PROTOTIPAÇÃO VIRTUAL DE SISTEMAS EMBARCADOS: ESTUDO DE CASO EM ROBÓTICA MÓVEL

Bruno Almeida da Silva (eng.fis.bruno@gmail.com) – Grupo de Automação e Controle, Universidade de Brasília (UnB).

Jones Yudi (jonesyudi@unb.br) – Grupo de Automação e Controle, Universidade de Brasília (UnB).

RESUMO

Com os avanços das tecnologias associadas à indústria microeletrônica, sistemas completos de aquisição de dados, processamento e tomada de decisões já podem ser encontrados em dispositivos eletrônicos compactos. O advento de novas tendências, como a Internet das Coisas, Indústria 4.0 e Sistemas Cyber-Físicos, tem demandado aplicações cada vez mais complexas associadas a tais dispositivos eletrônicos, também classificados como Sistemas Embarcados. De forma geral, um sistema embarcado possui restrições de tamanho, custo e consumo de energia não encontradas em outros tipos de sistemas computacionais. O projeto de sistemas embarcados modernos deve ser extremamente otimizado para contemplar não apenas as restrições gerais, mas também atender aos requisitos das aplicações. A complexidade do projeto envolve a seleção de sensores e atuadores, interfaces de entrada e saída de dados, software embarcado, sistema operacional e hardware, sendo necessário o envolvimento de equipes multidisciplinares. O ciclo de desenvolvimento e testes de todos os componentes do sistema demanda diversas iterações, visto que modificações em uma parte implicam alterações em outras. Cada iteração possui um custo de desenvolvimento e prototipação. Neste trabalho, discutimos a utilização de prototipação virtual como alternativa à elaboração de protótipos físicos, mostrando um caso de uso aplicado ao desenvolvimento de um robô móvel autônomo.

Palavras chave: desenvolvimento de produto; sistemas embarcados; prototipação virtual

1. INTRODUÇÃO

A tecnologia microeletrônica tem avançado rapidamente nas últimas décadas. A famosa Lei de Moore, previa um crescimento exponencial da capacidade de integração de circuitos integrados (número de transistores por mm^2, potencializando a quantidade e complexidade das aplicações suportadas. A Figura 1 mostra a evolução histórica de diversos parâmetros relacionados à indústria microeletrônica.

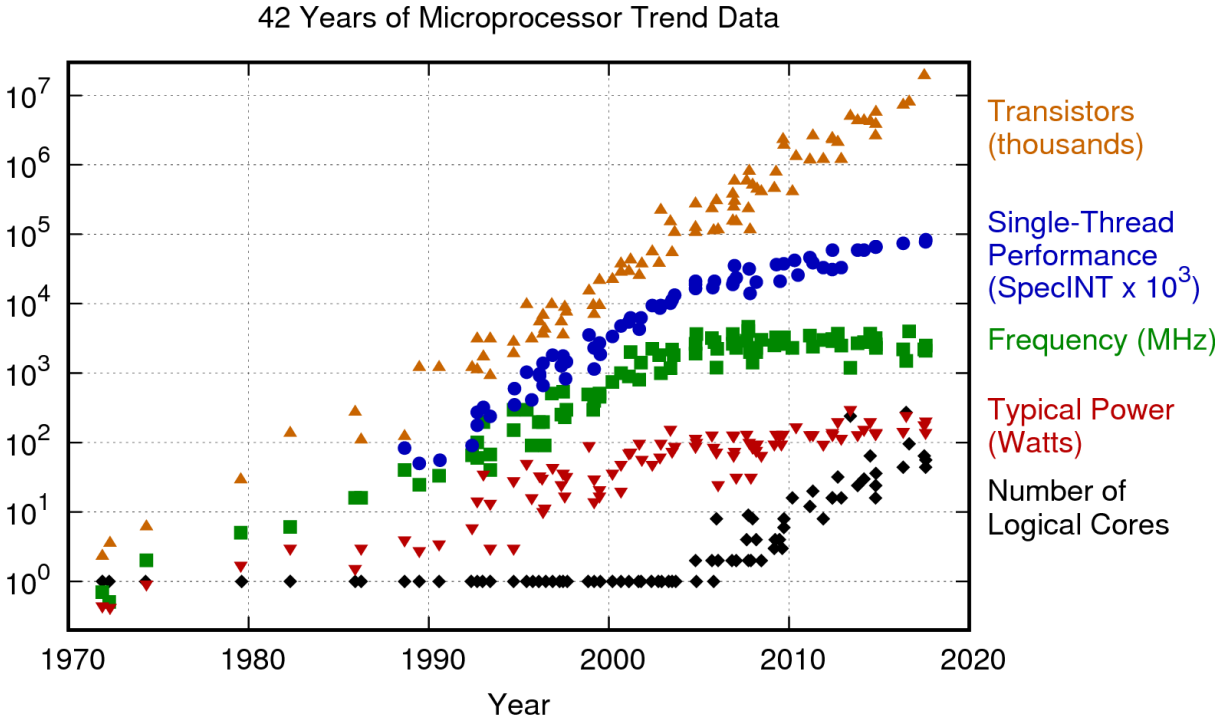

FIGURA 1 – Evolução de parâmetros relacionados à indústria microeletrônica (Rupp, K. 2018).

Observa-se na Figura 1 que a frequência de operação dos processadores (diretamente relacionada à sua velocidade de processamento) vem se estabilizando entre 10^3 e 10^4 MHz. Essa limitação de frequência deve-se a fenômenos físicos diversos ainda não solucionados pela indústria. A limitação da frequência de processamento não evitou o crescimento da demanda por velocidade, o que levou a indústria à rápida adoção de sistemas multi-processados. Na Figura 1 pode-se observar que o número de núcleos de processamento (*logical cores*) em um único circuito integrado já alcança as centenas.

O fortalecimento comercial de conceitos como Computação Ubíqua (em todos os lugares) e Computação Pervasiva (transparente ao usuário) tem sido visto em áreas diversas como a Realidade Aumentada, Internet das Coisas e Indústria 4.0. Sistemas Embarcados, nome genérico dado a sistemas de processamento de dados de pequeno/médio porte que funcionam

primordialmente sem um operador humano, têm sido a base para a implantação da nova sociedade digital, em que pessoas e objetos (“coisas”) estão cada vez mais interconectados. Devido ao grande avanço das tecnologias envolvidas, a quantidade de aplicações potenciais tem crescido de forma muito rápida, de modo que o ciclo de projeto de produtos tem de ser cada vez mais rápido e eficiente. Sistemas embarcados possuem requisitos de desempenho geralmente não presentes em sistemas computacionais comuns. Velocidade, segurança, tolerância a falhas e consumo de energia, entre outros, são alguns dos requisitos necessários nas aplicações que utilizam sistemas embarcados. Existem ainda os requisitos específicos de cada aplicação e a não-otimização dos sistemas computacionais envolvidos pode levar a aplicação à falha.

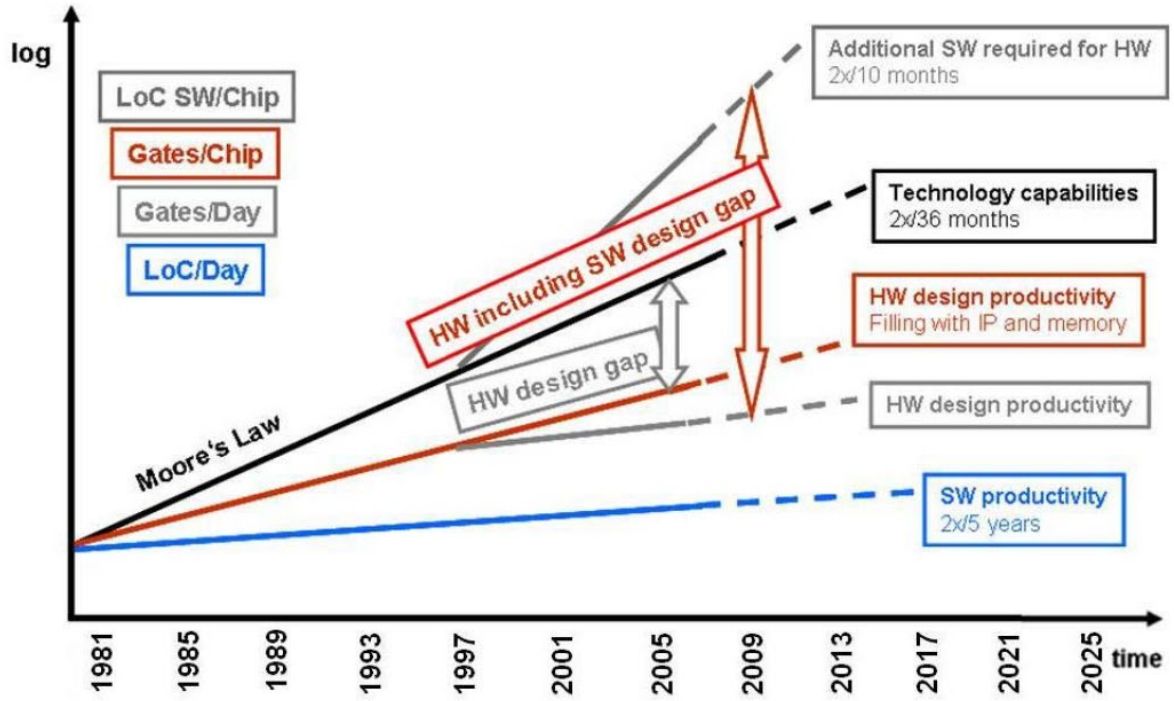

FIGURA 2 – Produtividade de desenvolvimento de HW/SW ao longo dos anos (ITRS 2009).

A Figura 2 mostra um fenômeno conhecido por *Design Productivity Gap* no ecossistema de desenvolvimento de sistemas de hardware e software. O gráfico mostra que, seguindo a Lei de Moore, a tecnologia microeletrônica dobra sua capacidade a cada 36 meses. Isso mostra que o avanço da capacidade de integração em tecnologias VLSI possibilitaria aplicações cada vez mais complexas em uma velocidade muito alta. Porém, a produtividade do desenvolvimento de software dobra apenas a cada 5 anos, de modo semelhante à produtividade do desenvolvimento de hardware. O projeto de sistemas embarcados necessita de grande otimização de desempenho,

o que é alcançado por meio de técnicas de projeto integrado hardware/software, cuja demanda dobra a cada 10 meses, de forma ainda mais rápida que o avanço da tecnologia microeletrônica. Isso implica em sérias limitações no tempo de desenvolvimento de sistemas embarcados, de modo que novas metodologias e ferramentas devem ser criadas/implementadas de modo a acelerar o processo de desenvolvimento.

O desenvolvimento de produtos mecatrônicos requer conhecimentos tanto genéricos quanto específicos em áreas distintas como Tecnologia da Informação, Mecânica e Eletrônica. O Diagrama-V mostra o ciclo de desenvolvimento de um sistema mecatrônico (Casner, et.al, 2017).

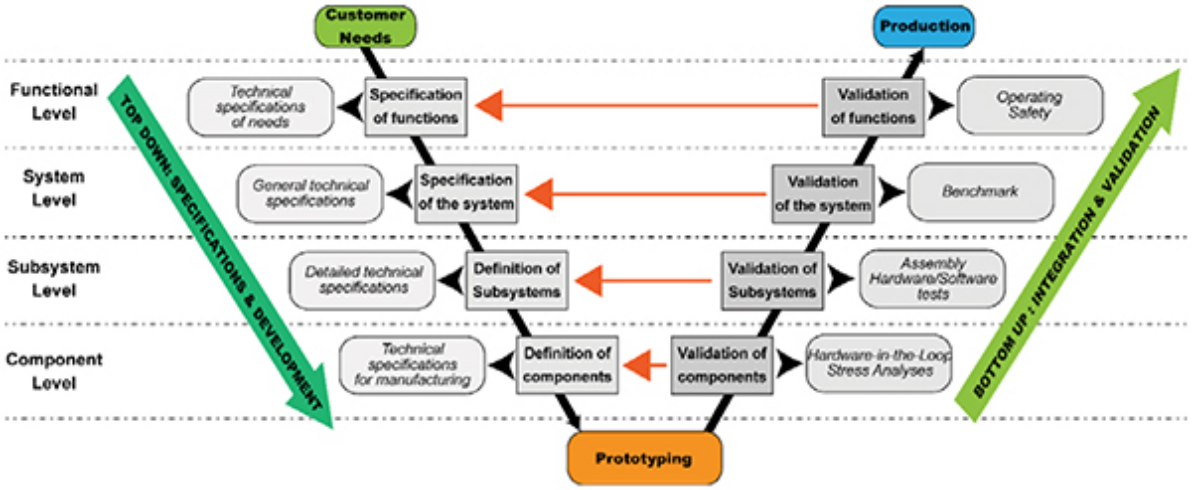

FIGURA 3 – Diagrama-V: ciclo de desenvolvimento de um sistema mecatrônico (Casner, et.al, 2017).

A iteração pelos passos de análise e desenvolvimento mostrados no Diagrama-V são geralmente demorados e requerem diversos profissionais de diferentes equipes. A tomada de decisões de projeto deve ser feita de forma cuidadosa e, na maioria das vezes, só é possível o teste de um número ínfimo de soluções em cada etapa. A prototipação física de um sistema mecatrônico pode ser dispendiosa e levar um período de tempo muito grande, impossibilitando a conclusão do projeto em prazos razoáveis.

Neste trabalho propomos a utilização de protótipos virtuais para acelerar o ciclo de desenvolvimento de sistemas multi-tecnótipos, com foco maior na parte de sistemas embarcados. Nos próximos tópicos mostraremos as vantagens e desvantagens da prototipação virtual e mostraremos um estudo de caso em uma aplicação de robótica móvel.

2. PROTOTIPAÇÃO VIRTUAL

Simulações computacionais já são amplamente utilizadas para prever a evolução de fenômenos físicos. Aspectos como tempo de simulação, recursos computacionais, precisão numéricas, entre outros, possuem grande influência nos resultados das simulações. Com os avanços da Tecnologia da Informação, simuladores cada vez mais potentes têm surgido, possibilitando um grau de precisão bastante elevado e minimizando a necessidade de construção de protótipos de produtos. Porém, cada fenômeno a ser simulado possui especificidades próprias e cada simulador é desenvolvido e otimizado com foco no fenômeno desejado.

A simulação de sistemas multi-tecnótipos envolve o desafio de integrarem-se simuladores distintos, respeitando as interações entre os diferentes fenômenos e o aspecto temporal que deve ser unificado para a simulação ter validade.

2.1 Ferramenta de simulação de arquiteturas de processamento

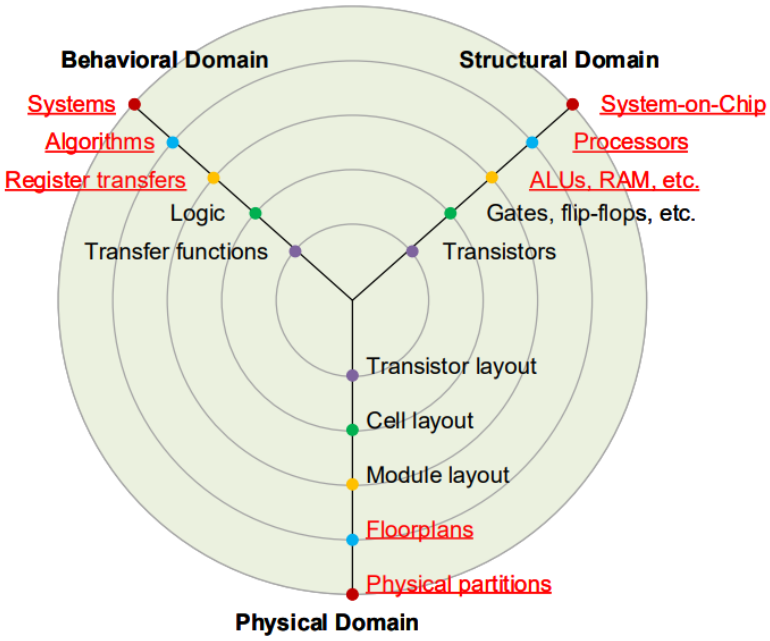

FIGURA 4 – Diagrama de Gajski-Kuhn mostrando os níveis de um projeto de sistema de processamento (J.Y.M.A. SILVA, 2018).

O Diagrama de Gajski-Kuhn, também conhecido como Diagrama-Y, mostra os níveis de abstração de um projeto de sistema de processamento (Figura 4). O projeto é geralmente dividido em três domínios: Comportamental, Estrutural e Físico. Existem simuladores

específicos para cada domínio, indo do nível algorítmico ao nível de interações eletrônicas entre transistores no circuito integrado.

Neste trabalho utilizamos uma biblioteca chamada SystemC, capaz de realizar simulações de sistemas em múltiplos níveis do Diagrama-Y. O SystemC é uma biblioteca escrita na linguagem C++ e que especifica diferentes formas de modelagem/descrição de arquiteturas de processamento (IEEE, 2011). Distintos modos de temporização também podem ser utilizados, permitindo simulações puramente comportamentais (sem temporização), simulações de tempo aproximado (estimativas de passagem temporal) e com precisão de ciclo (considerando um relógio). As abstrações temporais permitem aproximações realistas e de precisão variável quanto à velocidade de processamento de diferentes arquiteturas de processamento, permitindo uma exploração simplificada do espaço de projeto.

2.2 Ferramenta de Simulação de Sistemas Robóticos

O *Virtual Robot Experimentation Platform* (Plataforma Virtual de Experimentação de Robôs – V-REP), simulador de sistemas robóticos utilizado neste trabalho, é uma ferramenta para desenvolvimento de algoritmos, simulações de chão de fábrica, verificação e prototipação rápida de sistemas robóticos (COPELLIA ROBOTICS, c2019). O V-REP foi escolhido por ser um software portável (com versões para Mac, Windows e Linux), ter o código aberto, ser gratuito para aplicações educacionais e permitir a integração com aplicações externas por meio de uma API (do inglês *Interface de Programação de Aplicações*).

A simulação é feita com objetos, que se relacionam por meio de hierarquias, em uma cena. Esses objetos são classificados como juntas, formas, sensores, renderização, força, dentre outros, que juntos formam o robô e todo o ambiente de simulação (FREESE *et al.,* 2010). A Figura 5 mostra o cenário utilizado para a validação da ferramenta de prototipação desenvolvida neste trabalho, usando como exemplo o robô Pioneer P3-DX. A hierarquia da cena pode ser vista à esquerda da imagem.

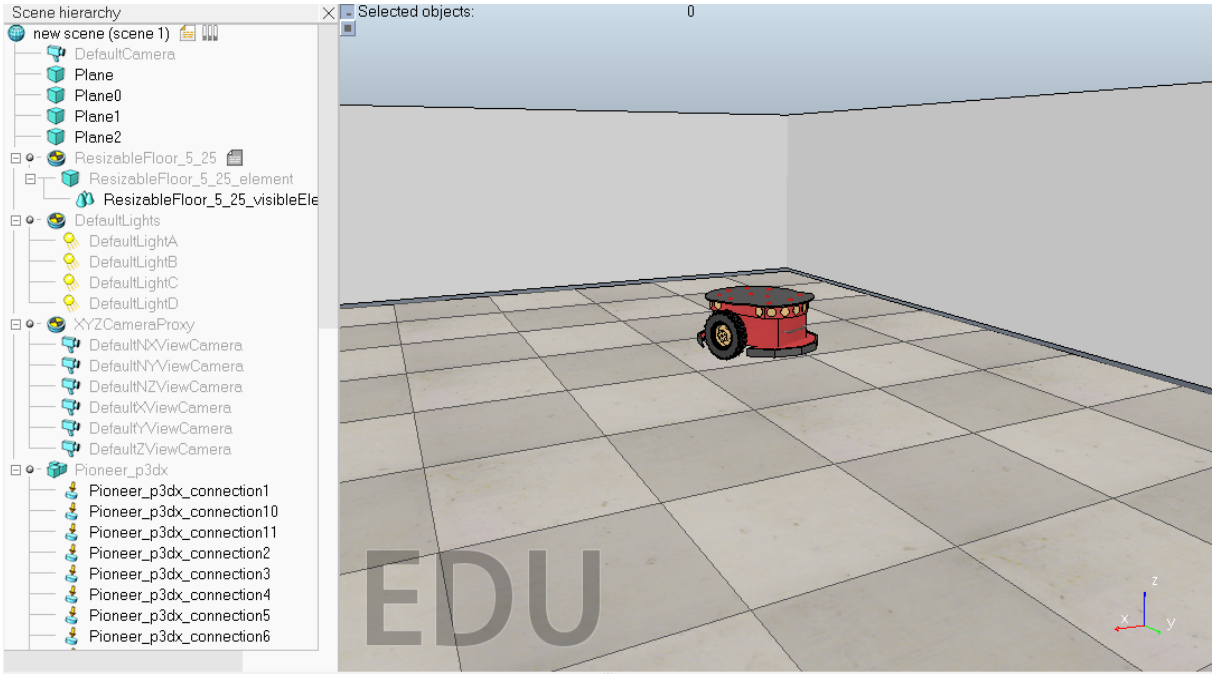

FIGURA 5 - Captura de tela da cena de um robô Pioneer P3-DX no simulador V-REP.

Este trabalho utiliza a API remota do V-REP para comunicação e implementação de um algoritmo de validação, permitindo que aplicações externas se conectem ao programa e interajam com ele por meio de funções nas linguagens de programação suportadas. A API foi escolhida por ser mais leve e mais simples do que a outra opção, chamada API remota B0 (COPELLIA ROBOTICS, [s.d.]). Dentre as linguagens de programação suportadas, escolheu-se o C++ por permitir integração com a biblioteca SystemC, que será discutida no próximo tópico deste texto.

3. ESTUDO DE CASO

Neste trabalho, propomos uma plataforma de prototipação virtual para robótica móvel, unindo um simulador de robótica comercial com uma biblioteca de simulação de arquiteturas de processamento embarcado.

O V-REP utiliza o modelo cliente/servidor para interagir com aplicações externas. O VPRES, framework desenvolvida neste trabalho, funciona como cliente que se conecta ao servidor, inserido internamente no V-REP, por meio de IPs e portas. O avanço temporal da simulação [e feito de acordo com a Figura 2, com passos de simulação constantes entre 100 µs e 10s.

É possível operar o V-REP de duas maneiras principais com a API remota: no modo assíncrono, em que o servidor avança livremente sem levar em conta a aplicação remota; e no modo síncrono, em que a API envia um comando para o servidor, solicitando o avanço da simulação. Este trabalho utiliza o modo síncrono para acoplar o avanço temporal entre V-REP e SystemC, fazendo com que o passo de simulação (Figura 6) avance o tempo igualmente nas duas plataformas.

Dois tipos de objetos foram utilizados no controle do robô Pioneer P3-DX: os sensores ultrassônicos (de proximidade) e motores (juntas) com 16 e 2 elementos, respectivamente. Os pontos amarelos no robô da Figura 5 são sensores de proximidade e os motores estão acoplados a cada uma das duas rodas. A API remota fornece funções tanto para **ler distâncias** entre os sensores e objetos próximos, quanto para configurar separadamente a **velocidade** dos motores. Existem três modos de operação para ambas as funções: bloqueante, não-bloqueante e fluxo de dados. O primeiro bloqueia a aplicação externa, esperando uma resposta do servidor (V-REP), e é usado quando se necessita de informação imediata do servidor para prosseguir a tarefa. O modo não-bloqueante permite que os dados sejam enviados para o V-REP sem esperar resposta imediata. No modo de fluxo de dados, o servidor antecipa que tipos de dados deverão ser enviados ao cliente, mandando a resposta em uma base de tempo regular e sem a necessidade da requisição por parte do cliente.

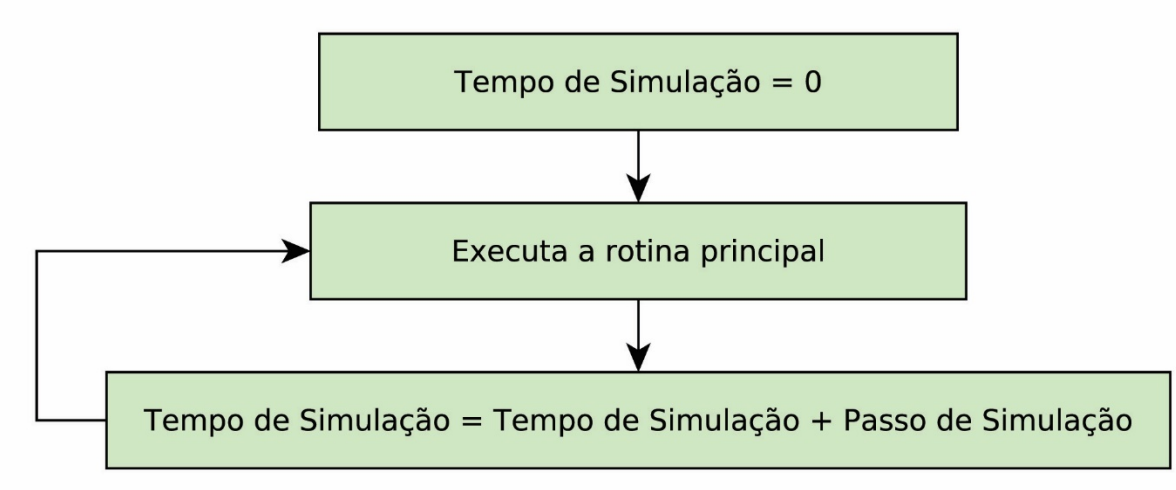

FIGURA 6 - Diagrama do laço temporal do V-REP.

Com o intuito de agilizar a comunicação do cliente e reduzir consideravelmente o tempo de simulação, utilizou-se o modo de fluxo de dados para receber dados sem atraso dos sensores

ultrassônicos, e o modo não-bloqueante para configurar a velocidade dos motores. A escolha foi feita pela necessidade de realimentação rápida por parte dos sensores (precisa-se da informação de distância que eles fornecem) e por não ser necessária a resposta dos motores, que só recebem a velocidade de rotação. A Figura 7 ilustra a comunicação entre V-REP e VPRES para um motor e um sensor ultrassônico do robô utilizado.

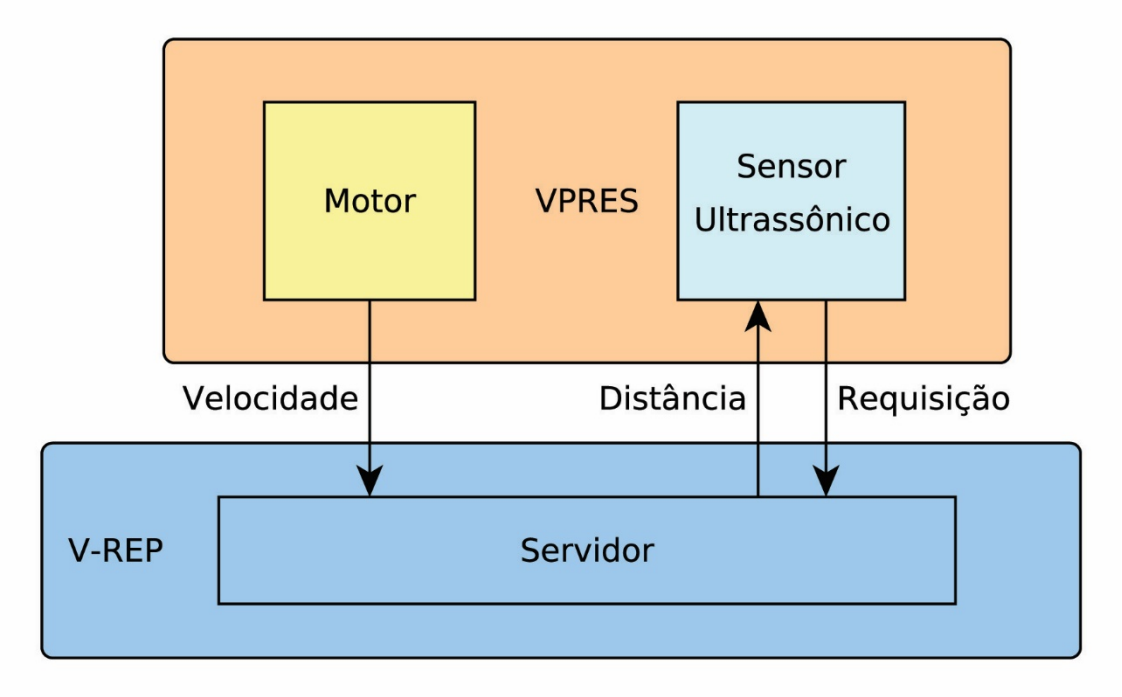

FIGURA 7 - Fluxo de dados entre VPRES e V-REP para um motor e sensor ultrassônico.

3.3 Arquitetura do VPRES

O VPRES foi pensado para permitir que aplicações particulares em SystemC pudessem ser inseridas da maneira mais direta possível na aplicação. Assim sendo, foi necessário montar um sistema que permitisse a sincronização independente de passo entre o V-REP e o SystemC, ao mesmo tempo em que lidasse com a comunicação dos objetos do V-REP à aplicação. Utilizou-se C++ para o desenvolvimento da aplicação, aproveitando a compatibilidade da API remota do V-REP e do SystemC, além da orientação a objetos da linguagem.

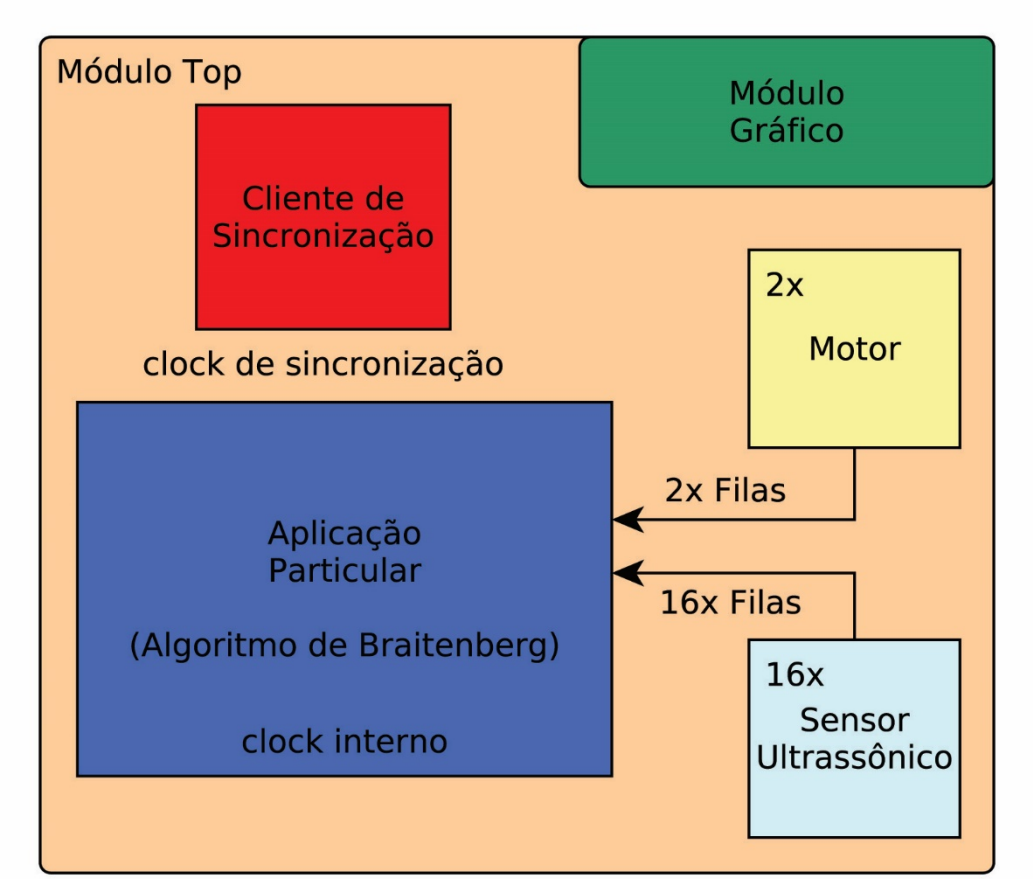

FIGURA 8 - Arquitetura do VPRES.

A Figura 8 mostra a arquitetura do VPRES, onde os quadrados representam módulos do SystemC. O cliente de sincronização, em vermelho, lida com a sincronização do tempo entre V-REP e SystemC por meio do relógio inserido dentro do módulo. Ressalta-se que ele não se liga a nenhum outro módulo do sistema, ou seja, não necessita de outros parâmetros para funcionar. O passo de simulação é dado pelo V-REP, o SystemC usa o mesmo intervalo de tempo, e deve ser configurado antes do início da simulação.

Os dois motores e dezesseis sensores ultrassônicos se comunicam com o V-REP de acordo com a Figura 7. Porém, suas ligações com o módulo de aplicação particular (caixa azul na Figura 8) são feitas com filas do tipo *first come, first served* (“primeiro a entrar, primeiro a sair”) de tamanho unitário. Esse tipo de fila retira o primeiro elemento que tiver sido inserido nela, após alguma requisição por parte da aplicação, e é travada para novas escritas até que isso aconteça. Neste contexto, a aplicação particular representa uma forma de validação do VPRES. Ela possui um relógio interno para simular um processador, que é independente do relógio do cliente de

sincronização. O algoritmo de Braitenberg foi escolhido para avaliar o desempenho do sistema e será discutido com mais detalhes na próxima Seção do texto. É importante destacar que o intuito da aplicação particular é depender somente das filas e ter a sintaxe do SystemC, para que seja trocada posteriormente por modelos de processadores mais sofisticados (ARM, MIPS, RISC-V, etc).

O módulo *Top* da aplicação em SystemC (Figura 8) instancia todos os módulos especificados anteriormente e implementa o módulo gráfico para acompanhamento em tempo real da trajetória do robô. Utilizou-se a biblioteca *QcustomPlot* que faz parte do *framework* Qt5 para o desenvolvimento de interfaces gráficas para C++. Além disso, os pontos da trajetória também são salvos em uma planilha para utilização posterior caso necessário.

4. RESULTADOS E DISCUSSÃO

O Algoritmo de Braitenberg foi derivado da publicação de (BRAITENBERG, 1986) e se encontra implementado na linguagem Lua para demonstração do robô Pioneer P3-DX no V-REP. Ele realiza operações aritméticas simples para se desviar de obstáculos colocados no cenário do simulador. O trabalho de (BERRI; GRASSI; OSORIO, 2015) adaptou o algoritmo para a linguagem C++ mas não utilizou orientação a objetos na implementação. Então, implementou-se o algoritmo neste trabalho levando em conta o paradigma de orientação a objetos C++ para melhor ajuste no SystemC.

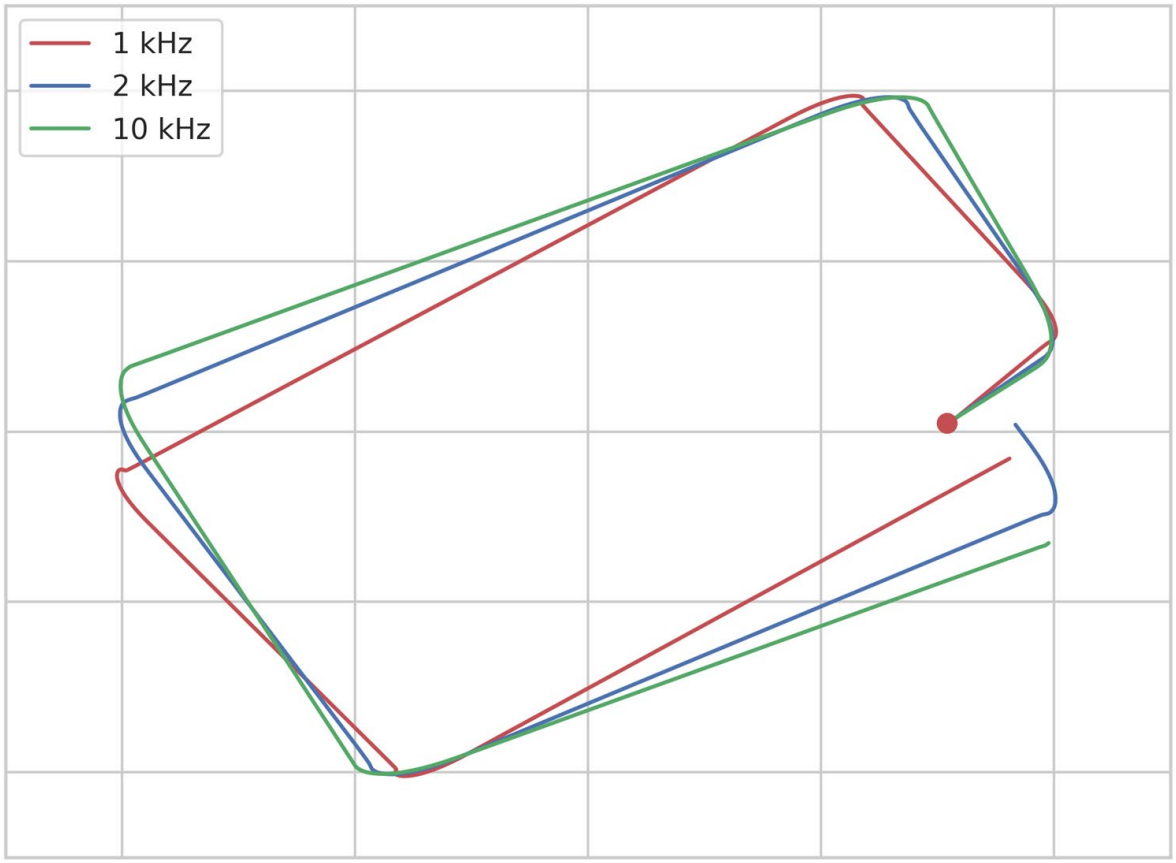

FIGURA 9 - Trajetória do robô Pioneer P3-DX para diferentes velocidades de processamento do Algoritmo de Braitenberg rodando sobre o processador simulado em SystemC.

Avaliou-se o desempenho do Algoritmo de Braitenberg para diferentes velocidades de processamento do processador simulado em SystemC (Figura 9). O robô tinha o espaço de 2,5m x 2,5m para se locomover, com paredes nas extremidades do quadrado e sem enhum objeto em seu caminho. Aplicou-se um modelo de temporização aproximada na aplicação, adicionando atrasos de tempo proporcionais às operações executadas no algoritmo. Por exemplo, após a atribuição de uma variável, adicionou-se um ciclo de atraso para simular a escrita rápida em memória.

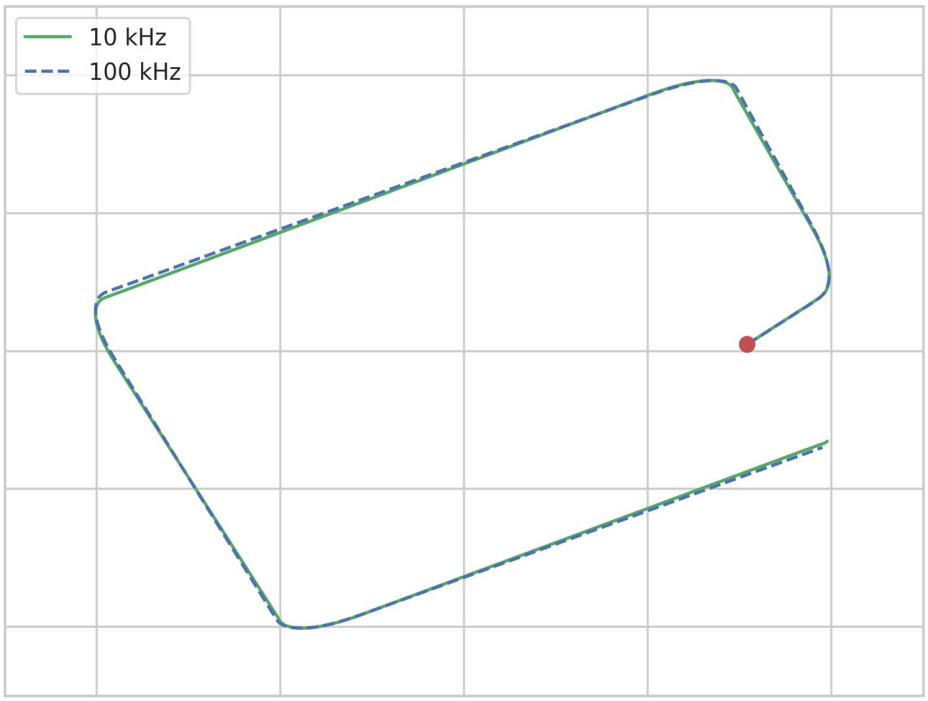

FIGURA 10 - Desempenho da aplicação para velocidades de processamento mais altas.

Considerou-se também que os sensores ultrassônicos foram lidos consecutivamente, como se houvesse um multiplexador para selecionar o dado de cada um deles separadamente. As velocidades dos motores foram configuradas no mesmo tempo de simulação, considerando que a aplicação tinha portas o suficiente para acionar os dois motores simultaneamente.

O intuito é avaliar a trajetória do robô com diferentes frequências e verificar se a arquitetura montada conseguiu levar em conta as limitações de processamento da aplicação. Se isso for verificado, supõe-se que o VPRES também será capaz de simular a temporização de diferentes arquiteturas de processadores ou qualquer módulo em SystemC que seja compatível com as filas de comunicação.

A Figura 9 mostra a trajetória do robô para diferentes velocidades do processador simulado em SystemC. Percebe-se que o modelo levou em consideração tais velocidades de processamento, ao observar-se as trajetórias resultantes. Verifica-se que a tarefa exige uma velocidade de processamento correspondente a 10KHz de frequência no processador para adquirir os dados e funcionar de forma ótima. Para frequências menores, percebem-se alterações significativas da rota, podendo causar um comportamento indesejado na aplicação.

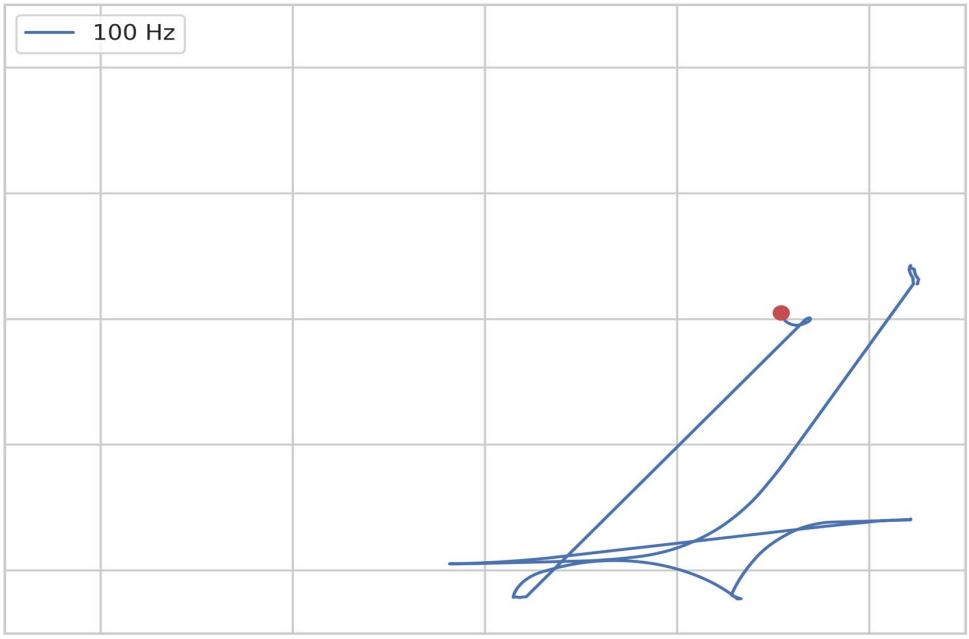

FIGURA 11 - Trajetória para velocidade de processamento muito baixa.

A Figura B reforça o que foi dito no parágrafo anterior. A trajetória não sofreu alteração considerável para valores dez vezes acima da frequência de 10 KHz, levando a considerar que essa seja a frequência próxima da ideal para o funcionamento da aplicação. Dessa maneira, não há perdas de desempenho por um processador mais lento e nem o uso ineficiente de equipamentos mais rápidos.

O caso de velocidades de processamento com frequências abaixo da amostragem mínima necessária está na Figura 11. Houve diferença na trajetória desde o início da simulação, resultando em um comportamento indefinido. Portanto, a aplicação não pode ter frequências de apenas 100 Hz para implementação do Algoritmo de Braitenberg no modelo de temporização utilizado.

5. CONCLUSÃO

Neste trabalho mostrou-se a criação e implementação de uma ferramenta, VPRES, capaz de realizar a integração entre dois simuladores de sistemas distintos (V-REP e SystemC). A utilização do VPRES torna possível que um projetista teste diferentes configurações de robô com distintas configurações de arquiteturas de processamento em um tempo razoavelmente curto.

O estudo de caso realizado possibilitou mostrar como diferentes velocidades de processamento em um mesmo sistema robótico leva a diferentes resultados. As simulações realizadas tiveram êxito em mostrar tanto a otimização do tempo de processamento (com a identificação da frequência ótima de operação da arquitetura de processamento) quanto a falha do sistema robótico, oriunda de uma baixa velocidade de processamento.

Com isso, indica-se a viabilidade de construção de uma metodologia para o desenvolvimento de sistemas multi-tecnótipos a partir da integração de simuladores de domínios físicos diferentes. Para a evolução deste trabalho, pretende-se comparar os sistemas virtualmente prototipados com protótipos físicos, de modo a estimar-se a diminuição no tempo de projeto quando protótipos virtuais são incluídos no processo de desenvolvimento do produto.

6. AGRADECIMENTOS

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

7. REFERÊNCIAS

RUPP, K. 42 Years of Microprocessor Trend Data. **https://www.karlrupp.net** acessado em 04/06/2019.

ITRS 2009. International Technology Roadmap for Semiconductors. ITRS. 2009.

CASNER, D., HOUSSIN, R., RENAUD, J., KNITTEL, D. “An Optimization-Based Embodiment Design Approach for Mechatronic Product Development”, 2017. In The Open Automation and Control Systems Journal, 2017, 9, (Suppl-1,M4) 27-47.

J.Y.M.A. SILVA, 2018.; Development of a design framework for Parallel Data Processing hardware architectures. Tese de doutorado em Engenharia Elétrica e Tecnologia da Informação. Ruhr-University Bochum, Alemanha. 2018.

IEEE, 2011. IEEE Standard for SystemC Language Reference Manual. IEEE Computer Society, 2011.

BERRI, R.A.; GRASSI Jr.,V.; OSORIO, F. S. "Simulação de Robôs Móveis e Articulados: Aplicações e Prática", Recife, Julho de 2015. Capítulo 6. 34a. JAI - Jornada de Atualização de Informatica, XXXV Congresso da SBC, pp.274-322. Disponível em: <https://www.sites.google.com/site/vrepjai/>. Acesso em: 11 de Junho de 2019.

BRAITENBERG, V. “Vehicles: Experiments in Synthetic Psychology”, 1989. MIT Press.

COPELLIA ROBOTICS, G. “V-REP – Virtual Robot Experimentation Platform”, c2019. Disponível em: <http://www.coppeliarobotics.com/index.html>. Acesso em: 11 de Junho de 2019.

COPELLIA ROBOTICS, G. “Virtual Robot Experimentation Platform – User Manual”, [s.d.]. Disponível em: <http://www.coppeliarobotics.com/helpFiles/index.html>. Acesso em: 11 de Junho de 2019.

FREESE, M., SINGH, S., OZAKI, F., e MATSUSHIRA, N. “Virtual robot experimentation platform V-REP: A versatile 3D robot simulator”, 2010. In Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp. 51–62.