Relatório

Controlador Fuzzy Para Simulador de Voo X-Plane 11

Gabriel Oliveira [1], Rafael Almaleh [2] e Vinicius Camargo [3]

[1] gabriel.luis@ufrgs.br
[2] hess.rafael95@gmail.com
[3] camargo@ufrgs.br

Data Início: 20/06/2020; Data Final: 10/07/2020;

Resumo: Neste trabalho foi desenvolvido um controlador *fuzzy* de ângulo de arfagem para o simulador de voo X-Plane 11. Após período inicial de familiarização com o simulador, foram obtidos dados de navegação compostos pelos sensores do ambiente virtual (ângulo de arfagem e derivada da arfagem) e as decisões de atuação do usuário (posição do *Joystick*). Estes dados foram gravados através de um *datalogger*, e utilizados, através do *fuzzy c-means* para identificações de padrões no comportamento do operador, para geração de *clusters fuzzy*. Com a chegada de dados novos, os mesmos tinham a sua pertinência a cada *cluster* avaliada e obteve-se a sua saída na forma de variação da posição do *joystick*. Assim, avaliou-se a performance do controlador *fuzzy* em comparação a um controlador proporcional tradicional. O controlador *fuzzy* apresentou, para uma perturbação positiva, erro de seguimento de 0,73 graus, tempo de acomodação de 13,28 segundos e *undershoot* zero, e para uma perturbação negativa, erro de seguimento de -0,55 grau, tempo de acomodação de 14,47 segundos e *overshoot* de 20%. Já o controlador proporcional apresentou, para uma perturbação positiva, erro de seguimento de -0,15 grau, tempo de acomodação de 1,97 segundos e *undershoot* de 18%, e para uma perturbação negativa, erro de seguimento de -0,61 grau, tempo de acomodação de 18,18 segundos e *overshoot* de 416%.

Abstract: In this project, a pitch angle fuzzy controller was developed for the X-Plane 11 flight simulator. After an initial familiarization period with the simulator, navigation data composed by the virtual environment sensors (pitch angle and pitch angle derivative) and user action (position of Joystick) were gathered. Using fuzzy c-means to identify patterns in the operator's behavior, this data was recorded using a datalogger to generate Fuzzy clusters. With the arrival of new data, they had their relevance to each cluster assessed and their output was obtained in the form of variation of the joystick position. The performance of the Fuzzy controller was then evaluated in comparison to a traditional proportional controller. The Fuzzy controller had, for a positive disturbance, a tracking error of 0.73 degrees, accommodation time of 13.28 seconds and zero undershoot, and for a negative disturbance, a tracking error of -0.55 degrees, accommodation time of 14.47 seconds and overshoot 20%. The proportional controller had, for a positive disturbance, a tracking error of -0.15 degrees, accommodation time of 1.97 seconds and a undershoot of 18%, and for a negative disturbance, an average tracking error of -0.61 degrees, maximum accommodation time of 18.18 seconds, and maximum overshoot of 416%

Keywords: controlador *fuzzy*, simulador de voo, X-Plane, *fuzzy, c-means*

---

1. Introdução

A área da aviação naturalmente apresenta sistemas de controle dado o fato de que muitos aviões são naturalmente instáveis e necessitam de um computador de bordo. Ademais, há aeronaves que possuem um sistema de piloto automático capaz de realizar a sua guiagem [1].

A modelagem aeronáutica, apesar de ser bem conhecida, é um modelo complexo, que utiliza equações aerodinâmicas e métodos computacionais para a sua realização. Desse modo, os autores deste trabalho propõem a ideia de, sem nenhuma modelagem matemática, apresentar um controlador capaz de manter uma tendência de voo da aeronave mesmo sobre perturbações externas. Para isto, foi

trabalhada a técnica *fuzzy*, de modo a encontrar padrões na pilotagem de uma certa aeronave e então, com base em novos dados, poder inferir qual ação o sistema deve tomar para manter a referência estabelecida.

Sistemas de controle *fuzzy* tem sido o novo paradigma de controle automatizado desde a introdução de conjuntos *fuzzy* por L. A. Zadeh em 1965. Seu raciocínio pode ser resumido na declaração de Zadeh "À medida que a complexidade aumenta, declarações de necessidade perdem significado e declarações significativas perdem precisão". Assim, o controlador *fuzzy* é uma tentativa de enfrentar os desafios da crescente complexidade dos processos a serem controlados e das tarefas a serem resolvidas por sistemas de controle automatizado. [2]

O sistema de controle *fuzzy* pode ser uma alternativa vantajosa às técnicas de controle convencionais se:

- O processo a ser controlado exibe um comportamento não linear;
- Nenhum modelo matemático do processo está disponível porque o esforço de modelagem é inaceitavelmente alto ou o processo não é bem compreendido;
- O conhecimento especializado desempenha um papel fundamental no controle do processo e deve ser adquirido e usado para controle automático;
- Uma relação não linear multidimensional deve ser representada de forma que pode ser entendida e modificada facilmente.

Essa técnica de controle tem a vantagem de poder representar sistemas de modelagem matemática complexa via regras linguísticas baseadas na experiência de um operador ou então via categorização dos dados em *clusters* de pertinência. A grande diferença de um *cluster* de pertinência para um normal é que um valor novo não é categorizado totalmente em um agrupamento, podendo pertencer em partes de vários agrupamentos, o que deixa o resultado com uma transição mais suave. [3]

Dado o fato de que a modelagem aeronáutica era complexa e desconhecida para os autores e de que o setor aeronáutico possui uma demanda natural por estratégias de controle, decidiu-se utilizar a técnica *fuzzy* para conseguir controlar o ângulo de arfagem de uma aeronave.

2. Metodologia experimental

2.1. O simulador de voo X-Plane 11

O simulador de voo X-Plane 11 é uma plataforma de alta fidelidade em relação a modelos de voo, permitindo, também, uma grande interação e personalização dos utilizadores. A sua fidelidade de modelo de voo é reconhecida pela *FAA*, órgão máximo da aviação civil americana[4]. Além de apresentar um modelo de voo fidedigno, o X-Plane é a plataforma ideal para conseguir um interfaceamento simples e rápido com outras plataformas. Essa característica é extremamente desejável no contexto deste trabalho, dado o fato de que os autores tinham o propósito de realizar o sensoriamento e atuação nas condições de voo e controles, respectivamente.

O software X-Plane é desenvolvido pela *LaminarResearch*. A versão 11, a mais atual, foi lançada em 2016. O modelo de voo difere dos concorrentes por implementar a teoria de elementos laminares[5]. Esse modelo é similar ao cálculo de elementos finitos para forças aerodinâmicas, calculando as forças e momentos atuantes na aeronave pelo cálculo das partes individuais que compõem a aeronave. Outros simuladores utilizam tabelas preenchidas com dados empíricos, o que pode se traduzir em um comportamento fiel para aeronaves bem conhecidas e avaliadas, mas que não serve para novas aeronaves desenvolvidas ou desconhecidas – como aeronaves antigas e protótipos, por exemplo. Essas diferenças podem ser sumarizadas em um caso bem simples: uma cadeira poderia voar num simulador tradicional se fossem colocados dados fictícios em uma tabela de forças, porém jamais voaria no X-Plane dado o fato de que a estrutura não é aerodinâmica.

O X-Plane funciona com *datarefs* e *commands*[6]. Um *dataref* nada mais é que uma variável do sistema. Essas variáveis compreendem desde o valor atual da frequência do rádio VHF do avião, até

a altitude da aeronave. Além disso, existem outras variáveis que servem como um identificador do avião, como a distância da porta principal até o centro de gravidade. Em resumo, os *datarefs* servem para caracterizar aeronaves e também registrar o seu estado e os de seus sistemas. Os *commands*, como diz seu nome, nada mais são que comandos que podem ser enviados para o simulador. Dentre eles existem comandos operacionais, como mudar a câmera, pausar a simulação; e há também comandos no sentido aeronáutico, como abaixar *flaps* e recolher trem de pouso.

Existem ocasiões em que os *datarefs* e os *commands* se confundem. Para a situação do trem de pouso, existem os comandos baixar e levantar trem; e existe o *dataref* com a posição do trem. Dessa forma, nota-se que um comando representa uma ação, com os *datarefs* sendo o estado resultante. Nada disso impede que possa se realizar a escrita em certos *datarefs*. Caso uma falha no trem ocorresse, poderia-se injetar um valor correspondente ao meio do caminho da sua excursão. Para os comandos de voo, pode-se escrever na variável correspondente à posição do *joystick* nos seus 3 eixos, correspondendo ao leme, aileron e estabilizador horizontal. O leme atua na guinada, o aileron na rolagem e o estabilizador na arfagem, conforme Figura 1.

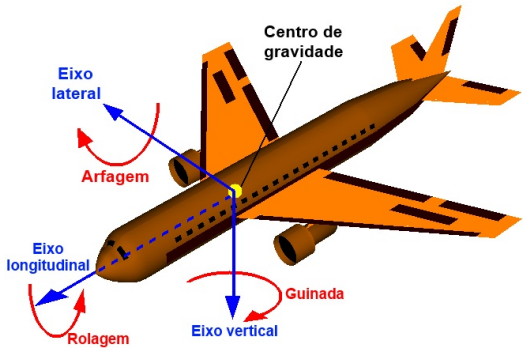

Figura 1. Eixos e momentos presentes em uma aeronave.
Fonte: http://canalpiloto.com.br/teorias-rotativas-05

Para realizar o controle do ângulo de arfagem, deve-se atuar no *joystick* para frente e para trás, variando o estabilizador. De modo a realizar esse comando, os autores seguiram a abordagem de escritura no *dataref* da arfagem do *joystick* via mensagens UDP, dado o caráter simples, eficaz e rápido da implementação. O *dataref* em questão é o *sim/controls/joystick/yoke_pitch_ratio*, variando de -1 a 1, onde valores negativos indicam nariz para baixo e valores positivos nariz para cima.[7]

2.2. *Aquisição de dados para o controlador*

Os dados de arfagem foram obtidos através da interface *socket* embutida no X-Plane. A implementação utilizada pode ser verificada no apêndice A. Após a definição dos *datarefs* e *commands* a serem enviados e recebidos pelo X-Plane, foi iniciado uma sequência de voos de treino, nos quais submeteu-se o avião a estímulos externos que alterariam seu ângulo de arfagem, seja pela definição manual no *dataref* correspondente ou pela simulação de eventos imprevisíveis (como por exemplo, uma tempestade). Foram então coletados o ângulo de arfagem da aeronave, a derivada da arfagem e a ação do usuário no *joystick* que corresponderia à normalização do sistema para seguir uma referência. Após a coleta, os dados foram normalizados, descartando-se os valores de erro de arfagem maiores que +-15 graus. Os dados obtidos seguem a forma apresentada na Tabela 1.

Tabela 1. Exemplo de dados capturados do simulador X-Plane 11

Erro Arfagem	Derivada do Erro	Posição do *Stick*
-28,74195	-3,30541	0,30054
-30,51777	-1,77582	0,30645
-34,32018	-3,80241	0,42801
-38,45888	-4,13870	0,43243
-42,94374	-4,48486	0,44581

Para manipulação dos dados, contou-se com o auxílio da biblioteca *pandas*, que permitiu a seleção eficaz do *range* dos dados. A implementação utilizada pode ser verificada no apêndice B.

A aeronave utilizada foi a *Cirrus Vision SF50*, que pode ser visualizada na Figura 2. Esta é uma aeronave do tipo monomotor *turbofan* e foi escolhida devido a ser uma aeronave de pequeno porte, potente e manobrável.[8]

Figura 2. Aeronave *Cirrus Vision SF50*.
Fonte: https://pt.wikipedia.org/wiki/Cirrus_Vision_SF50

2.3. Treinamento do Controlador

Através da utilização da biblioteca *skfuzzy* e de um sistema *fuzzy* criado pelos autores, foi desenvolvido um sistema de controle baseado nos dados coletados. O método *skfuzzy.cluster.cmeans* foi utilizado para criar os *clusters* do sistema, relacionando o erro e a derivada do erro do ângulo de arfagem, com a variação da posição do *stick* controlador da arfagem. A implementação utilizada pode ser verificada no apêndice C.

O uso de sete variáveis linguísticas para cada variável é justificado quando utiliza-se controladores *fuzzy* simples, isto é, controladores que que lidam com problemas como estabilização de pêndulos, por exemplo.

Uma vez os *clusters* identificados, projetou-se os mesmos na coordenada vertical (eixo Z), de modo a identificar os valores-chave para as funções de pertinência da saída. Os *clusters* foram, então, ordenados de 0 a 6 seguindo o critério de valor do eixo vertical crescente.

Uma função sigmóide é uma função cujo gráfico apresenta uma curva característica lembrando a letra "S". Possui forma real limitada, definida para todos os valores reais de entrada e possui apenas um ponto de inflexão, em x = 0. [9]. A definição matemática da função sigmóide está apresentada na equação 1.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Já uma função gaussiana é uma função cujo gráfico apresenta uma curva característica em forma de sino. Assim como a função sigmóide, possui forma real limitada, definida para todos os valores reais de entrada e possui apenas um ponto de inflexão.

A definição matemática da função gaussiana está apresentada na equação 2. Os parâmetros a, b e c representam constantes reais arbitrárias.

$$f(x) = a * e^{\frac{-(x-b)^2}{2c^2}} \tag{2}$$

Considerando essas definições e os respectivos gráficos e pontos de inflexão dessas funções, foram utilizadas funções de pertinência sigmoide e gaussiana criadas pelos autores, substituindo as funções tradicionais trapezoidal e triangular presentes na biblioteca *skfuzzy* a fim de criar uma transição mais suave no sistema, evitando solavancos. A implementação utilizada pode ser verificada no apêndice D.

Os clusters anteriormente projetados no eixo Z representam o que seriam os limites de pertinência máxima para cada *cluster*. Eles se traduzem no topo das funções gaussianas utilizadas nas funções centrais e no ponto de curvatura das funções sigmoides utilizadas nos extremos.

O treinamento procurou buscar a atuação do piloto em situações limites da aeronave, no entanto, estas situações não tendem a ocorrer naturalmente sem uma ação do próprio piloto. Deste modo, o sistema poderia aprender erroneamente que para um erro nulo ele deveria subir o nariz, quando, na verdade, o que se buscava era atingir uma situação de instabilidade.

Para contornar esse problema, prezou-se pela alteração brusca e forçada do ângulo de arfagem via modificação de *dataref*. Deste modo, conseguia-se fazer a aeronave sair da estabilidade (erro nulo) para os limites do sistema (+20 graus e -20 graus). Esta estratégia de reposicionamento foi replicada posteriormente para avaliação do sistema.

Propôs-se a utilização dos dados de treinamento da arfagem para atuação na rolagem apenas para manter a aeronave estabilizada neste eixo, permitindo uma atuação eficaz na variável controlada.

2.4. Atuação no sistema

Uma vez os *clusters* estabelecidos, para cada dado de entrada, utiliza-se o método *cmeans_predict* nos *clusters* projetados nos eixos horizontais para identificar a pertinência daquele dado novo.

Após, para cada um dos *clusters*, com base na pertinência dos dados de entrada para o *cluster* em questão, "ceifa-se"a função de pertinência do *cluster* via a operação *MIN* entre a pertinência calculada e a função de pertinência.

Após a realização desta etapa para todos os *clusters*, compõe-se a geometria final via agrupamento das geometrias (operação *MAX*) e, então, calcula-se a saída via método do centroide. A implementação utilizada pode ser verificada nos apêndices E e F.

Aqui revela-se a potência do método. Ao invés de inúmeras regras terem de ser estabelecidas para cobrir todas as combinações das entradas, a própria pertinência do dado de entrada em relação aos *clusters* já estabelece essas relações automaticamente. Isso permite um escalonamento do sistema para inúmeras variáveis de entrada, pois basta encontrar os grupamentos, calcular a pertinência dos dados para cada *cluster* e compor a figura final. Deste modo, seria possível, em um trabalho futuro, propor um modelo de controlador que levasse em questão outros sensores da aeronave, como de pressão atmosférica, velocidade, potência do motor, entre outros.

Para validação do sistema, observou-se o comportamento do controlador *fuzzy* para perturbações de +20 graus de arfagem e de -20 graus de arfagem, de modo a verificar se o sistema conseguia retomar uma referência de +10 graus de arfagem. Foram avaliados o tempo de estabilização, *overshoot* do sistema e erro de seguimento.

Utilizando a interface UDP com o simulador, os dados de navegação foram extraídos para o controlador a uma taxa de 0,5Hz. Para cada dado, o controlador verifica a pertinência do dado em

relação a cada *cluster* anteriormente definido, e define a saída defuzzificada do sistema a ser enviada para a entrada do simulador. A implementação utilizada pode ser verificada nos apêndices G e H.

3. Resultados

Os *clusters* obtidos para o sistema estão presentes na figura 3.

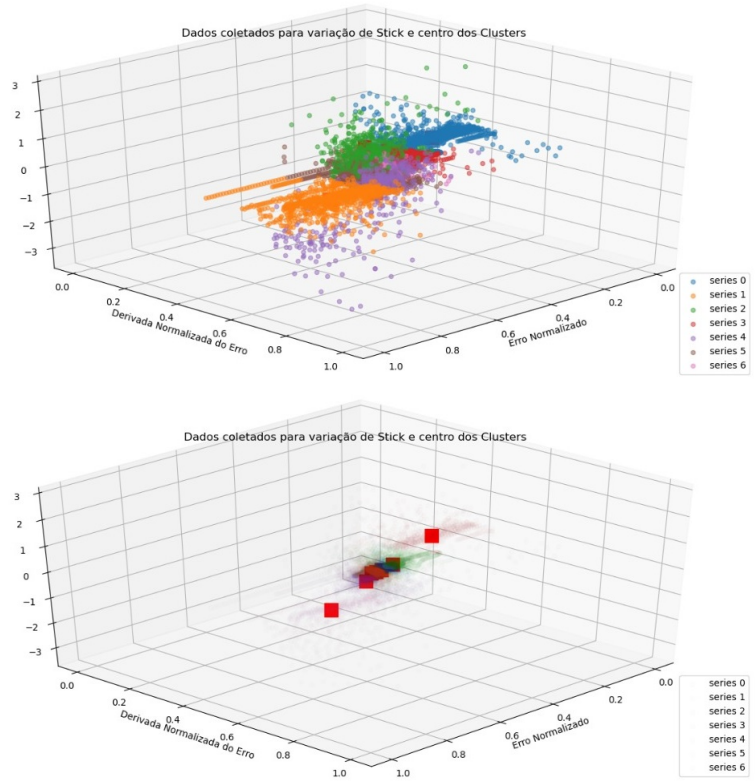

Figura 3. (**a**)*Clusters* obtidos no sistema. (**b**)Localização dos centroides.

Com as funções de pertinência resultantes da projeção dos *clusters* no eixo vertical presentes na Figura 4.

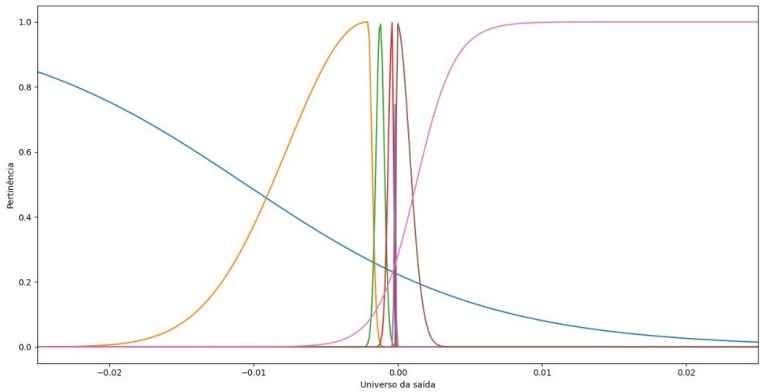

Figura 4. Função de pertinência para a saída do sistema.

O conjunto dos pontos das combinações das entradas e a saída do sistema resultante está mostrado na superfície de saída presente na figura 5. A superfície resultante foi considerada satisfatória, pois pode-se notar que quando ambos o erro e a derivada do erro são altos, o *stick* de saída é posicionado com um valor negativo alto. O oposto é verificado para valores baixos de erro e derivada do erro. A implementação utilizada pode ser verificada no apêndice I.

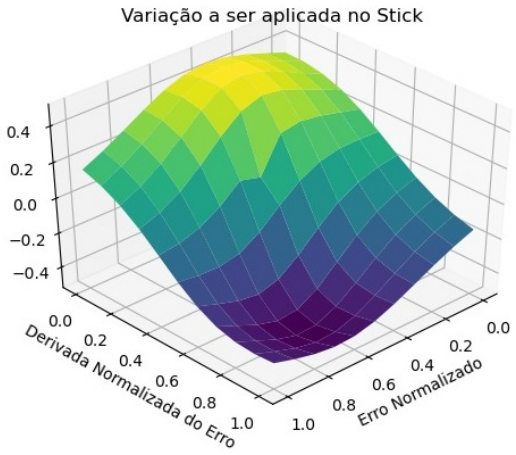

Figura 5. Saídas do sistema de controle resultante

O resultado do sistema para resposta às perturbações pode ser visto na Figura 6.

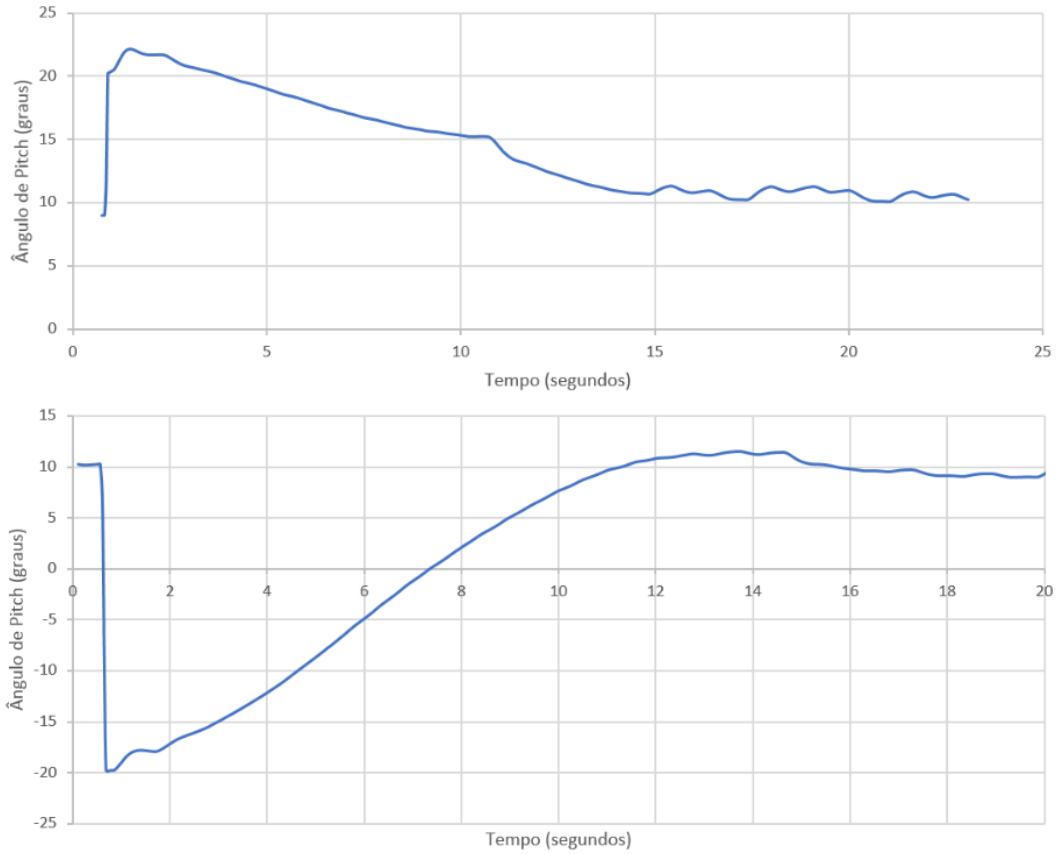

Figura 6. (**a**)Atuação do controlador para uma perturbação positiva no ângulo de arfagem. (**b**)Atuação do controlador para uma perturbação negativa no ângulo de arfagem.

Nota-se que, para uma perturbação positiva no ângulo de arfagem, o sistema atingiu a estabilidade inicial com erro de seguimento de 0,73 graus, tempo de estabilização de 13,28 segundos, e sem *undershoot*. Já para uma perturbação negativa, o sistema atingiu a estabilidade inicial com erro de seguimento de -0,55 graus, tempo de estabilização de 14,47 segundos, e com *overshoot* de 20%.

Os erros de seguimento também ficaram limitados a menos de 1 grau, que foi a resolução utilizada para seleção da referência.

Essas são características desejáveis, pois uma aeronave não pode apresentar uma resposta muito brusca nem *overshoot* elevado, pois isso se traduziria em desconforto para a tripulação, além de valores de acelerações g que podem danificar a estrutura de uma aeronave no mundo real.

Para fins de validação do controlador *Fuzzy*, foi feita a comparação dos resultados do mesmo utilizando um controlador proporcional. Este controlador foi idealizado utilizando a equação 3, de forma a escalonar o erro do ângulo, que varia de 0 a 1, para o universo de saída da posição do *stick* do ângulo de arfagem, que varia de -1 a 1. A constante proporcional K escolhida foi a unitária. Os resultados obtidos estão presentes na Figura 7.

$$stick\_pos = -K * (2 * Erro\_do\_angulo\_de\_arfagem - 1) \tag{3}$$

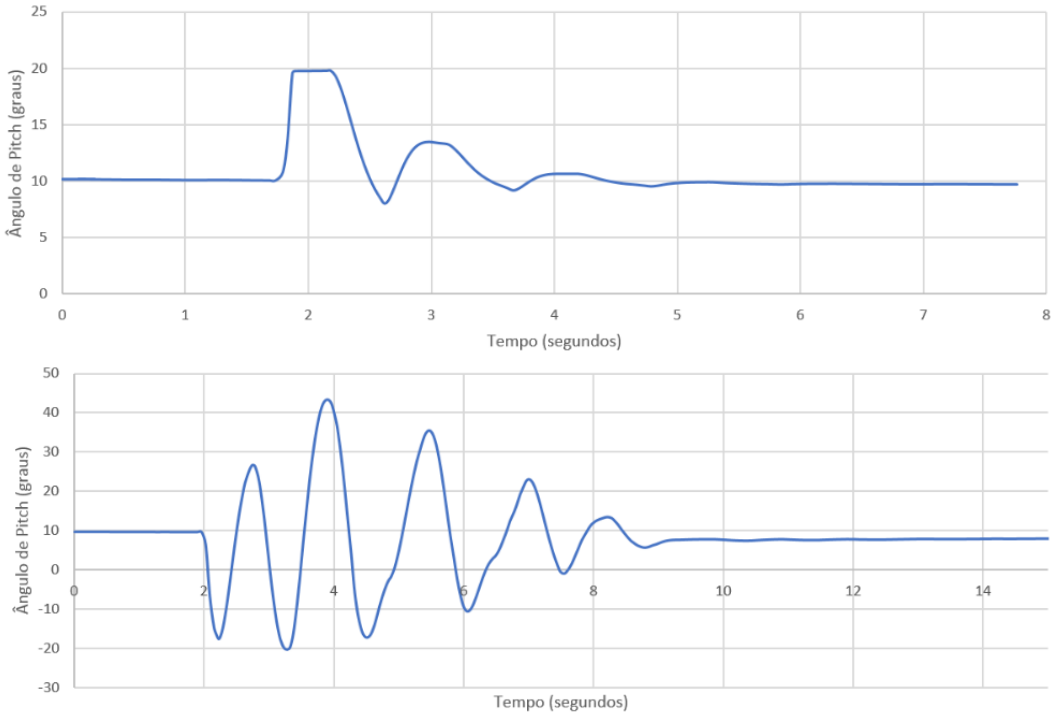

Figura 7. (**a**)Atuação do controlador para uma perturbação positiva no ângulo de arfagem. (**b**)Atuação do controlador para uma perturbação negativa no ângulo de arfagem.

Escolheu-se esse tipo de controlador pois o mesmo é de implementação bem simples e rápida, podendo ser um possível concorrente da estratégia *fuzzy*.

O controlador proporcional apresentou, para perturbação positiva, erro de seguimento de -0,15 graus, tempo de acomodação de 1,96 segundos e *undershoot* de 18%. Para perturbação negativa, o tempo de acomodação foi de 18,18 segundos, com erro de seguimento de -0,41 graus e *overshoot* de 416%.

Através da comparação entre o controlador *fuzzy* e o controlador proporcional, conclui-se que o controlador *fuzzy* possui uma resposta mais suave. O controlador proporcional aparente ser mais eficiente para perturbações positivas, porém a sua resposta mais rápida e brusca pode ser danosa aos tripulantes da aeronave. Além disso, para ambas as situações, foram apresentadas respostas oscilatórias indesejadas, que são eliminadas no sistema de controle *fuzzy*.

A oscilação mais pronunciada para perturbações negativas é resultado da tendência da aeronave de subir o nariz, assim, um movimento no *joystick* de subir o nariz é realimentado positivamente por essa tendência, fazendo com que a subida seja mais pronunciada, resultando num erro positivo mais elevado, que será traduzido em uma descida mais pronunciada, recomeçando o ciclo oscilatório.

4. Conclusão

Neste trabalho foi proposto e desenvolvido um controlador *fuzzy* de ângulo de arfagem para o simulador de voo X-Plane 11, cujos dados foram obtidos através da interface *socket* embutida no simulador. Após o treinamento do sistema de controle, o mesmo foi posto em prática para dados extraídos em tempo real do simulador, buscando normalizar perturbações no ângulo de arfagem da aeronave. Os resultados obtidos mostraram que o controlador *fuzzy* desenvolvido pelos autores obteve uma resposta satisfatória, conseguindo normalizar a perturbação induzida de forma suave e precisa, características extremamente importantes no âmbito aeronáutico, quando levados em consideração a saúde da tripulação e a estrutura da aeronave. Ao ser comparado com um controlador proporcional, foi demonstrado que o controlador *fuzzy* é extremamente superior, removendo possíveis respostas indesejadas do sistema. Para projetos futuros, sugere-se os seguintes tópicos de pesquisa:

- Realizar a comparação do controlador desenvolvido com outros tipos de sistemas de controle (PI, PD, PID, por exemplo), a fim de atestar a precisão do método desenvolvido;
- Implementar o controle do *roll* da aeronave, de forma a estabilizar a aeronave diante de tempestades com vento lateral, por exemplo;
- Implementar o controle de trajetória da aeronave, com o objetivo de permitir que a aeronave execute tarefas relacionadas com à navegação (por exemplo, seguir cartas aeronáuticas) com o mínimo de intervenções do usuário;
- Implementar o controlador desenvolvido em diferentes plataformas de hardware, para comparação do desempenho na velocidade de processamento e na transmissão de dados em relação à arquitetura apresentada neste trabalho.

Referências

1. Abzug, Malcolm; Larrabee, E.E. *Airplane stability and control: a history of the technologies that made aviation possible*; Cambridge Univ. Press, 2002.
2. Jaekel, J.; Mikut, R.; Bretthauer, G., Fuzzy Control Systems; 2004.
3. Klir, G. *Fuzzy sets and fuzzy logic : theory and applications*; Prentice Hall PTR: Upper Saddle River, N.J, 1995.
4. FAA-Certified X-Plane. https://www.x-plane.com/pro/certified/. Data de acesso: 8 de julho de 2020.
5. How X-Plane works. https://www.x-plane.com/desktop/how-x-plane-works/. Data de acesso: 8 de julho de 2020.
6. DataRefs Vs. Commands I: What's The Difference. https://developer.x-plane.com/2009/04/datarefs-vs-commands-i-whats-the-difference/. Data de acesso: 8 de julho de 2020.
7. Reference to X-Plane Datarefs. https://developer.x-plane.com/datarefs/. Data de acesso: 8 de julho de 2020.
8. Cirrus Vision SF50. https://en.wikipedia.org/wiki/Cirrus_Vision_SF50. Data de acesso: 8 de julho de 2020.
9. Mira, J. *From natural to artificial neural computation : International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995 : proceedings*; Springer-Verlag: Berlin New York, 1995.

Apêndice A Preparo dos sockets utilizados na comunicação com o simulador

```
# Creating Socket
UDP_IP = "127.0.0.1"
UDP_PORT = 49000
recSock = socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)
sendSock = socket.socket(socket.AF_INET,  socket.SOCK_DGRAM)

# Dataref subscribe
AGLDRef = "sim/cockpit2/gauges/indicators/pitch_AHARS_deg_pilot"
AGLDRefPadded = AGLDRef.ljust(400,"\0")
AGLDRefPadded = bytes(AGLDRefPadded,'utf-8')
```

```
RREFstr = 'RREF\0'
RREFstr = bytes(RREFstr,'utf-8')
requestAGL = RREFstr + pack('ii',2,1) + AGLDRefPadded

PRate = "sim/flightmodel/position/Q"
PRatePadded = PRate.ljust(400,"\0")
PRatePadded = bytes(PRatePadded,'utf-8')
requestPRate = RREFstr + pack('ii',2,2) + PRatePadded

Roll = "sim/cockpit2/gauges/indicators/roll_AHARS_deg_pilot"
RollPadded = Roll.ljust(400,"\0")
RollPadded = bytes(RollPadded,'utf-8')
requestRoll = RREFstr + pack('ii',2,3) + RollPadded

RRate = "sim/flightmodel/position/P"
RRatePadded = RRate.ljust(400,"\0")
RRatePadded = bytes(RRatePadded,'utf-8')
requestRRate = RREFstr + pack('ii',2,4) + RRatePadded

# Set pitch command
yawDREF = "sim/cockpit2/controls/yoke_pitch_ratio"
yawDREFPadded = yawDREF.ljust(500,"\0")
yawDREFPadded = bytes(yawDREFPadded,'utf-8')
DREFstr = 'DREF\0'
DREFstr = bytes(DREFstr,'utf-8')

JoyPPadded = yawDREF.ljust(400,"\0")
JoyPPadded = bytes(yawDREF,'utf-8')
requestJoyP = RREFstr + pack('ii',2,5) + JoyPPadded

# Set roll command
ailDREF = "sim/cockpit2/controls/yoke_roll_ratio"
ailDREFPadded = ailDREF.ljust(500,"\0")
ailDREFPadded = bytes(ailDREFPadded,'utf-8')
```

Apêndice B Método utilizado para coletar dados de treinamento

```
def getData():
    if firstRun:
        dataR = pd.read_excel('C:\\Workspace\\XPlane\\Dados Roll e Pitch 953.xlsx')
        dataR = dataR.drop(dataR.index[0:953]) #Removing n-first rows

        data = dataR #using new data
        #Selecting only pitch within range
        data = data.loc[(abs(data['   pitch,__deg '])) <= 15]
        data = data.loc[:,'ErroPitch':]

        datas = data.loc[:,['ErroPitch','dErroP','dCmdPitch']]

        datas.loc[:,'dErroP'] = datas.loc[:,'dErroP'].mul(-1)
```

```
        datasR = datas
        datas.to_pickle('datasetP.pkl')
        datasR.to_pickle('datasetR.pkl')
    else:
        datas = pd.read_pickle('datasetP.pkl')
        datasR = pd.read_pickle('datasetR.pkl')

    return datas, datasR
```

Apêndice C Método utilizado para gerar o sistema fuzzy

```
def generateAutoFuzzy(datas):
    #Calcula os clusters
    datasnp = datas.to_numpy()
    n_clusters = 7
    cntrError, uError, u0Error, dError, jmError, pError, fpcError =
        fuzz.cluster.cmeans(datasnp.T, c=n_clusters, m=2,
        error=0.005, maxiter=1000, init=None)
    #Ordena a coordenada Z dos clusters
    cntrStick = []
    for i in range (0,n_clusters):
        cntrStick.append(cntrError[i,2])
    cntrStick.sort()
    # Cria as funções de pertinencia da saída (Z)
    mfList = list()
    for i in range(0,n_clusters):
        if i == 0:
            mf = CSigMemberFunction(str(i),[cntrStick[i+1],cntrStick[i]])
        elif i == (n_clusters-1):
            mf = CSigMemberFunction(str(i),[cntrStick[i-1],cntrStick[i]])
            toto =3
        else:
            mf = CGauss2MemberFunction(str(i),
                [cntrStick[i-1],cntrStick[i],cntrStick[i+1]])
        mfList.append(mf)

        #Plots para debug
    fig3 = plt.figure()
    ax3 = fig3.add_subplot(111, projection='3d')

    ax3.set_title('Trained Model')

    cluster_membership = np.argmax(uError, axis=0)
    for j in range(n_clusters):
        ax3.scatter(datasnp[cluster_membership == j, 0],
                    datasnp[cluster_membership == j, 1],
                    datasnp[cluster_membership == j, 2], 'o',
                    label='series ' + str(j),alpha=0.5)

    for pt in cntrError:
        ax3.scatter(pt[0], pt[1],pt[2], marker='s',c='r',s=15**2)
```

```
    plt.xlabel('Erro Normalizado')
    plt.ylabel('Derivada Normalizada do Erro')
    plt.title('Dados coletados para variação de Stick e centro dos Clusters')
    plt.legend(loc='lower right')
    plt.show()

    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    x = np.arange(-1,1.0001,0.0001)
    i = 0
    for f in mfList:
        ax.plot(x,f(x))
    plt.title('Funções de Pertinência para a Saída')
    ax.set_ylabel('Pertinência')
    ax.set_xlabel('Universo da saída')
    ax.set_xlim([-0.025,0.025])
    plt.show()

    #Saída é lista com as funções de pertinencia
    Out = CFuzzyVar(mfList,[-1,1],'Stick')

    return Out,cntrError #retorna as funções de pertinencia e os Clusters
```

Apêndice D Definição da biblioteca Fuzzy desenvolvida pelos autores

```
class CDataFuzz:
    def _init_(self,ref,val):
        self._ref = ref
        self._val = val
    def val(self):
        return self._val
    def ref(self):
        return self._ref
    def set(self,ref,val):
        self._ref = ref
        self._val = val
    def _call_(self,input):
        return self._val(input)

class CGauss2MemberFunction:
    def _init_(self, name : str, Lims : List[float]):
        self._name = name
        self._low = Lims[0]
        self._med = Lims[1]
        self._high = Lims[2]
    def _call_(self,x):
        return gauss2mf(x,self._low,self._med,self._high)
    def name(self):
        return self._name
    def lims(self):
        return [self._low,self._med,self._high]
```

```
class CSigMemberFunction:
    def _init_(self, name : str, Lims : List[float]):
        self._name = name
        self._low = Lims[0]
        self._high = Lims[1]
    def _call_(self,x):
        return sigmf(x,self._low,self._high)
    def name(self):
        return self._name
    def lims(self):
        return [self._low,self._high]

class CTriMemberFunction:
    def _init_(self, name : str, triLims : List[float]):
        self._name = name
        self._low = triLims[0]
        self._med = triLims[1]
        self._high = triLims[2]
    def _call_(self,x):
        return trimf(x,self._low,self._med,self._high)
    def name(self):
        return self._name
    def lims(self):
        return [self._low,self._med,self._high]

class CTrapMemberFunction:
    def _init_(self, name : str, trapLims : List[float]):
        self._name = name
        self._low = trapLims[0]
        self._medLow = trapLims[1]
        self._medHigh = trapLims[2]
        self._high = trapLims[3]
    def _call_(self,x):
        return trapmf(x,self._low,self._medLow,self._medHigh,self._high)
    def name(self):
        return self._name

class CFuzzyVar:
    def _init_(self,funList : List[CTriMemberFunction],limits:List[float],name:str):
        self._memberFunctions = funList
        self._name = name
        self._limits = limits
    def _call_(self,x):
        y = list()
        for fn in self._memberFunctions:
            y.append(fn(x))
        return y
    def name(self):
        return self._name
```

```
    def lims(self):
        return self._limits
    def _getitem_(self,key):
        for fn in self._memberFunctions:
            if key == fn.name():
                return CDataFuzz(self,fn)
        return CDataFuzz(self,self._memberFunctions.end())

# função de pertinência triangular
def trimf (x, a , b , c ):

    y = list()
    if type(x) is np.ndarray: x = list(x)
    elif type(x) is not list: x = [x]
    else: x = list(x)
    for el in x:
        val = max(min((el-a)/(b-a),(c-el)/(c-b)),0.0)
        y.append(val)
    return np.array(y)

# função de pertinência trapezoidal
def trapmf(x,a,b,c,d):
    y = list()
    if type(x) is np.ndarray: x = list(x)
    elif type(x) is not list: x = [x]
    else: x = list(x)
    for el in x:
        val = max(min((el-a)/(b-a),1.0,(d-el)/(d-c)),0.0)
        y.append(val)
    return np.array(y)

# função de pertinência sigmoidal
def sigmf(x,down,top):
    w = 1/((top - down)/(x.max()-x.min()))
    c = np.minimum(top,down) + abs((top-down)/2)
    y = list()
    if type(x) is np.ndarray: x = list(x)
    elif type(x) is not list: x = [x]
    else: x = list(x)
    for el in x:
        try:
            den = 1 + math.exp(-w*(el-c))
        except OverflowError:
            den = float('inf')
        val = 1/(den)
        y.append(val)
    return np.array(y)

# função de pertinência gaussiana
def gauss2mf(x,a,b,c):
```

```
    s1 = float(abs(a-b))/3
    s2 = float(abs(b-c))/3
    y = list()
    if type(x) is np.ndarray: x = list(x)
    elif type(x) is not list: x = [x]
    else: x = list(x)
    for el in x:
        if el <= b:
            val = math.exp(-(el-b)*2/(2*s1*2))
        else:
            val = math.exp(-(el-b)*2/(2*s2*2))
        y.append(val)
    return np.array(y)
```

Apêndice E Cálculo da saída de um sistema fuzzy

```
def outCalc(outVar : CFuzzyVar, centroids, newNormData):

    #ordena os centroides pela coord Z
    centroids = centroids[centroids[:,2].argsort()]

    #realiza a predict com base na projecao nas dimensoes de entrada
    u, u0, d, jm, p, fpc = fuzz.cluster.cmeans_predict(
    newNormData.T, centroids[:,0:2], 2, error=0.005, maxiter=1000)

    x = np.arange(-1,1.001,0.001)
    #para cada cluster, com base na pertinencia, calcula a geometria da saída
    finalY = np.zeros_like(x)
    for i, pert in enumerate(u):
       mf = outVar[str(i)].val()
       pertLim = np.full_like(x,pert)
       y = mf(x)
       y = np.minimum(y,pertLim)
       finalY = np.maximum(finalY,y) #Compoe a saída via MAX

    defuzzedOut = defuzzCentroid(x.tolist(),finalY.tolist()) #Defuzzyfica a saída
    return defuzzedOut
```

Apêndice F Método utilizado para defuzzyficação dos centróides

```
def defuzzCentroid(x,y):
        x = np.array(x)
        y = np.array(y)
        num = np.sum(x*y)
        den = np.sum(y)
        if den == 0 and num == 0:
            return 0;
        out = num/den
        return out.tolist()
```

Apêndice G Método utilizado para leitura em tempo real do simulador

```
while (true):
    if run: #se está rodando
        #recebe dados
        datarec = receivedata()

        #define os erros e derivadas
        errop = datarec[0] - c_ref
        derrop = datarec[1]

        error = datarec[2]
        derror = datarec[3]

        if fuzzy: #se fuzzy

            #normaliza e calcula as saidas
            norm_inp = normalizenewdata([[errop,derrop]],datas)
            norm_inr = normalizenewdata([[error,derror]],datasr)
            commandp = outcalc(outvar, centroids, norm_inp)
            commandr = outcalc(outvarr, centroidsr, norm_inr)

            #incrementa comando conforme saída
            commandr = commandr*0.5 + oldcmdr
            commandp = commandp + oldcmd

            #evita saturação
            if commandp > 1:
                commandp = 1
            elif commandp < -1:
                commandp = -1

            if commandr > 1:
                commandr = 1
            elif commandr < -1:
                commandr = -1
        else: #se nao fuzzy, proporcional
            k = 1
            nerro= normalizenewdata([[errop,derrop]],datas)
            commandp = -k*(nerro[0][0]*2 - 1)
            #roll ainda é fuzzy
            norm_inr = normalizenewdata([[error,derror]],datasr)
            commandr = outcalc(outvarr, centroidsr, norm_inr)
            commandr = commandr*0.5 + oldcmdr
            #evita fuzzy
            if commandr > 1:
                commandr = 1
            elif commandr < -1:
                commandr = -1

        print(f'error | derror | command ||| ||| {errop} | {derrop} | {commandp} ')
```

```
        sendcommand(commandp,commandr) #envia os comandos
    try:  # used try so that if user pressed other than the given key error will not be shown
        if k.is_pressed('s'):  # if key 's' is pressed
            if run:
                #se rodando, para o controle e cancela recebimento dos dados
                requestprate = rrefstr + pack('ii',0,2) + pratepadded
                requestagl = rrefstr + pack('ii',0,1) + agldrefpadded
                requestroll = rrefstr + pack('ii',0,3) + rollpadded
                requestrrate = rrefstr + pack('ii',0,4) + rratepadded

                recsock.sendto(requestagl,(udp_ip,udp_port))
                recsock.sendto(requestprate,(udp_ip,udp_port))
                recsock.sendto(requestroll,(udp_ip,udp_port))
                recsock.sendto(requestrrate,(udp_ip,udp_port))
                run = false
                print("stopped")
        elif k.is_pressed('r'):
            if run == false:
                #se parado, roda e pede dados novamente ao xplane
                requestprate = rrefstr + pack('ii',2,2) + pratepadded
                requestagl = rrefstr + pack('ii',2,1) + agldrefpadded
                requestroll = rrefstr + pack('ii',2,3) + rollpadded
                requestrrate = rrefstr + pack('ii',2,4) + rratepadded

                recsock.sendto(requestagl,(udp_ip,udp_port))
                recsock.sendto(requestprate,(udp_ip,udp_port))
                recsock.sendto(requestroll,(udp_ip,udp_port))
                recsock.sendto(requestrrate,(udp_ip,udp_port))
                run = true
                print("running")
        elif k.is_pressed('f'):
            #alterna entre os controladores
            fuzzy = not fuzzy
            print(f'using fuzzy? {fuzzy}')
    except:
        pass
    #atualiza referencia, valores antigos e GUI
    c_ref = w1.get()
    oldcmd = commandp
    oldcmdr = commandr
    master.update_idletasks()
    master.update()
```

Apêndice H Métodos utilizados para I/O com o simulador

```
#método para receber dados
def receiveData():
    data = recSock.recv(1024)
    data = data[5:]
    test = len(data)
    id1, altitude, id2, PRate, id3, roll, id4, RRate = unpack('ifififif',data)
```

```
    if id1 == 1 and id2 == 2 and id3 == 3 and id4 == 4:
        out = [altitude,PRate, roll, RRate]

    return out

#método para enviar comandos ao xplane
def sendCommand(commandP, commandR):

    commandRoll = DREFstr + pack('f',commandR) + ailDREFPadded
    sendSock.sendto(commandRoll,(UDP_IP,UDP_PORT))

    commandYaw = DREFstr + pack('f',commandP) + yawDREFPadded
    sendSock.sendto(commandYaw,(UDP_IP,UDP_PORT))
```

Apêndice I Plotagem da superfície de saída

```
err = np.arange(0,1.1,0.1)
derr = err
z = list()
zR = list()
for elerr in err:
    for elderr in derr:
        result = outCalc(OutVar,centroids,np.array([[elerr,elderr]]))
        resultR = outCalc(OutVarR,centroidsR,np.array([[elerr,elderr]]))
        z.append(result)
        zR.append(resultR)

X,Y = np.meshgrid(err,derr)
Z = np.reshape(z,(int(1.1/0.1),int(1.1/0.1)))
ZR = np.reshape(zR,(int(1.1/0.1),int(1.1/0.1)))

fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.plot_surface(X, Y, Z,rstride=1,cstride=1,cmap='viridis',edgecolor='none')
plt.xlabel('Erro Normalizado')
plt.ylabel('Derivada Normalizada do Erro')
plt.title('Variação a ser aplicada no Stick')

plt.show()
```

Apêndice J TEMPLATE