Comparação de métodos de otimização Local e Meta-Heurístico para seguimento de caminho do manipulador robótico KUKA KR 6 R700 sixx

Lucas Alves Borges * **Dean Bicudo Karolak** * **Fadul Ferrari Rodor** *
Luiz Felipe Pugliese * **Giovani Bernardes Vitor** *

* *Laboratório de Robótica, Sistemas Inteligentes e Complexos RobSIC, UNIFEI, Itabira, MG,*
E-mails: lucas.lab2@hotmail.com; dean.karolak@unifei.edu.br; fadulrodor@unifei.edu.br; pugliese@unifei.edu.br; giovanibernardes@unifei.edu.br

Abstract: Performing the inverse kinematics of a robotic manipulator is always a challenge, especially when it comes to several degrees of freedom. Although manipulators are increasingly used in industry, *Digital Twins* techniques need to be extended to facilitate the handling of this equipment. Nevertheless, there are already many methods proposed to solve this issue. In this work, the comparison of three methods for solving inverse kinematics is performed, these being local optimization methods, such as the Inverse Jacobian, Levenberg-Marquardt, and a global optimization metaheuristic method, the Particle Swarm Optimization (PSO). The algorithms were programmed in C++, which were applied to Gazebo software to perform a 3D simulation of the robot motions. To create an interface between the real system and the simulation environment, the ROS framework was used. In the tests performed, all results demonstrated high accuracy, with the Inverse Jacobian and the Levenberg-Marquardt standing out.

Resumo: A realização da cinemática inversa de um manipulador robótico é sempre um desafio, principalmente quando se trata de vários graus de liberdade. Apesar dos manipuladores serem cada vez mais utilizados na indústria, as técnicas de *Digital Twins* precisam ser ampliadas para facilitar o manuseio destes equipamentos. No entanto, já existem vários métodos propostos para a solução da cinemática inversa. Neste trabalho é realizada a comparação de três métodos de cinemática inversa, sendo os métodos de otimização local, como o jacobiano inverso, Levenberg-Marquardt e um método meta-heurístico de otimização global, a otimização por enxame de partículas (PSO). Os algoritmos foram programados em C++, utilizando-se o *software* Gazebo para a simulação 3D do robô. Para realizar uma interface do sistema real com o ambiente de simulação, utilizou-se o *framework* ROS. Nos ensaios realizados, todos os resultados demonstraram alta exatidão, com destaque maior para o jacobiano inverso e o Levenberg-Marquardt.

Keywords: Inverse Kinematics; PSO; Levenberg-Marquardt; Inverse Jacobian; Tracking.
Palavras-chaves: Cinemática inversa; PSO; Levenberg-Marquardt; Jacobiano inverso; Caminho.

1. INTRODUÇÃO

Desde a revolução industrial, novas tecnologias que sejam capazes de aumentar a escala de produção e elevar a qualidade dos produtos vem sendo constantemente demandadas. A partir disso, emergiu a quarta revolução industrial, ou Indústria 4.0, que possui foco em automação de processos e conectividade (Lu, 2017). A automação se tornou uma tendência na indústria mundial, em que, o maior nível de tecnologia garante um melhor desempenho no mercado, otimizando os processos e garantindo uma maior produção e qualidade dos produtos com redução de custo (Haseeb et al., 2019).

Com a grande aplicação de robôs nos processos de fabricação, surge uma demanda por conhecimento teórico e prático. O controle, integração e manutenção dos robôs são tarefas complexas e por vezes desafiadoras, exigindo cada vez mais a utilização de recursos computacionais para simular os equipamentos, o ambiente de trabalho e as restrições envolvidas no processo (Jung et al., 2020).

A proposta deste trabalho é realizar o controle de um manipulador robótico modelo KUKA KR 6 R700 sixx por meio do *software* de simulação Gazebo, utilizando a *framework* ROS. Este irá criar uma camada de abstração que possibilite diversas aplicações no ambiente de simulação, de forma que seja aplicável também no sistema real.

Para realizar o controle, utiliza-se a cinemática inversa do manipulador, por meio de dois algoritmos numéricos de otimização local e um meta-heurístico de otimização global. Para os métodos numéricos é utilizado o jacobiano inverso (JI) (Whitney, 1969), e o método de Levenberg-Marquardt (LM) (Madsen et al., 2004), ambos utilizando uma aproximação do jacobiano para cálculo da trajetória. E para o método meta-heurístico é utilizado o algoritmo de enxame de partículas (*Particle Swarm Optimization* - PSO) (Kennedy and Eberhart, 1995). A comparação entre os métodos é feita por meio de ensaios relacionados a se atingir pontos individuais, bem como o seguimento de caminho, analisando o erro, quantidade de iterações e o tempo de processamento.

A simulação de algoritmos na indústria é muitas vezes realizada por *softwares* dedicados, fornecidos pelo próprio fabricante do robô, para fazer a comunicação com os seus equipamentos, o que os torna dependente da fabricante e também requerendo esforço adicional para aprendizagem da linguagem de programação existente para aquele manipulador em específico. Diante disto, atualmente existe um esforço coletivo para se utilizar uma ferramenta universal que consiga padronizar os desenvolvimentos relacionados à indústria, adotando para este fim o *framework* de robótica ROS. Logo, para construir tecnologia já fundamentada nesta concepção, tanto a academia quanto as indústrias estão empenhadas em estreitar tal distanciamento e juntos conseguirem avanços (Quigley et al., 2009).

Sendo assim, baseando-se na nova tendência chamada de *Digital Twins* (Boschert et al., 2018), foi utilizado o ROS, um *framework* com várias ferramentas adaptadas para robôs. Este recurso cria uma interface de comunicação, proporcionando uma camada de abstração capaz de gerenciar uma aplicação independente se esta está sendo executada em ambiente simulado ou real, conforme a Figura 1.

O projeto realizado, foi estruturado no ambiente de simulação, utilizando o *framework* ROS. A implementação do algorítimo no ambiente de operação real se dará em trabalhos futuros.

Este trabalho está estruturado como segue: A seção 2 apresenta um panorama de trabalhos relacionados a este e outros métodos utilizados, ressaltando suas características. A seção 3 descreve em detalhes a implementação da solução proposta neste trabalho. A seção 4 apresenta os resultados quantitativos e qualitativos acerca da metodologia implementada, e, por fim, a seção 5 apresenta as considerações finais sobre a pesquisa realizada.

2. FUNDAMENTAÇÃO TEÓRICA

A realização da cinemática inversa de um manipulador robótico simples, com poucos graus de liberdade pode ser deduzido de forma algébrica com um certo esforço, mas em um manipulador de seis juntas, isto acaba se tornando inviável por se tratar de um sistema de seis equações não lineares.

Métodos de otimização meta-heurístico como enxame de partículas (PSO) são frequentemente estudados devido a sua característica de fácil implementação, com resultados de erros satisfatórios. No trabalho realizado por Alkayyali

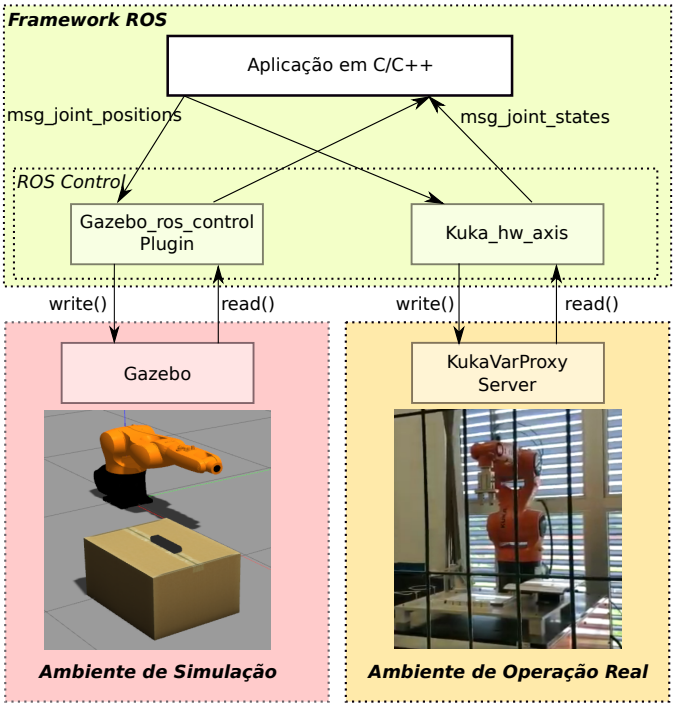

Figura 1. Plataforma de Simulação para Ensaios no manipulador KUKA KR 6 R700 sixx.

and Tutunji (2019), obtiveram um erro na faixa de 0,1% utilizando o manipulador KUKA KR6 K900, similar ao utilizado neste projeto.

Este algoritmo também é utilizado em aplicações de hiper-redundância como proposto por Trabasso et al. (2019), que utiliza em seu projeto um robô-cobra de 8 graus de liberdade para aplicação de selante em placas da fuselagem de uma aeronave. Utilizando aplicação de seguimento de trajetória, o autor obteve uma redução de massa de até 1 kg.

Em uma comparação com outros métodos meta-heurísticos, o PSO de comportamento quântico (QPSO) utilizado por Dereli and Köker (2020), em um manipulador redundante de sete graus de liberdade, obtiveram uma convergência mais rápida e com menor erro que com outros métodos de partículas, como o colônia de vaga-lume e o próprio PSO.

Outras formas de solução para o problema apresentado é a utilização de métodos numéricos de otimização local, como o de Newton-Raphson (Serenza, 2015; Rocha and Bessa, 2012) e mínimos quadrados (Wang et al., 2017), que consideram uma matriz jacobiana para minimização do erro. O método de Levenberg-Marquardt é um aperfeiçoamento do método de Gauss-Newton, o que reduz o problema de singularidade da matriz jacobiana (Nilsson, 2009), inserindo uma componente ortogonal á matriz. Sugihara (2011) obteve com este algoritmo uma convergência rápida, atingindo um erro estipulado de 10 mm em 2,23 ms.

Para este trabalho, um dos métodos apresentados é o proposto por Farzan and DeSouza (2013), que se aproxima do método de Newton-Raphson, utilizando a ponderação de um fator de atenuação α. Este procedimento utiliza uma aproximação da matriz jacobiana que facilita a implementação e possui uma convergência rápida, no qual o

autor obteve um erro inferior a 1 mm, necessitando de 20 a 30 iterações para a maioria dos casos.

O projeto foi realizado utilizando o *framework* ROS cujos resultados puderam ser implementados diretamente no manipulador. O ambiente de simulação escolhido foi o *software* de renderização 3D e simulação Gazebo (Koenig and Howard, 2004). Este simulador possui código aberto, o que facilita a construção e a simulação da dinâmica do robô KUKA KR 6 R700 sixx. A junção destas duas ferramentas torna este um ambiente de simulação interessante, capaz de realizar aplicações *online*, utilizando o mesmo código que pode também ser implementado no sistema real.

3. MATERIAIS E MÉTODOS

3.1 Manipulador Robótico

O manipulador utilizado para este projeto trata-se do KUKA KR 6 R700 sixx, que possui 6 graus de liberdade, 6 eixos rotativos e sua estrutura e limitações totalizam o espaço de trabalho apresentado na Figura 2. O modelo do robô pode ser representado pelos parâmetros de Denavit-Hartenberg (DH) (Denavit et al., 1955) apresentados na Tabela 1. Esta é uma técnica padrão para representar robôs e modelar seus movimentos, de forma a identificar os parâmetros cinemáticos de qualquer manipulador robótico com exatidão e eficiência. A partir deste modelo, é possível realizar uma análise dos movimentos dinâmicos, diferenciais e Jacobianos, do manipulador, de forma a reduzir a variabilidade na performance cinemática (Niku, 2010).

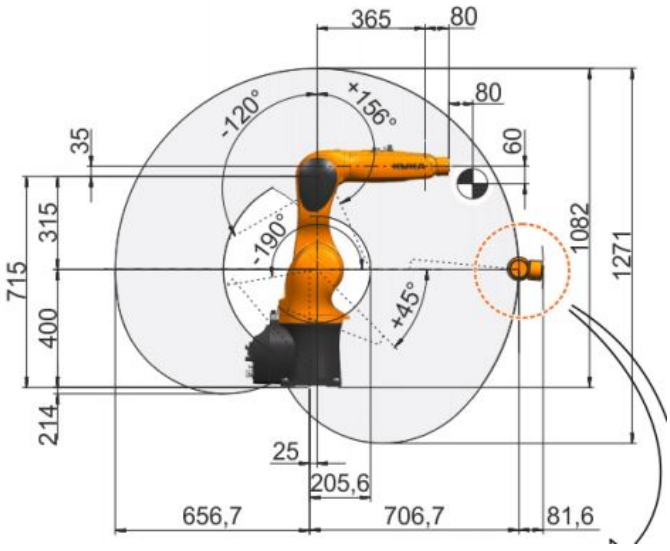

Figura 2. Espaço de trabalho do robô KUKA (2015).

Tabela 1. Parâmetros Denavit-Hartenberg do manipulador robótico.

junta	θ	d (mm)	a (mm)	α (rad)
1	θ_1	400	25	$-\pi/2$
2	θ_2	0	315	0
3	$\theta_3 + \pi/2$	0	-35	$\pi/2$
4	θ_4	365	0	$-\pi/2$
5	θ_5	0	0	$\pi/2$
6	θ_6	80	0	π

3.2 Cinemática Direta

A cinemática estuda os movimentos do manipulador, levando em consideração as equações que regem o seu comportamento. Para a cinemática direta, tem por finalidade obter a posição do efetuador final a partir da posição das articulações. Como o manipulador possui 6 juntas, tem-se um sistema de 6 equações não lineares complexo. Portanto, para resolver este problema, são utilizados os parâmetros DH para encontrar a matriz de transformação homogênea total (A_0^n), a qual informa a posição e o sentido do efetuador final, dado um vetor de ângulos de junta.

Inicialmente,necessita-se encontrar a matriz de transformação homogênea parcial (A_i), que relaciona a posição da junta i em relação a junta i_{+1}. Isto é feito por meio de uma série de movimentos de rotações R e translações T em torno dos eixos de cada junta (Niku, 2010):

$$A_i = R_{(z,\theta)} T_{(z,d)} T_{(x,a)} R_{(x,\alpha)} \tag{1}$$

Ao realizar este procedimento para todas as juntas, tem-se a matriz de transformação homogênea total composta por:

$$A_0^n = A_0 A_1 ... A_n = \begin{bmatrix} r_1 & r_2 & r_3 & P_x \\ r_4 & r_5 & r_6 & P_y \\ r_7 & r_8 & r_9 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Cuja matriz r e o vetor P representam respectivamente, o sentido e a posição do efetuador final.

3.3 Cinemática Inversa

A cinemática inversa é utilizada para encontrar os valores dos ângulos das juntas necessários para que o efetuador final possa chegar no local desejado. Para o caso de manipuladores com 6 graus de liberdade, encontrar estes valores de ângulo envolve um sistema de 6 equações não lineares, o que torna a cinemática inversa um desafio.

Para este trabalho, o foco foi apenas na posição do efetuador final, desconsiderando o sentido de orientação que o efetuador chegará no ponto desejado. Desta forma, para encontrar a solução deste sistema para o manipulador estudado, deseja-se minimizar a função de custo dada pela equação (3).

$$e = \sqrt{(P_{xf} - P_{xi})^2 + (P_{yf} - P_{yi})^2 + (P_{zf} - P_{zi})^2} \tag{3}$$

O erro e trata-se da distância euclidiana entre o ponto P_f que o efetuador final deseja chegar e o ponto P_i onde o efetuador se encontra. A minimização deste erro foi realizada por três métodos descritos nas Seções 3.4, 3.5 e 3.6.

3.4 Jacobiano Inverso

Este procedimento refere-se a um método numérico de otimização local que utiliza uma matriz jacobiana, baseada no modelo do robô, para minimizar uma posição de forma iterativa (Nilsson, 2009). Devido a complexidade das derivadas recorrentes do modelo, Farzan and DeSouza (2013) utilizaram uma aproximação numérica para realizar a matriz jacobiana.

Dado os valores de ângulos das juntas $\boldsymbol{Q}$, obteve-se por meio da cinemática direta, um vetor de posições e orientações $\boldsymbol{X}$.

$$\boldsymbol{Q} = \begin{bmatrix} q_1(i) \\ q_2(i) \\ q_3(i) \\ q_4(i) \\ q_5(i) \\ q_6(i) \end{bmatrix}, \quad f(\boldsymbol{Q}) = \boldsymbol{X} = \begin{bmatrix} x(i) \\ y(i) \\ z(i) \\ \phi_r(i) \\ \phi_p(i) \\ \phi_y(i) \end{bmatrix} \tag{4}$$

Uma aproximação da matriz jacobiana foi obtida a partir de uma estimação numérica, aplicando pequenas perturbações arbitrárias em cada junta individualmente conforme

$$J_c = \frac{\partial \boldsymbol{X}}{\partial q} = \boldsymbol{X} - f\left(\boldsymbol{Q} + \begin{bmatrix} \vdots \\ 0 \\ 0{,}01 \\ 0 \\ \vdots \end{bmatrix}\right) \tag{5}$$

em que J_c é uma coluna da matriz jacobiana, de forma que, realizado este procedimento para cada junta, encontra-se a matriz jacobina J_i. Com isto, pode-se encontrar uma variação de $\boldsymbol{Q}$ que reduz a função de custo (3) dado pela equação (6).

$$\Delta \boldsymbol{Q} = J_i^{-1} \alpha (\boldsymbol{X}_f - \boldsymbol{X}_i) \tag{6}$$

O fator de atenuação α é um parâmetro escolhido entre 0 e 1 que dita o tamanho do passo da iteração. Com um valor mais alto para o fator de atenuação, o sistema pode ficar muito rápido e ultrapassar o objetivo desejado, causando oscilações no sistema e tornando-o divergente. No entanto, se este valor for baixo demais, o sistema pode apresentar lentidão e causar uma convergência assintótica errônea (Farzan and DeSouza, 2013).

Por este ser um otimizador local, deve-se considerar uma configuração inicial, a qual, neste caso, são utilizados os ângulos de juntas atuais. Uma vez que o vetor $\boldsymbol{Q}$ for encontrado e enviado para o robô, é possível recalculá-lo enquanto o manipulador realiza sua dinâmica, a fim de refinar o resultado e obter um erro menor, durante o processo de simulação.

3.5 Levenberg-Marquardt

O método de Levenberg-Marquardt se trata de uma técnica de otimização local baseada em uma modificação do método de Gauss-Newton, o qual utiliza uma minimização iterativa por mínimos quadrados (Madsen et al., 2004).

Para a realização do algoritmo, foi utilizado o código disponível na biblioteca *Mobile Robot Programming Toolkit* (MRPT, 2021), a qual inclui diversos códigos para aplicação direta na robótica.

Neste código foi estimado o valor da matriz jacobiana usando incrementos para cada variável individual, conforme foi realizado na equação (5), a partir da configuração de juntas do robô naquele instante de tempo. Posteriormente, é realizada a minimização da função de custo da equação (3), de forma iterativa, dado por

$$\Delta Q = -(J_i^T J_i + \lambda I)^{-1} J_i^T e \tag{7}$$

em que e é o erro da equação (3). Para este cálculo é utilizado o jacobiano pseudo-inverso, o que reduz o problema de singularidade da matriz. Além disso, o fator de amortecimento λ adiciona um vetor ortogonal que também reduz a singularidade do cálculo, e influencia no tamanho do passo (Nilsson, 2009). Sua escolha foi embasado no trabalho de Madsen et al. (2004), em que seu valor está relacionada ao tamanho dos elementos da matriz hessiana.

$$\lambda = 10^{-3} \max(diag(J_i^T J_i)) \tag{8}$$

Assim como no método anterior, por este ser um otimizador local, após enviar os valores de juntas encontrados para o robô, é possível recalcular a cinemática inversa enquanto o manipulador realiza sua dinâmica, a fim de reduzir o erro.

3.6 Exame de partículas

Ao contrário dos métodos citados anteriormente, o PSO é uma técnica de otimização global, que consiste em definir em todo o espaço de trabalho n partículas, uniformemente distribuídas, com uma posição, orientação e velocidade aleatória que convergem em busca da melhor solução para a função de custo. Para esta implementação, foi utilizado um algoritmo publicamente disponível [1], que usa como função para atualizar as posições das partículas a equação (9), em que V_i é a velocidade da partícula.

$$\Delta \boldsymbol{Q} = V_{i+1} = wV_i + c_1 V_{local} + c_2 V_{global} \tag{9}$$

Para a qual

$$V_{local} = r_1(V_{best_p} - V_i) \tag{10}$$

$$V_{global} = r_2(V_{best_g} - V_i) \tag{11}$$

As velocidades das partículas são alteradas a cada iteração, considerando o melhor resultado obtido pela partícula V_{local}, e o melhor resultado obtido entre todas as partículas V_{global}. O parâmetro de ajuste w é uma ponderação de inercia (Kennedy and Eberhart, 1995), c_1 e c_2 são coeficientes de aceleração chamados de parâmetros cognitivo e social, respectivamente, já os parâmetros r_1 e r_2 são valores aleatórios distribuídos uniformemente entre [0, 1]. A cada iteração é atualizada a posição da partícula, considerando a sua velocidade e analisado o erro de cada uma em relação a equação (3). Desta forma, com o passar das iterações, as partículas convergem para o melhor resultado global dentro do espaço de trabalho.

Uma particularidade deste método é que, para simulação de caminho, (Seção 4.3), é preciso limitar o espaço de juntas entre a execução de um ponto à outro. Desta forma, impede-se que o manipulador encontre uma configuração de juntas muito diferente da obtida anteriormente, evitando que o dispositivo faça movimentos inconvenientes.

3.7 Caminhos

Para realizar a simulação dos experimentos e análise dos resultados, foram utilizados três caminhos, que são discretizadas com um passo de 0,1 rad, dividindo o movimento em 63 pontos. As curvas determinadas são:

[1] Algoritmo PSO utilizado disponível em: `https://github.com/Rookfighter/pso-cpp`

(1) lemniscata, descrita por:

$$\begin{aligned} x &= \frac{100\sqrt{2}\cos(t)}{\text{sen}(t)^2+1} \\ y &= -700 \\ z &= \frac{100\sqrt{2}\cos(t)\text{sen}(t)}{\text{sen}(t)^2+1} + 430 \end{aligned} \tag{12}$$

(2) *rhodonea*, descrita por:

$$\begin{aligned} x &= 100\cos(2t)\cos(t) \\ y &= 50\cos(2t)\cos(t) - 700 \\ z &= 100\cos(2t)\text{sen}(t) + 430 \end{aligned} \tag{13}$$

(3) Círculo, descrito por:

$$\begin{aligned} x &= 100\cos(t) \\ y &= 100\text{sen}(t) + 430 \\ z &= 50\cos(t) - 700 \end{aligned} \tag{14}$$

4. RESULTADOS

4.1 Configurações Utilizadas

As simulações foram realizadas em um computador com processador Intel(R) Core i5-7200Ur a 2.50GHz, com 8GB de memória RAM.

Para que fosse possível realizar os cálculos necessários, foi criado uma série de funções que realizam inversão, multiplicação e transposição de matriz. Da mesma forma, foram desenvolvidas funções para cinemática direta, inversa, leitura e escrita de dados do manipulador. E para simular o robô KUKA KR 6 sixx, foi utilizado o ambiente de simulação proporcionado pelo *software* GAZEBO, implementando o sistema ROS com código programado em C++. A visualização da representação do manipulador pode ser visto na Figura 3.

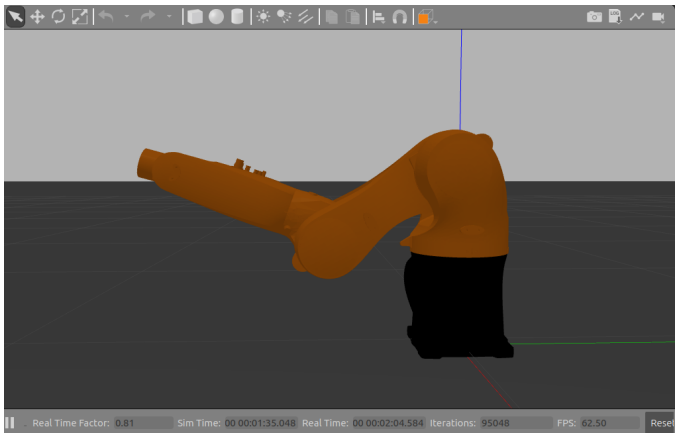

Figura 3. Representação do manipulador utilizado no Gazebo.

Durante a criação do algoritmo da cinemática inversa pelo jacobiano inverso, foi necessário escolher o valor do fator de atenuação α da equação (6). Para isso, foi feito um código no *software* MATLAB a fim de analisar a quantidade de iterações necessárias para convergência de três pontos específicos, sendo estes, pontos com distância euclidiana d entre o objetivo e a posição do efetuador final, apresentado na Tabela 2. O critério de convergência é atingido quando o erro é menor que 0,01 mm.

Tabela 2. Quantidade de iterações para o jacobiano inverso.

d (mm)	$\alpha = 0{,}004$	$\alpha = 0{,}005$	$\alpha = 0{,}006$
300	15	11	15
960	15	11	15
1160	36	13	10

Para um α menor, é necessário mais iterações para convergir, uma vez que o deslocamento realizado, entre uma iteração e outra, é muito pequeno. No entanto, a necessidade de 15 iterações observadas em $\alpha = 0{,}006$ ocorre devido ao tamanho elevado do passo entre as iterações, no qual, o algoritmo ultrapassa o objetivo desejado, causando oscilações no sistema. Portanto, o valor definido para realizar as próximas simulações foi $\alpha = 0{,}005$.

O código do PSO foi inicializado com 100 partículas distribuídas aleatoriamente, de forma uniforme em todo o espaço de trabalho. Para os parâmetros de ajuste foi utilizada a configuração *default*, na qual a ponderação da inércia w escolhida é unitária e para os parâmetros de ajuste c_1 e c_2 é aplicado o valor 2.

O critério de convergência utilizado para todos os métodos, foi de finalizar o cálculo da cinemática inversa quando a variação do erro for menor que 10^{-6} mm.

4.2 Simulação com Pontos Individuais

Inicialmente, para comparar a eficácia das técnicas, foi considerada a convergência para 3 pontos, conforme a Figura 4. Após cada execução, aguarda-se 5 segundos para garantir que a dinâmica do robô atinja o menor erro possível. Após realizar este teste 32 vezes, obteve-se os dados apresentados na Tabela 3, em que i é a quantidade de iterações, t o tempo de execução do código e e é o erro obtido com o seu desvio padrão.

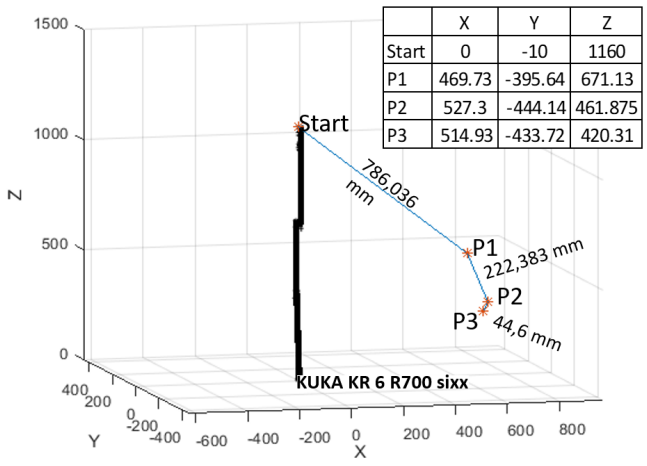

Figura 4. Representação dos pontos utilizados.

Pelo resultado apresentado na Tabela 3, observa-se que, tanto o jacobiano inverso quanto o Levemberg-Marquardt possuem erro baixo, no qual o primeiro obteve um resultado melhor para o P2 e P3, com uma quantidade de iterações muito inferior aos demais. Já o PSO, de forma geral, obteve valores de erros muito próximos aos outros dois métodos, contudo, as ocorrências de eventuais picos

Tabela 3. Dados do teste de pontos individuais.

	P1			P2			P3		
	JI	LM	PSO	JI	LM	PSO	JI	LM	PSO
i max	23	130	317	25	136	317	21	105	297
i mín	22	103	50	21	117	26	20	90	29
i med	**22,844**	112,219	241,656	**22,063**	129,844	193,906	**20,375**	99,594	212,125
t máx [ms]	21,546	27,391	853,374	19,607	30,576	836,941	16,675	24,566	783,836
t mín [ms]	10,379	18,132	141,601	12,331	22,789	74,340	10,589	16,558	85,882
t méd [ms]	**15,089**	21,796	636,180	**14,733**	26,034	510,221	**13,546**	19,856	558,266
e máx [mm]	0,00764	0,00168	1,4561	0,002229	0,004022	30,0759	0,000736	0,002384	25,8102
e mín [mm]	9,16E-05	8,63E-05	0	9,16E-05	0,00015	0	0	6,82E-05	0
e méd [mm]	0,001004	**0,000548**	0,04572	**0,000373**	0,000693	4,847263	**0,000132**	0,000504	1,441729
Desvio	0,00185	**0,000349**	0,257365	**0,000488**	0,000716	9,200058	**0,000244**	0,000454	5,092195

de erros acabam por elevar a média deste resultado. Pelo jacobiano inverso possuir o menor tempo de execução em todos os pontos, este se torna mais vantajoso para trabalhar em sistemas de tempo real. No entanto, apesar do Levemberg-Marquardt convergir com uma quantidade de iterações superior, o tempo de processamento dele é muito próximo do jacobiano inverso, o que se pressupõe, que este tempo possa ser reduzido em troca de um erro tolerável.

O PSO é um método de otimização global, que necessita de cálculos repetitivos dos erros de cada partícula, o que faz com que o tempo de execução seja muito longo, justificando os resultados apresentados na Tabela 3. Por sua vez, este procedimento evita que o sistema fique preso em mínimos locais.

4.3 Simulação de Caminho

A velocidade de movimento do manipulador altera conforme se aproxima do ponto desejado, devido ao controlador PID do robô. Portanto, para realizar um caminho suave, foi definido que quando a variação do erro obtido pelo efetuador final for menor que 0,5 mm, o manipulador irá se dirigir para o próximo ponto da curva.

O trajeto realizado pelo manipulador, utilizando cada método, pode ser visto nos resultados das Figuras 5, 6 e 7, sendo $\in$ o erro médio obtido e σ o desvio padrão em milímetros. Além disso, vídeos foram gravados demonstrando cada algoritmo durante a execução de um determinado caminho [2] .

A magnitude do erro apresentada nas Figuras 5, 6 e 7 deve-se ao critério definido para a troca de pontos. Caso o limite de variação do erro seja reduzido, o erro final será menor, porém, o manipulador apresentará grandes variações de velocidades no decorrer da trajetória.

O algoritmo de enxame de partículas apresenta um erro maior para todos os casos, mantendo uma média de erros superior a 10 mm, no entanto, este apresenta poucos picos de erro, tendo um resultado mais uniforme na maior parte do tempo. Isto se dá pelo fato de ser um método global, no qual o resultado não depende da sua posição atual, obtendo sempre um resultado similar.

Os gráficos das Figuras 8, 9 e 10 apresentam a convergência das juntas durante a execução da trajetória da rhodonea.

[2] Vídeos da execução dos caminhos disponíveis em: `https://bit.ly/3huu9a8`

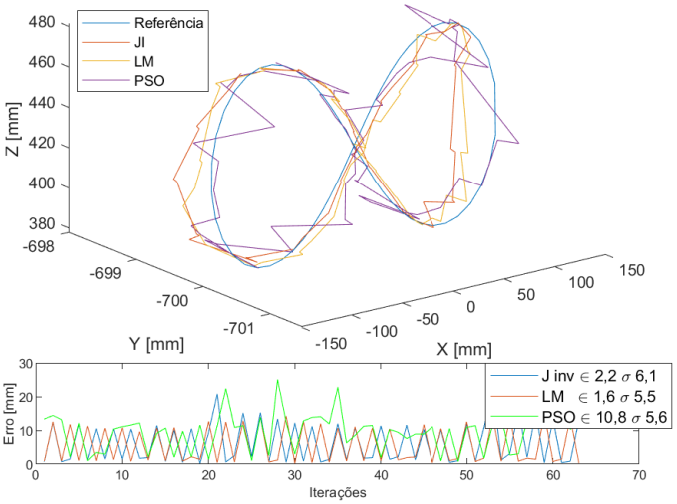

Figura 5. Caminho de uma lemniscata.

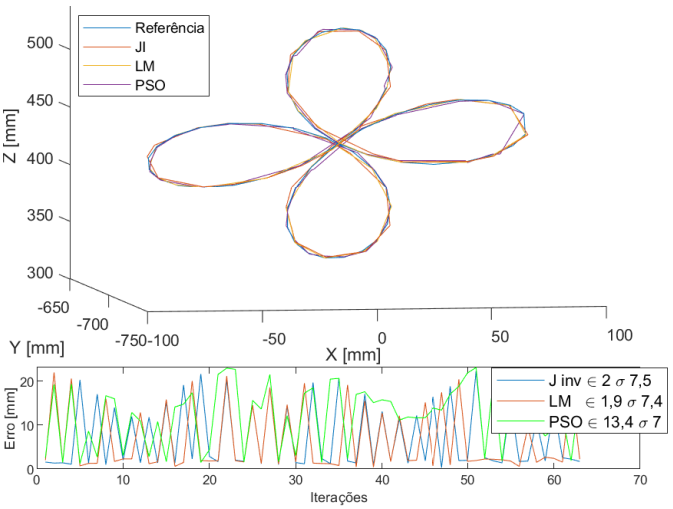

Figura 6. Caminho de uma rhodonea.

O método do jacobiano inverso está trabalhando com frequência na faixa de 60 Hz, enquanto que o Levemberg-Marquardt está trabalhando com frequência na faixa de 50 Hz. Portanto, de acordo com os resultados obtidos nas Figuras 8 e 9, observa-se que as juntas rastreiam a referência proposta, sendo limitados pela dinâmica de movimento do robô, constatando que as técnicas são eficazes para trabalhar com sistemas em tempo real. Referente ao PSO (Figura 10), devido ao seu tempo de processamento ser

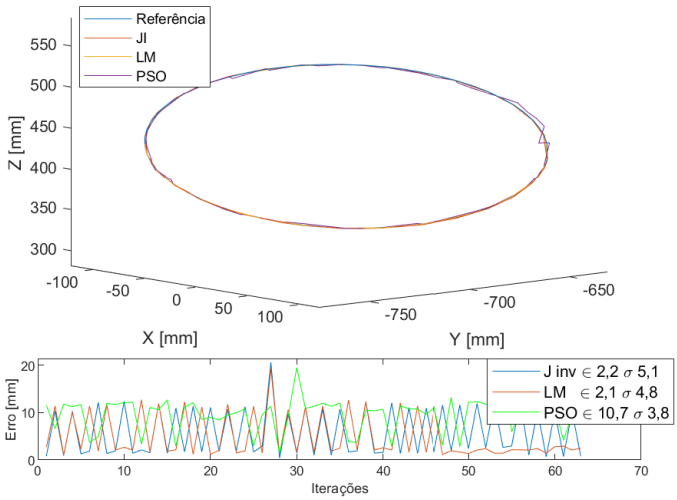

Figura 7. Caminho de um círculo.

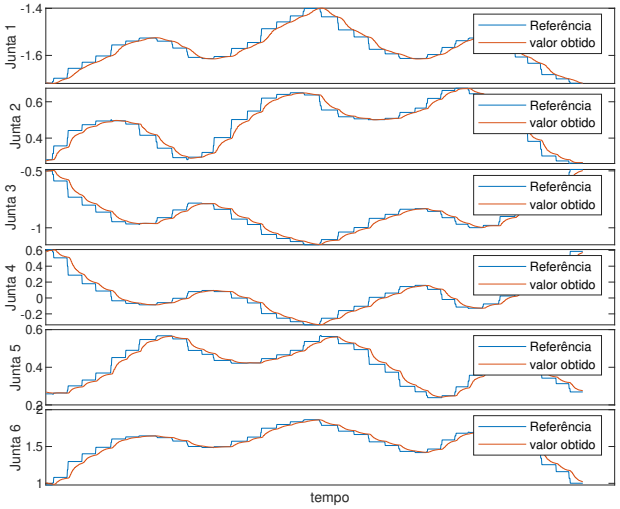

Figura 8. Convergência das juntas no Jacobiano Inverso.

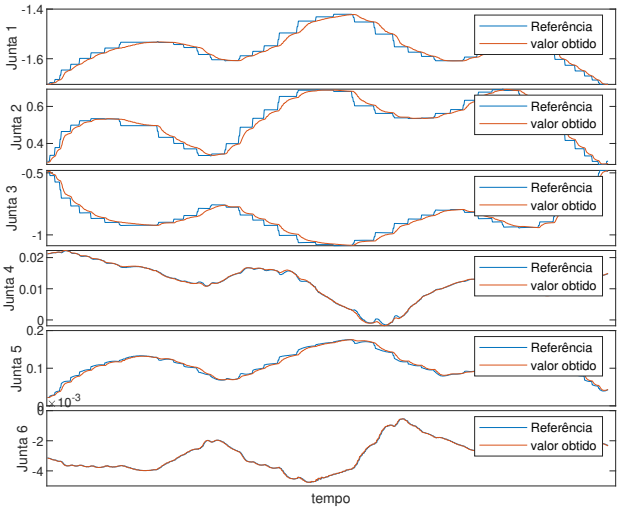

Figura 9. Convergência das juntas no Levemberg-Marquardt.

elevado, acaba por ter uma baixa amostragem de dados, obtendo valores de posição menos precisos.

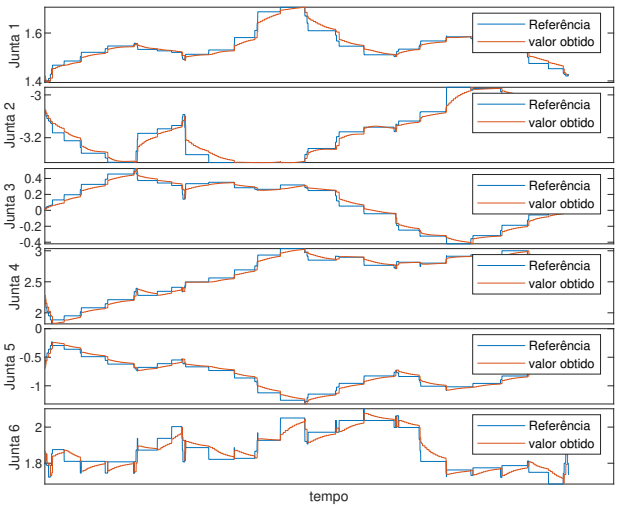

Figura 10. Convergência das juntas no Enxame de Partículas.

5. CONCLUSÃO

Analisando o comportamento dos métodos no teste de pontos individuais, constatou-se que todos possuem erros muito baixos em regime permanente, tendo erro médio na escala de 10μm, salvo o PSO, que, para as configurações utilizadas, possui erro com escala de 100μm. O jacobiano inverso consegue atingir erros menores e com poucas iterações, o que o torna uma opção atrativa para aplicações de exatidão. Entretanto, para esta tarefa deve ser aprimorado, pois sua forma de otimização local, associada ao fator de atenuação utilizado, pode levar o sistema a um erro superior em algumas convergências.

Durante a simulação de caminho, é preciso equilibrar a relação entre o erro e o comportamento uniforme do manipulador. Com o critério para troca de pontos da curva escolhida, o método de Levemberg-Marquardt se destaca em relação aos demais, obtendo erro médio inferior a $3mm$. Apesar do jacobiano inverso obter um resultado muito próximo a esse, a sua natureza oscilatória, dado o tamanho do passo utilizado, acaba por convergir, em alguns momentos, com valores de erros superiores, elevando a média de erros. Devido ao baixo tempo de execução, esses dois métodos se tornam capazes de trabalhar com sistemas em tempo real, funcionando neste projeto entre 40 e 60 Hz. Vale ressaltar que, para o método Levemberg-Marquardt, pode-se reduzir mais seu tempo de processamento em troca de um erro maior.

Outra vantagem destes dois métodos é o fato de que, devido a sua otimização local, permite que o algoritmo possa recalcular a cinemática inversa diversas vezes, enquanto o robô realiza sua dinâmica, a fim de encontrar um erro cada vez menor, sem que afete o percurso desejado. De forma contrária, o PSO, que por ser um método global de inicialização aleatória, pode encontrar configurações muito distintas para um mesmo ponto. Além disso, a complexidade do método faz com que o tempo de execução seja muito elevado, o que diminui a amostragem de dados, causando perda de informações e desta forma, reduz a precisão do método. Em contrapartida, o PSO, por ser uma técnica de otimização global, possui a vantagem de

não ficar preso em mínimos locais, não sendo necessário utilizar de técnicas para contornar estes contratempos.

Para trabalhos futuros, deve ser aplicado os algorítimos no ambiente de operação real. Além disso, é recomendado um estudo mais aprofundado sobre os parâmetros de ajuste de cada método. Para o Jacobiano Inverso, a utilização de um fator de atenuação dinâmico pode ser um bom candidato para reduzir erros em pontos distantes, já o PSO uma análise dos parâmetros de ajuste pode melhorar o desempenho do método. Uma outra possibilidade é a utilização de alguma variação do enxame de partículas, como o QPSO, que teve resultados melhores em outros trabalhos. Não obstante a aplicação do código no robô físico é imprescindível para concretizar os resultados da plataforma de simulação e os algoritmos no sistema real.

REFERÊNCIAS

Alkayyali, M. and Tutunji, T.A. (2019). PSO-based algorithm for inverse kinematics solution of robotic arm manipulators. In *2019 20th International Conference on Research and Education in Mechatronics (REM)*, 1–6.

Boschert, S., Heinrich, C., and Rosen, R. (2018). Next generation digital twin. In *Proc. tmce*, 209–218. Las Palmas de Gran Canaria, Spain.

Denavit, J., Hartenberg, R., Mooring, B., Tang, G., Whitney, D., and Lozinski, C. (1955). Kinematic parameter identification for robotc manipulators. *Journal of applied mechanics*, 77(2), 215–221.

Dereli, S. and Köker, R. (2020). A meta-heuristic proposal for inverse kinematics solution of 7-dof serial robotic manipulator: quantum behaved particle swarm algorithm. *Artificial Intelligence Review*, 53(2), 949–964.

Farzan, S. and DeSouza, G.N. (2013). From DH to inverse kinematics: A fast numerical solution for general robotic manipulators using parallel processing. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2507–2513. IEEE.

Haseeb, M., Hussain, H.I., Ślusarczyk, B., and Jermsittiparsert, K. (2019). Industry 4.0: A solution towards technology challenges of sustainable business performance. *Social Sciences*, 8(5), 154.

Jung, H., Kim, M., Chen, Y., Min, H.G., and Park, T. (2020). Implementation of a unified simulation for robot arm control with object detection based on ros and gazebo. In *2020 17th International Conference on Ubiquitous Robots (UR)*, 368–372. doi:10.1109/UR49135.2020.9144984.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, 1942–1948. IEEE.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, 2149–2154. IEEE.

KUKA (2015). *KR AGILUS sixx: With W and C Variants Operating Instructions*. KUKA Roboter GmbH, Germany.

Lu, Y. (2017). Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6, 1–10. doi:10.1016/j.jii.2017.04.005. Disponível em https://doi.org/10.1016%2Fj.jii.2017.04.005.

Madsen, K., Nielsen, H., and Tingleff, O. (2004). *Methods for Non-Linear Least Squares Problems*. Technical University of Denmark, second edition.

MRPT (2021). Discover mrpt mobile robotics c++ libraries. Disponível em: https://www.mrpt.org.

Niku, S.B. (2010). *Introduction to robotics: analysis, control, applications*. WILEY, California, 2 edition.

Nilsson, R. (2009). *Inverse kinematics*. Master's thesis, Luleá University of Technology. Department of Computer Science and Electrical Engineering.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y., et al. (2009). ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 5. Kobe, Japan.

Rocha, F.F. and Bessa, W.M. (2012). Cinemática de um manipulador robótico utilizando uma abordagem computacional. Congresso de Matemática Aplicada e Computacional, Nordeste.

Serenza, E.N. (2015). Análise cinemática dos movimentos de um guindaste florestal. Universidade de Caxias do Sul.

Sugihara, T. (2011). Solvability-unconcerned inverse kinematics by the levenberg–marquardt method. *IEEE Transactions on Robotics*, 27(5), 984–991.

Trabasso, L.G., Negri, D., Kapp, W.A., Engström, M., Natal, G.S., and Johansen, K. (2019). Airframe sealing automation using snake-robot. *FT2019*.

Wang, X., Zhang, D., and Zhao, C. (2017). Inverse kinematics of a 7r 6-dof robot with nonspherical wrist based on transformation into the 6r robot. *Mathematical Problems in Engineering*, 2017.

Whitney, D.E. (1969). Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10(2), 47–53. doi:10.1109/TMMS.1969.299896.