Avaliação dos Sistemas Operacionais MQX Lite e FreeRTOS em Aplicações de Robótica Móvel

Fernando Emilio Puntel[1], Joildo Schueroff[1], Giann Carlos Spilere Nandi[1], Anderson Luiz Fernandes Perez[1]

[1]Laboratório de Automação e Robótica Móvel
Universidade Federal de Santa Catarina (UFSC)
88.900-000 – Araranguá – SC – Brazil

fernandopuntel@gmail.com, joildoschueroff@gmail.com,
giannnandi@gmail.com, anderson.perez@ufsc.br

Abstract. *The use of embedded operating systems in applications of mobile robotics is critical, as is common robots perform various tasks and thus having a hardware complex. Therefore, the choice of operating system for a robot must take into consideration factors such as performance, ease of use and compatibility of the system with the robot's hardware. In this article are described the results obtained from the comparison of the performance of embedded operating systems MQX Lite and FreeRTOS on a mobile robot.*

Resumo. *O uso de sistemas operacionais embarcados em aplicações de robótica móvel é fundamental, pois é comum robôs realizarem várias tarefas e com isso possuírem um hardware complexo. Por isso, a escolha do sistema operacional para um robô deve levar em consideração fatores como desempenho, facilidade de uso e compatibilidade do sistema com o hardware do robô. Neste artigo são descritos os resultados obtidos da comparação de desempenho dos sistemas operacionais embarcados MQX Lite e FreeRTOS em um robô móvel.*

1. Introdução

No passado o uso de robôs visava somente tarefas que poderiam causar algum risco de vida ou que fossem repetitivas e estressantes aos seres humanos. Entretanto, nos dias atuais, robôs são projetados e construídos para serem utilizados na realização de inúmeras tarefas, aspirar o pó de uma residência (FORLIZZI and DISALVO 2006), servir de enfermeira para uma pessoa idosa (KLEIN, SPENCER et al. 2012) até auxiliar em procedimentos na área médica (ENTSFELLNER, TAUBER et al. 2013).

A evolução tecnológica aliada ao barateamento dos custos de produção é responsável pela popularização da robótica. Com isso, robôs estão deixando a condição de peças de ficção científica para entrarem de vez nos lares das pessoas como um bem de consumo. Contudo, mesmo os robôs mais simples, possuem um hardware sofisticado composto por vários sensores e atuadores.

A complexidade do hardware e a necessidade de execução de várias tarefas inerentes aos robôs justificam a utilização de um sistema operacional embarcado (SOE). Um SOE é responsável por todo o gerenciamento dos recursos existentes no hardware

do robô, bem como, em algumas aplicações, prover uma interface amigável para o utilizador do robô (EBENHOFER, BAUER et al. 2013), (HSU-CHIH and CHING-CHIH 2009) e (BALFOUR 2010).

Neste trabalho é descrita a avaliação dos sistemas operacionais embarcados FreeRTOS e MQX Lite com relação a eficiência no gerenciamento e na execução das tarefas de um robô móvel explorador responsável por navegar em um ambiente aberto ou fechado desviando de obstáculos.

Este trabalho está organizado em mais 4 (quatro) seções, que são: a Seção 2 descreve as principais características do robô explorador utilizado nos experimentos; na Seção 3, são descritas as principais características dos sistemas operacionais FreeRTOS e MQX Lite; os resultados obtidos na avaliação da eficiência dos sistemas operacionais FreeRTOS e MQX Lite são descritos na Seção 4; na Seção 5 são feitas as considerações finais e são apresentadas algumas propostas para trabalhos futuros.

2. O Robô Explorador

O robô explorador é baseado em uma caminhonete de controle remoto e foi projetado para atuar em ambiente *indoor* ou *outdoor*. É dotado de uma câmera do tipo webcam, responsável pela captura das imagens do ambiente, 6 (seis) sensores de ultrassom, que detectam a proximidade do robô a algum objeto presente no ambiente, evitando assim o choque com este objeto e 2 (dois) motores, um motor de corrente contínua responsável pela tração do robô e um motor de passo nas rodas dianteiras responsável pela direção do robô.

A Figura 1 ilustra uma imagem do robô explorador onde é possível perceber o posicionamento da câmera e dos sensores de ultrassom.

Figura 1 – Robô Explorador

A navegação do robô é baseada nas informações visuais recebidas pela câmera e também pelas informações provenientes dos sensores de ultrassom. O objetivo da navegação visual é permitir que o robô siga um determinado objeto, desta forma, conforme o objeto se movimente no ambiente, o robô também se movimentará, mantendo o objeto sempre no alvo, ou seja, no foco da câmera.

A Figura 2 ilustra a arquitetura de controle de software desenvolvida para o robô explorador. Esta arquitetura é baseada em dois níveis de controle, sendo o primeiro, o nível deliberativo, responsável pelo processamento das imagens provenientes da câmera e o segundo nível, o reativo, responsável pela interpretação dos sinais recebidos pelos sensores de ultrassom e o controle dos motores.

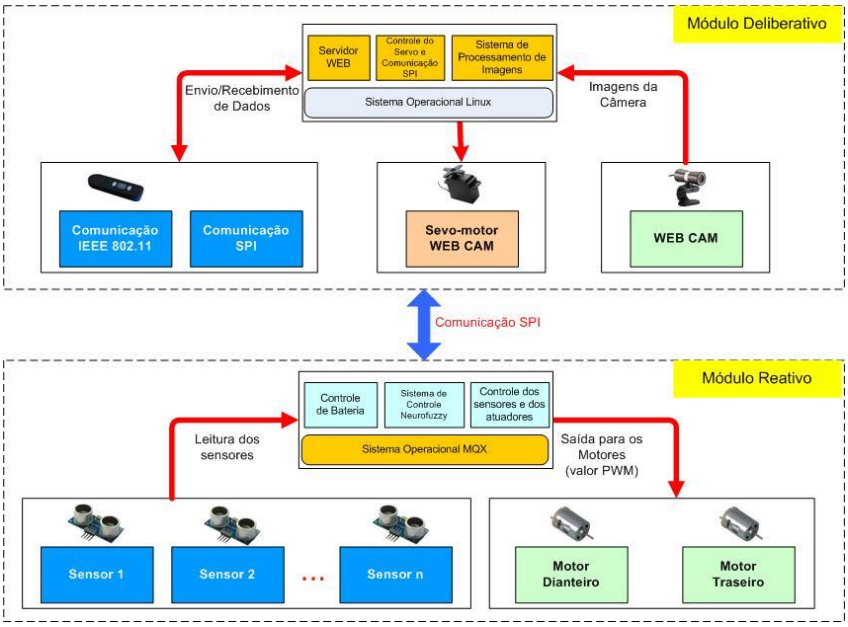

Figura 2. Arquitetura de controle de software do robô explorador

A Figura 3. **Arquitetura de controle de hardware do robô exploradorilustra** a arquitetura de controle de hardware utilizado no robô explorador.

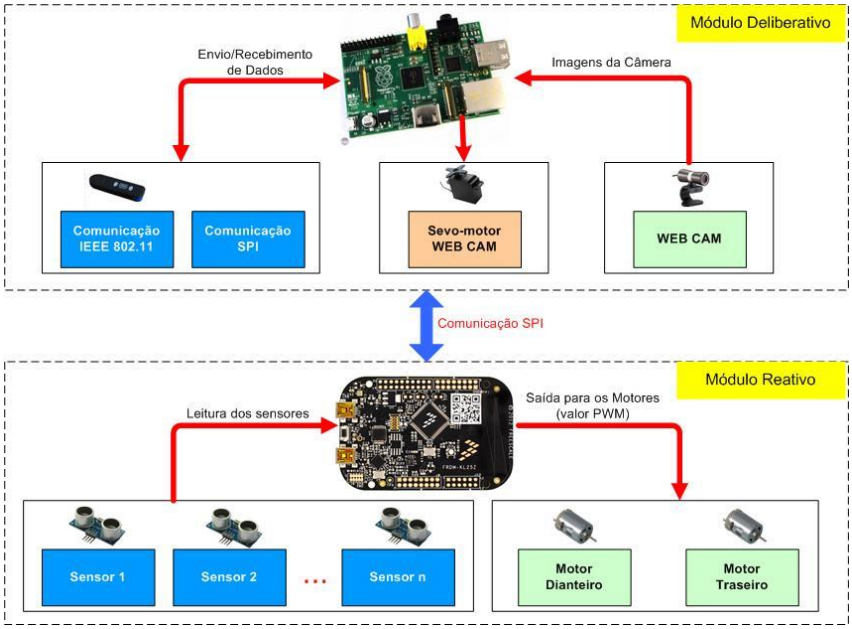

Figura 3. Arquitetura de controle de hardware do robô explorador

No módulo deliberativo uma placa do tipo Raspberry Pi modelo B é responsável pelo execução do sistema de processamento de imagem, controle do servo motor da câmera e outras funcionalidades de alto nível.

O módulo reativo é composto por uma placa do tipo *Freedom Board* da Freescale equipada com microcontrolador de núcleo ARM de 32 bits. Neste módulo de controle são executadas três tarefas, sendo uma para o controle e acionamento dos motores, uma para a leitura dos sensores e a última para o controle do nível de bateria do robô. As tarefas existentes neste módulo são o objeto de estudos deste artigo, ou seja, é neste módulo que os sistemas operacionais FreeRTOS e MQX Lite são avaliados.

3. Sistema Operacional MQX Lite e FreeRTOS

Para realização dos experimentos foram escolhidos dois sistemas operacionais embarcados, o MQX Lite e o FreeRTOS. Nas subseções seguintes serão descritas sucintamente as principais características do sistema operacional MQX Lite (Subseção 3.1) e do sistema operacional FreeRTOS (Subseção 3.2).

É importante salientar que esses sistemas foram escolhidos por sua simplicidade e facilidade de programação e também porque são compatíveis com o microcontrolador Kinetis L Cortex M0+ presente na placa *Freedom Board* da Freescale (FREESCALE 2012), a qual está presente no módulo reativo do sistema de controle do robô explorador, apresentado na seção anterior.

3.1 Sistema Operacional MQX Lite

O MQX Lite é um Sistema Operacional de Tempo Real (SOTR) totalmente integrado com a plataforma *Freedom Board* da Freescale. Mesmo sendo um SOTR de pequeno porte, oferece suporte as principais funções de um SO embarcado, tais como semáforos, gerenciamento de memória e gerenciamento de *threads*.

O MQX Lite suporta componentes do *Process Expert*, um sistema de desenvolvimento para criar, configurar, otimizar e migrar componentes de software para plataformas da Freescale.

O MQX Lite se destaca por possuir uma quantidade de componentes menor, permitindo assim que aplicativos que consomem menos de 4KB (*Kilobyte*) de RAM (*Random Acess Memory*) sejam executados. O *kernel* do MQX Lite é baseado no SOTR MQX, a Figura 4 ilustra quais são os componentes suportados por ambos os sistemas.

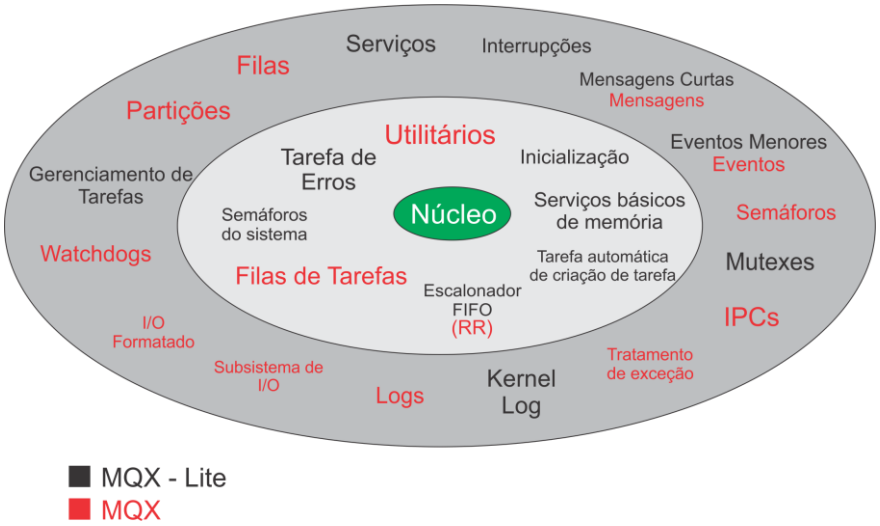

Figura 4. Comparação entre os sistemas operacionais MQX e MQX Lite
Extraído e Adaptado de: (FREESCALE 2013)

O escalonador de processos do MQX Lite é baseado no algoritmo FIFO (*First in, First out*). Possui vários componentes que são divididos em principais e opcionais. Os componentes principais representam as funções do MQX Lite que serão incluídas na imagem executável após a compilação do projeto que se está desenvolvendo, isto é, apenas os componentes inclusos na placa. Porém um aplicativo pode estender suas funcionalidades, configurando os componentes do núcleo e adicionando outros componentes (FREESCALE 2013).

Para alocação de memória o MQX Lite fornece um mecanismo de alocação de memória leve, muito parecido com as funções da biblioteca padrão da linguagem de programação C *malloc()* e *free()*. A diferença é que o componente de alocação de memória fornece um mecanismo seguro tanto para alocação como para a liberação de memória para tarefas concorrentes (FREESCALE 2013).

3.2 Sistema Operacional FreeRTOS

O FreeRTOS é um SOTR de código aberto e possui uma licença GPL (*General Publi License*) (LICENSE 2014). Possui um núcleo pequeno, com um escalonador de processos baseado no algoritmo de prioridades, também suporta semáforos binários e filas de mensagens (BARRY 2009).

O *kernel* do FreeRTOS é totalmente escrito na linguagem C e é composto por quatro arquivos, o que facilita qualquer tipo de manutenção. A Figura 5 ilustra a estrutura do *kernel* do sistema operacional FreeRTOS.

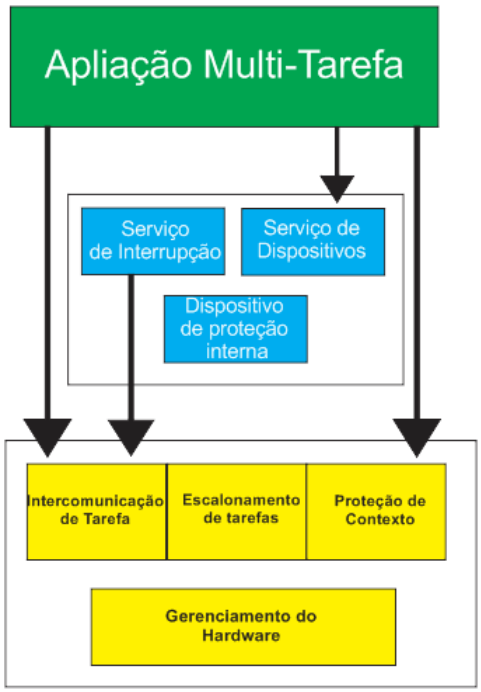

Figura 5. Estrutura do *kernel* do sistema operacional FreeRTOS
Extraído e adaptado de: (SHANCAO, JING et al. 2012)

Atualmente o FreeRTOS possui versões para 23 (vinte e três) arquiteturas que variam de 8 a 32 bits. Suas principais características são a portabilidade, escalabilidade e simplicidade, além de suportar um número elevado de tarefas (INAM, MAKI-TURJA et al. 2011).

O FreeRTOS possibilita ao usuário acesso facilitado aos recursos de hardware, permitindo que o desenvolvimento de sistemas de tempo real seja mais ágil (GALVÃO 2010).

4. Avaliação dos Sistemas Operacionais MQX Lite e FreeRTOS

A avaliação dos sistemas operacionais MQX Lite e FreeRTOS foram baseadas em testes realizados em bancada. Foram analisados os tempos efetivos de execução de duas tarefas do módulo reativo: a tarefas de controle dos motores e a tarefa de controle dos sensores.

A tarefa sensores é responsável pela leitura dos dados dos sensores e realiza a operação de escrita destes dados em variáveis alocadas na memória principal, a tarefa motores realiza a operação de leitura destes dados, para garantir a exclusão mútua foi utilizado o mecanismo de sincronização de processos semáforo.

Foram realizados dois experimentos onde o primeiro atestou a entrada em execução de ambas às tarefas após serem selecionadas pelo escalonador de processos e o segundo avaliou a execução das tarefas manipulando os atuadores.

Em ambos os experimentos foi atribuído uma prioridade padrão para as tarefas, ou seja, 8 (oito) no MQX Lite e 0 (zero) no FreeRTOS. Os dados dos experimentos foram obtidos com o auxílio de um osciloscópio.

Para testar a integridade dos dados obtidos através do osciloscópio, ambos os experimentos foram realizados 6 (seis) vezes. Após os testes, foram avaliados os dados obtidos e verificado que em todas as ocasiões os SOEs se comportaram de maneira semelhante.

As configurações utilizadas no osciloscópio durante os testes estão presentes nas figuras dos experimentos descritos a seguir.

4.1 Experimento 1 – Entrada das Tarefas em Execução

Este experimento teve como objetivo validar a entrada em execução das tarefas e a efetividade do escalonador de tarefas. Na Figura 6 e na Figura 7 são ilustrados o comportamento dos dois sistemas operacionais.

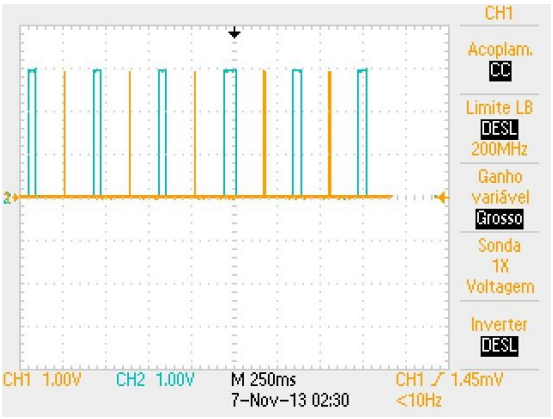

Figura 6. Resultado obtido com o sistema operacional MQX Lite

Figura 7. Resultado obtido com o sistema operacional FreeRTOS

Conforme ilustrado na Figura 6 e na Figura 7, o gráfico em azul representa a tarefa “sensores” e o gráfico em laranja representa a tarefa “motores”, ilustrando o tempo que cada tarefas permaneceu em execução.

Neste experimento a tarefa sensores somente realiza a leitura dos sensores dianteiro do robô. E a tarefa motores faz a verificação do dado recebido a partir dos sensores. Como neste experimento nenhum objeto está a menos de 35 centímetros dos sensores dianteiros a tarefa motores não realiza nenhuma operação.

A atividade das tarefas é sincronizada com o uso de um semáforo, onde a tarefa sensores obtém o semáforo, faz a leitura dos sensores instalados na dianteira do robô e o libera para que a tarefa “motores” possa executar.

Ao final da execução cada tarefa faz uma chamada de sistema *yield()* para liberar o processador para que outra tarefa entre em execução. Para melhorar a visualização do gráfico foi necessário adicionar um *delay* de 150 ms (millissegundos) em cada tarefa. Com os resultados obtidos é possível perceber que ambos os sistemas operacionais obtiveram desempenho semelhante.

4.2 Experimento 2 – Execução das Tarefas Manipulando os Atuadores

O objetivo deste experimento é avaliar o desempenho das tarefas na manipulação dos dados obtidos a partir de um sensor e, dependendo do resultado ativar os atuadores. Na Figura 8 e na Figura 9 são ilustrados os gráficos de desempenho de ambos os sistemas operacionais embarcados.

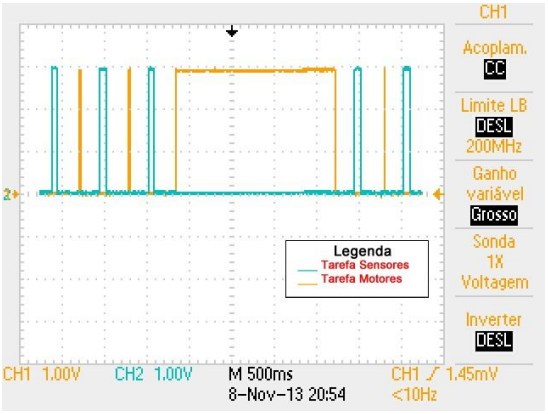

Figura 8. Avaliação MQX Lite com atuadores

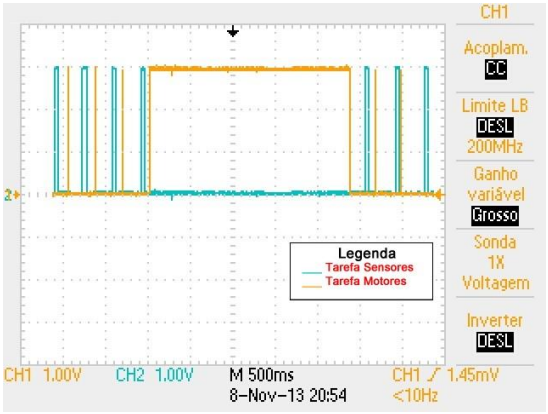

Figura 9. Avaliação FreeRTOS com atuadores

Neste experimento, assim como no experimento 1 (um), foi utilizado um semáforo para sincronizar as atividades entre as tarefas “sensores” e “motores”. Conforme ilustrado na Figura 8 e na Figura 9, o gráfico em azul representa a tarefa “sensores” e o gráfico em laranja representa a tarefa “motores” e o tempo que cada tarefa permaneceu em execução.

Para avaliação foi utilizado o sensor dianteiro e traseiro. Neste caso quando o sensor dianteiro identificasse um objeto a menos de 35 cm, a tarefa “sensores” liberava o semáforo para que a tarefa “motores” imediatamente mudasse o sentido do motor traseiro, e então o valor recebido pelo sensor traseiro era verificado, como ilustra o gráfico na cor laranja na Figura 8 e na Figura 9, até que a distância do sensor dianteiro tivesse a uma distância de 45 cm do objeto para continuar para frente.

Neste experimento é possível verificar que o sistema embarcado MQX Lite foi mais eficaz na operação de mudança do sentido do motor enquanto o FreeRTOS obteve melhor desempenho na leitura dos sensores.

Salienta-se que em ambos os experimentos em alguns momentos o processador foi utilizado para executar tarefas específicas do próprio SO.

Vale ressaltar que o robô possui 3 (três) sensores instalados na parte dianteira e 3 (três) sensores instalados na parte traseira do robô, porém neste experimento foi utilizado 1 (um) sensor dianteiro e 1 (um) sensor traseiro.

5. Considerações Finais

Neste artigo foi apresentado os resultados da comparação de desempenho dos sistemas operacionais embarcados MQX Lite e FreeRTOS aplicados em um robô móvel do tipo explorador de ambientes. Os resultados obtidos demonstram que ambos os sistemas operacionais são indicados para aplicações de robótica móvel.

A API e o material de apoio de ambos os sistemas foram utilizados para o desenvolvimento dos experimentos. Devido ao FreeRTOS ser portável para uma maior diversidade de arquiteturas e ser um sistema *open source*, a documentação disponível é superior ao MQX Lite.

O algoritmo desenvolvido para ambos os sistemas embarcados se mostrou bastante robusto, demonstrando um desempenho favorável nos experimentos realizados em bancada.

Destaca-se o tamanho reduzido dos sistemas embarcados utilizados, facilitando a experimentação e permitindo o uso de uma plataforma com pouca memória. Destaca-se também que o escalonador de tarefas de ambos os sistemas operacionais se mostraram eficientes.

Como trabalhos futuros pretende-se realizar a implementação de uma tarefa para controle de bateria, a implementação de um sistema de controle baseado em redes neurais artificiais e lógica fuzzy (sistema *neuro-fuzzy*) e a integração do módulo reativo com o módulo deliberativo a partir da comunicação via protocolo SPI (*Serial Peripheral Interface*).

Agradecimentos

Os autores, Fernando Emilio Puntel, Joildo Schueroff e Giann Carlos Spileri Nandi, agradecem a Universidade Federal de Santa Catarina pela bolsa de estudos.

Referências

BALFOUR, J. D. (2010). Efficient embedded computing, Stanford University.

BARRY, R. (2009). Using the FreeRTOS real time kernel: a practical guide, Real Time Engineers Limited.

EBENHOFER, G., H. BAUER, et al. (2013). A system integration approach for service-oriented robotics. Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on.

ENTSFELLNER, K., R. TAUBER, et al. (2013). Development of universal gripping adapters: Sterile coupling of medical devices and robots using robotic fingers. Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on.

FORLIZZI, J. and C. DISALVO (2006). Service robots in the domestic environment: a study of the roomba vacuum in the home. Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction, ACM.

FREESCALE (2012). FRDM-KL25Z User's Manual. **1:** 14.

FREESCALE (2013). "Freescale MQX Lite real-time operating system - users guide." **1**.

GALVÃO, S. d. S. L. (2010). Modelagem Formal do Sistema Operacional de Tempo Real FreeRTOS Utilizando o Método B, UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE.

HSU-CHIH, H. and T. CHING-CHIH (2009). "FPGA Implementation of an Embedded Robust Adaptive Controller for Autonomous Omnidirectional Mobile Platform." Industrial Electronics, IEEE Transactions on **56**(5): 1604-1616.

INAM, R., J. MAKI-TURJA, et al. (2011). Support for hierarchical scheduling in FreeRTOS. Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on.

KLEIN, J., S. J. SPENCER, et al. (2012). "Breaking It Down Is Better: Haptic Decomposition of Complex Movements Aids in Robot-Assisted Motor Learning." Neural Systems and Rehabilitation Engineering, IEEE Transactions on **20**(3): 268-275.

LICENSE, G. G. P. (2014). "Disponível em:< http://www.gnu.org/licenses/gpl.html>." Acesso em: 27 de janeiro de 2014.

SHANCAO, N., Z. JING, et al. (2012). Analysis and Implementation of Migrating Real-Time Embedded Operating System FreeRTOS Kernel Based on S3C44B0 Processor. Information Science and Engineering (ISISE), 2012 International Symposium on.