Solução de cinemática de plataforma de Stewart baseada em aprendizado de máquina

Gustavo G. Silva * **Mateus Giesbrecht** **

* *Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, SP, (e-mail: g169320@unicamp.br).*
** *Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, SP (e-mail: mateus@fee.unicamp.br)*

Abstract: This article presents a solution based on machine learning to the forward kinematic equations of the robotic manipulator known as Stewart platform. The Stewart platform is a parallel robotic manipulator composed by a fixed base and a platform that can move with six degrees of freedom, both connected by six linear actuators. The inverse kinematics problem is trivial, whereas the forward kinematics problem has no analytic solution, in general. Hence, a computational model of the robot was developed for the solution of the inverse kinematics, generating data for training an artificial neural network for the solution of forward kinematics.

Resumo: Neste artigo é apresentada uma solução baseada em aprendizado de máquina para as equações da cinemática direta do manipulador robótico conhecido como plataforma de Stewart. A plataforma de Stewart é um manipulador paralelo composto por uma base fixa e uma plataforma que se move com seis graus de liberdade, conectadas por seis atuadores lineares. O problema da cinemática inversa é trivial, enquanto o da cinemática direta em geral não possui solução analítica. Assim, um modelo computacional do manipulador foi desenvolvido para a solução da cinemática inversa, gerando os dados para o treinamento de uma rede neural artificial para a solução da cinemática direta.

Keywords: Stewart platform; kinematics; machine learning; artifitial neural networks; robotics.
Palavras-chaves: Plataforma de Stewart; cinemática; aprendizado de máquina; redes neurais artificiais; robótica.

1. INTRODUÇÃO

Manipuladores robóticos são empregados em diversos segmentos da indústria e pesquisa, devido a sua boa capacidade de repetibilidade e precisão de movimento (Craig, 2005). Com a crescente demanda por automação de processos tais manipuladores têm sido amplamente estudados.

Em particular, um tipo de manipulador robótico paralelo conhecido como plataforma de Stewart (Stewart, 1965) consiste de uma base fixa e uma plataforma móvel, ligadas por seis atuadores lineares, também chamados de pernas. Este manipulador recebe destaque por permitir a movimentação da plataforma (também chamada de efetuador final) com seis graus de liberdade.

Uma dificuldade de se trabalhar com a plataforma de Stewart está na resolução de suas equações cinemáticas. A cinemática inversa consiste em obter os deslocamentos de cada um dos seis atuadores lineares em função da posição do efetuador final. Já na cinemática direta deseja-se obter a posição espacial da plataforma a partir dos deslocamentos dos atuadores.

A solução da cinemática inversa da plataforma de Stewart é trivial e envolve a solução de seis equações algébricas. Já a solução da cinemática direta envolve a resolução de um sistema de seis equações não lineares para a obtenção das seis variáveis que representam os deslocamentos dos atuadores, e em geral não possui solução analítica, dependendo da geometria do manipulador (Craig, 2005).

Tradicionalmente são utilizados métodos numéricos para solucionar a cinemática direta, como o método de *Newton-Raphson* (Cardona, 2015). No entanto, tal abordagem pode ser computacionalmente custosa, o que compromete a performance do sistema em aplicações de tempo real, por exemplo. Outras soluções exploram características geométricas do manipulador para simplificar as equações (Selig and Li, 2009).

Algumas abordagens na literatura utilizam de algoritmos de aprendizado de máquina para estimar a solução. Morell et al. (2012) utilizaram uma *Support Vector Machine* (SVM) para estimar a solução. Contudo, como o algoritmo SVM produz uma única saída, os autores precisaram utilizar seis SVMs para o problema em questão. Em outro trabalho foi utilizado um algoritmo de rede neural artificial (Yee and Lim, 1997). O algoritmo é treinado sobre um conjunto de dados contendo pares [entrada, saída], no qual a entrada é a posição dos atuadores e a saída é a posição do efetuador final. Porém, os autores necessitaram de etapas a mais para obter uma solução satisfatória, retroalimentando o valor de saída da rede para calcular uma nova solução. Outros autores trabalharam com múltiplas redes em cascata para obter melhores resultados (Geng and Haynes, 1991) (Zhukov et al., 2019).

Neste trabalho propõe-se uma solução para o problema da cinemática direta da plataforma de Stewart utilizando puramente uma rede neural artificial. Para tanto, um modelo computacional da plataforma foi construído e o algoritmo da rede neural implementado, ambos em linguagem *Python* e com apoio de ferramentas de código aberto. A solução obtida a partir da rede neural artificial foi comparada com a obtida por meio de um método numérico, utilizando como métrica de desempenho o erro quadrático médio.

Este trabalho é estruturado da seguinte forma: na seção 2 são apresentados os conceitos gerais da plataforma de Stewart, sua geometria e suas equações cinemáticas. Na seção 3, discorre-se acerca dos conceitos de aprendizado de máquina e redes neurais artificiais. Na seção 4 apresenta-se a metodologia computacional utilizada na aplicação da rede neural artificial à solução das equações cinemáticas do manipulador. Na seção 5 apresentam-se o resultados obtidos. Por fim, na seção 6 são apresentadas as considerações finais.

2. PLATAFORMA DE STEWART

A plataforma de Stewart é um tipo de manipulador paralelo de 6 graus de liberdade (Stewart, 1965). Ela consiste de uma base, fixa, ligada a uma plataforma móvel por meio de seis atuadores lineares. Cada atuador conecta-se à base e à plataforma por juntas universais. No caso deste manipulador, a plataforma também é chamada de efetuador final. Uma representação gráfica simplificada do modelo da plataforma de Stewart estudado neste trabalho é apresentada na Figura 1.

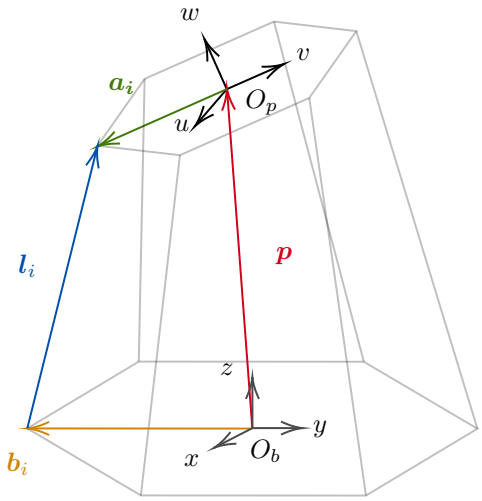

Figura 1. Modelo vetorial da plataforma de Stewart.

2.1 Cinemática da plataforma de Stewart

Para a formulação da cinemática é conveniente definir dois sistemas de coordenadas: O_b, centrado na base, e O_p, centrado na plataforma. Neste caso, "centro" significa o centro da circunferência que passa por todas as juntas do corpo em questão. Dessa forma, as posições das juntas da base são escritas no sistema de coordenadas da plataforma como um conjunto de vetores $\boldsymbol{b_i}$ $(i = 1, ..., 6)$. Similarmente, os vetores $\boldsymbol{a_i}$ $(i = 1, ..., 6)$, escritos no sistema de coordenadas da plataforma, ligam o centro da plataforma à i-ésima junta da plataforma.

Seja $\boldsymbol{p} = [x_p, y_p, z_p]$ o vetor posição do efetuador final, ou seja, que parte da origem do sistema de coordenadas O_b da base e vai até o sistema de coordenadas O_p da plataforma. Este vetor representa a translação do efetuador final. Na situação de repouso, $\boldsymbol{p} = [0, 0, h]$, em que h é a altura nominal da plataforma em relação à base.

A rotação do efetuador final é dada por três rotações sucessivas ao longo do sistema de coordenadas O_p e é definida em termos dos ângulos de Euler α, β e γ. Ou seja, pelo produto das três matrizes de rotação em torno de cada um dos eixos coordenados: $R = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha)$. Onde:

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{2}$$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{3}$$

Com isso, é possível escrever a equação da cinemática inversa:

$$\|\boldsymbol{l_i}\| = l_i = \|\boldsymbol{p} + R \cdot \boldsymbol{a_i} - \boldsymbol{b_i}\|, \quad i = 1, ..., 6 \tag{4}$$

Em (4), $\boldsymbol{l_i}$ é o vetor que representa o i-ésimo atuador linear ou perna do manipulador, $\boldsymbol{p}$ o vetor posição do efetuador final, $\boldsymbol{b_i}$ e $\boldsymbol{a_i}$ os vetores das i-ésimas juntas da base e da plataforma, respectivamente. Sendo d_i o comprimento da perna na situação de repouso, o deslocamento do i-ésimo atuador linear é dado pela diferença $l_i - d_i$.

Na cinemática direta, são conhecidos os comprimentos das seis pernas do manipulador e deseja-se obter a posição do efetuador final correspondente. Este problema envolve a resolução de um sistema de seis equações não lineares nas seis variáveis $x, y, z, \alpha, \beta, \gamma$, conforme a equação a seguir:

$$l_i^2 = (\boldsymbol{p} + R\boldsymbol{a_i} - \boldsymbol{b_i})^T(\boldsymbol{p} + R\boldsymbol{a_i} - \boldsymbol{b_i}), \quad i = 1, ..., 6 \tag{5}$$

3. REDES NEURAIS ARTIFICIAIS

Uma rede neural artificial (RNA) é um tipo de algoritmo de aprendizado de máquina que utiliza combinações lineares das variáveis de entrada ponderadas por pesos ou *weights*. Tais combinações são aplicadas a uma função não linear para gerar variáveis intermediárias. Essa arquitetura é comumente conhecida como *perceptron* multicamadas, pois pode-se empilhar várias camadas em sequência até obter uma camada de saída, correspondente às variáveis de saída do modelo, conforme a Figura 2.

Um modelo básico de uma RNA pode ser descrito como uma sequência de transformações funcionais (Bishop, 2006). Primeiro, faz-se uma sequência de M combinações

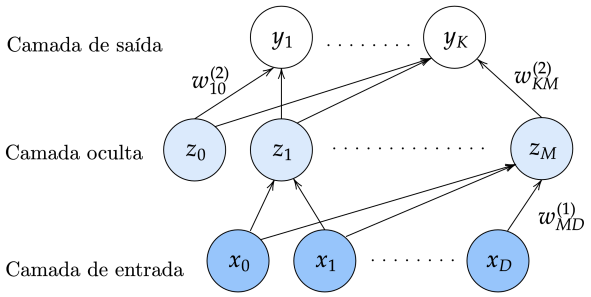

Figura 2. Um exemplo de estrutura de *perceptron* multicamadas.

lineares das variáveis de entrada $x_1, ..., x_D$, em que M é o número de neurônios da camada oculta:

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \quad j = 1, \ldots, M \tag{6}$$

onde superscrito (1) indica que o parâmetro pertence à primeira camada da rede. Os termos $w_{ji}^{(1)}$ são os pesos (*weights*) entre as entradas e as unidades ocultas, enquanto w_{j0} são chamados de *biases* ou viéses e correspondem aos termos constantes da combinação linear. As quantidades a_j são conhecidas como ativações. Em seguida elas são transformadas usando uma função diferenciável não linear chamada função de ativação $h(\cdot)$:

$$z_j = h(a_j^{(1)}) \tag{7}$$

As quantidades z_j são as saídas dos neurônios da camada oculta. Tais unidades são novamente combinadas para obter as ativações de saída (Bishop, 2006):

$$a_k^{(2)} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}, \quad k = 1, \ldots, K \tag{8}$$

em que $w_{kj}^{(2)}$ são os pesos entre as unidades ocultas e as saídas e K é o número total de saídas da rede. Essa transformação corresponde à segunda camada da rede. Finalmente, as ativações são transformadas usando uma função de ativação apropriada para se obter as saídas da rede y_k. A escolha da função de ativação depende do tipo de problema. Em problemas de regressão, a função de ativação da camada de saída é a função identidade, de forma que $y_k = a_k^{(2)}$ (Bishop, 2006).

Combinando os vários estágios de cálculos acima, pode-se obter uma função única para a saída da rede:

$$y_k(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \tag{9}$$

Em (9) fica claro que, neste caso, o modelo da rede neural contém dois estágios de processamento. Contudo, esse modelo pode ser facilmente generalizado para incorporar mais camadas, unidades ocultas e diferentes funções de ativação (Bishop, 2006).

3.1 Funções de ativação

As funções de ativação são funções não lineares aplicadas às combinações lineares das unidades da camada anterior. Essas funções são responsáveis por captar relações não lineares entre as variáveis de entrada e as de saída, fazendo com que a rede tenha uma boa capacidade de aproximação para vários tipos de função. As principais funções de ativação são: sigmóide (*sigmoid*), apresentada na equação (10), tangente hiperbólica (*tanh*), equação (11) e *ReLU* (*rectified linear unit*), equação (12) (Zhang et al., 2020).

$$sigmoid(x) = \frac{1}{1 + \exp(-x)} \tag{10}$$

$$tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)} \tag{11}$$

$$ReLU(x) = max(x, 0) \tag{12}$$

3.2 Função de custo

Para se determinar se a predição dada pelo modelo de regressão está próxima do valor esperado (e o quão próxima ela está) é necessária uma métrica. Tal métrica é dada pela função de custo, que determina o quão distante a predição está do valor esperado. Em geral, o custo é dado por número real não negativo, de forma que, quanto mais próximo de zero for o custo, melhor a predição do modelo.

Para problemas de regressão, como é o caso do problema da cinemática direta de uma plataforma de Stewart, uma função de custo apropriada é dada pela soma dos quadrados dos erros de predição entre as entradas $\mathbf{x}$ e as saídas $\mathbf{y}$ (Bishop, 2006).

3.3 Treinamento da rede neural

Ao se treinar uma rede neural, deseja-se obter os valores dos parâmetros da rede, ou seja, dos pesos e vieses, que minimizam o valor da função de custo. Sendo assim, a tarefa se resume a encontrar os valores dos parâmetros tal que o gradiente da função de custo se anule:

$$\nabla E(\mathbf{w}) = 0 \tag{13}$$

Os valores de $\mathbf{w}$ que satisfizerem (13) são chamados de pontos estacionários. Eles podem ser pontos de mínimo, de máximo, ou ponto de sela. Logicamente deseja-se encontrar um ponto de mínimo, que minimize o valor da função de custo. Porém, em geral as funções de erro apresentam uma dependência bastante não linear com relação aos pesos e vieses. Consequentemente, existem vários pontos estacionários no espaço dos pesos (Bishop, 2006). Um ponto de mínimo que corresponde aos valores dos pesos e vieses tais que a função de custo assume o menor valor possível é chamado de mínimo global. Qualquer outro ponto de mínimo é chamado de mínimo local. Para redes neurais, muitas vezes é suficiente procurar por alguns pontos de mínimo locais e comparar os valores da função de custo correspondente a cada um desses pontos até encontrar uma solução que satisfaça a um critério de exatidão da predição (Bishop, 2006).

A maioria das técnicas para encontrar pontos de mínimo envolve estimar um valor inicial para o vetor de pesos $\mathbf{w}_0$ e então mover-se no espaço dos pesos de forma iterativa:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{14}$$

Diferentes algoritmos são utilizados para determinar o incremento $\Delta\mathbf{w}^{(\tau)}$.

3.4 Gradiente descendente estocástico

Uma técnica para treinar a rede neural baseia-se em atualizar o vetor de pesos tomando um incremento na direção negativa à do gradiente da função de custo:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{\tau}) \tag{15}$$

onde o parâmetro $\eta > 0$ é conhecido como taxa de aprendizado (*learning rate*). Após a atualização do vetor $\mathbf{w}$, o gradiente é novamente avaliado e o processo é repetido. Tal algoritmo é conhecido como gradiente descendente. Como a função de custo é definida pra um conjunto de treino, a cada iteração é necessário que todo o conjunto de treino seja processado para que se possa avaliar ∇E. Técnicas que utilizam todo o conjunto de treinamento são chamadas de processamento em lote ou *batch* (Bishop, 2006). No entanto, quando o conjunto de dados é muito grande esse processo pode ser computacionalmente custoso e lento, já que a função custo representa a média dos erros associados a cada exemplo do conjunto de dados. Uma abordagem mais prática consiste em selecionar de forma aleatória um subconjunto do conjunto de exemplos, chamado *minibatch*. A cada iteração seleciona-se um *minibatch* de tamanho fixo, calcula-se o gradiente da função custo com respeito aos parâmetros do modelo e, por fim, atualiza-se os valores dos parâmetros usando a taxa de aprendizado (Zhang et al., 2020).

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n\mathbf{w}^{(\tau)} \tag{16}$$

Essa técnica é conhecida como gradiente descendente estocástico. Uma vantagem deste tipo de algoritmo com relação às técnicas em *batch* é que o primeiro é capaz de lidar de forma muito melhor dados redundantes (Bishop, 2006).

3.5 O algoritmo de retro-propagação

O cálculo do gradiente da função de custo com relação aos parâmetros da rede neural pode demandar muito esforço computacional e possui complexidade de ordem $O(W^3)$ (Bishop, 2006). Um algoritmo conhecido retro-propagação mostrou-se bastante eficiente nesse cálculo (Rumelhart et al., 1986). Usando informações sobre o gradiente ele é capaz de reduzir a complexidade para uma ordem $O(W^2)$ (Bishop, 2006).

Em suma, o algoritmo consiste em percorrer a rede neural na ordem reversa, ou seja, da camada de saída para a camada de entrada, aplicando a regra da cadeia do cálculo diferencial para calcular o gradiente da função de custo com respeito a cada um dos pesos da rede. A eficiência desse algortimo deve-se ao fato de que ele elimina a necessidade do cômputo direto das derivadas parciais da função de custo com respeito aos pesos. Conhecendo-se a função de custo e percorrendo a rede na ordem reversa os cálculos dessas derivadas podem ser feitos utilizando operações computacionalmente pouco custosas como somas e produtos.

Para aplicar o algoritmo, a função de custo deve satisfazer duas condições. Primeiro, deve ser possível escrevê-la como uma média de funções de custo avaliadas para cada elemento do conjunto de dados de entrada. A segunda condição é que deve ser possível representar a função de custo como uma função das saídas da rede neural (Nielsen, 2015).

4. METODOLOGIA

4.1 Modelo da plataforma de Stewart

Desenvolveu-se um modelo computacional de plataforma de Stewart, implementado utilizando a linguagem de programação *Python*. A base é definida por uma circunferência de raio R_b e por um conjunto de seis ângulos θ_{bi} $(i = 1, ..., 6)$, que determinam as posições angulares das juntas conectadas a uma das extremidades das pernas do manipulador. Neste modelo, foi considerado um raio $R_b =$ 300 mm e um conjunto $\theta_{bi} = [40, 60, 120, 140, 260, 280]$, em que os ângulos são dados em graus. De forma similar à base, a plataforma é definida pelo raio R_p e ângulos θ_{pi}, $(i = 1, ..., 6)$. Deve-se ressaltar, porém, que as coordenadas das juntas da plataforma são escrita no sistema de coordenadas O_p da plataforma. Ou seja, a componente vertical é nula. Considerou-se um raio $R_p = 200$ mm e $\theta_{bi} = [80, 100, 200, 220, 320, 340]$, com ângulos em graus.

Além das posições das juntas, também definiram-se os limites de operação de cada eixo do manipulador, ou seja, os intervalos contendo as posições do espaço que o efetuador final é capaz de alcançar dentro das limitações do manipulador. Em um manipulador real, tais limitações são dadas pela geometria e pela construção mecânica do robô. O lugar geométrico de todas as posições espaciais alcançáveis pelo efetuador final, incluindo as rotações em torno de cada um dos três eixos coordenados, constitui o espaço de trabalho do manipulador.

Tabela 1. Limites de operação do efetuador final.

Coordenada	Intervalo de operação	Unidade
x	[-25,25]	mm
y	[-25,25]	mm
z	[275,300]	mm
α	[-10,10]	°
β	[-10,10]	°
γ	[-15,15]	°

4.2 Implementação da rede neural

Para a implementação da rede neural, utilizou-se o *framework scikit-learn* (Pedregosa et al., 2011). O *scikit-learn* é um API (*Application Programming Interface*) que reúne ferramentas e funções para análise de dados e aprendizado de máquina para a linguagem de programação *Python*. Utilizou-se o modelo MLPRegressor, o qual implementa uma rede neural do tipo MLP para problemas de regressão.

Foram testadas diferentes topologias de rede, alterando o número de camadas ocultas e o número de neurônios nas camadas ocultas. Também foram testadas diferentes funções de ativação, como as funções tanh, sigmóide e ReLU. Além disso, diferentes algoritmos de otimização como o algoritmo SGD, Adam e LBFGS foram utilizados, e o desempenho da rede para cada caso foi observado.

Outros parâmetros como a taxa de aprendizagem e número de iterações também foram alterados a fim de se obter o melhor resultado possível.

Ademais, uma fração do conjunto de testes foi usada para validação da rede. Em todos os casos, 20% do conjunto de testes foi usado para validação, o que corresponde a 16000 amostras.

Como os dados de saída estão em intervalos diferentes, é necessário transformá-los para que estejam dentro de uma mesma faixa de valores. Dessa forma, valores em escalas diferentes não têm influência sobre os pesos da rede neural. Além disso, o pré-processamento é recomendável para melhorar o desempenho computacional da rede (Pedregosa et al., 2011). Optou-se por transformar os dados para que pertencessem ao intervalo [0,1]. Para tanto, calculou-se a média e o desvio padrão de cada variável do conjunto de dados de entrada (os comprimentos l_i das pernas do manipulador). Subtraiu-se de cada elemento a média e dividiu-se o resultado pelo desvio padrão. O mesmo foi feito para cada uma das variáveis de saída $[x, y, z, \alpha, \beta, \gamma]$.

Para avaliar o desempenho da rede neural, o erro quadrático médio entre a predição e o valor esperado foi calculado utilizando o conjunto de teste. Além disso, o algoritmo de rede neural foi comparado com uma solução método numérico, comparando o o erro entre o valor obtido e o esperado e o tempo de execução. O valor esperado foi obtido a partir da solução da cinemática inversa para cada posição do efetuador final.

Para a implementação do método numérico foi utilizada a biblioteca de computação científica *SciPy* (Virtanen et al., 2020). Nela, a função *fsolve* implementa um método numérico para encontrar raízes de funções. O parâmetro *xtol* da função *fsolve* define o valor do erro relativo entre duas iterações consecutivas do algoritmo para o qual o algoritmo deve ser encerrado. Usando esse parâmetro estabeleceu-ce um erro desejado para a solução e avaliou-se o tempo requerido para encontrar a solução a partir de uma estimativa inicial para a solução igual à condição de repouso do manipulador.

5. RESULTADOS

A rede foi treinada sobre o mesmo conjunto de dados de treino utilizando diferentes topologias, ou seja, número de camadas ocultas e números de neurônios nas camadas. O número de iterações ou *epochs* utilizado para todos os testes foi 300. O número de *minibatches* para o algoritmo de otimização foi definido como 200 e foi utilizada uma taxa de aprendizagem constante $\eta = 0.001$.

A métrica de desempenho utilizada foi o erro quadrático médio, avaliado sobre o conjunto de teste. De acordo com a documentação do *sklearn*, Se $\hat{\mathbf{y}}_\mathbf{i}$ for a predição do valor de saída para a i-ésima entrada, e $\mathbf{y_i}$ for o valor real da saída, então o erro quadrático médio (*mean squared error* ou *MSE*) sobre n amostras é calculado da seguinte forma:

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n_{amostras}} \sum_{i=0}^{n_{amostras}-1} (\mathbf{y_i} - \hat{\mathbf{y}}_\mathbf{i})^2 \qquad (17)$$

Os melhores resultados obtidos estão representados na Tabela 2. É importante ressaltar que o valor do erro quadrático médio pode variar para uma mesma topologia de rede e para um mesmo conjunto de dados, dada a natureza estocástica do algoritmo de otimização utilizado no treinamento da rede.

Tabela 2. Erro de validação para diferentes topologias de rede.

Camadas	Neurônios por camada oculta	Função	MSE
5	12, 14, 15, 23, 18	ReLU	0.0498669
3	13, 18, 14	ReLU	0.0726869
2	100, 100	ReLU	0.0019983
2	100, 100	Sigmóide	0.0036778
2	50, 50	tanh	0.0024999
3	100, 100, 50	ReLU	0.0087441

O melhor resultado foi obtido utilizando uma rede neural com duas camadas ocultas, cada qual com 100 *hidden units* e função de ativação ReLU. As funções de ativação das camadas ocultas são a função ReLU. O algoritmo de otimização para o treinamento da rede utilizado foi o algoritmo Adam (Kingma and Ba, 2017), uma variação do gradiente descendente estocástico. Para essa situação obteve-se a curva que mostra o erro entre a saída da rede e a resposta esperada correspondente no conjunto de validação, chamado de erro de validação, para cada iteração do algoritmo de otimização.

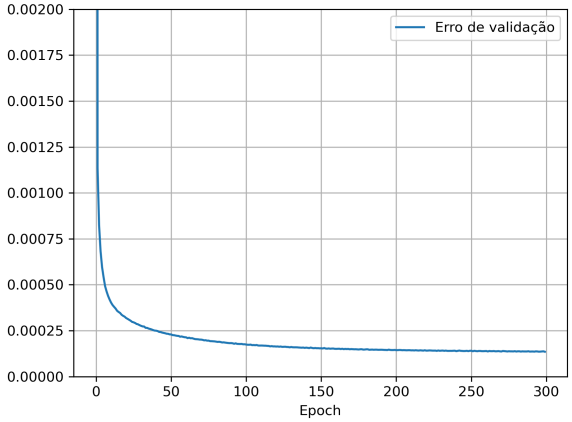

Figura 3. Erro de validação a cada iteração durante o treinamento da rede neural.

Na Tabela 3 são mostradas algumas predições obtidas com a rede neural que possuiu o menor erro de validação. Para cada conjunto, na primeira linha tem-se os valores esperados de cada variável de saída; na segunda linha vê-se a corresponde predição obtida a partir da rede e, na terceira, o erro percentual entre o valor predito e o valor esperado. Nota-se que para a maioria dos valores o erro é inferior a um por cento.

Tabela 3. Comparação entre predições e valores esperados.

x [mm]	y [mm]	z [mm]	α [°]	β [°]	γ [°]
-17.1167	4.0052	284.5223	-6.5987	8.03534	-11.9463
-17.1124	4.0596	284.5511	-6.5765	8.04487	-11.9818
0.0248%	1.3603%	0.0101%	0.3354%	0.1187%	0.2969%
4.1115	7.9505	297.4579	-9.5456	-2.2654	-3.6415
4.1252	7.9362	297.4810	-9.5449	-2.2415	-3.6846
0.3346%	0.1797%	0.0078%	0.0074%	1.0553%	1.1836%
1.8948	14.8617	290.5014	8.2182	9.1561	-13.0740
1.8579	14.8461	290.477	8.2121	9.1512	-13.0890
1.9472%	0.1047%	0.0083%	0.0734%	0.0543%	0.1150%
8.8691	12.4259	276.2763	1.5564	4.5463	7.1273
8.7571	12.4703	276.2571	1.5511	4.5535	7.1078
1.2625%	0.3571%	0.0069%	0.3362%	0.1580%	0.2733%
-24.3131	9.5505	281.3803	3.1225	-3.5383	-12.8574
-24.3874	9.5633	281.4125	3.1135	-3.5413	-12.8427
0.3055%	0.1337%	0.0114%	0.2864%	0.0866%	0.1141%
14.6091	23.1215	297.6881	-2.1240	-9.3761	-7.5446
14.5422	23.0170	297.6696	-2.1084	-9.3872	-7.5374
0.4580%	0.4518%	0.0062%	0.7338%	0.1182%	0.0956%

6. CONCLUSÃO

Neste trabalho foi proposto um método para a resolução da cinemática direta de uma plataforma de Stewart utilizando aprendizado de máquina, especificamente uma rede neural artificial. O melhor resultado obtido corresponde a um erro quadrático médio de 0.0019983. O método proposto pode ser utilizado em aplicações nas quais a precisão requerida não seja rigorosamente alta. Pode-se também utilizar o modelo para simular uma geometria do manipulador ou para fins de prototipagem, por exemplo. Além disso, o tempo com que se obtém uma estimativa para a pose da plataforma, da ordem de milissegundos, é pequeno o suficiente para algumas aplicações em tempo real.

Deve-se notar, ainda, que a qualidade da solução é afetada pela dimensão do espaço de trabalho do manipulador. No caso de uma plataforma de Stewart que opera em uma região menor do espaço, ou seja, para a qual os intervalos de translações e rotações são pequenos, o erro da predição tende a ser menor. Isso ocorre porque o conjunto de dados de treino e de testes está limitado a um intervalo menor, fazendo com que o algoritmo de rede neural tenha mais facilidade em aproximar a relação entre entrada e saída no intervalo.

AGRADECIMENTOS

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

REFERÊNCIAS

Bishop, C.M. (2006). *Pattern recognition and machine learning*. springer, New York.

Cardona, M. (2015). A new approach for the forward kinematics of general stewart-gough platforms. In *2015 IEEE Thirty Fifth Central American and Panama Convention (CONCAPAN XXXV)*, 1–6. doi:10.1109/CONCAPAN.2015.7428478.

Craig, J.J. (2005). *Introduction to robotics: mechanics and control*. Pearson Education International, Upper Saddle River, 3rd edition.

Geng, Z. and Haynes, L. (1991). Neural network solution for the forward kinematics problem of a stewart platform. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, 2650–2651. IEEE Computer Society.

Kingma, D.P. and Ba, J. (2017). Adam: A method for stochastic optimization.

Morell, A., Acosta, L., and Toledo, J. (2012). An artificial intelligence approach to forward kinematics of stewart platforms. In *2012 20th Mediterranean Conference on Control Automation (MED)*, 433–438. doi:10.1109/MED.2012.6265676.

Nielsen, M.A. (2015). *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088), 533–536.

Selig, J.M. and Li, H. (2009). A geometric newton-raphson method for gough-stewart platforms. In A. Kecskeméthy and A. Müller (eds.), *Computational Kinematics*, 183–190. Springer Berlin Heidelberg, Berlin, Heidelberg.

Stewart, D. (1965). A platform with six degrees of freedom. *Proceedings of the institution of mechanical engineers*, 180(1), 371–386.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272. doi:10.1038/s41592-019-0686-2.

Yee, C.S. and Lim, K.B. (1997). Forward kinematics solution of stewart platform using neural networks. *Neurocomputing*, 16(4), 333 – 349. doi:https://doi.org/10.1016/S0925-2312(97)00048-9. URL `http://www.sciencedirect.com/science/article/pii/S0925231297000489`.

Zhang, A., Lipton, Z.C., Li, M., and Smola, A.J. (2020). *Dive into Deep Learning*. Berkeley. `https://d2l.ai`.

Zhukov, Y.A., Korotkov, E., Zhukova, V., and Abramov, A. (2019). Neural network solution of the direct kinematics problem for a hexapod with ball-screw drives of legs. In *IOP Conference Series: Materials Science and Engineering*, volume 656, 012061. IOP Publishing.