[n] \cdot \sqrt{2}\cos\left(\frac{2\pi n}{N}\right) \tag{1}$$

$$y_I[n] = y[n] \cdot -\sqrt{2}\sin\left(\frac{2\pi n}{N}\right) \tag{2}$$

Considerando a análise para um único símbolo X_m no intervalo $N_0 \leq n < N_0 + N$, o sinal recebido é dado por $y[n] = x_m[n] + z[n]$. O desenvolvimento analítico para ambas as componentes é apresentado a seguir:

2.1.1 Desenvolvimento da Componente Real ($y_R[n]$)

Substituindo a expressão de $x_m[n]$ e aplicando a propriedade distributiva, temos:

$$\begin{aligned} y_R[n] &= \left[\sqrt{2}\left(\text{Re}\{X_m\}\cos\left(\frac{2\pi n}{N}\right) - \text{Im}\{X_m\}\sin\left(\frac{2\pi n}{N}\right)\right) + z[n]\right] \cdot \sqrt{2}\cos\left(\frac{2\pi n}{N}\right) \\ &= 2\text{Re}\{X_m\}\cos^2\left(\frac{2\pi n}{N}\right) - 2\text{Im}\{X_m\}\sin\left(\frac{2\pi n}{N}\right)\cos\left(\frac{2\pi n}{N}\right) + z'[n] \end{aligned}$$

Onde $z'[n] = z[n]\sqrt{2}\cos(2\pi n/N)$ é o ruído transladado em frequência. Utilizando as identidades trigonométricas de arco duplo (Anexo 1):

$$\cos^2(\theta) = \frac{1+\cos(2\theta)}{2} \quad \text{e} \quad \sin(\theta)\cos(\theta) = \frac{\sin(2\theta)}{2}$$

Obtém-se:

$$y_R[n] = \text{Re}\{X_m\} + \text{Re}\{X_m\}\cos\left(\frac{4\pi n}{N}\right) - \text{Im}\{X_m\}\sin\left(\frac{4\pi n}{N}\right) + z'[n] \tag{3}$$

2.1.2 Desenvolvimento da Componente Imaginária ($y_I[n]$)

De forma análoga:

$$\begin{aligned} y_I[n] &= \left[\sqrt{2}\left(\text{Re}\{X_m\}\cos\left(\frac{2\pi n}{N}\right) - \text{Im}\{X_m\}\sin\left(\frac{2\pi n}{N}\right)\right) + z[n]\right] \cdot \left(-\sqrt{2}\sin\left(\frac{2\pi n}{N}\right)\right) \\ &= -2\text{Re}\{X_m\}\cos\left(\frac{2\pi n}{N}\right)\sin\left(\frac{2\pi n}{N}\right) + 2\text{Im}\{X_m\}\sin^2\left(\frac{2\pi n}{N}\right) + z''[n] \end{aligned}$$

Aplicando a identidade trigonométrica (Anexo 1) $\sin^2(\theta) = \frac{1-\cos(2\theta)}{2}$:

$$\begin{aligned} y_I[n] &= -2\text{Re}\{X_m\}\left[\frac{1}{2}\sin\left(\frac{4\pi n}{N}\right)\right] + 2\text{Im}\{X_m\}\left[\frac{1-\cos\left(\frac{4\pi n}{N}\right)}{2}\right] + z''[n] \\ &= \text{Im}\{X_m\} - \text{Im}\{X_m\}\cos\left(\frac{4\pi n}{N}\right) - \text{Re}\{X_m\}\sin\left(\frac{4\pi n}{N}\right) + z''[n] \end{aligned}$$

2.1.3 Análise de Resultados

Em ambas as decomposições, observa-se que o resultado é composto por:

1. Um termo continuo que representa a informação original (Re$\{X_m\}$ ou Im$\{X_m\}$);
2. Componentes de alta frequência localizadas no dobro da frequência da portadora ($4\pi/N$);
3. Um termo de ruído aleatório.

Para isolar os valores dos símbolos é necessária a aplicação de um **filtro passa-baixa** em ambas as saídas, eliminando as componentes de alta frequência e atenuando o ruído fora da banda de interesse.

2.2 Aplicação em Código Python

```python
import numpy as np

# Importando os dados do csv
ySignal = np.loadtxt("rx_sgn_y.csv", dtype=complex)

# Total de amostras
total_samples = len(ySignal)

# Especificacao dos parametros dados no projeto
N = 21
Ns = 100
fc = 1000
Ts = 1/ (N*fc)


# Criando vetor de tempo discreto
n = np.arange(total_samples)

# Etapa 1 - Decomposicao do sinal ------------
arg = (2 * np.pi / N) * n

yReal = np.sqrt(2) * np.cos(arg) * ySignal # Sinal contendo componentes
    reais dos simbolos

yImg = -np.sqrt(2) * np.sin(arg) * ySignal # Sinal contendo componentes
    imaginarias dos simbolos
```

Listing 1: Código Decomposição

3 Filtragem

Visando um sistema com resposta finita ao impulso (**FIR**), fácil implementação e maior familiaridade (Lista Computacional 01), optou-se pela utilização do Filtro de Média Móvel que, atenuando os sinais de alta frequência, preserva os dados de interesse com distorções mínimas.

3.1 Análise Matemática

A fim de provar sua eficiência como Filtro Passa-Baixa, segue-se o seguinte passo a passo matemático:

- Sabe-se que a convolução discreta é dada pela forma:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} h[k] \cdot x[n-k]$$

- No caso da média móvel, a resposta ao impulso $h[k]$ vale $1/N$ para as amostras no intervalo de 0 a $N-1$ e 0 para as demais. Portanto:

$$y[n] = \frac{1}{N} \sum_{k=0}^{N-1} x[n-k]$$

- Intuitivamente, considera-se que ruídos de alta frequência serão atenuados, pois variações bruscas são suavizadas pela média. Para comprovar rigorosamente, analisa-se a resposta em frequência.

- A Resposta em Frequência, por sua vez, é obtida pela Transformada de Fourier de Tempo Discreto (DTFT) da resposta ao impulso:

$$H(e^{j\omega}) = \sum_{k=0}^{N-1} h[k] e^{-j\omega k} = \frac{1}{N} \sum_{k=0}^{N-1} (e^{-j\omega})^k$$

- Por se tratar de uma progressão geométrica finita de razão $r = e^{-j\omega}$, aplica a fórmula $S_N = \frac{1-r^N}{1-r}$:

$$H(e^{j\omega}) = \frac{1}{N} \frac{1 - e^{-j\omega N}}{1 - e^{-j\omega}}$$

- Para obter a relação em senos, coloca-se em evidência a exponencial de "meio ângulo"tanto no numerador quanto no denominador ($e^{-j\omega N/2}$ e $e^{-j\omega/2}$, respectivamente):

$$H(e^{j\omega}) = \frac{1}{N} \frac{e^{-j\omega N/2} \left(e^{j\omega N/2} - e^{-j\omega N/2}\right)}{e^{-j\omega/2} \left(e^{j\omega/2} - e^{-j\omega/2}\right)}$$

- Pela identidade de Euler ($2j \sin(\theta) = e^{j\theta} - e^{-j\theta}$), substitui-se os termos entre parênteses:

$$H(e^{j\omega}) = \frac{1}{N} \frac{e^{-j\omega N/2} \cdot 2j \sin(\omega N/2)}{e^{-j\omega/2} \cdot 2j \sin(\omega/2)}$$

- Simplificando os termos $2j$ e agrupando as exponenciais restantes, chega-se à expressão final que evidencia a atenuação e a fase linear:

$$H(e^{j\omega}) = \frac{1}{N}\frac{\sin(N\omega/2)}{\sin(\omega/2)}e^{-j\omega(N-1)/2}$$

Para verificar o comportamento, analisa-se apenas o módulo da resposta em frequência (Ganho):

$$|H(e^{j\omega})| = \left|\frac{1}{N}\frac{\sin(N\omega/2)}{\sin(\omega/2)}\right|$$

A prova de que o sistema é Passa-Baixa se dá pelos seguintes pontos críticos:

1. **Baixas Frequências ($\omega \to 0$):** Utilizando o limite fundamental trigonométrico ($\lim_{x\to 0}\frac{\sin(ax)}{\sin(x)} = a$), temos:

$$\lim_{\omega\to 0}|H(e^{j\omega})| = \frac{1}{N}\cdot\frac{N/2}{1/2} = 1$$

   Isso comprova que frequências muito baixas passam pelo sistema sem sofrer atenuação.

2. **Altas Frequências (ex: $\omega = \pi$):** Na maior frequência possível em tempo discreto (frequência de Nyquist), o termo $\sin(\omega/2)$ no denominador torna-se $\sin(\pi/2) = 1$. O ganho passa a ser limitado por:

$$|H(e^{j\pi})| = \left|\frac{\sin(N\pi/2)}{N}\right| \leq \frac{1}{N}$$

   Este resultado demonstra que a amplitude do sinal nessa frequência é reduzida a uma fração $1/N$ da amplitude original. Em outras palavras, o ruído de alta frequência sofre um **fator de atenuação de N vezes**.

Também demonstrando matematicamente que, quanto maior a ordem N do filtro, maior será a atenuação das altas frequências:

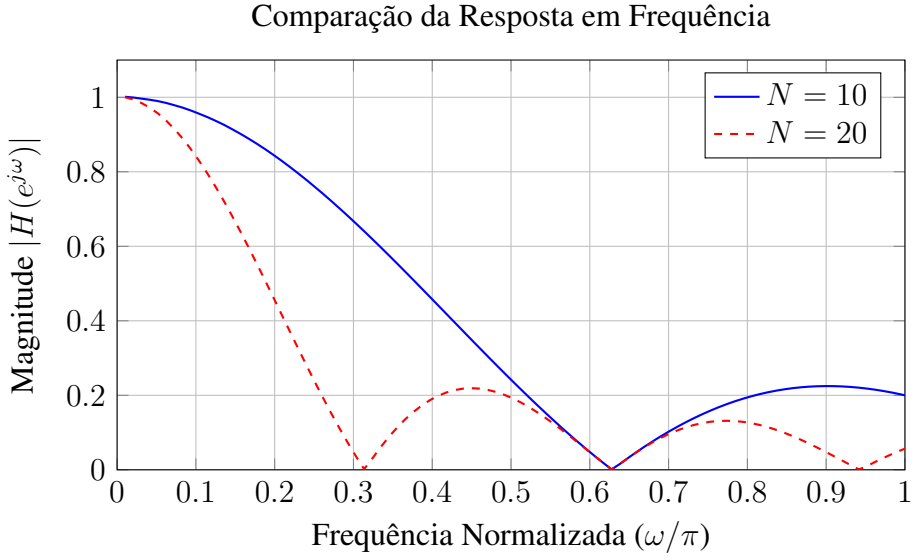

Figura 1: Gráfico da magnitude da resposta em frequência para um filtro de média móvel para tamanhos diferentes de intervalo.

Tendo comprovado a característica Passa-Baixa do filtro de Média Móvel, resta definir a ordem N ideal para a aplicação proposta. Denota-se que, cada símbolo observado na etapa anterior, é composto por uma janela temporal de **21 amostras**. Além disso, as portadoras são definidas com periodicidade ajustada para completar ciclos inteiros dentro desse intervalo.

Para um filtro de média móvel, a anulação completa de uma componente senoidal ocorre quando a janela do filtro abrange exatamente um número inteiro de períodos dessa senoide. A soma de uma senoide pura sobre seu período completo é nula:

$$\sum_{n=0}^{N-1} \cos\left(\frac{2\pi k}{N} n + \phi\right) = 0, \quad \text{para } k \text{ inteiro e não nulo.}$$

Desta forma, optou-se por configurar a ordem do filtro como $N = 21$. Essa escolha alinha a janela de filtragem exatamente com a duração do símbolo, garantindo:

- **Máxima Rejeição:** As componentes de alta frequência são canceladas pela soma, pois completam ciclos inteiros na janela de 21 amostras.
- **Integração Coerente:** O filtro soma toda a energia do sinal DC referente àquele símbolo, maximizando a relação sinal-ruído antes da decisão.

3.2 Análise Visual e Validação

A fim de comprovar as afirmações teóricas e a escolha da ordem do filtro, realizou-se uma análise visual baseada em simulações computacionais utilizando os dados do sinal recebido.

3.2.1 Resposta em Frequência (Domínio da Frequência)

Como a etapa de demodulação (multiplicação pela portadora) gera componentes indesejados no dobro da frequência da portadora ($2\omega_c$), e sabendo que o sistema foi projetado tal que a portadora completa um ciclo a cada N amostras ($\omega_c = \frac{2\pi}{N}$), a interferência ocorre em:

$$2\omega_c = \frac{4\pi}{N}$$

Observa-se no gráfico que este ponto corresponde exatamente ao segundo nulo da função ($k = 2$). Isso valida a escolha de $N = 21$, pois garante que a componente de alta frequência oscilatória seja matematicamente anulada, e não apenas atenuada.

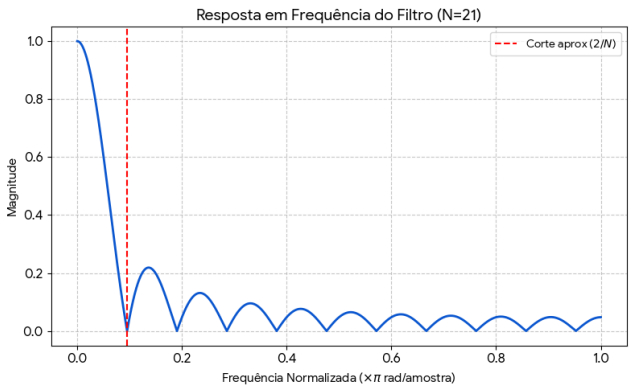

Figura 2: Magnitude da Resposta em Frequência do Filtro ($N = 21$). Note os nulos periódicos que eliminam harmônicos da frequência de amostragem do símbolo.

3.2.2 Filtragem Espectral

Nota-se que o processo de mistura gera réplicas de alta frequência distantes da banda base. O filtro de média móvel atua drasticamente sobre essas componentes, atenuando o ruído e preservando apenas o lóbulo principal em baixas frequências, que contém a informação útil. A ”limpeza”do espectro confirma a característica passa-baixa seletiva do filtro projetado.

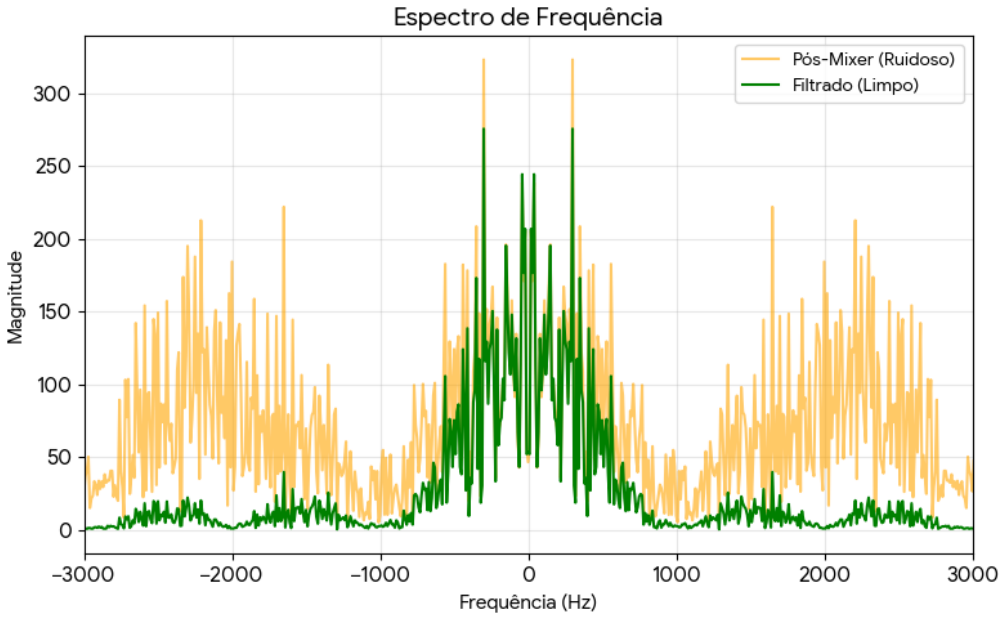

Figura 3: Espectro de Frequência: O sinal ruidoso pós-mixer (laranja) é suavizado pelo filtro (verde), eliminando componentes de alta frequência.

3.3 Aplicação em Código Python

```
# Etapa 2 - Filtro(Media Movel) --------------
h = np.ones(N) / N

# Convolucao
yReal_mm = np.convolve(yReal, h, mode='full')

yImg_mm = np.convolve(yImg, h, mode='full')
```

Listing 2: Código Filtragem

4 Amostragem e Estimação

Nesta etapa final, o sinal filtrado é discretizado e processado para a recuperação da informação original, transformando as sequências temporais em comandos de movimentação.

4.1 Processo de Amostragem

Após a filtragem por média móvel, os sinais $y_{R,mm}[n]$ e $y_{I,mm}[n]$ atingem a estabilidade ao final de cada período de símbolo. A amostragem é realizada de forma sincronizada a cada N amostras, extraindo o valor médio acumulado para compor a estimativa complexa $\hat{X}_m$:

$$\hat{X}_m = y_{R,mm}[idx] + j \cdot y_{I,mm}[idx]$$

Onde o índice de amostragem é definido por $idx = (m \cdot N) + (N - 1)$, para $m = 0, 1, \ldots, N_s - 1$. Este instante de amostragem garante que o filtro tenha integrado toda a energia do símbolo atual, maximizando a relação sinal-ruído antes da tomada de decisão.

4.1.1 Aplicação em Código Python

```python
# Amostragem --------------------------------
simbolos_estimados = []

for m in range(Ns): # Para todos simbolos transmitidos
    idx = (m * N) + (N - 1) # Indice do final de cada periodo de simbolo, que sera amostrado

    val_real = yReal_mm[idx] # Armazena a parte real do simbolo
    val_img = yImg_mm[idx] # Armazena a parte imaginaria do simbolo

    simbolos_estimados.append(val_real + 1j * val_img) # Coloca esse valor num vetor com todos simbolos estimados

simbolos_estimados = np.array(simbolos_estimados) # Converte para um numpy array
```

Listing 3: Etapa 1 Python

4.2 Critério de Decisão Euclidiana

Devido ao ruído presente no canal, as estimativas $\hat{X}_m$ sofrem deslocamentos em relação à constelação ideal. Para identificar o símbolo transmitido, utiliza-se o critério de *Distância Euclidiana Mínima*, mapeando a estimativa para o ponto S mais próximo dentro do alfabeto $\mathcal{S}_{4QAM}$:

$$X_m = \arg \min_{S \in \mathcal{S}_{4QAM}} |\hat{X}_m - S|$$

Onde $\mathcal{S}_{4QAM} = \{S_1, S_2, S_3, S_4\}$. A sequência decodificada X_m representa os vetores unitários de movimentação que alimentam o modelo cinemático para o cálculo da trajetória final do drone.

4.2.1 Aplicação em Código Python

```python
# Etapa 3 - Decisor ------------------------------

# Constelacao Ideal
constelacao = [
    1/np.sqrt(2) + 1j/np.sqrt(2),  #S1
    -1/np.sqrt(2) + 1j/np.sqrt(2), #S2
    -1/np.sqrt(2) - 1j/np.sqrt(2), #S3
    1/np.sqrt(2) - 1j/np.sqrt(2)   #S4
]

simbolos_decodificados = [] # Array que armazena os simbolos que foram decodificados em sequencia

for s_estimado in simbolos_estimados:
    distancias  = [abs(s_estimado - ponto) for ponto in constelacao] # Calcula a distancia entre nosso ponto estimado e os 4 pontos ideais, salvando-as em um array
    indice_menor = np.argmin(distancias) # Escolhe o indice da menor distancia (ponto mais proximo)
    simbolos_decodificados.append(constelacao[indice_menor]) # Adiciona o ponto mais proximo como o ponto que foi transmitido
```

Listing 4: Etapa 1 Python

5 Resultados

5.1 Diagrama de Blocos

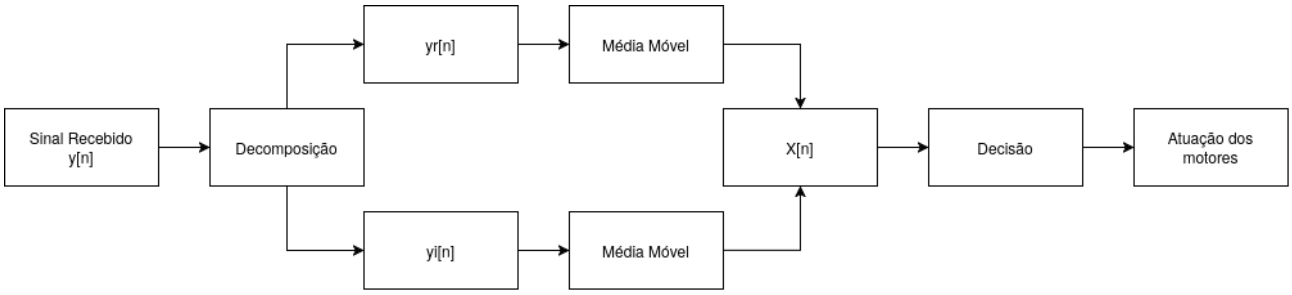

Figura 4: Diagrama de Blocos

Após o sinal ser passado pelo demodulador (saída do bloco de decisão) se tem os seguintes resultados.

5.2 Constelação

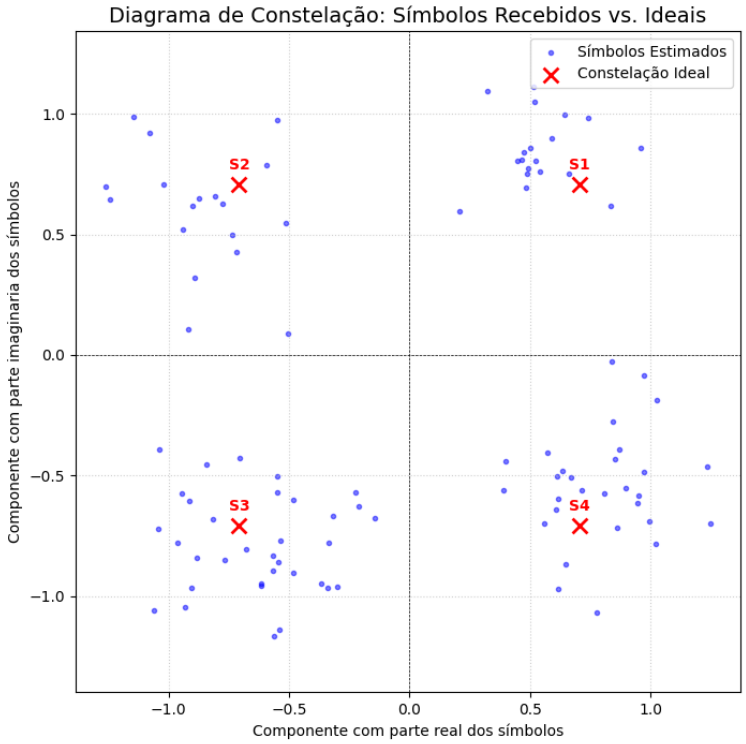

Figura 5: Constelação Pontos Estimados

O Gráfico apresenta a dispersão dos pontos estimados após a demodulação, mas antes do estágio de decisão (mapeamento para os símbolos ideais mais próximos $S_i \in \mathcal{S}_{4QAM}$). A dispersão observada deve-se à presença do ruído aditivo $z[n]$ inerente ao canal de transmissão. Nota-se, contudo, que a variância do ruído não foi suficiente para ultrapassar as fronteiras de decisão, garantindo que os símbolos fossem decodificados corretamente, conforme validado na análise de trajetória.

Ademais, à título de comparação, coloca-se a saída do misturador ao lado da constelação filtrada, denotando uma óbvia correção dos símbolos:

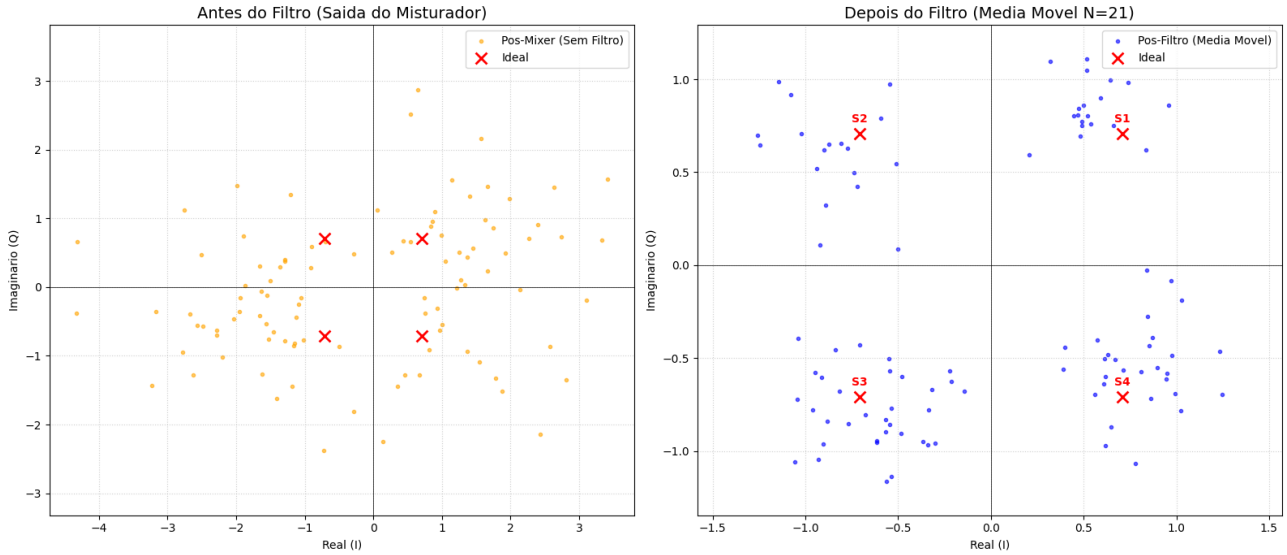

Figura 6: Comparação Pré e Pós Filtro

5.3 Trajetória

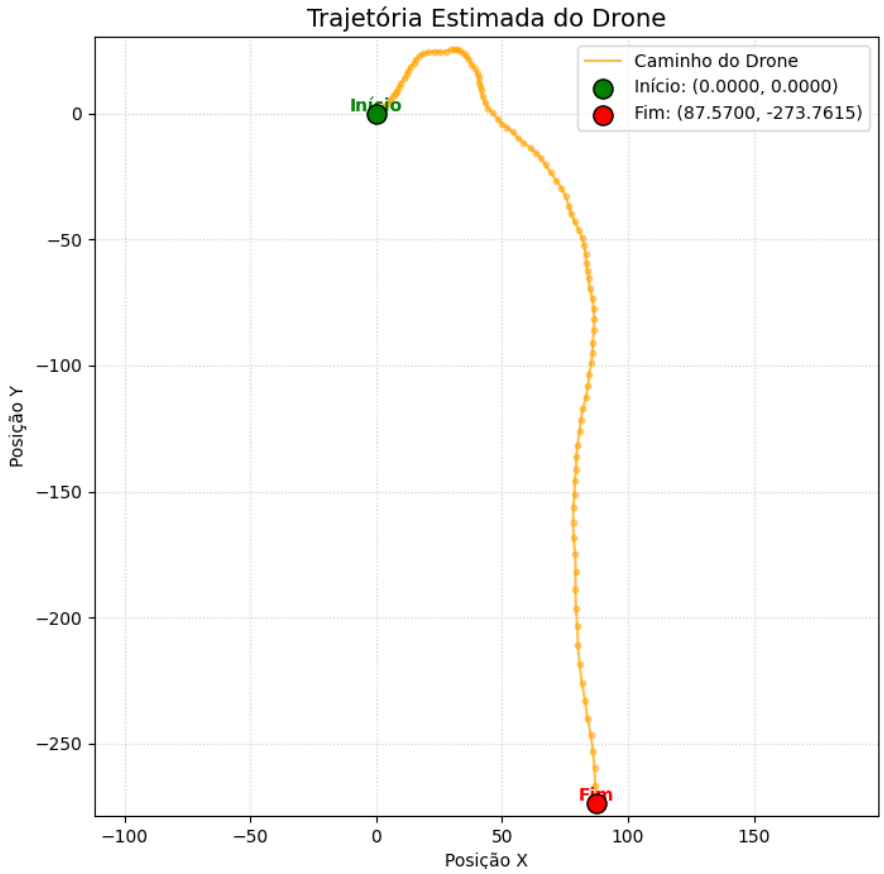

Figura 7: Trajetória do Drone

Conforme observado na Figura, o percurso realizado pelo drone e seu ponto final coincidem com as coordenadas de referência esperadas. Este resultado comprova a eficácia e a robustez do sistema demodulador proposto, demonstrando que, mesmo sob a influência do ruído $z[n]$ evidenciado na constelação, a integridade da informação de controle foi preservada.

6 Anexo 1 - Desenvolvimento das Identidades Trigonométricas

Desenvolvimento das Identidades trigonométricas

Identidades que serão utilizadas:

- $e^{\pm j\theta} = \cos\theta \pm j\,\mathrm{sen}\,\theta$
- $\cos\theta = \frac{1}{2}(e^{j\theta} + e^{-j\theta})$
- $\mathrm{sen}\,\theta = \frac{1}{j2}(e^{j\theta} - e^{-j\theta})$

I) $\cos^2\theta$

$$= \left(\frac{e^{j\theta}+e^{-j\theta}}{2}\right)\left(\frac{e^{j\theta}+e^{-j\theta}}{2}\right) = \frac{1}{4}\left(e^{j2\theta} + \overbrace{e^{j\theta-j\theta}}^{0} + \overbrace{e^{j\theta-j\theta}}^{0} + e^{-j2\theta}\right) =$$

$$= \frac{1}{4}\left(\cos(2\theta) + j\,\mathrm{sen}(2\theta) + 1 + 1 + \cos(2\theta) - j\,\mathrm{sen}(2\theta)\right) = \frac{1}{4}\left(2\cos(2\theta) + 2\right) = \frac{1}{2}\left(\cos(2\theta) + 1\right) \blacksquare$$

$$\cos^2\theta = \frac{1}{2}\left(1 + \cos 2\theta\right) \blacksquare$$

II) $\mathrm{sen}^2\theta$

$$= \left(\frac{e^{j\theta}-e^{-j\theta}}{j2}\right)\left(\frac{e^{j\theta}-e^{-j\theta}}{j2}\right) = -\frac{1}{4}\left(e^{j2\theta} - e^{0} - e^{0} + e^{-j2\theta}\right) = -\frac{1}{4}\left(\cos(2\theta) + j\,\mathrm{sen}(2\theta) - 1 - 1 + \cos(2\theta) - j\,\mathrm{sen}(2\theta)\right) =$$

$$= -\frac{1}{4}\left(2\cos(2\theta) - 2\right) = \frac{1}{2}\left(1 - \cos 2\theta\right) \blacksquare$$

$$\mathrm{sen}^2\theta = \frac{1}{2}\left(1 - \cos 2\theta\right) \blacksquare$$

III) $\mathrm{sen}(\theta)\cos(\theta)$

$$= \left(\frac{e^{j\theta}-e^{-j\theta}}{j2}\right)\left(\frac{e^{j\theta}+e^{-j\theta}}{2}\right) = \frac{1}{j4}\left(e^{j2\theta} + e^{0} - e^{0} - e^{-j2\theta}\right) = \frac{1}{j4}\left(e^{j2\theta} - e^{-j2\theta}\right) =$$

$$= \frac{1}{j4}\left(\cos(2\theta) + j\,\mathrm{sen}(2\theta) - \cos(2\theta) + j\,\mathrm{sen}(2\theta)\right) = \frac{1}{j4}\left(j2\,\mathrm{sen}(2\theta)\right) = \frac{1}{2}\cdot \mathrm{Sen}2\theta \blacksquare$$

$$\mathrm{sen}(\theta)\cos(\theta) = \frac{1}{2}\cdot\mathrm{sen}(2\theta) \blacksquare$$

Figura 8: Desenvolvimento Trigonométrico

7 Anexo 2 - Código Completo

7.1 Código Principal

```
import numpy as np
import matplotlib.pyplot as plt

# Importando os dados do csv
ySignal = np.loadtxt("rx_sgn_y.csv", dtype=complex)

# Total de amostras
total_samples = len(ySignal)

# Especificacao dos parametros dados no projeto
N = 21
Ns = 100
fc = 1000
Ts = 1/ (N*fc)


# Criando vetor de tempo discreto
n = np.arange(total_samples)

# Etapa 1 - Decomposicao do sinal ------------
arg = (2 * np.pi / N) * n

yReal = np.sqrt(2) * np.cos(arg) * ySignal

yImg = -np.sqrt(2) * np.sin(arg) * ySignal



# Etapa 2 - Filtro(Media Movel) --------------
h = np.ones(N) / N

# Convolucao
yReal_mm = np.convolve(yReal, h, mode='full')

yImg_mm = np.convolve(yImg, h, mode='full')

# Amostragem ---------------------------------
simbolos_estimados = []    # Com filtro (Sinal limpo)
simbolos_sem_filtro = []   # Sem filtro (Sinal sujo saindo do mixer)

for m in range(Ns):
    # Indice de amostragem (final do simbolo)
    idx = (m * N) + (N - 1)

    # 1. Captura do sinal FILTRADO (O que voce ja tinha)
    val_real = yReal_mm[idx]
    val_img = yImg_mm[idx]
    simbolos_estimados.append(val_real + 1j * val_img)

```

```
    # 2. Captura do sinal SEM FILTRO (Direto do Mixer)
    # Isso vai mostrar a interferencia de 2*fc
    raw_real = yReal[idx]
    raw_img = yImg[idx]
    simbolos_sem_filtro.append(raw_real + 1j * raw_img)

simbolos_estimados = np.array(simbolos_estimados)
simbolos_sem_filtro = np.array(simbolos_sem_filtro)


# Etapa 3 - Decisor ------------------------------

# Constelacao Ideal
constelacao = [
    1/np.sqrt(2) + 1j/np.sqrt(2),  #S1
    -1/np.sqrt(2) + 1j/np.sqrt(2), #S2
    -1/np.sqrt(2) - 1j/np.sqrt(2), #S3
    1/np.sqrt(2) - 1j/np.sqrt(2)   #S4
]

simbolos_decodificados = []

for s_estimado in simbolos_estimados:
    distancias  = [abs(s_estimado - ponto) for ponto in constelacao]
    indice_menor = np.argmin(distancias)
    simbolos_decodificados.append(constelacao[indice_menor])


# Teste da rota ----------------------------------------
cx, cy = 0, 0 # Posicoes Iniciais
vx, vy = 2, 2 # Velocidades
path_x = [cx]
path_y = [cy]
alpha = 0.5
tau = 1

for sym in simbolos_decodificados:
    vx = vx + alpha * sym.real
    vy = vy + alpha * sym.imag

    cx = cx + tau * vx
    cy = cy + tau * vy

    path_x.append(cx)
    path_y.append(cy)

print(f"Coordenada final: ({cx:.4f}, {cy:.4f})")

# Gerar graficos ---------------------------------

# Grafico sem comparacoes
plt.figure(figsize=(8, 8))

```

```
# Plotar os simbolos estimados
plt.scatter(simbolos_estimados.real, simbolos_estimados.imag,
            color='blue', alpha=0.5, label='Simbolos Estimados', marker='.')

# Plotar a constelacao ideal
const_x = [p.real for p in constelacao]
const_y = [p.imag for p in constelacao]
plt.scatter(const_x, const_y, color='red', marker='x', s=100, linewidths=2, label='Constelacao Ideal')

# Adicionar legendas para cada simbolo ideal
for i, ponto in enumerate(constelacao):
    plt.annotate(f'S{i+1}', (ponto.real, ponto.imag), textcoords="offset points",
                 xytext=(0,10), ha='center', fontweight='bold', color='red')

# Formatacao do grafico
plt.axhline(0, color='black', linestyle='--', linewidth=0.5) # Eixo X
plt.axvline(0, color='black', linestyle='--', linewidth=0.5) # Eixo Y
plt.grid(True, linestyle=':', alpha=0.6)
plt.title('Diagrama de Constelacao: Simbolos Recebidos vs. Ideais', fontsize=14)
plt.xlabel('Componente com parte real dos simbolos')
plt.ylabel('Componente com parte imaginaria dos simbolos')
plt.legend(loc='upper right')
plt.axis('equal')

plt.show()

# Constelacoes com comparacao
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))

# --- GRAFICO 1: SEM FILTRO (Esquerda) ---
ax1.scatter(simbolos_sem_filtro.real, simbolos_sem_filtro.imag,
            color='orange', alpha=0.6, label='Pos-Mixer (Sem Filtro)', marker='.')

# Desenha a constelacao ideal em cima para referencia
const_x = [p.real for p in constelacao]
const_y = [p.imag for p in constelacao]
ax1.scatter(const_x, const_y, color='red', marker='x', s=100, linewidths=2, label='Ideal')

ax1.set_title('Antes do Filtro (Saida do Misturador)', fontsize=14)
ax1.set_xlabel('Real (I)')
ax1.set_ylabel('Imaginario (Q)')
ax1.grid(True, linestyle=':', alpha=0.6)
ax1.axhline(0, color='black', linewidth=0.5)
ax1.axvline(0, color='black', linewidth=0.5)
ax1.legend()
ax1.axis('equal')
```

```
149
150 # --- GRAFICO 2: COM FILTRO (Direita) ---
151 ax2.scatter(simbolos_estimados.real, simbolos_estimados.imag,
152             color='blue', alpha=0.6, label='Pos-Filtro (Media Movel)',
    marker='.')
153
154 # Desenha a constelacao ideal
155 ax2.scatter(const_x, const_y, color='red', marker='x', s=100, linewidths
    =2, label='Ideal')
156
157 # Adiciona legendas S1, S2... apenas no grafico limpo para nao poluir
158 for i, ponto in enumerate(constelacao):
159     ax2.annotate(f'S{i+1}', (ponto.real, ponto.imag), textcoords="offset
    points",
160                  xytext=(0,10), ha='center', fontweight='bold', color='
    red')
161
162 ax2.set_title('Depois do Filtro (Media Movel N=21)', fontsize=14)
163 ax2.set_xlabel('Real (I)')
164 ax2.set_ylabel('Imaginario (Q)')
165 ax2.grid(True, linestyle=':', alpha=0.6)
166 ax2.axhline(0, color='black', linewidth=0.5)
167 ax2.axvline(0, color='black', linewidth=0.5)
168 ax2.legend()
169 ax2.axis('equal')
170
171 plt.tight_layout() # Ajusta o espacamento para nao sobrepor textos
172 plt.show()
173
174 # --- Grafico da Trajetoria do Drone ---
175 plt.figure(figsize=(8, 8))
176
177 # 1. Plotar a linha da trajetoria
178 plt.plot(path_x, path_y, linestyle='-', color='orange', alpha=0.7, label=
    'Caminho do Drone', zorder=1)
179 plt.scatter(path_x, path_y, color='orange', s=10, alpha=0.5) # Pontos
    intermediarios pequenos
180
181 # 2. Ponto de Inicio
182 plt.scatter(path_x[0], path_y[0], color='green', s=120, edgecolors='black
    ', zorder=5, label=f'Inicio: ({path_x[0]:.4f}, {path_y[0]:.4f})')
183
184 # 3. Ponto de Fim
185 plt.scatter(path_x[-1], path_y[-1], color='red', s=120, edgecolors='black
    ', zorder=5, label=f'Fim: ({path_x[-1]:.4f}, {path_y[-1]:.4f})')
186
187 # 4. Anotacoes de texto
188 plt.text(path_x[0], path_y[0] + 1, 'Inicio', color='green', fontweight='
    bold', ha='center')
189 plt.text(path_x[-1], path_y[-1] + 1, 'Fim', color='red', fontweight='bold
    ', ha='center')
190
191 # --- Ajustes de Escala e Enquadramento ---
```

```

# 'equal' garante que 1 metro em X seja igual a 1 metro em Y na tela
plt.axis('equal')

# Adiciona uma margem de seguranca para os rotulos nao ficarem cortados
margin = 5
plt.xlim(min(path_x) - margin, max(path_x) + margin)
plt.ylim(min(path_y) - margin, max(path_y) + margin)

plt.title('Trajetoria Estimada do Drone', fontsize=14)
plt.xlabel('Posicao X')
plt.ylabel('Posicao Y')
plt.grid(True, linestyle=':', alpha=0.6)
plt.legend(loc='best')

plt.show()
```

Listing 5: Etapa 1 Python

7.2 Outros Gráficos

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.fft import fft, fftshift, fftfreq

# CONFIGURACOES GERAIS
N = 21              # Tamanho do filtro
fc = 1000           # Frequencia da portadora
fs = N * fc         # 21000 Hz

# GRaFICO 1: MAGNITUDE DA RESPOSTA EM FREQUENCIA TEORICA:
# Definicao do filtro
h = np.ones(N) / N

# Calculo da resposta em frequencia
num_pontos_fft = 4096 # alto numero de pontos para suavizar
H = fft(h, num_pontos_fft)

# Eixo da frequencia normalizada
H_mag = np.abs(H[:num_pontos_fft//2])
w_norm = np.linspace(0, 1, len(H_mag))

plt.figure(figsize=(10, 5))
plt.plot(w_norm, H_mag, 'royalblue', linewidth=2)

# Estilizacao para ficar igual ao seu relatorio
plt.title(f'Resposta em Frequencia do Filtro (N={N})')
plt.xlabel(r'Frequencia Normalizada ($\times \pi$ rad/amostra)')
plt.ylabel('Magnitude')
plt.grid(True, linestyle='--', alpha=0.6)
plt.axvline(2/N, color='red', linestyle='--', label='Corte aprox.')
plt.legend()
plt.tight_layout()
```

```
plt.show()


# GR FICO 2: ESPECTRO DE FREQUENCIA

# Gerar um sinal sintetico similar ao do drone para o grafico
np.random.seed(42)
num_simbolos = 200
t = np.arange(num_simbolos * N)


# Sinal de dados
dados = np.repeat(np.random.choice([-1, 1], num_simbolos), N)

# Misturador: Sinal * cos(2*pi*fc*t) * cos(2*pi*fc*t) gera DC + 2fc
# Aqui simulamos direto o resultado do mixer: Parte DC + Parte em 2*fc + Ruido
sinal_pos_mixer = (dados * 1.0) + (dados * np.cos(2 * np.pi * (2*fc/fs) * t)) + 0.5 * np.random.randn(len(t))

# Aplicar o Filtro
sinal_filtrado = np.convolve(sinal_pos_mixer, h, mode='same')

# Calculo do FFT para o Grafico
freqs = fftshift(fftfreq(len(t), d=1/fs))
X_mixer = fftshift(fft(sinal_pos_mixer))
X_filt = fftshift(fft(sinal_filtrado))

# Magnitude normalizada para escala do grafico
mag_mixer = np.abs(X_mixer)
mag_filt = np.abs(X_filt)

# Plotagem
plt.figure(figsize=(10, 6))

# Plot do sinal ruidoso
plt.plot(freqs, mag_mixer, color='orange', alpha=0.6, linewidth=1, label='P s -Mixer (Ruidoso)')

# Plot do sinal filtrado
plt.plot(freqs, mag_filt, color='green', linewidth=1.5, label='Filtrado (Limpo)')

# Configura es de Eixo e Texto
plt.title('Espectro de Frequ ncia')
plt.xlabel('Frequencia (Hz)')
plt.ylabel('Magnitude')
plt.legend(loc='upper right', frameon=True, fancybox=True, framealpha=1, borderpad=1)
plt.grid(True, alpha=0.3)

# Limites iguais a imagem ]
plt.xlim(-3000, 3000)
```

```
plt.tight_layout()
plt.show()
```

Listing 6: Outros Graficos Python