Rastreamento de Trajetória de um Quadricóptero com Controle Robusto e Rede Neural Profunda ⋆

Paulo V. G. Simplício * **João R. S. Benevides** * **Roberto S. Inoue** ** **Marco H. Terra** *

* *Departamento de Engenharia Elétrica e de Computacão, Universidade de São Paulo, SP, (e-mails: paulogalvao@usp.br, jrsbenevides@usp.br, terra@sc.usp.br).*
** *Departamento de Computacão, Universidade Federal de São Carlos, SP, (e-mail: rsinoue@ufscar.br)*

Abstract: In this paper, a control architecture is proposed for the trajectory tracking problem of a quadrotor subject to parametric uncertainties and wind disturbances. The proposed architecture combines a deep neural network (DNN) with a recursive robust linear-quadratic regulator. The former method is responsible for modifying the reference signal sent to the controller in order to correct the quadrotor's trajectory, the latter approach deals with the parametric uncertainties associated with the aircraft model. A Gazebo-based 3D simulator was used to demonstrate the performance of the proposed architecture compared to the proportional-integral-derivative (PID) controller and the stand-alone robust regulator (without DNN assistance).

Resumo: Neste artigo, uma arquitetura de controle é proposta para o problema de rastreamento de trajetória de um quadricóptero sujeito a incertezas paramétricas e distúrbios de vento. A arquitetura proposta combina uma rede neural profunda com um regulador linear quadrático robusto (RLQR). Enquanto a primeira é responsável por modificar o sinal de referência enviado ao controlador com o objetivo de corrigir a trajetória do quadricóptero, o segundo lida com as incertezas paramétricas associadas ao modelo da aeronave. Um simulador 3D baseado no Gazebo foi utilizado para demonstrar o desempenho da arquitetura proposta em relação ao controlador proporcional-integral-derivativo (PID) e ao RLQR isolado (sem auxílio da rede neural).

Keywords: Robust control; Quadrotor; Deep learning; Disturbances; UAV.
Palavras-chaves: Controle robusto; Quadricóptero; Aprendizado profundo; Distúrbios; VANT.

1. INTRODUÇÃO

Nos últimos anos, houve um aumento considerável de pesquisas relacionadas ao desenvolvimento de técnicas de controle aplicadas em quadricóptero. Este aumento está diretamente relacionado com as características particulares que este tipo de aeronave apresenta. Sua construção, com hélices rotativas, fornece uma boa estabilidade em voo pairado e facilidade para decolar e pousar em locais com espaço limitado (Mahony et al., 2012). Essas particularidades combinadas com técnicas de controle avançadas possibilitaram a utilização do quadricóptero nos mais variados processos, desde filmagem e fotografia até aplicações militares severas (Valavanis e Vachtsevanos, 2014).

Dentre as técnicas de controle mais utilizadas, destacam-se as convencionais, classificadas em lineares e não lineares, e as técnicas de controle inteligente (Abdelmaksoud et al., 2020). Diferentemente das técnicas de controle convencionais, o controle inteligente pode se adaptar à falta de informações da planta e do ambiente de operação. Desse modo, pode-se operar em condições em que se tenha mudança nos parâmetros do processo e distúrbios com magnitudes desconhecidas (Liu, 2017). Neste caso, destacam-se: controle *fuzzy*, redes neurais artificiais (RNAs) e algumas variações do controle preditivo baseado em modelo (Kim et al., 2020).

A aplicação de RNAs em quadricópteros tem recebido uma crescente atenção por pesquisadores do campo da robótica e automação. Segundo Abdelmaksoud et al. (2020), utilizar RNAs em conjunto com controladores, lineares ou não, aumenta a robustez e a eficiência do sistema de controle proposto. Nesse contexto, alguns trabalhos utilizam, ainda, redes neurais profundas (DNNs) [1] por sua capacidade de aproximar funções não lineares complexas com desempenho superior. No trabalho de Sarabakha e Kayacan (2019), foi utilizada uma DNN para melhorar o rastreamento de trajetória de um quadricóptero. A rede proposta foi treinada de forma *offline*, com ajuda de um controlador PID; e *online*, utilizando lógica *fuzzy*. No trabalho de Varshney et al. (2019), uma DNN foi treinada para ser capaz de prover desempenho similar ao de um controlador preditivo baseado em modelo. Os resultados demonstraram um bom desempenho da rede implementada com redução do custo computacional.

⋆ Este trabalho foi realizado com apoio da CAPES Proc. 88887.509571/2020-00, FAPESP Proc. 2017/05668-0, CNPq Proc. 421131/2018-7, e do Instituto Nacional de Ciência e Tecnologia (Proc. FAPESP: 465755/2014-3 e Proc. CNPq: 2014/50851-0).

[1] Do inglês *Deep Neural Networks*.

O controle inteligente utilizando RNAs se mostrou capaz, também, de reduzir os efeitos causados por distúrbios de vento em quadricópteros. No trabalho de Sierra e Santos (2019), uma estratégia de controle inteligente que combina uma RNA com controlador PID é proposta para atenuar distúrbio de vento e variações na massa da aeronave. Já em Bellahcene et al. (2021), uma rede neural de base radial é utilizada em conjunto com um controlador $\mathcal{H}_\infty$, tanto para atenuar os erros de modelagem quanto para reduzir os efeitos do distúrbio de vento. De modo equivalente, é possível encontrar trabalhos na literatura que apresentam diferentes arquiteturas de RNAs para tratar este problema. Entretanto, é notável a deficiência de trabalhos que combinam DNNs com controladores robustos, fazendo-se necessária uma melhor exploração desta linha de pesquisa.

Com essa motivação, neste trabalho é proposta uma arquitetura de controle formada por um regulador linear quadrático robusto (RLQR) e uma DNN para controle de posição de um quadricóptero. A arquitetura proposta é inspirada no trabalho de Li et al. (2017), que combina um controlador proporcional-derivativo com uma DNN para melhorar o desempenho do quadricóptero em condições normais de voo. Aqui, distúrbios de vento, incertezas paramétricas e um modelo simplificado do quadricóptero serão considerados a fim de avaliar o desempenho da arquitetura proposta.

O RLQR utilizado baseia-se em parâmetros de penalidade e mínimos quadrados regularizados, fornecendo um algoritmo que não depende de parâmetros de ajuste em aplicações *online*, apenas das matrizes de ponderação e incertezas previamente definidas. A DNN, a partir da trajetória desejada fornecida, tem o objetivo de produzir uma trajetória de referência factível para o controlador com base em experiências passadas. Além disso, adiciona uma capacidade adaptativa à arquitetura do controle, sendo fundamental para otimizar o desempenho do controlador durante o seguimento de trajetória.

2. ARQUITETURA DE CONTROLE BASEADA EM DNN E RLQR

O problema de rastreamento de trajetória de quadricópteros é um dos ramos mais importantes no campo da robótica aérea. O principal objetivo consiste em implementar técnicas de controle, de modo que um quadricóptero possa seguir uma trajetória predefinida. Quando sujeito a distúrbios de vento, diferentes técnicas podem ser incorporadas à arquitetura de controle como alternativa para reduzir o erro de trajetória. A arquitetura proposta neste trabalho é apresentada na Figura 1, sendo o RLQR o responsável pelo rastreamento de trajetória e a DNN responsável por modificar o sinal de referência que é passado ao controlador, melhorando o desempenho durante o seguimento de trajetória. O quadricóptero escolhido é o Parrot$^{\text{TM}}$ Bebop 2.0 pela sua facilidade de integração com o sistema ROS (*Robot Operating System*).

2.1 Modelo Matemático para o Parrot^{TM} Bebop 2.0

O quadricóptero possui características que tornam sua modelagem matemática complexa. Contudo, visando facilitar esta tarefa, um modelo dinâmico simplificado foi proposto

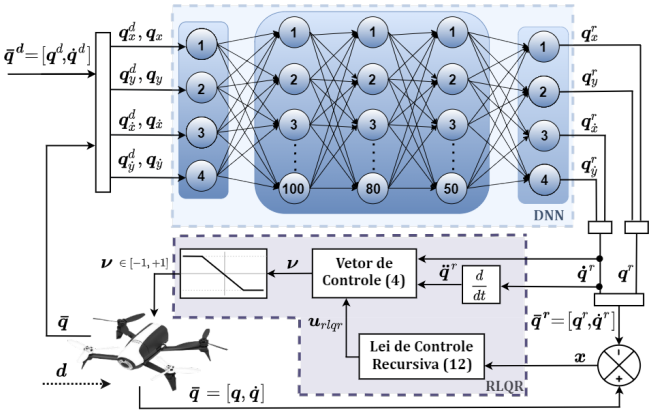


Figura 1. Arquitetura de controle baseada em DNN e RLQR.

em Santana et al. (2014) para um Parrot$^{\text{TM}}$AR.Drone e adaptado para o Bebop 2.0 em Benevides et al. (2019a).

O Bebop 2.0 é controlado com uma entrada de controle do tipo $\nu = [\nu_{\dot{x}} \ \nu_{\dot{y}} \ \nu_{\dot{z}} \ \nu_{\dot{\psi}}]^T$, em que ν representa os comandos de velocidade linear nos eixos x, y e z, além da velocidade angular em torno do eixo z, representada por $\dot{\psi}$. Considerando que a estabilização de atitude é garantida pelo sistema de controle interno do Bebop 2.0, um modelo simplificado é dado por:

$$\ddot{q}(t) = \Lambda \dot{q}(t) + \Gamma \nu(t), \tag{1}$$

em que $\dot{q} = [q_{\dot{x}} \ q_{\dot{y}} \ q_{\dot{z}} \ q_{\dot{\psi}}]^T$, $\Lambda \in \mathbb{R}^{4\times4}$ e $\Gamma \in \mathbb{R}^{4\times4}$ são matrizes contendo parâmetros a serem identificados.

A fim de realizar o rastreamento de trajetória, o modelo pode ser reescrito em função do espaço de estado do erro, onde o controlador atuará. Considerando o sinal $\bar{q}^r$ fornecido pela DNN (Figura 1), temos:

$$x(t) = \begin{bmatrix} \dot{q}(t) - \dot{q}^r(t) \\ q(t) - q^r(t) \end{bmatrix}, \tag{2}$$

como sendo o erro de trajetória. Dessa forma, o espaço de estado do erro, considerando distúrbios externos, é dado por:

$$\dot{x}(t) = \underbrace{\begin{bmatrix} \Lambda & 0 \\ I & 0 \end{bmatrix}}_{A} x + \underbrace{\begin{bmatrix} I \\ 0 \end{bmatrix}}_{B} u + \underbrace{\begin{bmatrix} \Xi \\ 0 \end{bmatrix}}_{B_d} d, \tag{3}$$

em que $\Xi = diag(1/m\, I_{3\times3}, \ 1/I_z I_{1\times1})$ é a matriz que mapeia distúrbios externos, com m sendo a massa do quadricóptero e I_z o momento de inércia com respeito ao eixo Z; d é o distúrbio externo que afeta o quadricóptero e u é utilizado para calcular a entrada de controle real enviada ao Bebop 2.0, dada por:

$$\nu = \Gamma^{-1}(u + \ddot{q}^r - \Lambda \dot{q}^r). \tag{4}$$

Para implementação do controlador, podemos discretizar (3) usando o método de Euler, resultando em:

$$x_{i+1} = F_i x_i + G u_i + G_d d_i, \tag{5}$$

em que $F_i = I + TA(t)$, $G = TB$, $G_d = TB_d$, e T é o tempo de amostragem.

2.2 Regulador Linear Quadrático Robusto - RLQR

No trabalho de Terra et al. (2014), foi desenvolvido um regulador linear quadrático robusto para sistemas lineares de tempo discreto sujeitos a incertezas paramétricas. Este regulador se baseia em parâmetros de penalidade e mínimos quadrados regularizados, fornecendo um algoritmo que não depende do ajuste de nenhum parâmetro auxiliar, o que facilita o uso em aplicações *online*.

Considere o modelo linear de tempo discreto com incertezas paramétricas no espaço de estados:

$$x_{i+1} = (F_i + \delta F_i)x_i + (G_i + \delta G_i)u_i; \quad i = 0, ..., N, \quad (6)$$

no qual $x_i \in \mathbb{R}^n$ é o vetor de estado, $u_i \in \mathbb{R}^m$ é a entrada de controle, $F_i \in \mathbb{R}^{n\times n}$ e $G_i \in \mathbb{R}^{n\times m}$ são matrizes de parâmetros nominais com dimensão apropriada do sistema e N é um inteiro que define o número de iterações. O estado inicial x_0 é constante e conhecido, e as matrizes $\delta F_i \in \mathbb{R}^{n\times n}$ e $\delta G_i \in \mathbb{R}^{n\times m}$ são matrizes de incertezas desconhecidas, modeladas como:

$$[\delta F_i \;\; \delta G_i] = H_i \Delta_i [E_{F_i} \;\; E_{G_i}]; \quad i = 0, ..., N, \quad (7)$$

sendo $H_i \in \mathbb{R}^{n\times k}$, $E_{F_i} \in \mathbb{R}^{n\times n}$ e $E_{G_i} \in \mathbb{R}^{l\times m}$ são matrizes conhecidas e $\Delta_i \in \mathbb{R}^{k\times l}$ uma matriz arbitrária com $\|\Delta_i\| \leq 1$. O RLQR é obtido através da solução do seguinte problema de otimização:

$$\min_{x_{i+1}, u_i} \max_{\delta F_i, \delta G_i} \{\tilde{J}_i^u(x_{i+1}, u_i, \delta F_i, \delta G_i)\}, \quad (8)$$

sendo $\tilde{J}_i^u$ o funcional de custo quadrático regularizado, definido como:

$$\begin{aligned}
&\tilde{J}_i^\mu(x_{i+1}, u_i, \delta F_i, \delta G_i) = \\
&= \begin{bmatrix} x_{i+1} \\ u_i \end{bmatrix}^T \begin{bmatrix} P_{i+1} & 0 \\ 0 & R_i \end{bmatrix} \begin{bmatrix} x_{i+1} \\ u_i \end{bmatrix} + \\
&+ \left\{ \left(\begin{bmatrix} 0 & 0 \\ I & -G_i \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & -\delta G_i \end{bmatrix} \right) \begin{bmatrix} x_{i+1} \\ u_i \end{bmatrix} - \right. \\
&\left. - \left(\begin{bmatrix} -I \\ F_i \end{bmatrix} x_i + \begin{bmatrix} 0 \\ \delta F_i \end{bmatrix} x_i \right) \right\}^T \begin{bmatrix} Q_i & 0 \\ 0 & \mu I \end{bmatrix} \left\{ \bullet \right\}, \quad (9)
\end{aligned}$$

em que $P_{i+1} \succ 0$, $Q \succ 0$, pondera os estados, $R \succ 0$, pondera as entradas de controle e $\mu > 0$ é um parâmetro de penalidade fixo, responsável pela garantia de permanência da igualdade em (6).

De acordo com Terra et al. (2014) o RLQR se baseia na solução $(x_{i+1}^*(\mu), u_i^*(\mu))$ do problema de otimização $min{-}max$, no qual o intuito é obter a menor magnitude do estado e a menor magnitude de ação de controle nos piores casos de incertezas paramétricas para cada parâmetro de penalidade $\mu > 0$. A solução ótima pode ser encontrada de forma recursiva através de (10) e (11),

$$\begin{bmatrix} x_{i+1}^*(\mu) \\ u_i^*(\mu) \\ \tilde{J}_i^\mu(x_{i+1}^*(\mu), u_i^*(\mu)) \end{bmatrix} = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & x_i^*(\mu)^T \end{bmatrix} \begin{bmatrix} L_i, \mu \\ K_i, \mu \\ P_i, \mu \end{bmatrix} x_i^*(\mu) \quad (10)$$

com

$$\begin{aligned}
\begin{bmatrix} L_i \\ k_i \\ P_i \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & I \\ 0 & 0 & -I & \mathcal{F}_i^T & 0 & 0 \end{bmatrix} \\
&\times \begin{bmatrix} P_{i+1}^{-1} & 0 & 0 & 0 & I & 0 \\ 0 & R_i^{-1} & 0 & 0 & 0 & I \\ 0 & 0 & Q_i^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sum_i (\mu, \lambda_i) & \mathcal{I} & -\mathcal{G}_i^T \\ I & 0 & 0 & \mathcal{I} & 0 & 0 \\ 0 & I & 0 & -\mathcal{G}_i^T & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ -I \\ \mathcal{F}_i \\ 0 \\ 0 \end{bmatrix}
\end{aligned} \quad (11)$$

sendo

$$\Sigma_i = \begin{bmatrix} \mu^{-1} I - \hat{\lambda}_i^{-1} H_i H_i^T & 0 \\ 0 & \hat{\lambda}_i^{-1} I \end{bmatrix}, \qquad \mathcal{I} = \begin{bmatrix} I \\ 0 \end{bmatrix},$$

$$\mathcal{G}_i = \begin{bmatrix} G_i \\ E_{G_i} \end{bmatrix}, \; \mathcal{F}_i = \begin{bmatrix} F_i \\ E_{F_i} \end{bmatrix}.$$

Além disso, quando o parâmetro de penalidade $\mu \to \infty$, a robustez do controlador é alcançada. A lei de controle ótima é dada por:

$$u_i^* = K_i x_i^*. \quad (12)$$

Os detalhes sobre a garantia de convergência e estabilidade podem ser encontrados em Terra et al. (2014).

2.3 Rede Neural Profunda - DNN

Redes neurais profundas integram o campo do aprendizado de máquina dentro da grande área da inteligência artificial. São compostas por múltiplas camadas de processamento com capacidade de aprender grandes representações de dados com vários níveis de abstração (Chollet, 2017). De forma sumária, DNNs se diferenciam de redes neurais comuns, principalmente, pelo número de camadas ocultas e pela capacidade de tratar uma grande quantidade de dados não estruturados.

O objetivo principal da DNN, neste trabalho, é mapear um sinal de referência ($\bar{q}^r$) a cada instante e enviar ao controlador, baseado nos valores da trajetória desejada ($\bar{q}^d$) e atual ($\bar{q}$) do quadricóptero. Para isto, considerando o método de aprendizado supervisionado, a rede é treinada utilizando a trajetória atual do quadricóptero $\{(\bar{q}_i, \bar{q}_{i+1})\}$ como entrada e a trajetória desejada $\bar{q}^d = \{q_x^d, q_y^d, q_x^d, q_y^d\}$ como rótulo. O aprendizado é realizado de forma *offline* e o conhecimento adquirido pela rede é aplicado *online* junto ao RLQR.

Durante o funcionamento (Figura 1), a rede obtém informações tanto do gerador de trajetória como do sistema de odometria do quadricóptero. A entrada da rede passa a ser composta pelo vetor $\{(\bar{q}_i, \bar{q}_i^d)\}$ e a saída gerada para uma trajetória circular, em que z e ψ são iguais a zero, é composta pelo vetor $\bar{q}^r = \{q_x^r, q_y^r, q_x^r, q_y^r\}$. A ideia é que se a trajetória atual for igual à trajetória desejada, a rede deverá fornecer esse valor de referência guardado para alcançar o rastreamento com menor erro possível. Sendo assim, quando a trajetória do quadricóptero for diferente da desejada, o mapeamento aprendido pela rede irá gerar um sinal de referência factível para o controlador, visando corrigir o curso do quadricóptero.

Os detalhes correspondentes às características da rede, assim como os algoritmos e as ferramentas utilizadas para treinamento, são explorados na Seção 3.4.

3. METODOLOGIA

Nesta seção, são apresentados os detalhes e as ferramentas utilizadas para implementação da arquitetura proposta.

3.1 Sistema ROS

O ROS (Quigley et al., 2009) é uma plataforma *open source* constituída de uma série de ferramentas que facilita a programação de diferentes tipos de robôs. Seu funcionamento acontece através da integração entre diferentes nós que carregam informações sobre o sistema em questão. Esses nós podem representar, por exemplo, sensores ou atuadores, e as mensagens geradas por cada um deles são publicadas em tópicos acessíveis a outros nós. Dessa forma, é possível ler as informações geradas por sensores e enviar comandos para atuadores de maneira prática.

3.2 Simulador Parrot-Sphinx

O *Parrot-Sphinx* é um simulador desenvolvido pela empresa Parrot$^{\text{TM}}$, fabricante oficial do Bebop. Atualmente é disponibilizado em código aberto e permite que o usuário controle o Bebop 2.0 em diversos ambientes, utilizando os recursos físicos e visuais do Gazebo. Além de simular a dinâmica do Bebop 2.0 com fidelidade, permite configurar as características físicas do ambiente, como velocidade linear do vento nos eixos x, y e z. Desta maneira, a utilização deste simulador permite desenvolver sistemas funcionais para o mundo real, visto que os algoritmos desenvolvidos para o Bebop 2.0 no *Parrot-Sphinx* podem ser aplicados da mesma forma no modelo da aeronave real.

3.3 Trajetória Desejada

A trajetória desejada utilizada nos experimentos possui um formato circular no plano xy com uma altura definida em $z = 1$ m e raio de $0,3$ m. Um nó do ROS é responsável por publicar no tópico `/bebop/waypoint` tanto a posição quanto a velocidade desejada a uma frequência de 50 Hz. Esse tópico, acessível ao nó de controle e à DNN, disponibiliza $q^d = [q^d_x, q^d_y, q^d_z, q^d_\psi]^T$ e $\dot{q}^d$ da seguinte forma:

$$q^d = \left[A\cos\omega t \;\; A\sin\omega t \;\; 0 \;\; 0\right]^T,$$
$$\dot{q}^d = \left[-\omega A\sin\omega t \;\; \omega A\cos\omega t \;\; 0 \;\; 0\right]^T,$$

em que $\omega = 0,5$ rad/s é a velocidade angular, dada pela razão entre a velocidade média $v = 0,15$ m/s e o raio $A = 0,3$ m.

3.4 Características e Treinamento da DNN

Além das camadas de entrada e saída, a arquitetura da rede é composta por três camadas ocultas contendo 100, 80 e 50 neurônios, respectivamente. Tanto o número de camadas quanto o número de neurônios em cada uma delas foram escolhidos de forma empírica. Para isto, foram realizadas uma série de treinamentos em que buscou-se ajustar a rede com base no desempenho apresentado para a tarefa proposta. Dessa forma, foi possível alcançar a arquitetura apresentada na Figura 1.

O treinamento foi realizado utilizando o modo de aprendizado supervisionado através do fornecimento de um conjunto de amostras rotuladas. Para criação do *dataset*, foram realizados voos com a aeronave utilizando o RLQR. Desse modo, foi possível obter as amostras de dados com informações da trajetória atual e desejada do quadricóptero a cada instante de tempo. No total foram obtidas 2000 amostras para compor o *dataset*, sendo que 90% destas foram utilizadas para a etapa de treinamento e 10% para a etapa de testes.

A partir da obtenção das amostras, o conjunto de treinamento da rede foi formado por $\{(\bar{q}_i, \bar{q}_{i+1}),\ \bar{q}^d_i\}$, em que $\bar{q}_i$ e $\bar{q}_{i+1}$ formam a entrada de treinamento da rede e representam a trajetória do quadricóptero no instante i e $(i+1)$, respectivamente; e $\bar{q}^d$ representa seu correspondente dado de saída (rótulo). Note que, $\{(\bar{q}_i, \bar{q}_{i+1})\} \rightarrow \bar{q}^d_i$ (treinamento) é um mapeamento aproximado de $\{(\bar{q}_i, \bar{q}^d_i)\} \rightarrow \bar{q}^r_i$ (Figura 1). Dessa forma, quando a trajetória atual do quadricóptero for igual à desejada, a rede entende que o rastreamento está sendo perfeito e $\bar{q}^r_i$ será igual a $\bar{q}^d_i$. Contudo, quando a trajetória atual do quadricóptero for diferente da trajetória desejada, a rede fornece um sinal de referência com base em experiências passadas, com o intuito de fazer com que $\bar{q}_i$ convirja para $\bar{q}^d_i$.

Para desenvolver a DNN apresentada, foi utilizada a linguagem Python através do *framework Keras* com *backend* em *Tensorflow*. A Tabela 1 apresenta os parâmetros utilizados para o treinamento.

Tabela 1. Parâmetros de treinamento da rede.

Funções de ativação	ReLU
Função de perda	Erro quadrático médio (EQM)
Algoritmo de otimização	*RMSprop*
Taxa de aprendizagem	0,008
Inicialização dos pesos	*Xavier Initialization*
Inicialização dos *biases*	0
Número máximo de épocas	20
Tamanho do *batch*	32

A escolha da função de ativação (ReLU) se dá por esta ser uma das mais eficientes no treinamento de redes neurais profundas. Já em relação ao algoritmo de otimização, além do *RMSprop* utilizado, foram testados o gradiente descendente estocástico e o *adaptive moment estimation* (Adam). Para o problema tratado neste artigo, notou-se um melhor desempenho da rede após o treinamento com o algoritmo *RMSprop*. Os demais parâmetros de treinamento da rede foram ajustados de forma empírica, buscando encontrar o melhor desempenho sem que ocorresse sobreajuste dos pesos sinápticos.

Com os parâmetros apresentados na Tabela 1, verificou-se que a rede levou cerca de 1125 iterações para convergir. A métrica utilizada para avaliação foi o erro absoluto médio (EAM), em que representa o valor absoluto da diferença entre a predição da rede e o conjunto de amostras rotuladas. Após o treinamento, a rede foi avaliada com o *dataset* de teste e com voos experimentais utilizando

o *Parrot-Sphinx*, demonstrando desempenho satisfatório para o problema tratado neste trabalho.

3.5 Comunicação Entre os Nós do ROS

Para implementação dos controladores e geração da trajetória desejada, foi utilizado o sistema *ROS* na versão *Kinetic Kame* com os pacotes *bebop_autonomy* [2] e *drone_dev* (Benevides et al., 2019a) instalados. Para funcionamento do sistema, foram necessários 4 nós do ROS atuando em conjunto. São eles:

- *drone_driver*: responsável pela comunicação com o Bebop 2.0;
- *drone_dev*: nó onde são implementados os controladores, como o RLQR e PID, cuja lei de controle é enviada à aeronave;
- *traj_planner*: responsável por enviar a trajetória desejada q^d e $\dot{q}^d$ a cada instante de tempo;
- *traj_network*: responsável por enviar um sinal de referência q^r e $\dot{q}^r$ ao quadricóptero a cada instante de tempo. Este nó comporta a DNN treinada.

Vale ressaltar que para o treinamento da rede neural e implementação da arquitetura de controle proposta, foi utilizado um *notebook* Dell Inspiron 14 7460 com sistema operacional Ubuntu na versão 16.04 LTS. O *hardware* utilizado possui processador Intel Core i7-7500U com 2 núcleos de até 3,5GHz, memória RAM de 16 GB DDR4 e placa de vídeo Nvidia Geforce 940MX.

4. RESULTADOS

Com intuito de verificar a eficiência da arquitetura proposta, foram realizados voos no ambiente *Parrot-Sphinx* com e sem a aplicação de distúrbio de vento na aeronave. Para efeito de comparação, os resultados da arquitetura proposta foram comparados com os resultados dos controladores PID padrão e RLQR isolado (i.e. sem auxílio da DNN). Vale ressaltar que, neste artigo, os detalhes referentes ao funcionamento do controlador PID foram omitidos devido a sua consolidação na literatura e por este servir apenas como comparação. A trajetória de referência enviada possui formato circular (Seção 3.3) e os parâmetros desconhecidos das matrizes $\Lambda \in \mathbb{R}^{4\times4}$ e $\Gamma \in \mathbb{R}^{4\times4}$ do modelo do Bebop 2.0 (1) foram previamente identificados no trabalho de Benevides et al. (2019b).

4.1 Parâmetros dos Controladores

Os parâmetros de controle do RLQR foram definidos heuristicamente a partir de análises em uma série de voos experimentais. Já para o controlador PID, foi utilizado o segundo método de Ziegler-Nichols como ponto de partida para definição das matrizes de ganho. Os valores definidos para ambos os controladores são apresentados a seguir.

a) Parâmetros do RLQR

$$Q = diag\,(1,5;\ 1,4;\ 1;\ 1;\ 1;\ 1;\ 10;\ 5), \qquad R = 0,15\,I_{4\times4},$$

$$E_F = [E_{F_1}\ E_{F_2}], \quad E_{F_1} = diag(0,442;\ 0,413;\ 0,295;\ 0,295),$$

$$E_{F_2} = diag\,(0,285;\ 0,285;\ 2,85;\ 1,425),\ E_G = 0,0187\,I_{4\times4}.$$

[2] http://wiki.ros.org/bebop_autonomy - 2015.

b) Parâmetros do PID

$$K_p = diag(0,25;\ 0,25;\ 0,75;\ 1),$$
$$K_i = diag(0,001;\ 0,001;\ 0,01;\ 0),$$
$$K_d = diag(0,5;\ 0,5;\ 0,25;\ 1).$$

4.2 Índices de Desempenho

A fim de realizar uma análise estatística do desempenho da arquitetura proposta e de cada controlador implementado, foi utilizada a norma ℓ_1 e o erro médio absoluto para as variáveis de posição controladas, sendo:

$$E(i) = \|q_i - q_i^d\|_1, \tag{13}$$

a norma ℓ_1, e

$$\overline{E}(i) = \frac{1}{N}\sum_{i=1}^{N} E(i), \tag{14}$$

o erro médio absoluto, em que q_i e q_i^d são os vetores de posição e posição desejada, respectivamente; i corresponde ao valor de uma iteração e N é a quantidade de iterações.

Para calcular o percentual de melhoria da arquitetura proposta em relação aos controladores RLQR isolado e PID, e o percentual de melhoria do RLQR isolado em relação ao controlador PID, foi utilizado o seguinte índice:

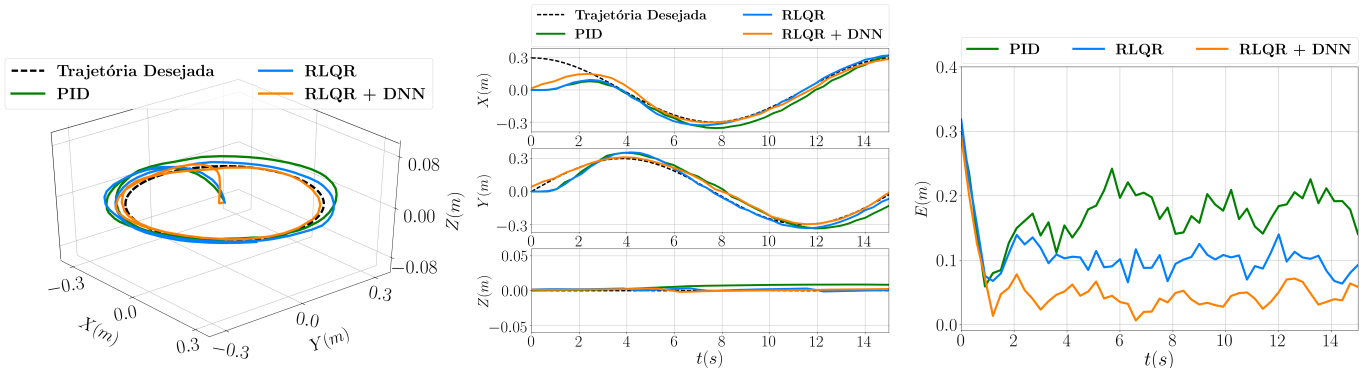
$$I^{\%} = \left(1 - \frac{\overline{E}}{\overline{E}_{ref}}\right) \times 100\,, \tag{15}$$

em que $\overline{E}$ é o erro médio absoluto tanto para a arquitetura proposta quanto para o RLQR isolado e $\overline{E}_{ref}$ é o erro médio absoluto para RLQR isolado e PID.

4.3 Análise dos Resultados

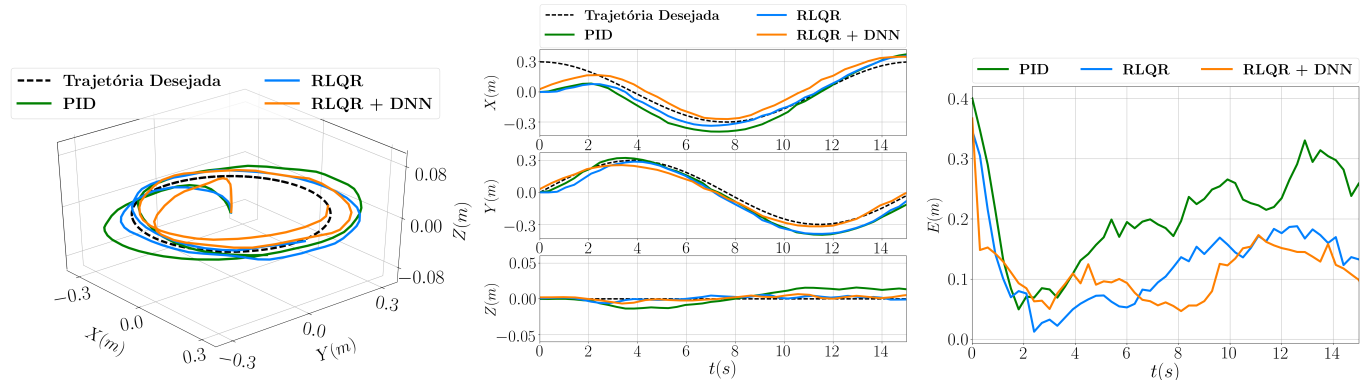
Os resultados apresentados a seguir compreendem dois conjuntos de experimentos. No primeiro, foi analisado o desempenho da arquitetura RLQR + DNN e dos controladores RLQR e PID sem a aplicação de distúrbio de vento. No segundo, tanto a arquitetura proposta quanto os controladores são avaliados com a aplicação de distúrbio de vento constante durante todo o tempo da trajetória. O distúrbio foi configurado com uma velocidade média de 5 m/s ao longo do eixo x do quadricóptero.

a) Desempenho para Condições Normais de Voo: A Figura 2 apresenta o desempenho da arquitetura proposta frente aos controladores RLQR isolado e PID para condições normais de voo. A vista 3D do rastreamento de trajetória é apresentada na Figura 2(a), seguida das variáveis de posição controladas (x, y, z) na Figura 2(b) e da evolução do erro apresentada na Figura 2(c). Para este caso, pode-se perceber que o conjunto proposto RLQR + DNN apresentou um melhor desempenho no problema de seguimento de trajetória em relação aos demais controladores. Já o controlador PID foi o que apresentou o pior desempenho. Isto pode ser confirmado através de uma análise dos índices de desempenho apresentados na Tabela 2. Neste caso, a arquitetura RLQR + DNN apresentou o melhor desempenho frente a todos os outros índices avaliados, tendo um percentual de melhoria em relação



(a) Vista 3d. (b) Variáveis de posição controladas. (c) Erros de posição.

Figura 2. Rastreamento de trajetória circular no plano xy sem adição de distúrbio de vento.



(a) Vista 3d. (b) Variáveis de posição controladas. (c) Erros de posição.

Figura 3. Rastreamento de trajetória circular no plano xy sob influência de distúrbio de vento.

ao controlador PID de 70, 63% e uma melhoria de 51, 94% em relação ao controlador RLQR isolado.

Tabela 2. Índices de desempenho para condições normais de voo.

Métrica	PID	RLQR	RLQR+DNN
$\overline{E}$ [m]	0,1765	0,1079	0,0519
Erro Máximo [m]	0,2979	0,3185	0,2925
Melhoria [%]	− − −	38,88%	70,63% * / 51,94% **

b) Desempenho com Adição de Distúrbio de Vento: Neste caso, a Figura 3 apresenta o desempenho da arquitetura proposta frente aos controladores RLQR isolado e PID para o segundo experimento, em que a aeronave ficou sob influência de distúrbio de vento no eixo x. A Figura 3(a) apresenta a vista 3D do rastreamento de trajetória, seguida da Figura 3(b), que apresenta as variáveis de posição controladas (x, y, z). A evolução do erro para este caso é apresentada na Figura 3(c). Deste modo, pode-se notar que, assim como para o primeiro experimento avaliado, a arquitetura RLQR + DNN apresentou um melhor desempenho em relação aos controladores PID e RLQR isolado. Como esperado, o controlador PID apresentou o pior desempenho. A partir dos índices apresentados na Tabela 3 é possível visualizar um erro menor por parte da arquitetura proposta, correspondendo a um percentual de melhoria de 48, 70% e 20, 13% em relação aos controladores PID e RLQR isolado, respectivamente.

Tabela 3. Índices de desempenho para voos sob influência de distúrbios de vento.

Métrica	PID	RLQR	RLQR+DNN
$\overline{E}$ [m]	0,2028	0,1303	0,1041
Erro Máximo [m]	0,4007	0,3472	0,3673
Melhoria [%]	− − −	35,77%	48,70% * / 20,13% **

5. CONCLUSÃO

Neste trabalho, uma arquitetura de controle robusta formada por uma DNN e um RLQR foi proposta para controle de posição de um quadricóptero afetado por incertezas paramétricas e distúrbios de vento. Dois conjuntos de experimentos foram realizados a fim de comparar o desempenho da combinação RLQR + DNN frente aos controladores PID e RLQR isolado. Os resultados globais demonstraram um melhor desempenho da arquitetura proposta tanto para condições normais de voo quanto para voos sob a influência de distúrbio de vento. A combinação de técnicas de aprendizado profundo com um controlador robusto recursivo apresentou grande potencial no problema de rastreamento de trajetória e deverá ser melhor explorada. Trabalhos futuros consistirão em validar a arquitetura proposta em experimentos práticos, a partir da criação de um ambiente com velocidade de vento controlada. Além disso, uma extensão deste trabalho consistirá na combinação da

* Percentual de melhoria em relação ao PID.
** Percentual de melhoria em relação ao RLQR isolado.

DNN com outros controladores presentes na literatura, como o PID, utilizado neste trabalho.

REFERÊNCIAS

Abdelmaksoud, S.I., Mailah, M., e Abdallah, A.M. (2020). Control Strategies and Novel Techniques for Autonomous Rotorcraft Unmanned Aerial Vehicles: A Review. *IEEE Access*, 8, 195142–195169.

Bellahcene, Z., Bouhamida, M., Denai, M., e Assali, K. (2021). Adaptive neural network-based robust $\mathcal{H}_\infty$ tracking control of a quadrotor UAV under wind disturbances. *International Journal of Automation and Control (IJAAC)*, 15, 28–57.

Benevides, J.R.S., Inoue, R.S., Paiva, M.A.D., e Terra, M.H. (2019a). ROS-based robust and recursive optimal control of commercial quadrotors. *IEEE International Conference on Automation Science and Engineering (CASE)*, 2019, 998–1003.

Benevides, J.R.S., Inoue, R.S., Paiva, M.A.D., e Terra, M.H. (2019b). Parameter Estimation Based on Linear Regression for Commercial Quadrotors. *Anais do 14º Simpósio Brasileiro de Automação Inteligente*.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Company.

Kim, J., Gadsden, S.A., e Wilkerson, S.A. (2020). A Comprehensive Survey of Control Strategies for Autonomous Quadrotors. *Canadian Journal of Electrical and Computer Engineering*, 43(1), 3–16.

Li, Q., Qian, J., Zhu, Z., Bao, X., Helwa, M.K., e Schoellig, A.P. (2017). Deep neural networks for improved, impromptu trajectory tracking of quadrotors. *IEEE International Conference on Robotics and Automation (ICRA)*, 5183–5189.

Liu, J. (2017). *Intelligent Control Design and MATLAB Simulation*. Springer Singapore.

Mahony, R., Kumar, V., e Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation Magazine*, 19(3), 20–32.

Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., e Ng, A.Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Santana, L.V., Brandao, A.S., Sarcinelli-Filho, M., e Carelli, R. (2014). A trajectory tracking and 3D positioning controller for the AR.Drone quadrotor. *International Conference on Unmanned Aircraft Systems (ICUAS)*, 756–767.

Sarabakha, A. e Kayacan, E. (2019). Online Deep Learning for Improved Trajectory Tracking of Unmanned Aerial Vehicles Using Expert Knowledge. *IEEE International Conference on Robotics and Automation (ICRA)*, 7727–7733.

Sierra, J.E. e Santos, M. (2019). Wind and Payload Disturbance Rejection Control Based on Adaptive Neural Estimators: Application on Quadrotors. *Complexity*, 2019, 1–20.

Terra, M.H., Cerri, J.P., e Ishihara, J.Y. (2014). Optimal robust linear quadratic regulator for systems subject to uncertainties. *IEEE Transactions on Automatic Control*, 59(9), 2586–2591.

Valavanis, K. e Vachtsevanos, G. (2014). *Handbook of Unmanned Aerial Vehicles*. Springer-Verlag GmbH.

Varshney, P., Nagar, G., e Saha, I. (2019). DeepControl: Energy-Efficient Control of a Quadrotor using a Deep Neural Network. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 43–50.