Workflow para Desenvolvimento de Aplicações Robóticas com Requisitos de Tempo Real

Matheus Leitzke Pinto[0000−0003−3571−222X] **e** André Schneider de Oliveira[0000−0002−8295−366X]

Universidade Tecnológica Federal do Paraná, Curitiba, Brasil
matheus.pinto@ifsc.edu.br, andreoliveira@utfpr.edu.br

Resumo: O uso de sistemas de tempo real (STR) na robótica se torna necessário, à medida que robôs possuem mais restrições de tempo impostas pela necessidade de aplicações que forneçam segurança. Devido à carência de materiais e métodos que auxiliem no desenvolvimento de aplicações robóticas com restrições temporais, no presente trabalho é desenvolvido e apresentado um *workflow* completo desde a determinação da plataforma computacional utilizada como STR, até a implementação da aplicação em um robô operacional. Na avaliação de sistemas computacionais, foi desenvolvido um *benchmark* denominado Hart-ROS, como a ferramenta de análise do desempenho de tempo real. Após validação do STR, uma aplicação de robô móvel com restrições de tempo foi desenvolvida, cujos materiais utilizados são de fácil disponibilidade no mercado brasileiro e internacional. Além do exposto, as metodologias empregadas são baseadas na literatura de tempo real e apresentadas de forma aprofundada. Até onde foi conduzido esse estudo, nenhum trabalho apresentou de forma sistemática por meio de etapas, o desenvolvimento de um robô que atenda aos requisitos temporais de uma aplicação, de forma que possa ser replicável por grande parte da comunidade. Pode-se verificar que o *workflow* proposto é uma solução aberta e abrangente no desenvolvimento de aplicações robóticas, pois foi possível criar um sistema que atende aos requisitos temporais de projeto, com base científica como aporte.

Palavras chave: *Workflow* · Robótica · Sistemas de Tempo Real · Desempenho de Tempo Real · Hart-ROS.

1 Introdução

Um robô que interage com o meio deve, em muitos casos, responder aos estímulos dentro de um intervalo de tempo, caso contrário, sua ação pode ser prejudicada, inutilizada ou causar danos físicos. Isso faz com que, em grande parte dos casos, um robô deva operar sob o controle de um **sistema de tempo real** (STR). Um STR pode ser visto como um sistema computacional que fornece serviços que atendam aos **requisitos temporais** da aplicação, em especial de uma aplicação robótica. Isso ocorre devido à **previsibilidade temporal** que se pode garantir *a priori*, em tempo de projeto, sobre seus serviços.

Considera-se que existe uma carência de ferramentas e pesquisas bem estabelecidas no desenvolvimento de robôs, onde são necessárias restrições de tempo. Por outro lado, também considera-se que é possível desenvolver robôs com requisitos de tempo real, utilizando ferramentas disponíveis no mercado, por meio de um ***workflow*** bem definido. Segundo [13], um *workflow* pode ser descrito como sendo uma sequência de atividades ordenadas no tempo, onde cada atividade realiza alguma parte do trabalho, com a finalidade de atingir um determinado objetivo. Relacionado ao trabalho atual, esse objetivo é a criação de um robô que possa atender aos requisitos temporais de uma aplicação específica. Além disso, os métodos e ferramentas utilizados em cada atividade devem ser bem definidos de forma a alcançar esse objetivo. Posto isto, o *workflow* proposto para o **desenvolvimento de aplicações robóticas com requisitos de tempo real** pode ser definido da seguinte maneira.

1. Avaliação de uma possível plataforma para ser utilizada como o sistema computacional do robô. A plataforma deverá fornecer os serviços de tempo real, assim como a previsibilidade temporal nas operações de *software*. Fatores como recursos, custo, complexidade e principalmente **desempenho de tempo real**, devem ser considerados na escolha da plataforma. No desempenho de tempo real será obtida a qualidade do sistema em atender aos requisitos temporais da aplicação à qual se deseja.
2. A implementação das **tarefas de tempo real**, unidades de *software* executando concorrentemente funções específicas do robô. Estas são geralmente implementadas sobre um **modelo de tarefas** imposto pelo uso do **escalonador de tempo real** utilizado, sendo este o responsável pela gerência das tarefas no sistema.
3. A determinação dos tempos de execução das tarefas, sendo necessárias técnicas para a obtenção segura do tempo de execução mais demorado – denominado ***Worst-Case Execution Time*** (WCET).
4. Após determinação dos WCET de cada tarefa, é possível aplicar um **teste de escalonamento** no conjunto de tarefas, um teste analítico, de forma a verificar se os requisitos temporais do sistema poderão ser atendidos ao longo da operação.
5. Por fim, é necessária a junção de todas as tarefas para a execução concorrente, de forma a verificar se as restrições temporais do sistema serão atendidas durante a operação do robô. Em um sistema crítico, algum mecanismo de **tolerância a faltas** deve ser integrado para manter a operação correta do robô, mesmo quando violações de tempo ocorrerem.

Dessa forma, a contribuição do trabalho corrente está na implementação e apresentação de ferramentas e metodologias, de forma que seja possível o desenvolvimento de aplicações robóticas com requisitos de tempo real com base no *workflow* apresentado. Não se objetiva o desenvolvimento de sistemas críticos em um primeiro momento, e dessa forma, mecanismos de tolerância a faltas não serão abordados.

Salienta-se que este trabalho gerou duas publicações em dois congressos internacionais de robótica [8, 7]. O artigo [8] foi considerado entre os dez melhores

do congresso, sendo convidado para publicação no *Journal of Intelligent and Robotic Systems*.

O presente trabalho está dividido nas seguintes seções: a seção 1 é a seção corrente, com a apresentação do trabalho; na seção 2 é apresentado o *benchmark* desenvolvido para a avaliação do desempenho de tempo real da plataforma computacional no estágio inicial do *workflow* proposto. Utilizou-se duas plataformas de sistemas embarcados como estudo de caso. No final deste, são apresentados os resultados e discussões obtidos com o *benchmark* acerca do desempenho de tempo real das plataformas embarcadas; as etapas do *workflow* pós avaliação do desempenho de tempo real, são apresentadas na seção 3, utilizando como estudo de caso um robô móvel seguidor de linha dotado de interfaces com o meio. Relacionadas a estas interfaces, foram desenvolvidas tarefas de tempo real sobre uma plataforma embarcada. Também, todo o resultado acerca do desenvolvimento do robô móvel é avaliado nessa seção, tais como medições dos tempos de execução das tarefas, teste de escalonabilidade e avaliação da operação do robô no ambiente; por fim, na seção 4 são apresentadas as conclusões e discussões finais relacionadas às duas seções de desenvolvimento.

2 O *Benchmark* Hart-ROS

O Hart-ROS é um *benchmark* **orientado a aplicação**, isso significa que o **desempenho de tempo real** é avaliado por meio de **aplicações sintéticas** em execução no sistema alvo, o qual se quer avaliar. Seu funcionamento foi apresentado de forma breve em [8].

Uma aplicação sintética não realiza nenhuma atividade prática em um robô operacional. Por outro lado, aplica o mesmo estresse no sistema que uma aplicação rodando em um cenário real semelhante, de forma que seja possível capturar informações deste. As aplicações sintéticas são compostas de tarefas de tempo real. Em determinadas aplicações é executado um **código sintético**, sendo utilizado o Kilo-Whetstone [2, 17]. Além disso, em grande parte dos casos, as unidades de execução da aplicação robótica se comunicam por meio de mensagens. Para tanto, o *benchmark* fornece uma interface de comunicação entre as tarefas sintéticas. Tal interface irá utilizar os serviços do sistema de comunicação da plataforma avaliada. As mensagens das aplicações do *benchmark* também serão sintéticas, ou seja, serão compostas por um conjunto de *bytes* sem significado, mas que irão se tornar um carga para o sistema de comunicação, de forma que o mesmo seja avaliado.

Dessa forma, cada experimento no *benchmark* está relacionado com cenários que representam aplicações práticas. Cada cenário varia de acordo com o número de tarefas, arranjo destas, suas frequências, se há comunicação por meio de mensagens, entre outros. É executado um experimento por vez, cada um começando com um conjunto de **tarefas base** que irá rodar por um período pré-determinado, chamado de **passo do experimento**. Se todos os *deadlines* forem atendidos, alguns parâmetros do conjunto serão aumentados e um novo passo será executado. Esses parâmetros dependem do experimento especifico. O experimento finaliza

quando é atingido o **ponto de ruptura** (*breakdown point*), decorrente de: *deadlines* não atendidos no passo corrente ou; limitação do sistema no aumento dos parâmetros do próximo passo (ex.: falta de memória).

Ao final de cada passo, o *benchmark* fornece os parâmetros das tarefas e os resultados referentes aos tempos de resposta e *deadlines*. Com os resultados em mãos, é possível avaliar as limitações do sistema para aplicações de tempo real, quando um determinado número de tarefas, *workload* e frequência são atingidos. A partir disso pode-ser determinar se o sistema atinge as necessidades de um projeto.

2.1 Definições dos Experimentos

Os sistemas de controle robóticos avaliados são vistos como **nós** pelo *benchmark*, os quais podem ser processos do sistema operacional (SO), ou máquinas individuais. Um nó executa um conjunto de tarefas de tempo real cujas restrições de tempo devem ser atendidas para que o sistema atinja seus requisitos temporais. As tarefas são escalonadas por meio da política ***Rate-Monotonic*** (RM) [6]. Isso se deve ao fato de que muitos SOs comerciais tem nativamente um escalonador preemptivo de prioridades fixas, o que torna o RM fácil de ser implementado [5]. Quanto às tarefas que rodam os experimentos, estas serão denominadas **tarefas sintéticas**, pois irão executar códigos sintéticos, ou se comunicar por meio de mensagens sintéticas.

O *benchmark* fornece uma abstração de comunicação entre as tarefas sintéticas com o paradigma ***Publisher/Subscriber*** (PS) [14, 9]. Nesse mecanismo, mensagens são enviadas enviadas/lidas para/de **tópicos**. Tarefas que enviam mensagens para um tópico são denominadas ***publishers***, enquanto tarefas que recebem mensagens por meio de tópicos são denominadas ***subscribers***.

Foram desenvolvidos seis experimentos para execução pelo *benchmark*, representando um cenário de aplicação cada. Os três primeiros experimentos relacionados ao **domínio do processamento** são basicamente casos gerais que ocorrem em qualquer STR. Esses experimentos são inicializados com 5 tarefas base $\{\tau_1, \tau_2, \tau_3, \tau_4, \tau_5\}$, cujos parâmetros foram definidos utilizando como base [17]. O critério de aumento de cada passo do experimento foi definido conforme [17] também. Esses experimentos são definidos como:

- **Experimento 1** - a cada passo, o *workload* de cada tarefa base aumenta 10% do seu valor inicial. Esse experimento avalia o comportamento do sistema quando seu *workload* aumenta através do aumento do *workload* das tarefas.
- **Experimento 2** - a cada passo, a frequência de cada tarefa base aumenta 10% da sua frequência inicial. Esse experimento avalia o comportamento do sistema quando a carga de escalonamento do mesmo aumenta com o aumento das frequências das tarefas.
- **Experimento 3** - a cada passo, uma tarefa é inserida no sistema com os mesmos parâmetros de τ_3. Esse experimento avalia o comportamento do sistema quando seu *workload* e carga de escalonamento aumentam com o aumento do número de tarefas.

Nos experimentos relacionados ao **domínio da comunicação**, um nó é considerado o ***Master*** e outro nó é considerado o ***Slave***. No lado da aplicação do nó *Master*, os experimentos irão inicializar com um conjunto de tarefas base do tipo *publisher* $\{\tau_{P1}, \tau_{P2}, \tau_{P3}\}$, cujo objetivo é o envio de mensagens em tópicos individuais. No lado do *Slave*, tarefas base do tipo *subscriber* $\{\tau_{S1}, \tau_{S2}, \tau_{S3}\}$ irão ficar à espera das mensagens nos respectivos tópicos. O critério de aumento de cada passo do experimento foi definido conforme [4] e também por meio de análise do tempo de demora do experimento. Esses experimentos são definidos como:

- **Experimento 4** - a cada passo, o tamanho das mensagens base é acrescido pelo valor base (30 KiB). Esse experimento avalia o comportamento do sistema quando seu *workload* no domínio da comunicação aumenta com o acréscimo do tamanho das mensagens. Por existir um grande número de aplicações na robótica, também está presente uma variedade de tipos de mensagens com tamanhos variados, dessa forma justificando um experimento que verifique o impacto dos diferentes tamanhos.
- **Experimento 5** - aumentar a cada passo as frequências das tarefas (e consequentemente do envio de mensagens) em 10% os valores de base. Esse experimento avalia o comportamento do sistema quando a carga de escalonamento no domínio da comunicação aumenta com o aumento da frequência de envio de mensagens. Com uma justificativa semelhante ao Experimento 4, as diversas aplicações da robótica exigem frequências variadas. Por exemplo, sensores podem enviar amostras que variam na faixa dos Hz aos kHz.
- **Experimento 6** - uma tarefa com os mesmos parâmetros de τ_{S2} é inserida no *Slave* a cada passo. Esse experimento avalia o comportamento do sistema quando a carga de escalonamento no domínio da comunicação aumenta com o aumento do número de *subscribers* lendo do mesmo tópico. Esse é um comportamento comum na robótica, quando vários nós leem o valor enviado por um mesmo sensor.

2.2 Plataformas de Sistemas Embarcados para Avaliação

Como objetos de estudo de caso do *benchmark*, foram utilizadas duas plataformas de desenvolvimento em sistemas embarcados. Uma delas é o bem conhecido Raspberry Pi 3™ (RPi3), uma placa que contém como processador principal um *system-on-a-chip* (SoC) projetado para aplicações que utilizam SOs de propósito geral. Por outro lado, aplicamos um *co-kernel* de tempo real, o Xenomai, juntamente com uma versão do Linux para fornecer ao *benchmark* os serviços de tempo real necessários. Nessa placa, aplicamos os experimentos 1 à 3 para verificação da capacidade de tempo real do sistema em escalonar as tarefas em um único núcleo, e aplicamos os experimentos 4 à 6 para verificar os recursos de tempo real quando existe comunicação das tarefas entre núcleos.

A segunda placa é a plataforma NXP® FRDM-K22F, utilizada para aplicações com microcontroladores (MCUs). Aqui, aplicamos apenas os experimentos 1 à 3 para verificação da capacidade de tempo real do sistema em escalonar as tarefas

em um único núcleo. No restante da seção, serão apresentados alguns detalhes sobre cada um e como o *benchmark* será aplicado.

2.3 Resultados dos Experimentos nas Plataformas

Os resultados gerais obtidos dos experimentos no ponto de ruptura sobre o RPi3 estão apresentados nas tabelas 1 e 2. Na primeira estão presentes os resultados obtidos dos experimentos relacionados ao domínio do processamento. Em "Nº do Teste", indica-se o número do teste quando ocorreu o ponto de ruptura. Em "*Deadlines* Atendidos" estão indicados o número total de *deadlines* atendidos por todas as tarefas no teste. Em "*Deadlines* Perdidos" estão presentes os números totais de *deadlines* perdidos por instâncias que iniciaram suas execuções. Em "*Deadlines* Pulados", indicam-se quantos *deadlines* foram pulados, pois quando uma instância perde seus *deadline*, entrando nos períodos de ativação das instâncias seguintes, estas não serão mais criadas (serão puladas). Finalmente, são apresentados os valores de utilização do processador na última coluna.

Tabela 1. Resultados obtidos dos experimentos relacionados ao domínio do processamento durante o ponto de ruptura no Raspberry Pi 3.

Exp.	Nº do Teste	*Deadlines* Atendidos	*Deadlines* Perdidos	*Deadlines* Pulados	Utilização do Processador
1	50	3655	19	38	82,89%
2	49	21953	14	28	81,51%
3	28	14223	16	32	88,42%

Na tabela 2 estão apresentados os resultados obtidos dos experimentos relacionados ao domínio da comunicação. Seus campos são semelhantes ao da tabela 1, exceto pela ausência de um campo para a utilização do processador (já que o objetivo agora é a análise da comunicação) e a presença de campos individuais para os números de *deadlines* atendidos em cada nó. Ainda, os *deadlines* perdidos e pulados ocorrem apenas no *Slave* (sendo relacionados aos *deadlines* das mensagens).

Tabela 2. Resultados obtidos dos experimentos relacionados ao domínio da comunicação durante o ponto de ruptura no Raspberry Pi 3.

Exp.	Nº do Teste	*Deadlines* Atendidos (*Master*)	*Deadlines* Atendidos (*Slave*)	*Deadlines* Perdidos	*Deadlines* Pulados
4	58	2477	2475	1	2
5	151	25221	25219	1	2
6	54	2509	24822	0	0

Observando a tabela 1, para todos os experimentos do domínio do processamento, os resultados de *deadlines* perdidos correspondem às tarefas de menor prioridade. Como cada umas destas tem sua execução postergada até que não hajam tarefas de maior prioridade prontas, os resultados são consistentes do que se espera em um escalonamento RM. Ademais, para este escalonamento, independentemente do *workload* e da frequência das tarefas: o escalonamento de tempo real é garantido em tempo de projeto para cinco tarefas apenas se a utilização do processador for menor que 74.35% [6]. Dito isto, para todos os experimentos, as tarefas passam no teste proposto por [6], por uma grande margem, seja com o aumento do *workload*, frequência ou número de tarefas. Portanto, o sistema apresenta bom desempenho de tempo real para todos os cenários propostos relacionados ao domínio do processamento.

Nos experimentos relacionados ao domínio da comunicação, cujos resultados estão presentes na tabela 2, é apresentado um comportamento diferente dos experimentos relacionados ao domínio do processamento: os *deadlines* são perdidos pelas tarefas de maior prioridade, ao invés das menos prioritárias. Isso se deve à implementação do sistema de comunicação com PS, onde todas as tarefas no núcleo terão sua execução postergada no envio de qualquer mensagem. Isso irá causar um maior impacto nas tarefas mais prioritárias, pois possuem períodos de ativação menores.

Apesar disso, os resultados apresentados confirmam um bom desempenho de tempo real do sistema. No Experimento 4, o tamanho das mensagens no ponto de ruptura é próximo dos 2 MB. Dessa forma, o sistema apresenta uma previsibilidade de tempo (tarefas completando antes de seus *deadlines*) antes de 2 MB. Isso envolve um grande número de aplicação na robótica, tais como odometria por *encoders*, desvio de obstáculos por sensores ultrassônicos e muitos outros que não utilizam quantidade massiva de dados na transação de mensagens.

No Experimento 5, a tarefa mais prioritária alcança uma frequência próxima dos 3 kHz, o que abrange grande parte das aplicações em robótica [11, 12, 3].

Nota-se que no Experimento 6 o número máximo de tarefas criadas do Slave foi 58. Dessa forma, o sistema atingiu o ponto de ruptura por falta de memória, e não por perda de *deadlines*. Apesar disso, esse número é bastante expressivo para um sistema embarcado rodando em uma aplicação robótica, já que cada tarefa está em muitos casos relacionada com o tratamento de informação de um periférico.

Portanto, o sistema apresenta um desempenho de tempo real considerável nos cenários impostos relacionados ao domínio da comunicação.

Os resultados gerais da análise orientada a aplicação para o domínio do processamento no FRDM-K22F estão apresentados na tabela 3. Os campos são semelhantes ao da tabela 1.

Como no RPi3, os resultados no FRDM-K22F são os esperados quando utiliza-se o escalonamento RM. Todos os *deadlines* que não foram alcançados correspondem às tarefas de menor prioridade, pelo fato da execução destas ser postergada até que nenhuma tarefa de maior prioridade esteja pronta. No Experimento 2 não há *deadlines* perdidos ou pulados, pois o sistema atinge o ponto

Tabela 3. Resultados obtidos dos experimentos relacionados ao domínio do processamento durante o ponto de ruptura no FRDM-K22F.

Exp.	Nº do Teste	*Deadlines* Atendidos	*Deadlines* Perdidos	*Deadlines* Pulados	Utilização do Processador
1	57	3905	3	6	84,35%
2	21	12100	0	0	39,84%
3	27	14950	52	104	90,34%

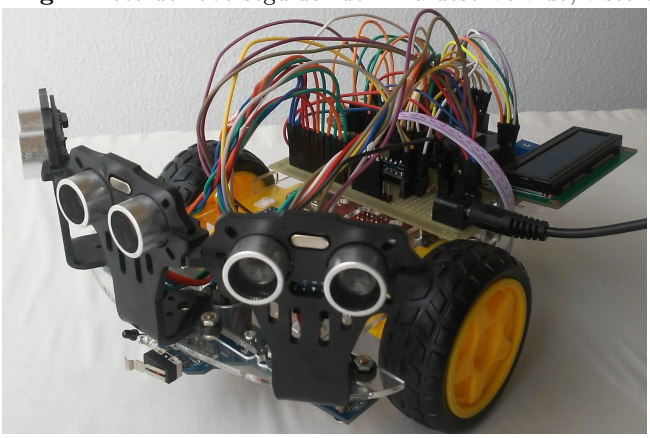
de ruptura por falta de granularidade do temporizador – o qual utilizou-se o *tick* do escalonador. Isso ocorreu pelo fato de seu período de ativação ser definido com o valor de $4,96$ ms para o próximo passo, menor do que o *tick* do escalonador de 5 ms. Para os outros dois experimentos, as tarefas passam no teste do RM [6] por uma grande margem. Por conseguinte, o sistema apresenta bom desempenho de tempo real para todos os cenários, mesmo para o Experimento 2 quando utiliza-se períodos de ativação maiores que o *tick* do escalonador.

3 Implementação de Robô Móvel utilizando Sistema Embarcado de Tempo Real

O robô criado é uma versão simplificada e miniaturizada de um robô seguidor de linha industrial, ou seja, este deve detectar e seguir uma trilha no formato de linha no chão. Uma apresentação parcial desse projeto foi apresentado em [7]. As tarefas de tempo real do robô foram desenvolvidas utilizando o modelo de etapas do *workflow* proposto, posteriores à avaliação da plataforma computacional para uso como STR. O mesmo deve estar dotado de sensores para detecção de obstáculos que podem aparecer no caminho (ex.: ser humano), de forma a parar o percurso. O tempo de resposta após a detecção do obstáculo deve ser garantido em tempo de projeto, para que nenhum acidente ocorra. O robô é composto por rodas diferenciais (*differential wheeled robot*) e uma roda de giro livre para controle de estabilidade. Na figura 1 é apresentada uma foto em perspectiva cavaleira.

O robô desenvolvido possui diversos requisitos temporais. Um destes é a obtenção das informações da linha no chão em tempo hábil, senão poderá desviar da rota. Também, deve enviar os comandos de atuação aos motores pela mesma razão anterior. Os deslocamentos angulares, lineares e determinação da *pose* devem ser feitos no mesmo intervalo de tempo para que os valores coincidam com os parâmetros suficientemente aproximados de posição e orientação do robô. Quanto à detecção de obstáculos, dada a máxima velocidade que o robô realiza, o tempo entre o surgimento do obstáculo e atuação nos motores deve ser suficiente para que o robô pare. Além disso, nenhuma amostra de obstáculo deve ser perdida, caso contrário uma colisão irá ocorrer. Por fim, deve apresentar as informações de *pose* por meio da interface homem-máquina (IHM) para que o usuário acompanhe de maneira satisfatória as informações do robô. Este último,

Fig. 1. Foto do robô seguidor de linha desenvolvido, visto em perspectiva cavaleira.



pode ser tratado por tarefas *soft real-time* ou *hard real-time*, conforme a importância que seja a apresentação das informações ao usuário.

3.1 Tarefas Desenvolvidas

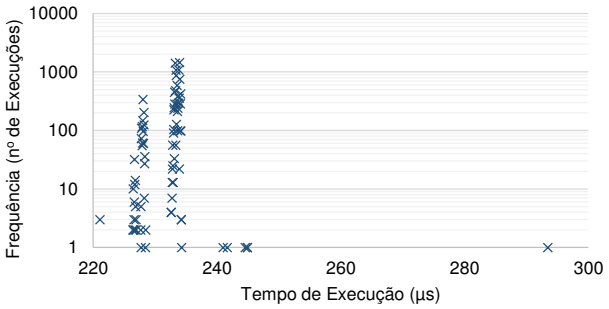
Na implementação do *software* de tempo real para controle do robô, referente à segunda etapa do *workflow* proposto, foram desenvolvidas sete tarefas para serem escalonadas através da política RM. Utilizou-se o RM para escalonar as tarefas por meio do escalonador preemptivo de prioridades fixas do FreeRTOS. As tarefas comunicam-se por mensagens em um modelo semelhante ao mecanismo PS. Aqui se define os períodos de ativação com base na necessidade de frequência de cada função do sistema.

Correspondente à terceira etapa do *workflow*, os códigos referentes à execução da tarefa são medidos um número suficientemente grande de vezes por meio de alguma ferramenta de instrumentação. Uma maneira usual de fazer isso é colocar o código à ser medido entre duas instruções de GPIO para geração de pulsos com largura igual aos tempos de execução do bloco de código [10, 1]. As tarefas devem ser alimentadas com um sequência de entradas de forma que se obtenham os estados de *hardware* mais representativos da aplicação com o intuito da geração dos WCET. Contudo, o método não fornece nenhuma garantia de que a execução mais longa observada (HOET) seja de fato o WCET. Contudo, alguns padrões para sistemas críticos consideram este um método válido para a determinação dos tempos de execução das tarefas [16, 1].

Como exemplo, na figura 2 é apresentado um gráfico com as medições de uma das tarefas, responsável pela determinação do deslocamento angular do robô, utilizando um analisador lógico. Há um total de 78 diferentes valores, sendo que o HOET ocorre apenas uma vez, e cujo valor é 293,42 μs. Por ser um caso tão isolado, e por depender de outro dispositivo computacional externo ao MCU (no caso, uma unidade inercial de movimento), é difícil determinar qual o motivo

específico na geração desse valor. Apesar disso, um HOET pode ser obtido e o WCET da tarefa foi definido como sendo 20% maior do que esse valor [1].

Fig. 2. Tempos de execução da tarefa τ_{imu}.



Por fim, são apresentadas as últimas etapas do *workflow*: teste de escalonamento no conjunto de tarefas e; junção de todas as tarefas para a execução concorrente, de forma a verificar se as restrições temporais do sistema serão atendidas durante a operação do robô. Utilizando os tempos de execução obtidos e os períodos de ativação definidos, é possível aplicar um teste exato na avaliação da escalonabilidade das tarefas desenvolvidas por meio da análise de seus tempos de resposta [15]. Para garantir que a tarefa atenda todos os seus requisitos temporais ao longo da operação do sistema, com base nos parâmetros obtidos, é necessário que seu tempo de resposta mais demorado (***worst-case response time*** - WCRT), seja menor que seu *deadline* relativo. Os tempos de bloqueio das tarefas desenvolvidas são considerados nulos, pois foram incorporados aos seus tempos de execução por meio da metodologia de medição denominada MBDTA (*measurement-based timing analysis*).

Na Tabela 4) são apresentados alguns dos principais parâmetros definidos para as tarefas desenvolvidas

Tabela 4. Relação entre as tarefas τ_i propostas, períodos de ativação T_i, WCETs C_i e tempos de resposta R_i. Onde i é um identificador da tarefa.

τ_i	$\tau_{isr-enc}$	$\tau_{isr-ult}$	τ_{enc}	τ_{imu}	τ_{pos}	τ_{obt}	τ_{twi}	τ_{mot}	τ_{lcd}
$T_i(ms)$	5	19	20	20	20	60	75	80	150
$C_i(\mu s)$	18	50	120	350	460	71	115	118	$31{\cdot}10^3$
$R_i(ms)$	$18{\cdot}10^{-3}$	$68{\cdot}10^{-3}$	0,538	0,538	1,066	1,069	1,184	1,302	33,39

Para a verificação da perda de *deadlines*, em cada tarefa foi colocado um código que compara seu tempo de resposta com o período de ativação. Caso, o

valor seja ultrapassado, o *deadline* não foi atingido, e o sistema envia um sinal luminoso. Nesse caso, o robô é parado.

Como a alimentação do robô foi determinada pela rede elétrica, então foi possível deixa-lo operando por seis horas ininterruptas. Sua operação foi interrompida pelo desligamento do sistema. Nesse intervalo de tempo, nenhum *deadline* foi perdido, pois o robô seguiu operando de maneira desejada, seja mantendo-se na linha, parando na presença de obstáculos e apresentado a *pose* correta na IHM. Portanto, o robô apresentou de forma satisfatória a operação desejada, conforme o planejamento da sua parte física e das tarefas de tempo real desenvolvidas.

4 Conclusão

Considera-se que é possível desenvolver robôs com requisitos de tempo real, utilizando ferramentas disponíveis no mercado, por meio de um *workflow* bem definido – ou seja, uma sequência de atividades ordenadas no tempo, onde cada atividade realiza alguma parte do trabalho, com a finalidade de atingir um determinado objetivo. Na primeira etapa do *workflow* proposto, na avaliação de sistemas computacionais, desenvolveu-se o *benchmark* denominado Hart-ROS. A ferramenta fornece uma métrica quantitativa, em termos de pontos de ruptura, de forma que seja possível avaliar a qualidade do sistema em atender aos requisitos temporais. Para avaliar a aplicabilidade do *benchmark* na avaliação de sistemas, foram verificadas duas plataformas de sistemas embarcados. Pode-se afirmar que a proposta de avaliação do desempenho de tempo real de sistemas computacionais por meio do *benchmark* desenvolvido atingiu o objetivo esperado, pois foi possível verificar características esperadas de STR sob as plataformas elencadas, e dessa forma, verificar o quão bom estas são em atender as demandas de aplicações com requisitos de tempo. Quanto às etapas seguintes do *workflow*, foi apresentado o desenvolvimento de um robô físico. Foi possível desenvolver e determinar que todo o conjunto de tarefas do sistema é escalonável em tempo real, sendo possível avaliar a operação do robô por diversas horas, verificando que: nenhum *deadline* foi perdido (cuja confirmação é realizada por cada tarefa) e; todas as funções foram corretamente executadas. Dessa forma, foi posto em evidência que é possível desenvolver robôs com requisitos de tempo real, utilizando ferramentas disponíveis no mercado. Como sugestões de trabalhos futuros, pode-se ampliar as funcionalidades do *benchmark*, seja aumentando o número de experimentos ou ampliando as configurações das tarefas. Além do exposto, outras aplicações robóticas (ex.: robôs aéreos) podem ser propostas de forma a verificar a aplicabilidade do *workflow* apresentado com diferentes ferramentas.

Referências

1. Abella, J., Hernandez, C., Quiñones, E., Cazorla, F.J., Conmy, P.R., Azkarate-Askasua, M., Perez, J., Mezzetti, E., Vardanega, T.: WCET analysis methods:

Pitfalls and challenges on their trustworthiness. In: 10th IEEE International Symposium on Industrial Embedded Systems (SIES). pp. 1–10. IEEE, Siegen (2015)
2. Curnow, H.J., Wichmann, B.A.: A synthetic benchmark. The Computer Journal **19**(1), 43–49 (1976)
3. Dudek, G., Jenkin, M.: Computational principles of mobile robotics. Cambridge university press, Cambridge (2010)
4. Kamenoff, N.I., Weiderman, N.H.: Hartstone distributed benchmark: requirements and definitions. In: Real-Time Systems Symposium, 1991. Proceedings., Twelfth. pp. 199–208. IEEE, San Antonio, Texas (1991)
5. Klein, M.H., Ralya, T., Pollak, B., Obenza, R., Harbour, M.G.: A Practioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publishers, Dordrecht (1993)
6. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) **20**(1), 46–61 (1973)
7. Pinto, M.L., de Oliveira, A.S.: Robotics Development using Real-Time Embedded System. In: 2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE). IEEE, Rio Grande, Brazil (2019)
8. Pinto, M.L., de Oliveira, A.S., Wehrmeister, M.A.: Real-Time Systems Evaluation for Robotics Using the Hart-ROS Benchmark. In: 2019 19th International Conference on Advanced Robotics (ICAR). pp. 296–301. IEEE, Belo Horizonte, Brazil (2019)
9. Rajkumar, R., Gagliardi, M., Sha, L.: The real-time publisher/subscriber inter-process communication model for distributed real-time systems: design and implementation. In: 1st IEEE Real-Time Technology and Applications Symposium. pp. 66–75. IEEE, Illinois, United States (1995)
10. Rapita Systems: Automating WCET Analysis for DO-178B & DO-178C (2020), https://www.rapitasystems.com/downloads/automating-wcet-analysis-do-178b-do-178c
11. Siciliano, B., Khatib, O.: Springer handbook of robotics. Springer, Berlin (2016)
12. Siegwart, R., Nourbakhsh, I.R., Scaramuzza, D., Arkin, R.C.: Introduction to autonomous mobile robots. MIT press, Massachusetts, 2 edn. (2011)
13. Sommerville, I.: Software engineering. Pearson Education, Boston, 9 edn. (2011)
14. Tanenbaum, A.S., Van Steen, M.: Distributed systems: principles and paradigms. Prentice-Hall, New Jersey (2007)
15. Tindell, K.W., Burns, A., Wellings, A.J.: An extendible approach for analyzing fixed priority hard real-time tasks. Real-Time Systems **6**(2), 133–151 (1994)
16. Wartel, F., Kosmidis, L., Gogonel, A., Baldovino, A., Stephenson, Z., Triquet, B., Quinones, E., Lo, C., Mezzetta, E., Broster, I., et al.: Timing analysis of an avionics case study on complex hardware/software platforms. In: 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 397–402. IEEE, Grenoble, France (2015)
17. Weiderman, N.H., Kamenoff, N.I.: Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems. Real-Time Systems **4**(4), 353–382 (1992)