EasyChair Preprint

№ 11080

Kinematic Synchronization Control for Digital Twin Counterparts Through the New SyncLMKD Method

Fabiano Stingelin Cardoso, Alvaro Rogério Cantieri and André Schneider de Oliveira

October 12, 2023

Controle Cinemático de Sincronização para as Contrapartes do Gêmeo Digital Através do Novo Método SyncLMKD

Fabiano Stingelin Cardoso* Alvaro Rogério Cantieri*
André Schneider de Oliveira**

**Universidade Tecnológica Federal do Paraná / Instituto Federal do Paraná, (e-mail: {fabiano.cardoso@, alvaro.cantieri}@ifpr.edu.br).*
*** Universidade Tecnológica Federal do Paraná, (e-mail: andreoliveira@utfpr.edu.br)*

Abstract: A digital twin can be seen as "the digital representation of a specific physical product, in the form of computer models, identified by their serial numbers, at a specific time and with sufficient fidelity to simulate the corresponding physical product", as described by Singh and Willcox (2018). In this way, the digital twin can have different levels of fidelity that represent the physical model in its life cycle and specific moment. In this paper, a new method for synchronizing the counterparts of the digital twin is developed, assigning sufficient fidelity levels to the demands of dynamic flexibility required by Industry 4.0. Experiments with the proposed approach were performed on a virtual simulation platform that supports the new SyncLMKD method. The obtained results verified that the SyncLMKD method is able both to track and measure the synchronization efficiency and to adjust the synchronization for the virtual and physical counterparts, for digital twin applications.

Resumo: Um gêmeo digital pode ser visto como "a representação digital de um produto físico específico, na forma de modelos computacionais, identificados por seus números de séries, em um momento específico e com fidelidade suficiente para simular o produto físico correspondente", conforme descrevem Singh e Willcox (2018). Dessa forma, o gêmeo digital pode ter diferentes níveis de fidelidade que representem o modelo físico em seu ciclo de vida e momento específico. Neste artigo é desenvolvido um novo método para a sincronização das contrapartes do gêmeo digital, atribuindo níveis de fidelidade suficientes às demandas de flexibilidade dinâmica requeridas pela Indústria 4.0. Experimentos com a abordagem proposta foram realizados em uma plataforma de simulação virtual que suporta o novo método SyncLMKD. Os resultados obtidos aferiram que o método SyncLMKD é capaz tanto de rastrear e medir a eficiência de sincronização como de ajustar a sincronização para as contrapartes virtual e física, para aplicações de gêmeos digitais.

Keywords: Industry 4.0; Digital twins; Synchronization; Dynamic flexibility; SyncLMKD Method.

Palavras-chaves: Indústria 4.0; Gêmeos digitais; Sincronização; Flexibilidade dinâmica; Método SyncLMKD.

1. INTRODUÇÃO

O gêmeo digital é descrito como uma entidade física, uma contraparte virtual e as conexões de dados entre elas. Ele está sendo cada vez mais explorado como um meio de melhorar o desempenho de entidades físicas, por meio do aproveitamento de técnicas computacionais, ativadas através da contraparte virtual ou pela contraparte física.

Grieves, o criador do conceito de gêmeo digital, e Vickers adaptaram, aplicaram e testaram o método de previsão e eliminação de falhas proposto por Turing (1950). Turing chamou esse método de "Jogo da Imitação". No teste adaptado por Grieves, um observador humano foi posicionado em uma sala que possui duas telas, onde foi apresentado o sistema baseado na tecnologia de gêmeos digitais. A primeira tela apresentou o vídeo da sala real onde um sistema físico estava presente. A segunda tela fez a representação através de um computador. Na sequencia, três testes foram passados para o observador humano realizar: a) teste visual sensorial; b) teste de desempenho; c) e teste de refletividade. Com base nos resultados aferidos pelo observador humano, alcançou-se o nível de fidelidade existente entre as contrapartes físicas e virtuais, Grieves e Vickers (2017).

Essa sincronização dos estados virtuais e físicos, ao ponto dos parâmetros virtuais terem os mesmos valores que os parâmetros físicos do gêmeo digital, é conhecida na literatura como "geminação", conforme Schleich et al. (2017). Assim, pode-se dizer que as entidades estão "geminadas" ou simplesmente "sincronizadas".

Mas, para garantir a consistência e o sincronismo entre os espaços físicos e virtuais em uma alta fidelidade, executar em

tempo online da simulação e manter o controle do gêmeo digital quando ativados, é essencial a "conjunção firme e coordenação entre recursos computacionais e físicos", segundo Baheti e Gill (2011). Contudo, algumas perturbações podem ocorrer, tais como: inconsistências causadas nos modelos virtuais; erro da pré-calibração; e distúrbios na entidade física e no ambiente de infraestrutura montado.

Ademais, outro problema grave é que, na prática, a representação do robô móvel e a sua interação com o ambiente físico e virtual não consegue seguir o caminho planejado para o alvo da posição, por conta das não linearidades existentes. Muitas pesquisas tentaram fechar essa lacuna por meio de soluções via software, como por exemplo, Müller et al. (2022), Liang et al. (2020), Kuts et al. (2020), porém, tendo em vista que o mundo físico é dinâmico e muito complexo pra ser inteiramente digitalizado, as técnicas apresentadas mostraram-se limitadas para lidar com a completude desses fatores não lineares em tempo online.

Nesse contexto, o presente artigo propõe o novo método da Diferença Cinemática Média Logarítmica de Sincronização - SyncLMKD que objetiva o rastreamento, a medição da eficiência e o ajuste de sincronização entre as contrapartes da aplicação do gêmeo digital. E a fim de validar esse novo método, foram realizadas simulações em um ambiente virtual. Também foi elaborada uma representação de aplicação para o gêmeo digital. Além disso, foram feitos testes com controle Proporcional, Integral e Derivativo – PID a fim de comparar com o método SyncLMKD.

Este artigo está organizado da seguinte forma: a modelagem do novo método SyncLMKD na seção 2; as principais características da arquitetura desenvolvida são apresentadas na seção 3; o controle de sincronização na seção 4; os resultados são demonstrados na seção 5; e as considerações finais e trabalhos futuros na seção 6.

2. MÉTODO SyncLMKD

A modelagem do método SyncLMKD surgiu a partir da análise das verossimilhanças encontradas entre os problemas e soluções que o Método da Diferença de Temperatura Média Logarítmica – DTML resolve relacionados à transferência de calor. E, aplicando a generalização abdutiva (Wertheimer, 2020), segundo o qual é possível generalizar a solução de um problema para aplicá-lo em outro, foi possível constatar que o método SyncLMKD é adequado para resolver problemas de cinemática para robôs móveis em ambientes dinâmicos. Principalmente quando se trata do espelhamento para a aplicação do gêmeo digital.

Nesse sentido, é relevante contextualizar o que torna o método DTML adequado para calcular a transferência de calor na interface do trocador.

Pois bem. O método da DTML é adequado quando a temperatura alvo é conhecida. Mas, não seria possível simplesmente fazer uma média das diferenças de temperatura, pois essa diferença de temperatura entre o fluido quente e frio varia ao longo do trocador de calor. Assim, usar a diferença de temperatura média logarítmica é a forma mais adequada para calcular a taxa de transferência de calor. Com isso, a taxa de transferência de calor é proporcional à diferença de temperatura entre os fluidos.

Na sequência, uma premissa será formulada para corroborar uma hipótese sobre a pertinência do novo método SyncLMKD.

Partindo do mesmo pressuposto anteriormente contextualizado, vale indagar: "Seria possível fazer simplesmente uma média das diferenças de movimentos cinemáticos entre o robô real e virtual?". Não, porque a diferença de posição entre o robô real e virtual varia ao longo do trajeto planejado. Assim chega-se a seguinte hipótese: usar a diferença de cinemática média logarítmica (método SyncLMKD) é a forma mais adequada para calcular a taxa de transferência de posição entre o robô real e o virtual.

Portanto, com base na hipótese acima, e aplicando o raciocínio generalizado abdutivo (que não será detalhado neste artigo), ainda faz-se necessário avaliar conjuntos de dados limitados a fim de certificar a hipótese. Para isso, foram analisados conjuntos de dados exportados para uma planilha do Excel, referentes a um experimento realizado para medir a transferência de calor em unidades de processamento de dados, Cardoso et al. (2019). Ademais, com o intuito de melhor embasar o entendimento da modelagem proposta, um vídeo está disponível no YouTube: https://www.youtube.com/watch?v=Ed4ix_jK-gc.

3. ARQUITETURA DESENVOLVIDA

A arquitetura foi desenvolvida no ambiente de simulação CoppeliaSim, anteriormente conhecido como Vrep, que foi instalado e configurado conjuntamente com o ROS e que permitiu testar as abordagens de controle, algoritmos de planejamento de caminhos, dentre outros.

3.1 ROS

Nesta seção, serão fornecidos detalhes adicionais das conexões com o ROS Melodic no ambiente de simulação CoppeliaSim. Ambos instalados no Ubuntu 18.04 LTS.

As linguagens de programação de alto nível, tais como Java, C++, Lua, Python, entre outras, trabalham com "objetos", que, por sua vez, precisam ser construídos pelo programador. Os objetos de portas "Com", por exemplo, estabelecem a comunicação com o robô por meio de seu endereço IP e o número de porta. Além desses, é necessário uma infinidade de outros objetos para interagir com sensores, como "Lidar" e "Odometria", que devem ser criados. E, assim que a execução termina, os objetos são desconectados e removidos. Basicamente, os objetos que trocam informações entre os scripts Lua no CoppeliaSim e os scripts C++ no ROS são os objetos, ou nós de acordo com o middleware ROS, apresentados na Figura 1.

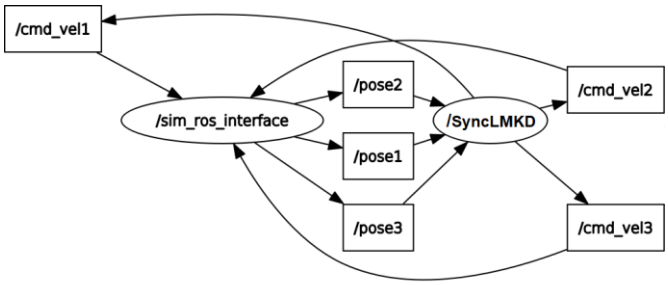

Fig. 1 Nós (objetos) que trocam informações entre o ROS e o CoppeliaSim.

Os nós /cmd_vel1, /cmd_vel2 e /cmd_vel3 representam as velocidades enviadas para os deslocamentos do Robot1, Robot2 e Path_Planning. Embora o nó /cmd_vel3 tenha sido criado, nenhuma velocidade lhe é passada. Portanto, a velocidade de deslocamento do Path_Planning (item 6 na Figura 2) é fornecida pelo planejamento de caminho do algorítimo BiTRRT. Já os nós /pose1, /pose2 e /pose3 representam, respectivamente, as odometrias do robô virtual, robô real (item 5 da Figura 2) e caminho idealizado (Path_Planning), sendo todos obtidos pelo nó principal /SyncLMKD e vinculados pelo nó de troca de mensagens /sim_ros_interface do próprio pacóte ROS.

3.2 CoppeliaSim

A Figura 2 apresenta detalhes da implementação da cena no CoppeliaSim, que faz a representação de uma aplicação de gêmeos digitais e demonstra a medição da eficiência e do ajuste de sincronização do movimento entre as contrapartes através do método SyncLMKD.

Tendo em vista que para os cálculos do SyncLMKD utiliza-se apenas as coordenadas x e y, os indicativos 1, 2 e 3 mostram três superfícies separadas para deslocamento, que estão alinhadas ao eixo z em zero, centralmente. Dessa forma, o indicativo 1 é a superfície do robô virtual, 2 a superfície do caminho idealizado e 3 a superfície do robô real, sendo que cada objeto possui uma superfície de deslocamento igual a do outro. Para a visualização conjunta, a superfície de deslocamento 2 e 3 foi configurada como transparente.

As contrapartes, Robot1 (na cor vermelha que representa o robô virtual) e Robot2 (na cor azul que representa o robô real), indicativo 5. O caminho idealizado “Path_Planning”, indicativo 6, utiliza o exemplo da cena “mobileRobotPathPlanning.ttt” do próprio CoppeliaSim que também foi configurado como transparente.

Inicialmente, pretendia-se riscar a superfície de deslocamento e obter os pontos de coordenadas para os cálculos do SyncLMKD. Mas isso não foi feito, pois inviabilizaria a questão do tempo de deslocamento do caminho idealizado, que compreende também o planejamento do tempo de deslocamento, afinal a proposta se baseia no movimento livre com graus de liberdade, previamente estabelecidos e configurados de acordo com o perfil de interface de deslocamento.

Outro fator foi a questão de tratar o desvio de obstáculos fixos e dinâmicos em tempo online. Para isso, é necessário um algoritmo de planejamento de caminho (e o exemplo já utiliza o algoritmo BiTRRT) e o método de otimização estocástica, que é usado para calcular os mínimos globais em espaços complexos.

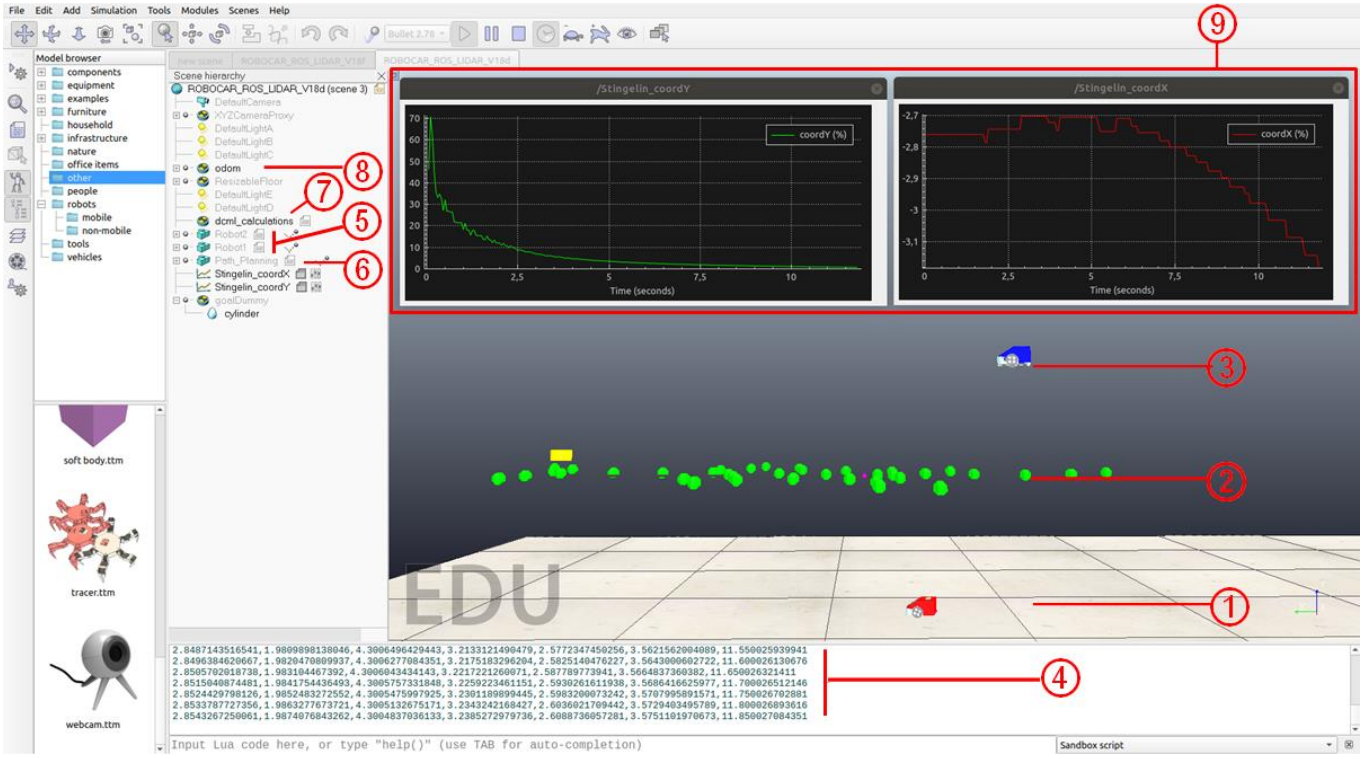

Fig. 2 Cena de simulação no CoppeliaSim.

O "dcml_calculations", indicativo 7, realiza os cálculos do método SyncLMKD, no caso da cena no CoppeliaSim, apenas para plotagem dos gráficos em tempo online.

Durante a execução da simulação, os dados de posição em função do tempo foram impressos na saída do "CoppeliaSim:info", indicativo 4. Na sequência da esquerda para direita mostram-se os seguintes dados de rastreamento de posição: robô virtual (eixo x), robô real (eixo x), caminho idealizado (eixo x), robô virtual (eixo y), robô real (eixo y), caminho idealizado (eixo y) e tempo decorrido em segundos.

Para plotar a sincronização dos robôs "virtual e real", dois gráficos foram incluídos na cena, indicativo 9: sendo um gráfico que mostra a eficiência e o ajuste de trajetória no eixo x, e o outro que mostra a eficiência e o ajuste de trajetória no eixo y.

Propositalmente, os robôs e o caminho idealizado foram posicionados no segundo quadrante da cena, indicativo 8, tendo em vista que as três superfícies de deslocamentos utilizam as mesmas referências de localização e odometria, porém em posições diferentes para os eixos x e y.

4. CONTROLE DE SINCRONIZAÇÃO

A abordagem do controle de sincronização apresentada baseia-se em um método dinâmico e de tempo online que independe de técnicas que geralmente atrapalham a execução direta. Por exemplo, técnicas de aprendizado de reforço e treinamento de RNA's que se baseiam na repetição da simulação para alcançar o aperfeiçoamento. Embora essas técnicas funcionem, elas demandam tempo e processamento extra. Nesse ponto, o novo método SyncLMKD mostra-se viável em alcançar os seguintes objetivos: a sincronização das contrapartes e o emprego numa aplicação para gêmeo digital. Assim, indo na contramão das propostas existentes no estado da arte, que, em suma, tratam separadamente ambas as coisas: a sincronização e a aplicação do gêmeo digital.

Outro aspecto relevante do método SyncLMKD é sua capacidade de manter a sincronização rígida e relaxada para as contrapartes na aplicação do gêmeo digital. Será rígida quando as contrapartes tiverem que manter a fidelidade do deslocamento. E será relaxada quando as contrapartes não tiverem que manter fidelidade, ou seja, com algum grau de liberdade entre elas. O grau de liberdade baseia-se na propagação logarítmica e no controle dessa propagação através do estabelecimento de *setpoint's* que o torna customizável ao longo de todo o trajeto (interface de deslocamento).

Dessa forma, o pseudocódigo do controle para o novo método de sincronização SyncLMKD é mostrado abaixo:

Algoritmo: SyncLMKD

1. **entradas:**
2. $posicao\_robo1[x,y]$,
3. $posicao\_robo2[x,y]$,
4. $posicao_$idealizado$[x,y]$,
5.
6. $tolerancia\_max$,
7. $tolerancia\_min$,
8. $vel\_linearR1$,
9. $vel\_linearR2$,
10. vel_linearIdeal,
11. vel_linearMin,
12. $vel_$angularErro;
13.
14. **enquanto** e$rro_distancia_robo1 > tolerancia_min$ e e$rro_distancia_robo2 > tolerancia_min$ **faça**
15.
16. e$rro_distancia_robo1 = distancia_euclidiana$;
17. e$rro_distancia_robo\ 2 = distancia_euclidiana$;
18.
19. $deltaR1\_X = posicao\_idealizado[x] - posicao\_robot1[x]$;
20. $deltaR2\_X = posicao\_idealizado[x] - posicao\_robot2[x]$;
21.
22. $deltaR1\_Y = posicao\_idealizado[y] - posicao\_robot1[y]$;
23. $deltaR2\_Y = posicao\_idealizado[y] - posicao\_robot2[y]$;
24.
25. $coordY = deltaR1\_Y - deltaR2\_Y\ /log(deltaR1\_Y\ /deltaR2\_Y)$
26. $coordX = deltaR1\_X - deltaR2\_X/log(deltaR1\_X/\ deltaR2\_X)$
27.
28. $flag\_sincY = coordY > 0$
29. $flag\_sincX = coordX > 0$
30.
31. $flag\_nan\_sincY = coordY = = NaN$
32. $flag\_nan\_sincX = coordX = = NaN$
33.
34. **se** $flag\_sincY$ **então**
35. **se** $deltaR1\_Y > tolerancia\_max$ **então**
36. $robo1 = vel\_linearR1$
37. **senão**,
38. $robo1 = vel\_linearIdeal$
39. **finaliza se**
40. **se** $deltaR2\_Y > tolerancia\_max$ **então**
41. $robo2 = vel\_linearR2$
42. **senão**,
43. $robo2 = vel\_linearIdeal$
44. **finaliza se**
45. **retorna** $robo1, robo2$
46. **senão**,
47. **se** $deltaR1\_Y > tolerancia\_max$ **então**
48. $robo1 = vel\_linearR1 + Ajuste(efeito\_derivativo)$
49. **senão**,
50. $robo1 = vel\_linearIdeal$
51. **finaliza se**
52. **se** $deltaR2\_Y > tolerancia\_max$ **então**
53. $robo2 = vel\_linearR2 + Ajuste(efeito\_\ derivativo)$
54. **senão**,
55. $robo2 = vel\_linearIdeal$
56. **finaliza se**
57. **retorna** $robo1, robo2$
58. **finaliza se**
59.
60. **se** $flag\_sincX$ **então**
61. **se** $deltaR1\_X > tolerancia\_max$ **então**
62. $robo1 = vel\_angularErro$
63. **finaliza se**
64. **se** $deltaR2\_X > tolerancia\_max$ **então**
65. $robo2 = vel\_angularErro$
66. **finaliza se**
67. **senão**,
68. **se** $deltaR1\_X > tolerancia\_max$ **então**
69. $robo1 = vel\_angularErro$
70. **finaliza se**
71. **se** $deltaR2\_X > tolerancia\_max$ **então**
72. $robo2 = vel\_angularErro$
73. **finaliza se**
74. **retorna** $robo1, robo2$
75. **finaliza se**
76.
77. **se** $flag\_nan\_sincY$ **então**
78. **se** $deltaR1\_Y > tolerancia\_min$ **então**
79. $robo1 = vel\_linearMin$
80. **senão**,
81. $robo1 = 0$
82. **finaliza se**

```
    se deltaR2_Y > tolerancia_min então
        robo2 = vel_linearMin
    senão,
        robo2 = 0
    finaliza se
    retorna robo1, robo2
finaliza se

se flag_nan_sincX então
    se deltaR1_X > tolerancia_min então
        robo1 = vel_angularErro
    senão,
        robo1 = 0
    finaliza se
    se deltaR2_X > tolerancia_min então
        robo2 = vel_angularErro
    senão,
        robo2 = 0
    finaliza se
    retorna robo1, robo2
finaliza se
finaliza enquanto
```

No pseudocódigo acima, inicialmente realiza-se as entradas das posições relativas e também se obtém os parâmetros de configuração para o relaxamento ou rigidez do deslocamento em relação ao alvo dinâmico e fixo. No caso do dinâmico é a posição idealizada, Path_Planning. E no caso do alvo fixo é a posição destino final, ou seja, calculada pela distancia_euclidiana entre o robô virtual e o cilindro amarelo e o robô real o cilindro amarelo. Posteriormente, um laço se encarrega de controlar a chegada até o destino final de deslocamento de ambos os robôs, virtual e real. E, dentro do laço, algumas flags são usadas para controlar momentos específicos dos ajustes.

A parte principal do SyncLMKD é calculada em coordY e coordX, para então se verificar quatro condições que retornam verdadeiro ou falso. Dentro de cada condição, flags, sinais são enviados para os ajustes de velocidades e angulares das contrapartes.

Onde:

• flag_sincY verifica se o retorno booleano da coordY é verdadeiro. Sendo isso afirmativo, significa que o Robot1 está a frente do Robot2 na coordenada Y, velocidades diferentes são enviadas para ambos os robôs com o objetivo de aproximarem-se. Quando coordY retorna falso, significa que o Robot2 ultrapassou Robot1 na coordenada Y em alguns poucos milímetros. Então a chamada da função ajuste derivativo é acionada e aplicada sobre o valor de velocidade do Robot2, fazendo com que Robot1 e Robot2 tenham velocidades ajustadas, ou seja, sincronizem-se. Enquanto a verificação de distância de ambos os robôs não alcançarem o limite máximo permitido em relação ao deslocamento do caminho idealizado na coordenada Y, ambos seguirão juntos até atingirem esse *setpoint*,

• flag_sincX verifica se o retorno booleano da coordX é verdadeiro ou falso, independente do retorno, uma vez que a coordenada Y já foi alcançada. Em ambos os sinais booleanos, calcula-se o erro de orientação e passa as respectivas velocidades de orientação aos robôs. Enquanto a verificação de distância de ambos os robôs não alcançarem o limite máximo permitido em relação ao deslocamento do caminho planejado na coordenada X, ambos seguirão juntos ajustando os ângulos e velocidades até atingirem o *setpoint*,

• flag_nan_sincY é o retorno de valor do tipo NaN para a coordY. Quando isso ocorre significa que tentou-se realizar a operação matemática da divisão de um número zero por outro zero e, posteriormente, tentou-se tirar o seu logarítmico, Log(0/0), uma operação não permitida na matemática. Dessa forma, compreende-se que a sincronização em relação ao eixo Y ocorreu entre as contrapartes Robot1, Robot2 e o caminho idealizado (Path_Planning),

• flag_nan_sincX é o retorno de valor do tipo NaN para a coordX, idem flag_nan_sincY. Dessa forma, entende-se que a sincronização em relação ao eixo X também ocorreu entre as contrapartes Robot1, Robot2 e o caminho idealizado (Path_Planning).

O valor percentual do SyncLMKD indica o momento para que o efeito derivativo seja iniciado e ocorra o processo de derivação até que as velocidades lineares fiquem próximas. O valor de ajuste de velocidade, retorno da função Ajuste (efeito_derivativo), é adicionado ao valor da velocidade pré-configurada, possibilitando, assim, que ambas as contrapartes sigam sincronizadas.

5. RESULTADOS

Durante a elaboração desta seção, pensou-se em comparar o método SyncLMKD com outros métodos de sincronização para as contrapartes de gêmeos digitais numa mesma aplicação. Porém, observou-se que o processo de geminação encontra-se num estágio embrionário do ponto de vista de soluções em software, não sendo possível realizar a comparação.

Os gráficos presentes nas Figuras 3 e 4 foram gerados a partir dos dados da simulação no CoppeliaSim, os quais sofrem interferência externa dos sinais de ajustes de velocidades advindos do método SyncLMKD presente no ROS. Inicialmente, os robots (virtual e real) seguem o deslocamento dinâmico do Path_Planning (esfera na cor roxa que se desloca ao longo do caminho idealizado), mantendo assim a distância de tolerância máxima previamente configurada. Após o alcance e acompanhamento do objeto dinâmico, Path_Planning, a sub rotina de tolerância mínima só se encerra quando se alcançar o objeto fixo (cilindro na cor amarela). Em ambos os casos, objeto fixo (cilindro amarelo) e objeto dinâmico (esfera na cor roxa), as coordenadas x e y são consideradas e desprezada a coordenada do eixo z.

Ao observar os contrapontos entre os sinais positivos e negativos nos gráficos das Figuras 3 e 4, se entende que ocorreu a ultrapassagem nesses determinados tempos de simulação logo nos primeiros milímetros de ultrapassagem, permitindo que a função do efeito derivativo do método SyncLMKD seja acionada, e então reajustadas as velocidades dos motores até alcançarem a sincronização entre as contrapartes que estão em rota de sincronização com o caminho idealizado (Path_Planning). É importante destacar que tanto para o eixo x quanto para o eixo y, nos gráficos das Figuras 3 e 4, a variação se manteve próxima de zero e os

pontos se mantiveram dispersos em zero ao longo do tempo. No entendimento do gráfico, isso demonstra que as diferenças cinemáticas não foram significativas.

Outro ponto relevante é em relação ao ajuste rígido e relaxado, entre o intervalo de tempo de simulação de 100 a 111 segundos, quando aplicado o efeito derivativo do método SyncLMKD ao alcançar o objeto dinâmico e fixo. Nesses respectivos tempos, a tolerância máxima estipulada como *setpoint* se manteve a diferença 0,10 cm entre as contrapartes e o caminho idealizado (objeto dinâmico). Enquanto que, ao se aproximar do objeto fixo, a tolerância mínima chegou ao que foi previamente configurado de 0,01 cm, demonstrando que tolera o relaxamento e a rigidez no controle do método SyncLMKD. Uma demonstração em vídeo está disponível no YouTube (https://youtu.be/Ej5aKW904q4).

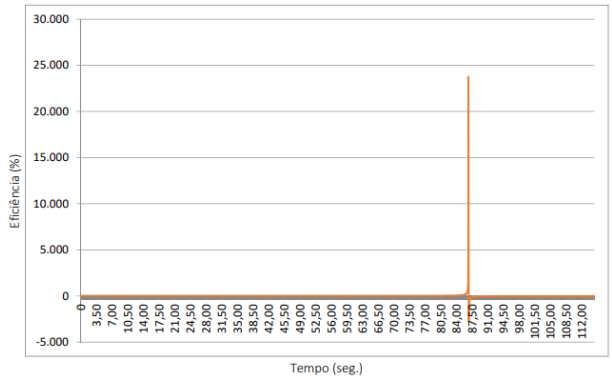

Fig. 3 Movimento no eixo x com o controle SyncLMKD.

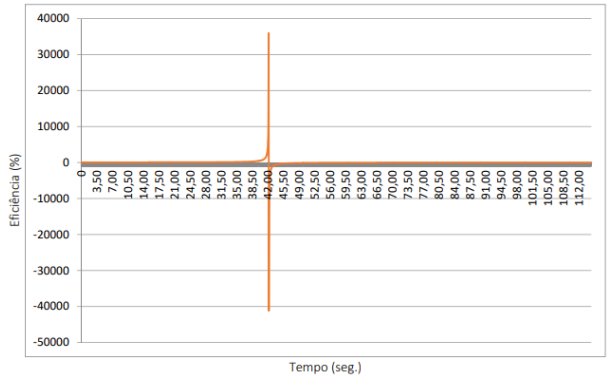

Fig. 4 Movimento no eixo y com o controle SyncLMKD.

É observada uma sutil elevação em rampa, antes da acentuação do pico, nos gráficos das Figuras 3 e 4. Conforme dados nas tabelas 1 e 2, isso ocorre porque, embora as contrapartes estejam em posições bem próximas, o caminho idealizado encontra-se mais distante delas. Dessa forma, quanto mais longe estiver o caminho idealizado, maior será a rampa mesmo que as contrapartes estejam sincronizadas. E quando ocorrer a confluência entre as três entidades (virtual, real e idealizado), a rampa será praticamente nula.

Tabela 1 Dados relevantes do SyncLMKD para aplicação do efeito derivativo eixo X

Tempo (seg)	Virtual (m)	Real (m)	Idealizado (m)	SyncLMKD (%)
86,60	5,2836	5,2818	6,7439	1167,9744
86,65	5,2860	5,2851	6,7439	2238,0235
86,70	5,2884	5,2883	6,7439	23763,0595
86,75	5,2908	5,2916	6,7439	-2765,8765
86,80	5,2932	5,2948	6,7439	-1302,2483

Tabela 2 Dados relevântes do SyncLMKD para aplicação do efeito derivativo eixo Y

Tempo (seg)	Virtual (m)	Real (m)	Idealizado (m)	SyncLMKD (%)
42,15	6,0829	6,0814	10,3508	12556,0946
42,20	6,0869	6,0864	10,3530	35916,2574
42,25	6,0909	6,0914	10,3552	-41171,1340
42,30	6,0949	6,0963	10,3573	-13060,0524
42,35	6,0989	6,1013	10,3595	-7734,5780

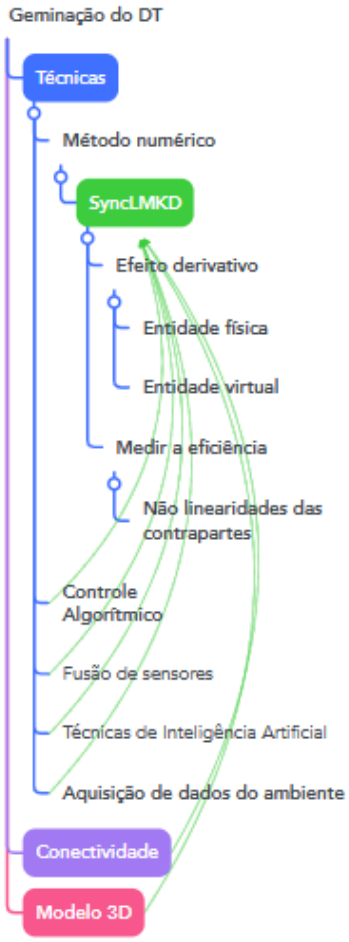

Fig. 5 Delimitação do método SyncLMKD.

O clássico controle Proporcional, Integral e Derivativo - PID foi embarcado em ambas as contrapartes, a fim de ser observada a sua eficiência através do método SyncLMKD. A Figura 6 refere-se ao deslocamento em y com o controle PID. Os pontos da linha do gráfico estão dispersos e espalhados, ou seja, ocorreu uma variação cinemática praticamente em todo o tempo de execução da simulação. Isso demonstra que o deslocamento sincronizado no eixo y teve baixa eficiência por meio do controle PID. No tempo 79,75 seg. e eficiência de 15% começa uma descida em curva. Isso ocorre porque o Path_Planning alcança a posição destino, permanecendo

parado, tornando mais fácil para o controle PID recalcular suas coordenadas.

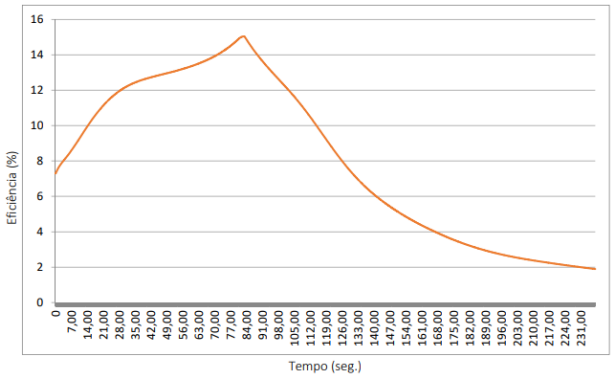

Fig. 6 Movimento no eixo y apenas com a medição da eficiência do controle PID através do SyncLMKD.

Na simulação com o uso do controle PID não ocorreu o acoplamento de ajuste de velocidades do método SyncLMKD. Ocorreu apenas a medição da eficiência do controlador PID em seguir o caminho idealizado. Conforme a Figura 5, o método SyncLMKD possibilita tanto aplicar o ajuste de velocidade entre as contrapartes, como medir a eficiência do controle. As técnicas de Fusão de sensores e de Inteligência Artificial não corroboram o presente trabalho.

6. CONCLUSÃO

Este artigo discutiu inicialmente a propriedade de reflexão ou fidelidade do gêmeo digital, problemas de controle e deslocamento e motivações para o emprego da nova abordagem com o método SyncLMKD. Também apresentou trabalhos que possuem alguma correlação com a sincronização e controle de gêmeos digitais, com destaque na abordagem para problemas de deslocamento envolvendo robôs móveis. Com isso, evidenciou-se que há um campo promissor de pesquisa na área de gêmeos digitais, contudo verificou-se também que ainda existem inúmeros desafios a serem vencidos. Posteriormente, foi demonstrada a aplicação da generalização abdutiva para a modelagem do novo método SyncLMKD. O ambiente de simulação preparado fez a representação da aplicação real de gêmeos digitais, que foi empregada para rastrear, medir a eficiência e controlar o ajuste do efeito derivativo.

Várias abordagens anteriores foram iniciadas, porém notou-se que a melhor abordagem seria começar pela mais simples para se constatar a viabilidade do método SyncLMKD. Dessa forma, a simulação foi simplificada por adotar 3 ações: 1) escolher robôs móveis diferenciais; 2) considerar apenas os eixos x e y para o deslocamento linear e angular; 3) alterar as velocidades dos motores das contrapartes. Nesse último caso, ambas as contrapartes e o caminho idealizado encontravam-se inicialmente dessincronizados por estarem em posições diferentes tanto para o eixo x quanto para o eixo y.

O controle do método SyncLMKD forneceu sinalizadores importantes que direcionaram a movimentação das contrapartes para seguir o objeto dinâmico. Quando ocorreu a aproximação para a intersecção entre as contrapartes, o controle do método SyncLMKD executou a rotina de efeito derivativo que garantiu a permanência de sincronização. Então, ambas as contrapartes seguiram juntas acompanhando o objeto dinâmico (Path_Planning). E, ao se aproximarem, mantiveram o relaxamento de 10 cm pré-configurado de aproximação, perfazendo uma fidelidade relaxada em relação ao objeto dinâmico. Na parte final do deslocamento, ao se aproximarem do objeto fixo (cilindro amarelo), mantiveram o relaxamento 0,01 cm. Comparativamente, o método SyncLMKD se mostrou melhor em relação ao PID clássico, principalmente no quesito de seguir o objeto dinâmico.

Na simulação objeto deste artigo, optou-se por investigar a não linearidade de estarem em posições e velocidades pré-configuradas diferentes. Em estudos futuros, pretende-se a realização de testes com o robô físico real, onde serão investigadas as não linearidades, tais como: controladores não calibrados ou mesmo diferentes; ambientes de simulação de fabricantes diferentes; atrito físico com a superfície real; massa inercial; pressão atmosférica; vibrações de motores e atuadores; dentre outros.

Portanto, o presente trabalho contribuiu com uma solução em software para o problema das contrapartes de robôs móveis não conseguirem seguir o caminho planejado numa aplicação para gêmeos digitais. Além disso, o controle SyncLMKD possibilitou que as contrapartes seguissem o caminho planejado de forma sincronizada, o que vai na contramão dos atuais trabalhos encontrados no estado da arte, que em suma tratam separadamente a sincronização de robôs e a aplicação de gêmeos digitais.

REFERÊNCIAS

Baheti, R., Gill, H. (2011). Cyber-physical systems. Impact Control Technol.

Cardoso, F., Nazareno, J., Negrão, E. (2019). Analysis of heat transfer refrigeration systems of data processing units. International Journal of Computer Applications,

Grieves, M., Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems.

Kuts, V., Cherezova, N., Šarkans, M., Otto, T. (2020). Digital twin: industrial robot kinematic model integration to the virtual reality environment. Journal of Machine Engineering,

Liang, C., Mcgee, W., Menassa, C., Kamat, V. (2020). Bi-directional communication bridge for state synchronization between digital twin simulations and physical construction robots.

Müller, M., Jazdi, N., Weyrich, M. (2022) Self-improving models for the intelligent digital twin: Towards closing the reality-to-simulation gap. IFAC-PapersOnLine.

Schleich, B., Anwer, N., Mathieu, L., Wartzack, S. (2017) Shaping the digital twin for design and production engineering. CIRP Annals - Manufacturing Technology,

Singh, V., Willcox, K. (2018). Engineering design with digital thread. AIAA Journal.

Turing, A. (1950) Computing machinery and intelligence.

Wertheimer, M. (2020). Productive Thinking.