

# 1 Metodologia

Todo o trabalho foi desenvolvido em ambiente simulado, utilizando o *software* Matlab®. Primeiramente, foram representados no Simulink® dois sistemas de quadricóptero seguindo a modelagem proposta por Balas (2007): ambos submetidos a uma gravidade  $g = 9,8 \text{ m/s}^2$  e com comprimento de cada haste  $l = 0,5 \text{ m}$ . Os dois sistemas diferem entretanto na massa do quadricóptero modelado em cada caso:  $m = 2,3 \text{ kg}$  e  $m = 5 \text{ kg}$ .

O sistema com massa  $m = 2,3 \text{ kg}$  foi então utilizado para mostrar o desacoplamento das entradas e a instabilidade do sistema. Para tanto, o modelo foi submetido a sinais em pulso em cada uma de suas entradas. Então, a partir da resposta do sistema a essas entradas, foram modelados dois controladores *fuzzy* para estabilizar a atitude e altitude do quadricóptero. Para tanto, foi utilizada a ferramenta *Fuzzy Logic Toolbox* do Matlab®.

A partir dos controladores *fuzzy* desenvolvidos e utilizando a ferramenta *Neuro-Fuzzy Designer* também do Matlab® foram modelados dois controladores *neuro-fuzzy* para controlar a atitude e altitude do *drone*.

Os controladores *fuzzy* e *neuro-fuzzy* foram então comparados tanto para o sistema com massa de  $m = 2,3 \text{ kg}$  quanto para o de  $m = 5 \text{ kg}$ . Os aspectos levados em conta para a comparação dos controladores foram:

- Variação apresentada;
- Tempo necessário para a estabilização;
- Oscilação;
- Sobrelevação apresentada;
- Gasto energético apresentado pelos controladores.

Por fim, o sistema com massa  $m = 2,3 \text{ kg}$ , para o qual os controladores foram desenvolvidos, foi submetido a um cenário que envolve ruídos de medição para verificar se o controle implementado se mostra robusto.

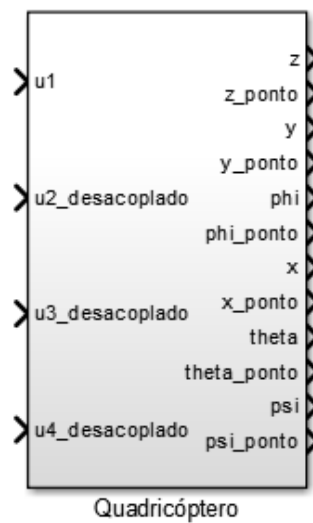
## 2 Desenvolvimento do Trabalho

Este capítulo trata da forma como o trabalho foi desenvolvido de forma a aplicar a metodologia proposta, descrevendo os processos utilizados para a verificação do desacoplamento e instabilidade do sistema (Seção 2.1); a modelagem dos controladores *fuzzy* (Seção 2.2) e *neuro-fuzzy* (Seção 2.3); e a descrição detalhada dos experimentos realizados (Seção 2.4).

### 2.1 Verificação do Desacoplamento das Entradas

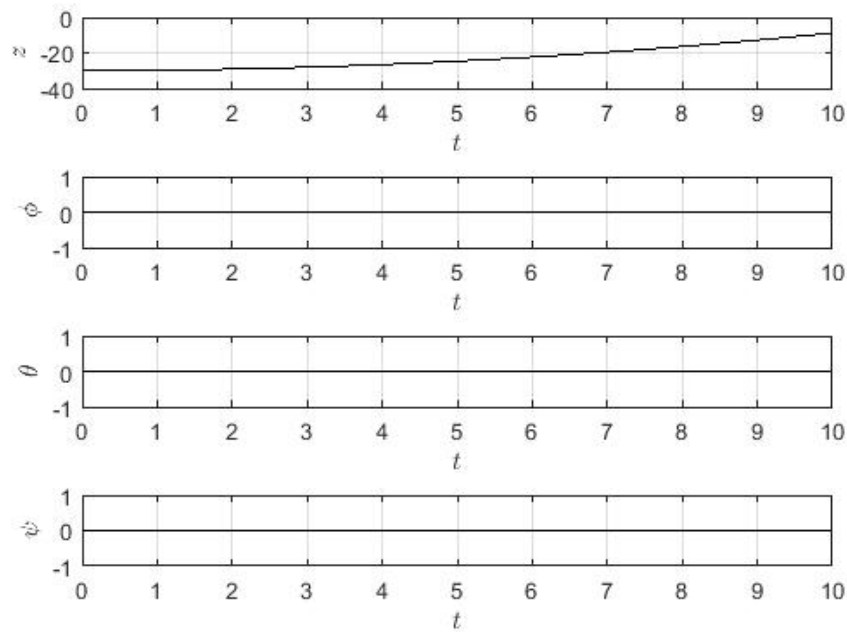
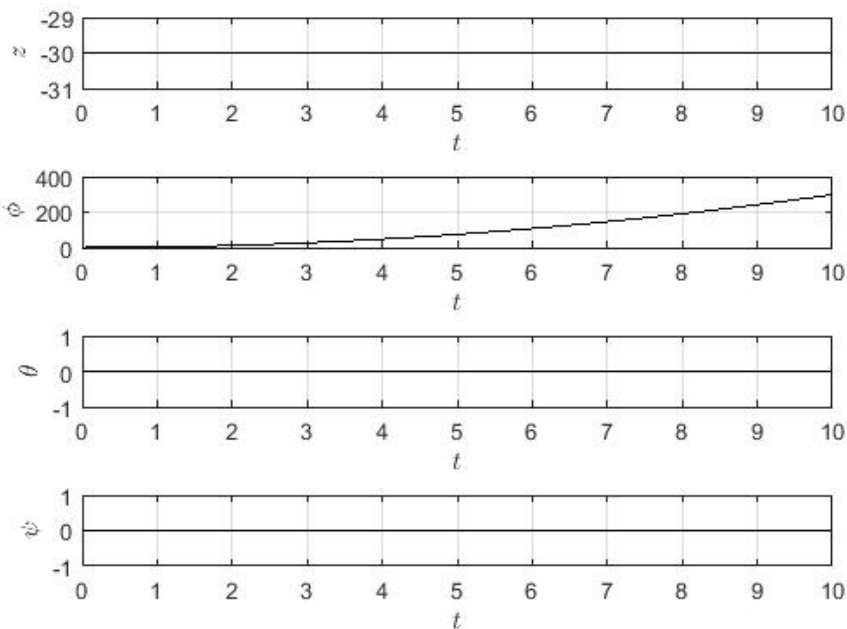
A representação do quadricóptero criada no Simulink® seguindo a modelagem de Balas (2007) é mostrada na Figura 1.

Figura 1 – Representação do quadricóptero no *Simulink*

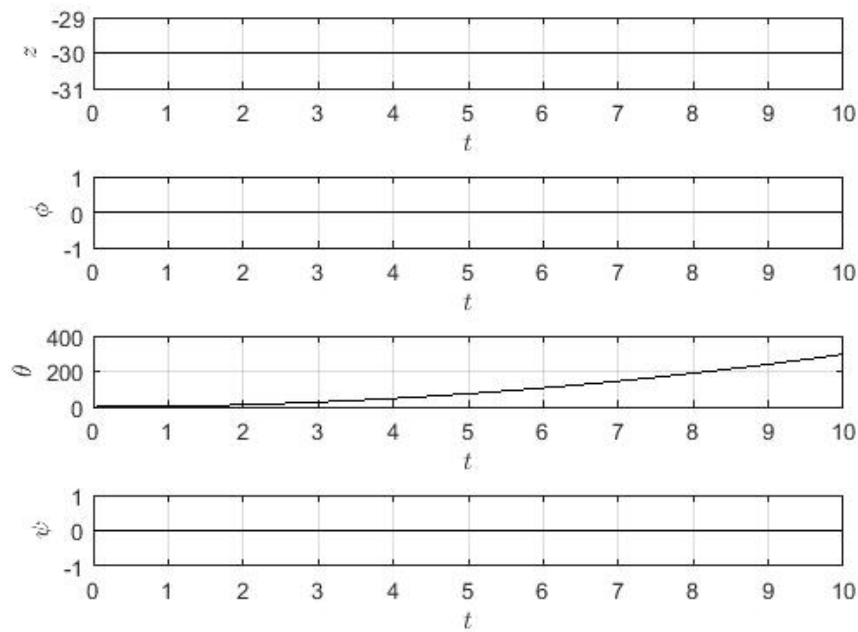
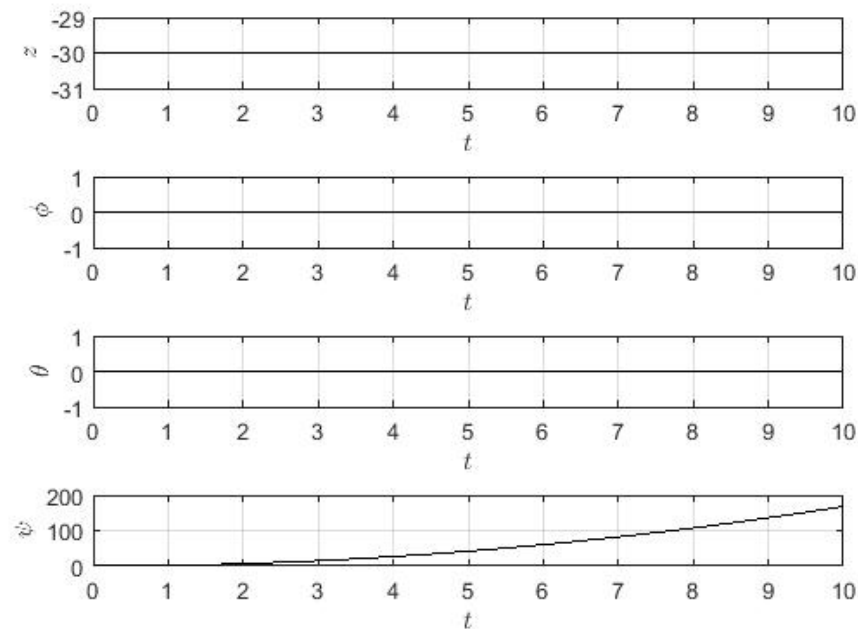


Como se pode ver, o sistema inclui os quatro sinais de entrada seguindo o desacoplamento desenvolvido ( $u1$ ,  $u2\_desacoplado$ ,  $u3\_desacoplado$  e  $u4\_desacoplado$ ) e com as doze saídas referentes às seis variáveis de configuração  $x, y, z, \phi, \theta, \psi$  indicadas por  $x, y, z, \phi, \theta, \psi$ , respectivamente; e suas respectivas variações  $\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}$  representadas por  $x\_ponto, y\_ponto, z\_ponto, \phi\_ponto, \theta\_ponto$  e  $\psi\_ponto$ .

Para mostrar o desacoplamento das variáveis, alternadamente foi aplicado um sinal de degrau a cada uma das entradas. Em cada um dos casos, somente uma entrada era submetida ao degrau, ao passo que as demais eram aterradas. As respostas, a cada um dos experimentos, das variáveis de configuração relativas à altitude e atitude do quadricóptero são mostradas nas Figuras 2, 3, 4 e 5 tomando como estado inicial um quadricóptero estável ( $\phi = \theta = \psi = 0$  rad) a trinta metros de altura ( $z = -30$  m).

Figura 2 – Resposta das saídas  $z$ ,  $\phi$ ,  $\theta$  e  $\psi$  a um entrada em degrau em  $u_1$ Figura 3 – Resposta das saídas  $z$ ,  $\phi$ ,  $\theta$  e  $\psi$  a um entrada em degrau em  $u_{2\_desacoplado}$ 

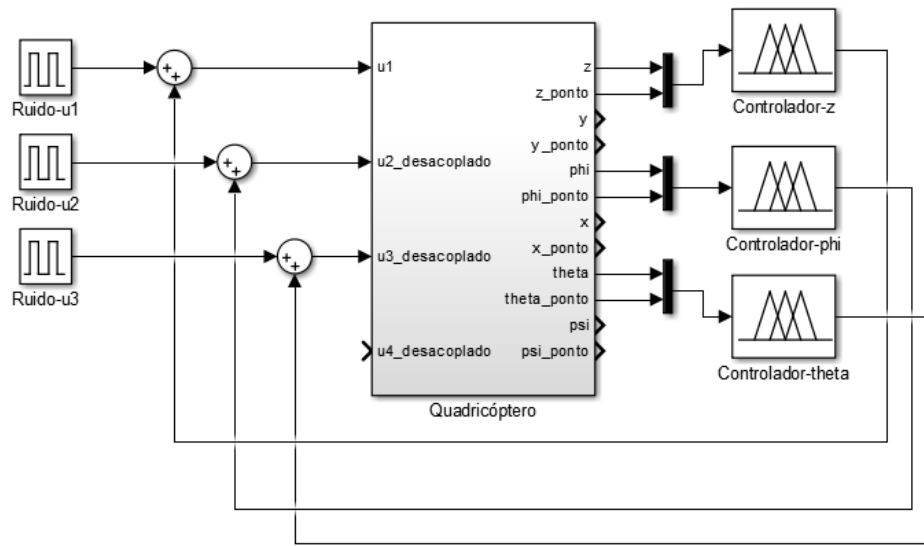
Como se pode ver, cada entrada afeta uma única saída e cada saída é afetada apenas por uma entrada. Com isso, mostra-se o desacoplamento existente que faz com que a entrada  $u_1$  somente interfira na variável de configuração  $z$ ;  $u_{2\_desacoplado}$  em  $\phi$ ;  $u_{3\_desacoplado}$  em  $\theta$ ; e  $u_{4\_desacoplado}$  em  $\psi$ .

Figura 4 – Resposta das saídas  $z$ ,  $\phi$ ,  $\theta$  e  $\psi$  a um entrada em degrau em  $u3\_desacoplado$ Figura 5 – Resposta das saídas  $z$ ,  $\phi$ ,  $\theta$  e  $\psi$  a um entrada em degrau em  $u4\_desacoplado$ 

## 2.2 Controladores Fuzzy

O projeto dos controlador *fuzzy* foi focado na estabilização de atitude e altitude do *drone* modelado de forma a se inserirem no sistema como é mostrado na Figura 6.

Figura 6 – Diagrama do sistema de controle de altitude utilizando controlador fuzzy



Como se pode ver, três controladores agem no sistema com o objetivo de torná-lo imune a distúrbios representados pelas entradas de ruídos. O Controlador-z diz respeito a um controlador de altitude ao passo que os Controlador-phi e Controlador-theta dizem respeito a controladores de atitude que, pelo fato de o quadricóptero ser simétrico em relação aos eixos  $x$  e  $y$ , puderam ser representados por um único controlador.

O controlador de altitude possui duas entradas e uma saída. As entradas são referentes à posição vertical do quadricóptero ( $z$ ) e sua respectiva velocidade ( $\dot{z}$ ), ao passo que a saída diz respeito ao sinal de controle a ser aplicado sobre o sistema para estabilizar sua altitude ( $u_1$ ).

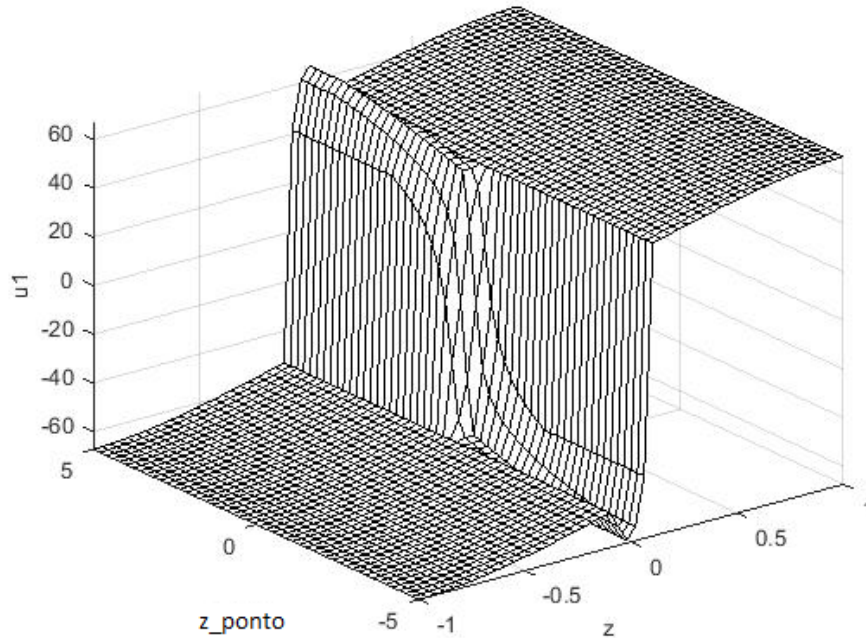
Utilizando o *Fuzzy Logic Toolbox* do MATLAB, cada variável linguística do controlador *fuzzy* foi dividida em três conjuntos: N (negativo), Z (zero) e P (positivo), tomando como base os trabalhos de [Maj e Butkiewicz \(2013\)](#) e [Gao et al. \(2014\)](#). As regras *fuzzy* definidas para este controlador são mostradas no Quadro 1, e a Figura 7 exibe seu equivalente em superfície.

Quadro 1 – Regras fuzzy para modelagem do controle de altitude

$z$	$\dot{z}$	$u_1$
N	-	N
P	-	P
Z	N	N
Z	Z	Z
Z	P	P

O controlador de atitude projetado também possui duas entradas e uma saída. Desta vez, entretanto, as entradas são referentes ao ângulo em relação ao eixo horizontal

Figura 7 – Superfície das regras do sistema de controle *fuzzy* para a altitude do quadricóptero



( $\phi$  ou  $\theta$ ) e sua respectiva variação ( $\dot{\phi}$  ou  $\dot{\theta}$ ). Mais uma vez, cada variável linguística foi dividida em três conjuntos: N, Z e P.

As regras que regem o controlador de atitude são sintetizadas no Quadro 2 e podem ser vistas na superfície de regras mostradas na Figura 8.

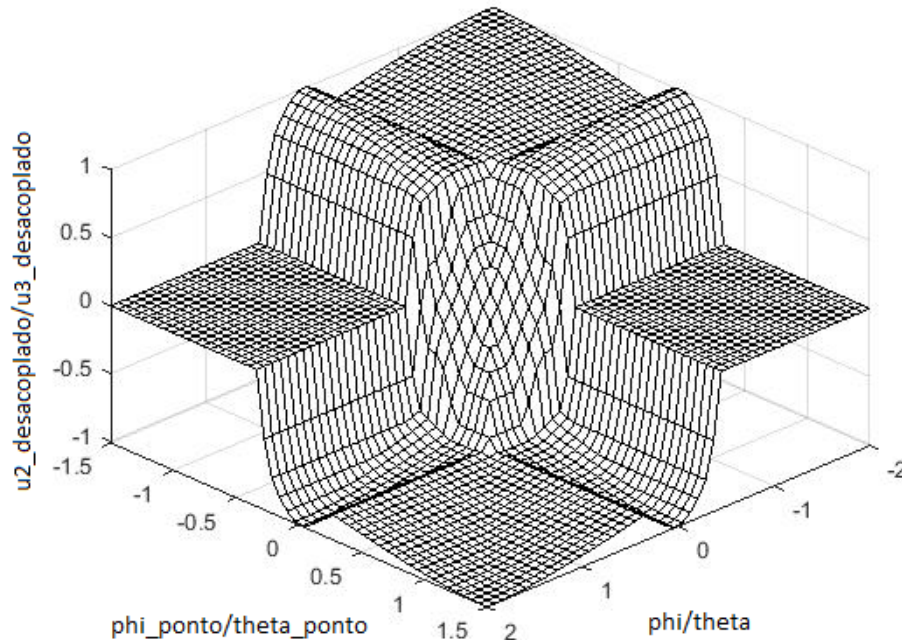
Quadro 2 – Regras fuzzy para modelagem do controle de atitude

$\phi/\theta$	$\dot{\phi}/\dot{\theta}$	$u_2/u_3$
P	P	N
P	Z	N
P	N	Z
N	N	P
N	Z	P
N	P	Z
Z	Z	Z
Z	N	P
Z	P	N

## 2.3 Controladores Neuro-Fuzzy

A partir dos controladores de atitude e altitude *fuzzy* projetados, foram propostos dois controladores do tipo neuro-*fuzzy*: um para cada dos casos.

Figura 8 – Superfície das regras do sistema de controle *fuzzy* para a atitude do quadricóptero



Para tanto, foram utilizados os códigos mostrados nos Apêndices A e B. No processo de criação do controlador de altitude neuro-*fuzzy*, foram gerados trezentos<sup>1</sup> pares de entradas e cada um deles foi submetido ao processo de inferência *fuzzy* utilizando o controlador previamente modelado e descrito na Seção 2.2. Dois terços desses dados foram utilizados para gerar o conjunto de treinamento, representado pela variável `train` e o um terço restante foi armazenado na variável `test` e utilizado para validação do treinamento. Então, utilizando o comando `mam2sug` do MATLAB, foi gerado um modelo fuzzy Sugeno a partir do Mamdani que havia sido modelado e este novo arquivo foi salvo sob o nome `fis_altitude_neuro.fis`.

Feito isto, utilizou-se o comando `anfisedit` para abrir o *Neuro-Fuzzy Designer* do MATLAB, cuja interface é mostrada na Figura 9. No campo marcado pelo número 2 na imagem (*Generate FIS*), clicou-se no botão *Load* e se selecionou o arquivo `fis_altitude_neuro.fis` que fora gerado pelo código executado. Após isto, no campo marcado pelo número 1 (*Load Data*), marcou-se *Training* e *worksp* para utilizar uma variável da área de trabalho do MATLAB para treinar a rede. Após clicar em *Load Data*, digitou-se `train`, nome da variável definida no código. Então, no campo marcado pelo número 3, marcou-se *Training Data* e se clicou no botão *Test Now* para executar o treinamento da rede. Após estes passos, a rede neuro-fuzzy foi devidamente treinada e sua estrutura, mostrada na Figura 10, pode ser obtida clicando no botão *Structure* logo

<sup>1</sup> Este valor foi arbitrado por corresponder a uma quantidade razoável para treinar a RNA sem que se alcance o sobre-parametrização, conhecido como *overfitting*.

acima do campo 3. Esta estrutura relaciona as variáveis de entrada e suas funções de pertinência, através das regras fuzzy, à saída do sistema e às suas funções de pertinência, em que cada componente representa um neurônio da RNA obtida.

Figura 9 – Interface gráfica da ferramenta *Neuro-Fuzzy Designer* com destaque aos três campos necessários para treinamento e teste da rede neuro-fuzzy

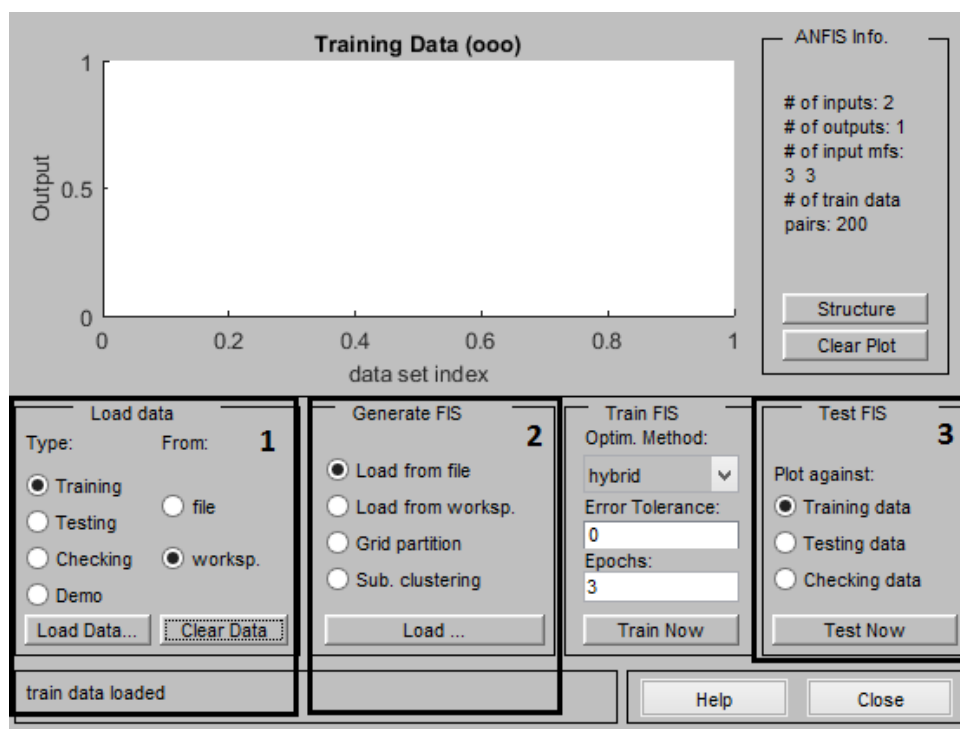
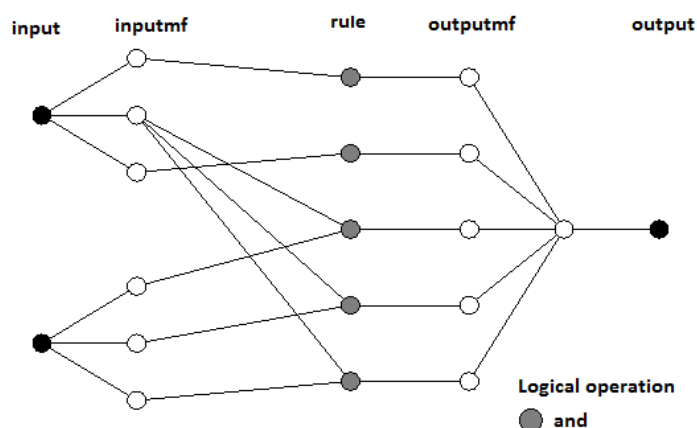


Figura 10 – Diagrama da RNA referente ao controlador neuro-fuzzy para altitude

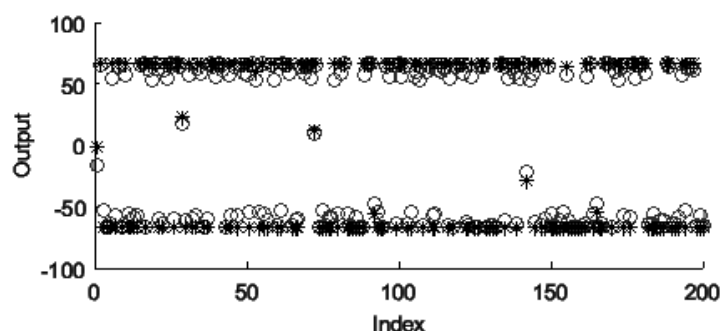


Após o término do treinamento, deve-se submeter a rede ao processo de teste. Para tanto, basta selecionar *Testing* no campo marcado pelo número 1, deixar marcada a opção *workspace*, clicar no botão *Load Data* e escolher a variável *test*, que também foi definida no código executado.



A Figura 11 mostra o gráfico obtido na ferramenta após o processo de treinamento, em que os círculos brancos mostram os dados utilizados no treinamento e os asteriscos pretos indicam o valor referentes a eles obtidos pela rede treinada.

Figura 11 – Resultado obtido pelo treinamento da RNA para controle de altitude



Um processo similar foi aplicado para modelar o controlador de atitude neuro-fuzzy, como mostra o Apêndice B. As Figuras 12 e 13 mostram o diagrama da RNA referente ao controlador neuro-fuzzy para atitude e o resultado obtido pelo seu treinamento respectivamente.

Figura 12 – Diagrama da RNA referente ao controlador neuro-fuzzy para atitude

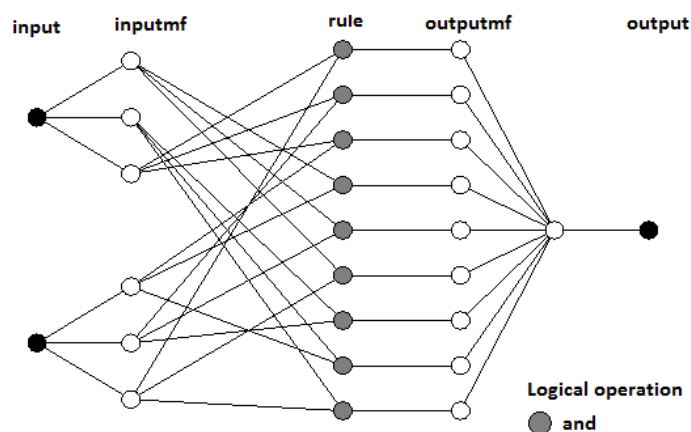
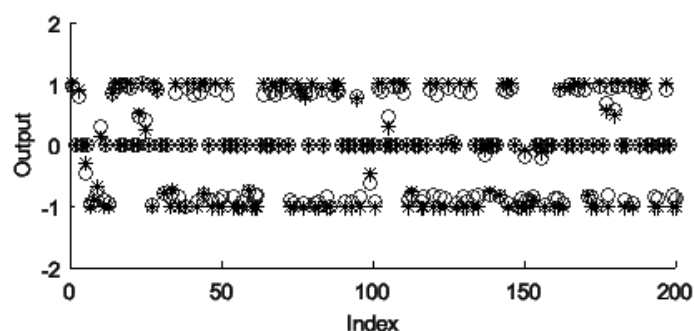


Figura 13 – Resultado obtido pelo treinamento da RNA para controle de atitude



O processo de treinamento determina o comportamento dos controladores neuro-fuzzy projetados, cujas superfícies de regras são exibidas nas Figuras 14 e 15.

Figura 14 – Superfície das regras do sistema de controle neuro-*fuzzy* para a altitude do quadricóptero

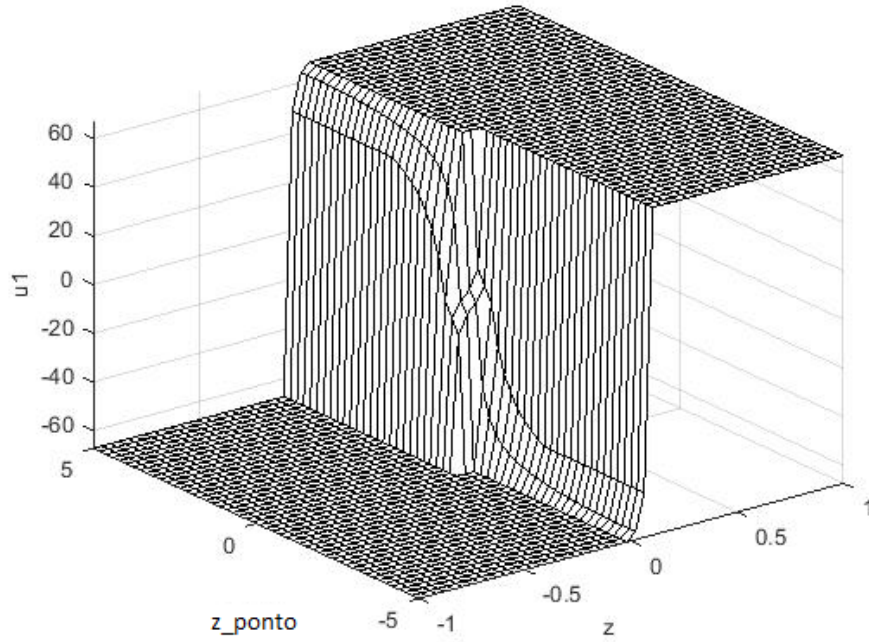
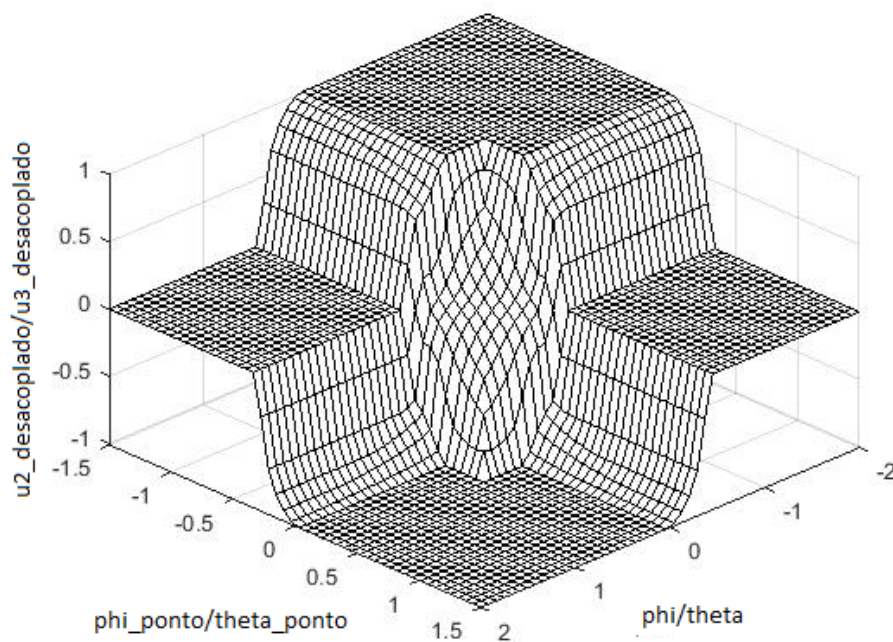


Figura 15 – Superfície das regras do sistema de controle neuro-*fuzzy* para a atitude do quadricóptero



## 2.4 Experimentos Realizados

Uma vez projetados os controladores *fuzzy* e *neuro-fuzzy*, o sistema foi submetido a distúrbios em pulso em atitude e altitude para verificar o funcionamento deles sob condições similares às mostradas quando nenhum controle agia sobre ele fazendo com que o sistema divergisse. Primeiramente, o comportamento de ambos os controladores foi verificado quando atuando sobre o sistema para os quais eles foram projetados, com  $g = 9,81 \text{ m/s}^2$ ,  $m = 2,3 \text{ kg}$  e  $l = 0,5 \text{ m}$ .

Em seguida, para testar a robustez de cada controlador, foi feita uma simulação em que eles atuam sobre um sistema cuja massa do quadricóptero é  $m = 5 \text{ kg}$ , valor este que foi escolhido por variar o parâmetro massa em mais de 100 %.

Por fim, foi testado o funcionamento do sistema quando um ruído de medição passa a fazer parte dele. Os resultados obtidos são mostrados no capítulo seguinte.

## Referências

BALAS, C. **Modeling and Linear Control of a Quadrotor**. Dissertação (Mestrado) — Cranfield University, Reino Unido, 2007. Citado 2 vezes nas páginas [1](#) e [2](#).

GAO, Q.; YUE, F.; HU, D. Research of stability augmentation hybrid controller for quadrotor uav. In: **Control and Decision Conference (2014 CCDC), The 26th Chinese**. [S.l.: s.n.], 2014. p. 5224–5229. Citado na página [5](#).

MAJ, W.; BUTKIEWICZ, B. Flying n-copter with fuzzy logic control. In: **Signal Processing Symposium (SPS), 2013**. [S.l.: s.n.], 2013. p. 1–6. Citado na página [5](#).

## Apêndices

## APÊNDICE A – Código para Criação de Modelo Neuro-Fuzzy para Altitude e Definição de Dados para Treinamento

```

1      % le arquivo fis referente ao controle de altitude
2      fismat = readfis('fis_altitude.fis');
3
4      % define numero de casos a serem avaliados (treinamento + teste)
5      n = 300;
6      % define conjunto de n entradas aleatorias para o sistema fuzzy
7      % respeitando o range de cada entrada
8      input = zeros(n,2);
9      for i=1:n
10         z_value = rand * 2 - 1;
11         z_dot_value = rand * 10 - 5;
12         input(i,:) = [ z_value z_dot_value ];
13     end
14
15     % avalia resposta fuzzy para cada entrada
16     output= evalfis(input,fismat);
17
18     % define data como vetor relacionando cada conjunto de entradas ...
19     % a saida
20     % - obtida pelo sistema fuzzy
21     data = [];
22     for i=1:n
23         data(i,:) = [ input(i,:) output(i) ];
24     end
25
26     % define que 2/3 dos dados obtidos serao usados para treinamento
27     % e 1/3 sera usado para teste da rede
28     train = data(1:2*n/3,:);    % dados para treinamento
29     test = data(2*n/3+1:n,:);  % dados para validacao do sistema ...
30     treinado
31
32     % gera modelo fuzzy Sugeno a partir do Mamdani modelado
33     sugFIS = mam2sug(fismat);
34     % salva modelo Sugeno em disco com o nome fis_altitude_neuro.fis
35     writefis(sugFIS, 'fis_altitude_neuro.fis');

```

## APÊNDICE B – Código para Criação de Modelo Neuro-Fuzzy para Atitude e Definição de Dados para Treinamento

```

1      % le arquivo fis referente ao controle de atitude
2      fismat = readfis('fis_atitude.fis');
3
4      % define numero de casos a serem avaliados (treinamento + teste)
5      n = 300;
6      % define conjunto de n entradas aleatorias para o sistema fuzzy
7      % respeitando o range de cada entrada
8      input = zeros(n,2);
9      for i=1:n
10         phi_value = rand * 4 - 2;
11         phi_dot_value = rand * 3 - 1.5;
12         input(i,:) = [ phi_value phi_dot_value ];
13     end
14
15     % avalia resposta fuzzy para cada entrada
16     output= evalfis(input,fismat);
17
18     % define data como vetor relacionando cada conjunto de entradas ...
19     % a saída
20     % obtida pelo sistema fuzzy
21     data = [];
22     for i=1:n
23         data(i,:) = [ input(i,:) output(i) ];
24     end
25
26     % define que 2/3 dos dados obtidos serao usados para treinamento
27     % e 1/3 sera usado para teste da rede
28     train = data(1:2*n/3,:);    % dados para treinamento
29     test = data(2*n/3+1:n,:);  % dados para validação do sistema ...
30     % treinado
31
32     % gera modelo fuzzy Sugeno a partir do Mamdani modelado
33     sugFIS = mam2sug(fismat);
34     % salva modelo Sugeno em disco com o nome fis_atitude_neuro.fis
35     writefis(sugFIS, 'fis_atitude_neuro.fis');

```