UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS ARARANGUÁ

Fernando Emilio Puntel

AVALIAÇÃO DOS SISTEMAS OPERACIONAIS EMBARCADOS MQX LITE E FREERTOS EM APLICAÇÕES DE ROBÓTICA MÓVEL

Araranguá, dezembro de 2013.

Fernando Emilio Puntel

AVALIAÇÃO DOS SISTEMAS OPERACIONAIS EMBARCADOS MQX LITE E FREERTOS EM APLICAÇÕES DE ROBÓTICA MÓVEL

Trabalho de Conclusão de Curso submetido à Universidade Federal de Santa Catarina, como parte dos requisitos necessários para a obtenção do Grau de Bacharel em Tecnologias da Informação e Comunicação.

Araranguá, dezembro de 2013.

Fernando Emilio Puntel

AVALIAÇÃO DOS SISTEMAS OPERACIONAIS EMBARCADOS MQX LITE E FREERTOS EM APLICAÇÕES DE ROBÓTICA MÓVEL

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Tecnologias da Informação e Comunicação", e aprovado em sua forma final pelo Curso de Graduação em Tecnologias da Informação e Comunicação.

Araranguá, dezembro de 2013

Prof. Dr. Vilson Gruber
Coordenador do Curso

Prof. Dr. Anderson Luiz Fernandes Perez
Orientador

Banca Examinadora:

Prof. Dr. Anderson Luiz Fernandes Perez
Presidente

Prof. Dr. Fábio Rodrigues de La Rocha

Prof. Dr. Gustavo Medeiros de Araújo

Dedico este trabalho aos meus pais Juceli
e Loidir, e a minhas irmãs Bruna e Luiza.

AGRADECIMENTOS

Agradeço a Deus por me iluminar nesta caminhada. A minha família que mesmo longe sempre me incentivou e esteve do meu lado nas minhas decisões, nunca deixando de acreditar em mim. Aos meus amigos e membros do Laboratório de Automação e Robótica Móvel - LARM que estiveram presentes nesta jornada, em especial aos colegas Elder Dominghini Tramontin e ao Igor Salvaro Raupp, que foram peças fundamentais para a elaboração deste trabalho. Agradeço em especial ao meu professor orientador Anderson Luiz Fernandes Perez, que como um pai me incentivou nos momentos mais difíceis e me estimulou para a busca de novos conhecimentos.

"É muito melhor lançar-se em busca de conquistas grandiosas, mesmo expondo-se ao fracasso, do que alinhar-se com os pobres de espírito, que nem gozam muito nem sofrem muito, porque vivem numa penumbra cinzenta, onde não conhecem nem vitória, nem derrota."

Theodore Roosevelt

RESUMO

Os robôs móveis estão sendo utilizados em diversas áreas, auxiliando os seres humanos desde a exploração espacial até a limpeza de residências. Devido a complexidade de algumas tarefas que os robôs devem cumprir o uso de sistemas operacionais embarcados 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 trabalho são descritos os resultados obtidos da comparação de desempenho dos sistemas operacionais embarcados MQX Lite e FreeRTOS em um robô explorador de ambientes.

Palavras-chave: robótica móvel, sistemas operacionais embarcados, robô explorador, MQX Lite, FreeRTOS

ABSTRACT

Mobile robots are being used in various areas, helping humans from space exploration to cleaning homes. Due to the complexity of some tasks robots should fulfill the use of embedded operating systems is essential, as is common robots perform various tasks and thus possess hardware complex. Therefore, the choice of operating system to a robot must take into consideration factors such as performance, ease of use and compatibility of the system with the hardware of the robot. In this work we describe the results achieved of the performance comparison of embedded operating systems MQX Lite and FreeRTOS in rover.

Keywords: mobile robots, embedded operating systems, rover, MQX Lite, FreeRTOS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Com o passar dos anos a tecnologia se tornou algo indispensável em nosso dia a dia. Novas tecnologias vem surgindo no mercado, com tamanho cada vez menor e com maior capacidade de processamento. Com estes avanços, a robótica que antes era associada somente a braços manipuladores, aplicados à construção de algum bem de consumo, também passou a ser utilizada no auxílio ao ser humano. Robôs são projetados e construídos para serem utilizados na realização de inúmeras tarefas, como no ensino e aprendizado (PEREZ et al., 2013), tecnologias assistivas (ZANOTTO et al., 2013), na medicina (ENTSFELLNER et al., 2013) entre outras áreas.

Robôs realizam tarefas perigosas, repetitivas e estressantes, a complexidade do hardware e a necessidade de execução de várias tarefas justifiquem a utilização de um SOE (Sistema Operacional Embarcado) em um robô. Um SOE é responsável por todo o gerenciamento dos recursos de hardware existentes no robô.

Um robô móvel é dotado de um sistema de controle para que possa navegar em um ambiente *indoor*[1] ou *outdoor* [2], em alguns casos devido a complexidade do problema, justifica-se o uso de um sistema operacional embarcado (AL-JARRAH; ROTH, 2013), (HAMBLEN; BEKKUM, 2013) e (VAISHAK; SHILPA, 2013).

Neste trabalho será descrita a avaliação dos sistemas operacionais embarcados FreeRTOS e MQX Lite em termos da 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 por meio de sensores e atuadores.

1.1 OBJETIVOS

Para melhor compreensão deste trabalho os objetivos foram separados em gerais e específicos que serão descritos a seguir.

[1] local fechado. Exemplo: sala, casa etc.
[2] local aberto. Exemplo: parque, campo etc.

1.1.1 Geral

Avaliar os sistemas operacionais embarcado MQX Lite e FreeRTOS em um robô móvel explorador de ambiente *indoor* e *outdoor*.

1.1.2 Específicos

1. Levantar o estado da arte em sistemas embarcados;
2. Levantar o estado da arte sobre robótica móvel e o desenvolvimento de sistema de controle para robôs móveis;
3. Estudar a arquitetura e a API (*Application Programming Interface*) de programação dos Sistemas Operacionais MQX Lite e FreeRTOS;
4. Avaliar os sistemas operacionais MQX Lite e FreeRTOS em uma aplicação de controle de um robô explorador;
5. Analisar e descrever resultados obtidos em (4).

1.2 JUSTIFICATIVA E MOTIVAÇÃO

A utilização de robôs para resolução de problemas é muito comum, algumas tarefas possuem um grau de complexidade maior. Uma das principais dificuldades é fazer com que o robô realize diversas tarefas simultâneas de maneira eficiente.

Em alguns sistemas robóticos é indispensável o uso de um sistema operacional embarcado, facilitando o gerenciamento de recursos, tais como memória e processamento. Também por permitir que o sistema de controle do robô seja separado em várias tarefas e *threads*. Algumas tarefas possuem nível crítico, ou seja, são obrigadas a responderem em um prazo determinado (LUO; JHA, 2000).

Um robô necessita controlar diversos sensores e atuadores em tempo real, se adaptando as mudanças no ambiente. Fazer com que robôs consigam executar as tarefas em tempo hábil, passando confiabilidade e eficiência, é um desafio muito complexo que proporciona uma oportunidade para pesquisas na área de sistemas operacionais embarcados.

Entre os sistemas operacionais embarcados, o FreeRTOS é o que

mais se destaca entre os *opensource* [3], por possuir uma arquitetura compatível com diversas plataformas, além de possuir uma comunidade muito atuante. Outro sistema embarcado que se destaca é o MQX Lite (FREESCALE, 2013a), uma versão reduzida do MQX (FREESCALE, 2013b) da Freescale.

1.3 METODOLOGIA

Este trabalho é uma pesquisa tecnológica que visa a avaliação dos sistemas operacionais embarcados MQX Lite e FreeRTOS em aplicações de robótica móvel.

O programas usados para a avaliação foram desenvolvidos na IDE (Ambiente Integrado de Desenvolvimento, do Inglês, *Integrated Development Environment*) CodeWarrior na linguagem de programação C.

Foram realizados experimentos em bancada com ambos os sistemas operacionais, no LARM (Laboratório de Automação e Robótica Móvel), com o auxílio de um osciloscópio.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho, além desta introdução, está dividido em mais 6 (cinco) capítulos e um anexo.

O **Capítulo 2** apresenta a definição de sistema operacional e seus componentes. Também são descritos Sistemas Operacionais de Tempo Real e Sistemas Operacionais Embarcados, com ênfase nos sistemas MQX Lite e FreeRTOS.

O **Capítulo 3** apresenta um breve histórico da robótica. São descritos os conceitos de sensores e atuadores, bem como as principais arquiteturas de controle utilizadas em robôs móveis.

O **Capítulo 4** apresenta a descrição do robô explorador, o qual foi utilizado para realizar os experimentos com os sistemas operacionais embarcados MQX Lite e FreeRTOS, bem como sua arquitetura de hardware e software.

No **Capítulo 5** são apresentados os resultados da avaliação dos sistemas operacionais MQX Lite e FreeRTOS, no gerenciamento de duas tarefas do módulo Reativo do robô explorador.

[3] Designa-se a um software de código aberto, também conhecido como software livre.

O **Capítulo 6** apresenta as considerações finais e algumas propostas para trabalhos futuros.

No **Anexo A** são descritas em detalhes algumas das principais funções da API (*Application Programming Interface*) dos sistemas operacionais MQX Lite e FreeRTOS.

2 SISTEMAS OPERACIONAIS EMBARCADOS

Neste capítulo apresenta-se a definição de sistema operacional e seus componentes. Também são descritos Sistemas Operacionais de Tempo Real e Sistemas Operacionais Embarcados, com ênfase nos sistemas MQX Lite e FreeRTOS.

2.1 DEFINIÇÃO DE SISTEMA OPERACIONAL

Um sistema de computação pode ser separado em quatro partes, sendo: hardware, sistema operacional, programas e o usuário. O hardware fornece recursos básicos de computação do sistema. Os programas, ou aplicativos como também são chamados, definem a forma como estes recursos são utilizados. O Sistema Operacional (SO) controla o hardware e o uso destes recursos. O usuário é quem utiliza os aplicativos que executam no sistema operacional. A Figura 1 ilustra uma visão abstrata dos componentes de um sistema computacional. (SILBERSCHATZ; GALVIN; GAGNE, 2010)

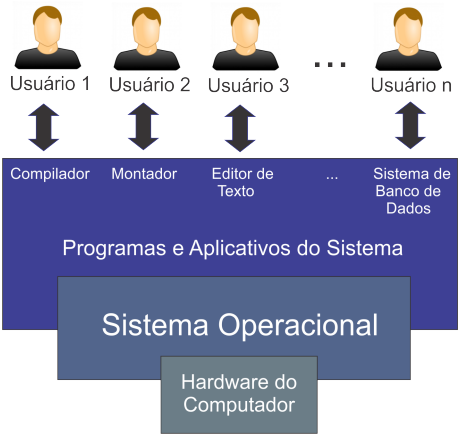

Figura 1 – Visão abstrata dos componentes de um sistema computacional
Extraído e adaptado de: (SILBERSCHATZ; GALVIN; GAGNE, 2010)

O sistema operacional pode ser visto de dois pontos de vista dis-

tintos: no primeiro, o usuário monopoliza os recursos e o SO deve facilitar e maximizar a atividade que este usuário está executando. Neste caso, o SO é projetado principalmente para permitir facilidade de uso; no ponto de vista do Sistema, o SO é o programa mais diretamente envolvido com o hardware, é considerado como um alocador de recursos, tais como: tempo de CPU (Unidade Central de Processamento, do Inglês, *Central Processing Unit*), espaço de memória, dispositivos de entrada/saída entre outros (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Indiferente da maneira como o sistema operacional possa ser visto não há uma definição adequada para este. Sistemas operacionais são criados para que seja possível utilizar um sistema computacional. A principal vantagem do SO é tornar o sistema computacional transparente, ou seja, de fácil utilização para o usuário.

Todo SO possui um núcleo, chamado de kernel, que é responsável pelo gerenciamento dos recursos. O kernel do SO é responsável pelo gerenciamento de processos, memória, arquivos e controle dos dispositivos de entrada e saída. Na seção 2.2.1 o kernel é descrito detalhadamente.

O kernel se mantém em execução durante o tempo em que o sistema computacional esteja ligado e disponibiliza serviços e recursos para aplicações que estão sendo executadas no SO. Para tanto, disponibiliza um conjunto de chamadas de sistema para prover acesso ao hardware, podendo assim realizar operações de leitura e escrita de dados.

Existem vários tipos de sistemas operacionais, sendo um deles o Sistema Operacional Embarcado (SOE) que é projetado para um hardware específico e com poucos recursos computacionais. Um SOE geralmente está presente em sistemas de automação industrial (BARAN; MAHAJAN, 2003), automação residencial (AL-ALI; AL-ROUSAN, 2004) e (SILVA, 2013), sistemas automotivos (NAVET et al., 2005) e até mesmo em equipamentos como leitores de DVD/CD, central de alarme e outros.

O Sistema Operacional de Tempo Real (SOTR) é outro tipo de SO. Um SOTR não precisa ser rápido, porém é preciso que o sistema possua um compromisso em executar tarefas respeitando um determinado limite de tempo. Existem dois tipos de SOTRs, os de tempo real crítico e os de tempo real não crítico. Na Seção 2.3 é descrito em detalhes a diferença entre esses dois tipos de SOTR.

2.2 COMPONENTES DE UM SISTEMA OPERACIONAL

Um SO possui serviços responsáveis por realizar chamadas de sistema que se comunicam com o *kernel*, tais como gerenciamento de processos, gerenciamento de memória e gerenciamento de armazenamento. Nas subseções seguintes são descritos os principais serviços suportados por um SO.

2.2.1 Kernel

O núcleo do SO, ou kernel, é responsável pelo gerenciamento de processos, memória, arquivos e os dispositivos de entrada e saída de dados. Para isso o kernel é a primeira parte de um SO a entrar em execução. Após o SO inicializar, o kernel garante que os aplicativos irão usufruir de memória e processador sem oferecer riscos a integridade da máquina.

A fiabilidade de todo SO, bem como seu desempenho, depende do kernel ser ou não confiável e bem estruturado. Um kernel para sistemas embarcados é diferente de um kernel de um sistema de propósito geral como Linux, Solaris e Windows, pois devem ser bastante simples e suas necessidades de gerenciamento são menos complexas, devido a quantidade de memória e a capacidade de processamento serem reduzidos (CRAIG, 2007).

O kernel constrói, a partir de uma máquina física, que pode conter um ou mais processadores, vários máquinas virtuais onde cada uma pode ser designada a um processo. Cada processo, dentro dos limites estabelecidos pelo SO, controla sua máquina virtual a fim de executar suas tarefas (JUNIOR, 2004).

O kernel é a parte do código mais crítica do SO, devido a responsabilidade de gerenciar da melhor maneira os recursos, fazendo com que os recursos fiquem ociosos o menor tempo possível. A Figura 2 ilustra a organização interna de um SO.

2.2.2 Gerenciamento de Processos

Um processo é a abstração de um programa em execução. Processos necessitam de recursos, tempo da unidade central de processamento (CPU), memória, arquivos e acesso aos dispositivos de entrada e saída. Para um processo executar, estes recursos são alocados tão logo

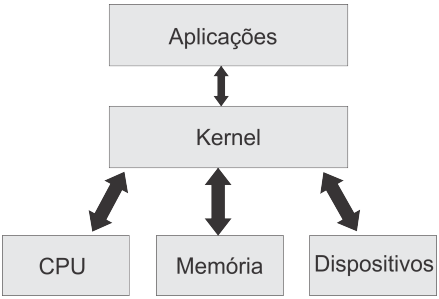

Figura 2 – Estrutura de um sistema operacional

o processo é criado ou durante sua execução (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Um programa de computador pode ser dividido em um conjunto de processos. Um SO possui processos criados por ele próprio e processos criados pelo usuário. Estes processos podem executar concorrentemente.

Atualmente os sistemas operacionais dão suporte a processos que possuam mais de uma thread. Thread é uma forma do processo se dividir em dois ou mais fluxos de execução que podem ser executado concorrentemente.

2.2.3 Gerenciamento de Memória

Para um programa executar em um sistema computacional é necessário que pelo menos uma parte deste programa esteja na memória principal. Para diminuir o tempo de resposta para o usuário e melhor organizar a utilização da CPU, o SO tende a manter alguns processos alocados na memória.

Para controlar os processos alocados na memória, existem vários algoritmos de gerenciamento, com abordagens diferentes, a eficácia de cada um depende da situação. A escolha do gerenciamento de memória depende de como o hardware foi projetado.

A função do gerenciador de memória é ter controle sobre os blocos de memória que estão em uso e quais não estão, e alocar e liberar memória quando os processos necessitarem de mais memória ou quando concluírem sua execução. Quando a memória principal não é suficiente para alocar todos os processos é preciso realizar o *swapping* entre a

memória principal e o disco rígido (memória virtual).

2.2.4 Gerenciamento de Armazenamento

O sistema de arquivos de um SO é o aspecto mais visível ao usuário final, possibilitando que o usuário organize seus arquivos em formas de pastas. Fornece um mecanismo para armazenamento de dados tanto do usuário como do próprio SO. O sistema de arquivos é dividido em uma coleção de arquivos e uma estrutura de diretórios ou pastas (SILBERSCHATZ; GALVIN; GAGNE, 2010).

A coleção de arquivos é mapeada pelo SO em dispositivos físicos. Estes dispositivos são do tipo não volátil, isto é, não ocorre perda de dados com falha de energia ou desligamento do sistema.

Para que um sistema consiga localizar e acessar arquivos no SO é preciso uma estrutura de diretórios que forneça e organize todos os arquivos que estão disponíveis no sistema. Cada arquivo possui algumas informações que podem ser: nome do arquivo, sua localização, tamanho, data e hora de criação, tipo etc.

O acesso a dados deve ser tratado com muito cuidado pelo SO, as aplicações fazem requisições de leitura e escrita de dados a quase todo momento e o SO deve garantir que a operação seja realizada da melhor maneira possível.

2.3 SISTEMAS OPERACIONAIS DE TEMPO REAL

Um SOTR é um SO de uso específico, ou seja, é utilizado em aplicações que além de necessitarem de uma respostas correta também necessitam que a tarefa seja executada dentro de um período de tempo estabelecido.

Os SOTRs estão presentes em alguns eletrodomésticos, tais como forno elétrico ou em um aparelho de DVD, também é possível identificar um SOTR em sistemas multimídias, como câmaras, e em aparelhos de comunicação. Estes sistemas também são encontrados em equipamentos maiores, como carros e aviões.

Existem dois tipos de SOTR:

- Sistema Operacional de Tempo Real Crítico: em um SOTR crítico é necessário que a tarefa execute no tempo previsto, por exemplo, o controlador de um freio ABS (*Anti-Lock Braking System*), onde a resposta deve ser imediata após o condutor do veículo pressio-

nar o pedal do freio. Caso a tarefa tenha sua execução atrasada, uma tragédia poderá acontecer.

- Sistema de Tempo Real Não Crítico: também são sistemas que tem o tempo como um parâmetro, mas a falha é aceitável, por exemplo, o leitor de um videogame é um sistema não-crítico, pois caso falhe não irá causar danos materiais ou até mesmo a perda de vidas humanas.

Um sistema de tempo real é classificado como previsível no domínio lógico e temporal quando, independente de variações que ocorram à nível de hardware, de carga e de falhas, o comportamento do sistema pode ser antecipado. Ou seja, é possível antecipar todos os prazos colocados a partir das interações com o ambiente que o sistema está inserido (OLIVEIRA; CARISSIMI; CARISSIMO, 2010).

Diferentemente de outros sistemas operacionais um SOTR possui algumas características típicas, tais como uso específico, na maioria dos casos seu tamanho é pequeno; produzido em massa e não dispendioso com requisitos específicos de tempo (SILBERSCHATZ; GALVIN; GAGNE, 2010).

2.4 SISTEMAS OPERACIONAIS EMBARCADOS

Os sistemas embarcados estão cada vez mais presentes em nosso dia a dia, mesmo que por vezes passem despercebidos. Aplicações como o controle de um portão eletrônico ou até mesmo o controle de um aparelho de ar condicionado em uma residência podem possuir um Sistema Embarcado (SE).

SEs são programados para aplicações específicas e podem executar em arquiteturas simples de 8 bits ou até mesmo em arquiteturas mais robustas com 32 ou 64 bits.

Geralmente aplicações embarcadas não necessitam de mecanismos complexos de proteção de memória e podem ser construídos em plataformas que não provém uma MMU (Unidade de Gerenciamento de Memória, do Inglês, *Memory Management Unit*) com os microcontroladores que podem ser baseados em arquiteturas RISC (Conjunto de Instruções Reduzido, do Inglês, *Reduced Instruction Set Computer*) ou CISC (Conjunto Complexo de Instruções, do Inglês, *Complex Instruction Set Computer*), trocando eficiência de pipeline por densidade de código (MARCONDES et al., 2006).

Um Sistema Operacional Embarcado (SOE) é utilizado quando

é preciso realizar uma atividade complexa, tendo assim que dividir o problema em tarefas específicas. Desta forma, o o programa é dividido em várias tarefas ou threads.

Além das tarefas do programa, o SOE também deve gerenciar as interrupções e tarefas do próprio SOE, o que faz com o processador pare de executar a tarefa corrente e desvie a execução para o tratador de interrupção. Após realizar o tratamento da interrupção, o processador retorna ao estado da tarefa antes da ocorrência da preempção. A Figura 3 ilustra um exemplo do fluxo do tratamento de interrupção.

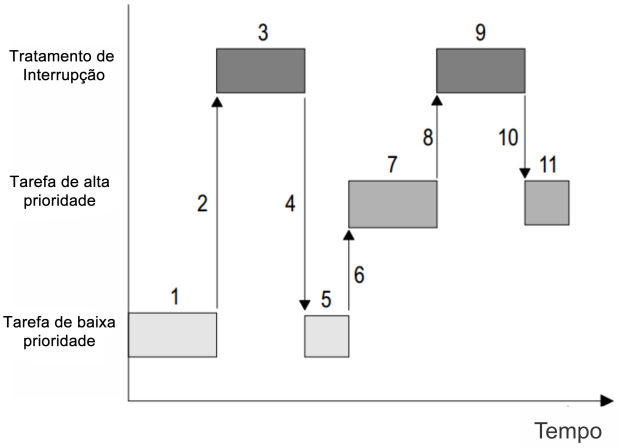

Figura 3 – Fluxo de tratamento de interrupções
Extraído e adaptado de: (PUMPKIN, 2010)

De acordo com o que é ilustrado na Figura 3, a tarefa (1) possui prioridade baixa e está executando, quando ocorre uma interrupção (2). A interrupção (3) é executada até (4), quando então a tarefa de baixa prioridade retoma sua execuçao (5). A mudança de contexto ocorre (6) quando a tarefa (7) de alta prioridade é executada. Essa mudança de contexto da tarefa é tratada pelo próprio programador, a tarefa de alta prioridade é interrompida (8-10) antes de continuar (11) (PUMPKIN, 2010).

Existem dois tipos de interrupções, as assíncronas que são causadas por alguma ação externa, por exemplo, em um computador do tipo desktop pode ser ocasionada pelo acionamento de um mouse ou teclado, no caso de um sistema embarcado pode ser por um comando via rádio;

e as síncronas que são causadas em consequência da instrução que está sendo executada, por exemplo, *overflow* [1] em operações aritméticas ou acesso ilegal a memória.

No projeto de um SOE é necessário avaliar diversos aspectos, como, tempo de resposta, consumo de energia, como o sistema irá se portar devido a pouca memória disponível entre outras funcionalidades.

Como todos os sistemas, os sistemas embarcados necessitam de algum tipo de entrada e saída, por exemplo em um sistema de um forno microondas, as entradas são os botões do painel frontal e sonda de temperatura e as saídas são a tela com os dados e a radiação de microondas. As entradas para o sistema geralmente são sensores, sinais de comunicação ou botões de controle. As saídas geralmente são monitores, sinais de comunicação ou alterações no mundo físico (SAGAR; AGARWAL, 2002).

Mesmo que cada SO embarcado seja projetado para uma atividade em particular, existem algumas características que os sistemas embarcados possuem em comum (VAHID; GIVARGIS, 1999). São elas:

- Simplicidade e funcionalidade: um sistema embarcado costuma executar somente um programa, repetidamente. Por exemplo, um pager é somente um pager. Diferente de um sistema para desktop que executa uma variedade de aplicativos.

- Condicionamento: todos os sistemas possuem restrições a métricas, mas os sistemas embarcados possuem uma quantidade de recursos limitada. É importante levar em consideração custo, tamanho, desempenho e potência.

- Reatividade e restrições temporais: muitos sistemas devem reagir conforme as mudanças ocorridas no ambiente em que estão, e deve calcular certos resultados em tempo real, sem atrasos.

A Tabela 1 lista as principais áreas onde os sistemas operacionais embarcados podem ser utilizados.

Nas seções 2.4.1 e 2.4.2 são descritos os SOEs MQX Lite e o FreeRTOS, respectivamente.

[1] Quando os dados resultantes de um processamento exigem mais espaço de armazenamento do que o fornecido no hardware ou no software de armazenamento de dados.

Tabela 1 – Exemplo de aplicações de sistemas embarcados

Extraído de: (HAMBLEN, 2007).

Área	**Exemplo de Aplicação**
Aeronaves e sistemas militares	Piloto automático, sistema de navegação, sistema de aterrisagem automática, sistema de orientação e controle dos motores.
Medicina	Tomografia computadorizada e sistemas de imagem de ultrassom.
Carros	Controle do motor, freio, sistema de tração, controle de airbag, controle do ar condicionado, GPS, rádio.
Comunicação	Satélites, roteadores de rede, switchs, hubs.
Eletrônicos de consumo	Televisor, DVD player, sistema de alarme, rádio, câmera, forno elétrico, máquina de lavar, controle de irrigação, calculadora, telefone celular, entre outros.
Dispositivos de entrada e saída do computador	Teclados, mouse, impressora, scanner, leitores e gravadores de DVD, placas de vídeo, monitores.
Equipamentos eletrônicos	Osciloscópio, multímetro, gerador de sinais.
Equipamentos industriais	Elevadores, sistema de vigilância, robôs, sistema de automação industrial.

2.4.1 MQX Lite

O MQX Lite é baseado no sistema operacional embarcado MQX (FREESCALE, 2013b) porém com recurso limitados. O MQX Lite possui uma quantidade de componentes menor, permitindo assim que aplicativos que consomem menos de 4 KB (*Kilobyte*) de RAM (*Random Access Memory*), sejam executados. O MQX Lite suporta componentes do *Process Expert* [2] (FREESCALE, 2010).

A Figura 4 ilustra a diferença entre os componentes suportados no MQX em relação ao MQX Lite.

MQX Lite se destaca por ser um SOE de tamanho reduzido e totalmente integrado com a plataforma Freedom Board KL25Z da Freescale. Mesmo sendo um SO embarcado de pequeno porte, ele oferece

[2] Sistema de desenvolvimento para criar, configurar, otimizar e migrar componentes de software para plataformas da Freescale.

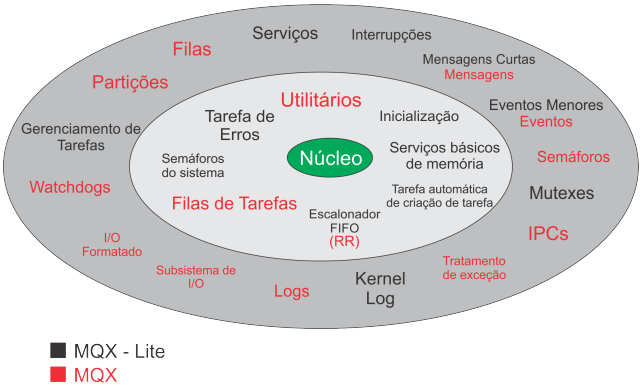

Figura 4 – Comparação entre os sistemas operacionais MQX e o MQX wLite.

Extraído e adaptado de: (FREESCALE, 2013a)

suporte as principais funções de um SOE, como utilização de semáforos e ao gerenciamento de processos.

O MQX Lite possui vários componentes, que são divididos em principais e opcionais. Os componentes principais são as funções do sistema 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.

O MQX Lite possui as seguintes características:

- Multitarefa preemptivo;
- Escalonador de tarefas baseado em prioridade;
- Sincronização de acesso a recursos, utilizando semáforos;
- Manipulação de interrupção.

No anexo A.1 são descritos as principais funções da API (*Application Programming Interface*) do sistema Operacional MQX Lite.

2.4.2 FreeRTOS

O FreeRTOS é um sistema operacional de tempo real, de código fonte aberto. Possui um núcleo pequeno e eficiente. O escalonador de tarefas do FreeRTOS é baseado em prioridades fixas, possui suporte a semáforos binários e filas de mensagens (AMINE; MOHAMED, 2013).

Atualmente o FreeRTOS é portável para 33 arquiteturas de microcontroladores de 8 a 32 bits, além de suportar várias ferramentas de desenvolvimento. Suas principais características são a portabilidade, escalabilidade e simplicidade (MISTRY, 2011).

Seu núcleo é totalmente escrito na linguagem de programação C, composto por quatro arquivos, que são facilmente portáveis e de fácil manutenção. O escalonador é baseado em prioridades fixa e suporta um número elevado de tarefas (INAM et al., 2011). A Figura 5 ilustra o diagrama do kernel do FreeRTOS.

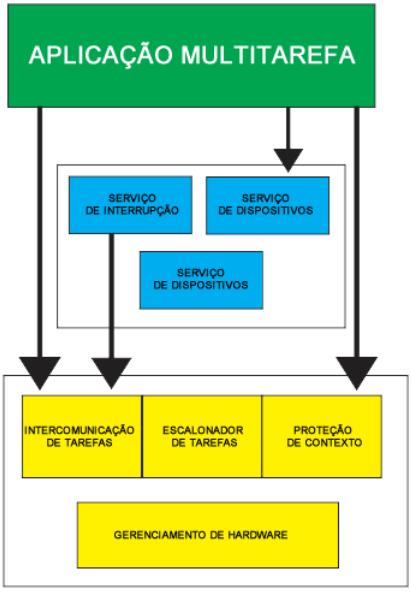

Figura 5 – Visão do kernel do sistema operacional FreeRTOS.
Extraído e adaptado de: (NIU et al., 2012)

Existem outras distribuições do FreeRTOS, um exemplo é o OpenRTOS, que possui um licenciamento diferente, a licença OpenR-

TOS retira a referência GPL[3] (Licença Pública Geral, do Inglês, *General Public License*) e outras implicações.

No anexo A.2 são descritas as principais funções da API (*Application Programming Interface*) do sistema operacional FreeRTOS.

[3]É um tipo de licença para software livre. Permite que os softwares sejam de código aberto garantindo que qualquer mudança seja liberada para todos os usuários.

3 ROBÓTICA MÓVEL

Neste capítulo é apresentado um breve histórico da robótica. São descritos os conceitos de sensores e atuadores, bem como as principais arquiteturas de controle utilizadas em robôs móveis.

3.1 BREVE HISTÓRICO DA ROBÓTICA

Há indícios que os primeiros robôs foram criados pelos gregos. Acredita-se que em Alexandria moradores criaram aparelhos para medir o tempo, a invenção mais famosa foi a Clepsidra (BRASIL, 2013), ou relógio de água, conforme a ilustração da Figura 6.

Figura 6 – Clepsidra ou relógio de água.
Extraído de: (BRASIL, 2013)

No século XVII, Jacques de Vaucanson construiu uma série de robôs mecânicos do tamanho de uma pessoa que eram capazes de tocar instrumentos musicais. A influência causada pela Revolução Industrial, que explodia na época, fez surgir uma série de invenções voltadas em sua maioria para a produção têxtil, como a fiandeira de fusos múltiplos de Hargreaves (1770), a máquina de fiar de Crompton (1779) e o tear mecânico de Cartwright (1785) (ZANELATTO, 2004).

Em 1950 o engenheiro Joseph F. Engelberger ficou conhecido como pai da robótica depois de fabricar o primeiro robô para industrialização, chamado Unimate (ilustrado na Figura 7).

Figura 7 – Imagem de um robô Unimate
Extraído de: (CHALLACOMBE et al., 2006)

Nos anos de 1951 e 1952 foram criadas duas tecnologias que contribuíram significativamente para o desenvolvimento da robótica. Uma delas foi o telecomando, que permitia controlar remotamente aparelhos utilizados para manusear materiais radioativos. A segunda, mais importante, desenvolvida no MIT (*Massachusetts Institute of Technology*) nos Estados Unidos da América, foi o comando numérico que controlava máquinas a partir de números. Na década 1960 foram construídos outros robôs importantes, resultado de uma união entre essas duas tecnologias(SICILIANO B.; KHATIB, 2008).

A NASA (*National Aeronautics and Space Administration*) buscava construir robôs para exploração de terrenos hostis, pois o envio de astronauta para outros planetas e o posterior retorno deles com segurança é uma tarefa muito complicada. Por este motivo sondas foram enviadas para o espaço, a fim de coletar dados (KURFESS, 2005).

Em 2003 a NASA iniciou uma missão chamada *Mars Exploration Rovers* (SQUYRES, 2008), com robôs exploradores idênticos, o *Spirit* e o *Opportunity*, que pousaram em janeiro de 2004 em Marte, a Figura 8 ilustra ambos os robôs. Os dois robôs pousaram em locais distintos em Marte com a intenção de localizar vestígios de água, além de fotografar o local para estudo e fazer análise de solo (KURFESS, 2005).

Hoje o robô explorador mais conhecido utilizado pela NASA é o robô *Curiosity* (Figura 9) enviado à Marte em 2012.

Para (BISHOP, 2009) um robô é uma máquina controlada por um computador. Com o avanço da tecnologia e o fácil acesso, empresas vem buscando construir robôs para fazer parte do dia-a-dia das pessoas, por exemplo para auxílio de pessoas com necessidades especiais ou com idade avançada (WALTERS et al., 2013). Com isso robôs passam

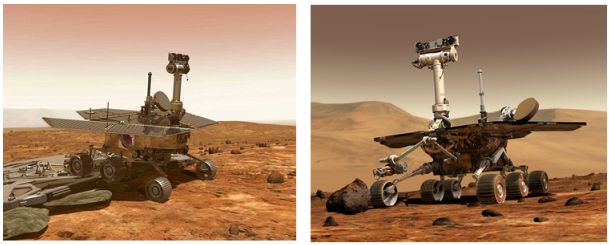

Figura 8 – Robôs Opportunity (a) e Spirit (b)
Extraído de: (NASA, 2013)

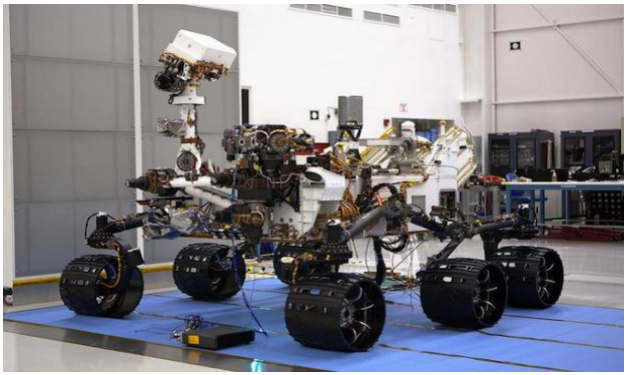

Figura 9 – Robô Curiosity
Extraído de: (NASA, 2013)

a ter características humanas, câmeras que simulam olhos e até mesmo braços robotizados.

Com o grande avanço da área da computação, passou-se a buscar cada vez mais tecnologias com maior recurso computacional, possibilitando que robôs menores fossem construídos com tecnologia de ponta e com maior capacidade de processamento, com isso a área de microcontroladores e sistemas operacionais embarcados tiveram um avanço significativo.

3.2 CICLO PERCEPÇÃO-AÇÃO EM ROBÓTICA MÓVEL

Um robô móvel pode obter informações do ambiente que está presente e a partir do seu conhecimento sobre determinado domínio realizar tarefas. A Figura 10 ilustra o ciclo de percepção e ação no realizado por um robô inserido em um ambiente qualquer.

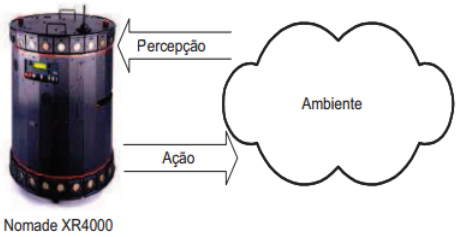

Figura 10 – Ciclo percepção-ação
Extraído de: (PEREZ, 2010)

Um Robô pode obter dados do ambiente das seguintes maneiras (RIBEIRO; COSTA; ROMERO, 2001):

- através de sensores;
- a partir de uma determinada ação a executar, eventualmente com o raciocínio para interpretar percepções e resolver problemas.
- executar ações a partir dos seus atuadores, modificando o ambiente e produzir novas situações.

Existem diversos tipos de robôs, cada um com suas especificidades, para cada robôs são necessários sensores e atuadores. Nas próximas subseções serão abordadas definições e tipos de sensores e atuadores.

3.2.1 Sensores

Existe um vasto número de sensores para área de robótica, com diferentes funcionalidades e diferentes tipos de interface. Os sensores são responsáveis por obter o maior número de informações possível do ambiente que possam ser úteis para o processamento.

É importante ter conhecimento do sensor que será utilizado na aplicação, conhecer seus requisitos e sua capacidade de funcionamento. Um exemplo é um sensor de temperatura de uma caldeira, é importante que o sensor utilizado suporte uma temperatura elevada, caso contrário, o sensor será danificado e poderá acontecer uma tragédia. A Figura 11 ilustra exemplos de sensores.

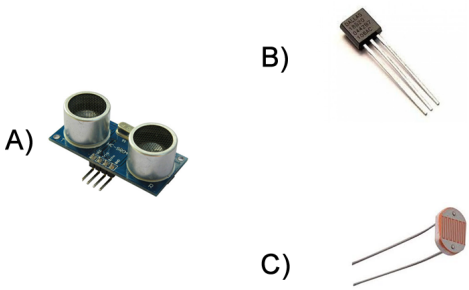

Figura 11 – Exemplos de sensores
A) Sensor Ultrassônico. B) Sensor de Temperatura. C) Sensor de Luminosidade

O dado que será transferido do sensor para a CPU (Unidade Central de Processamento, do Inglês, *Central Processing Unit*) pode ser tratado de duas maneiras, a primeira em que o dado é processado no próprio sensor com uma pequena central, e a outra é por meio de interrupção onde não existe tratamento de dado do sensor até a CPU (BRAUNL, 2008). A Tabela 2 lista alguns tipos de sensores e suas respectivas saídas.

3.2.2 Atuadores

Para que um robô possa interagir com o ambiente é preciso que este seja dotado de atuadores. Os atuadores podem responder a comandos elétricos, manuais ou mecânicos. O atuador mais comum é o motor, existem diferentes tipos de motores: motor de corrente contínua (DC, do Inglês *Direct Current*), motor de passo e servo motor.

Tabela 2 – Saída dos sensores

Adaptado de (BRAUNL, 2008).

Exemplo	**Saída do Sensor**
Sensor de Toque	Sinal Binário (0 ou 1)
Sensor de Luminosidade	Sinal Analógico (0 a 5V)
Giroscópio	Modulação por Largura de Pulso (PWM)
Módulo GPS	Serial Link (RS232 ou USB)
Câmera Digital	Paralelo

Os atuadores presentes em um robô são responsáveis por gerar força necessária para deslocar o sistema robótico. Devem ser capazes de acelerar e desacelerar o robô, suportando a quantidade de força exigida, e serem de fácil manutenção (NIKU, 2001).

Os motores DC são utilizados para que o robô consiga se deslocar de maneira mais rápida. São motores silenciosos, fácil de manusear e muitas vezes não consomem muita corrente. Os motores de passo, diferentemente dos motores DC, possuem duas ou mais bobinas que podem ser controladas de maneira independente (BRAUNL, 2008). A Figura 12 ilustra os tipos de motores.

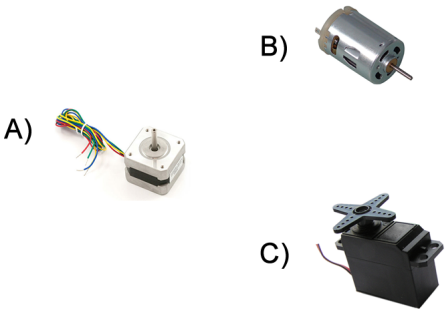

Figura 12 – Exemplos de motores
A) Motor de Passo. B) Motor DC. C) Servo Motor

Na Figura 13 é possível visualizar a presença de sensores e atua-

dores em um sistema robótico. Neste robô estão presentes: sensores de toque (a); sensor laser (b); sonar (c); motores (d).

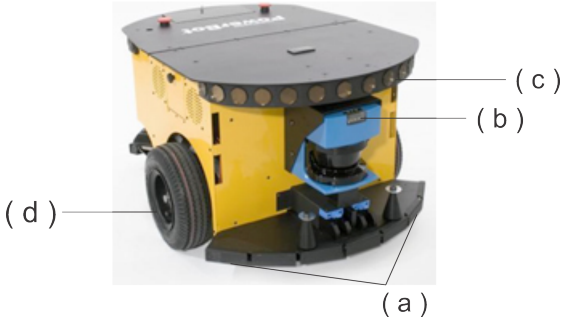

Figura 13 – Exemplo de um robô móvel com seus respectivos sensores e atuadores

Extraído e adaptado de: (ROBOTS, 2013)

3.3 SISTEMA DE CONTROLE DE ROBÔS

Para que os dados dos sensores sejam processados de maneira correta para determinar uma nova instrução aos atuadores é preciso que haja um sistema de controle. É importante que o sistema consiga processar todos os dados de maneira eficiente, de preferência, se adaptando ao ambiente quando houver modificações.

Para que o robô consiga interagir com o ambiente ele necessita de uma trajetória já planejada e então realizar o ciclo de percepção e ação, a partir da leitura dos sensores e manipulação os atuadores. A Figura 14 ilustra o diagrama de funcionamento de um sistema de controle.

O sistema de controle de um robô deve ser projeto conforme o problema ao qual se quer que o robô resolva. Nos casos em que o robô irá interagir com ambientes desconhecidos ou não estruturados é necessário que o programador possua conhecimento que não estão disponíveis no processo de desenvolvimento (ARKIN, 1998). Neste caso, é necessário que o robô tenha um alto grau de autonomia, para que o robô consiga tomar decisões em situações nas quais não foi programado (PEREZ, 2010).

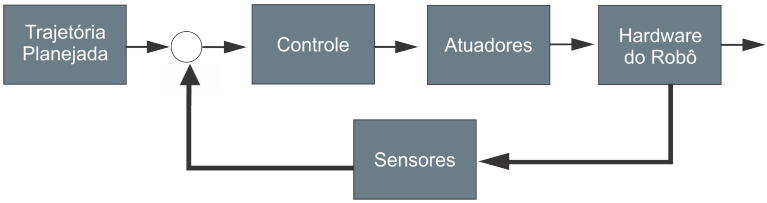

Figura 14 – Esquema de um sistema de controle para um robô
Adaptador de: (BAJD et al., 2010)

3.4 ARQUITETURAS DE CONTROLE PARA ROBÔS

A arquitetura de controle de um robô é responsável pela forma com que o robô irá se comportar no ambiente em que está interagindo. Este comportamento inclui técnicas para planejamento de trajetória, desvio de obstáculos, localização e interpretação de comandos.

Dentre as arquiteturas de controle para robôs destacam-se: a Reativa, a Deliberativa e a Híbrida. Nas seções 3.4.1, 3.4.2 e 3.4.3, respectivamente, essas arquiteturas são descritas em detalhes.

3.4.1 Arquitetura Reativa

Na arquitetura reativa o robô possui um sistema baseado em ação-reação, onde sensores obtém informações do ambiente, e a partir dessas informações é designada uma ou mais ações para os atuadores. O sistema de controle fica a cargo de atribuir a cada informação uma ação conveniente (KLIPP, 2013). Por exemplo, um robô que possua um sensor ultrassônico, caso detecte um objeto próximo a ele é enviado uma informação para o microcontrolador que então determina uma ação para os atuadores.

Na arquitetura reativa não se leva em conta o planejamento. A Figura 15 ilustra como é organizada a arquitetura reativa.

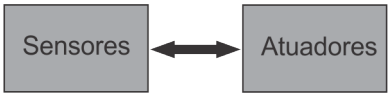

Figura 15 – Arquitetura Reativa

Como pode ser observado na Figura 15 os sensores representam a entrada de dados do ambiente e os atuadores as ações que serão realizadas (executadas) no ambiente. Nesta arquitetura não é preciso manter um histórico do ambiente, os sensores irão ler dados em tempo real e os atuadores irão responder.

3.4.2 Arquitetura Deliberativa

A arquitetura deliberativa, diferentemente da arquitetura reativa, possui um planejamento para realizar uma tarefa, este planejamento é feito por meio de raciocínio lógico, ou seja, é pré-definido. Um exemplo é um sistema especialista, onde todo processamento é baseado em regras. A Figura 16 ilustra a organização da arquitetura deliberativa.

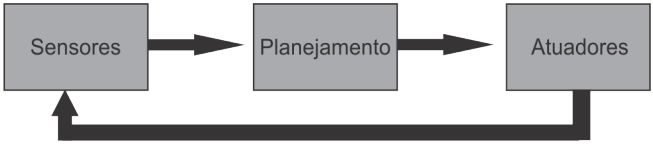

Figura 16 – Arquitetura Deliberativa

No módulo de planejamento pode ser adicionado alguma técnica de Inteligência Artificial em virtude da complexidade do problema.

3.4.3 Arquitetura Híbrida

A arquitetura híbrida combina elementos da arquitetura deliberativa e reativa. A arquitetura reativa é responsável pelos comandos que necessitam ser executados em um curto espaço de tempo e a deliberativa fica responsável pelo planejamento. Por exemplo, em uma situação onde o robô necessita chegar a um determinado local, a arquitetura deliberativa irá fazer o planejamento enquanto a reativa fica monitorando os sensores e os atuadores a fim de evitar que o robô se choque com algum obstáculo presente no ambiente (BRAUNL, 2008).

A Figura 17 ilustra a organização da arquitetura híbrida.

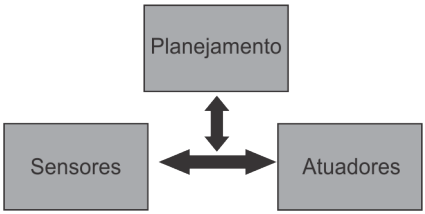

Figura 17 – Arquitetura Híbrida

4 APLICAÇÃO DOS SISTEMAS OPERACIONAIS EMBARCADOS MQX LITE E FREERTOS EM UM ROBÔ EXPLORADOR DE AMBIENTES

Neste capítulo apresenta-se a descrição do robô explorador, o qual foi utilizado para realizar os experimentos com os sistemas operacionais embarcados MQX Lite e FreeRTOS, bem como sua arquitetura de hardware e software.

4.1 DESCRIÇÃO DO ROBÔ EXPLORADOR

O robô explorador (PUNTEL et al., 2013) é composto por uma câmera do tipo webcam que é responsável pela captura de imagens do ambiente e 6 (seis) sensores de ultrassom, que são responsável pela identificação de objetos próximos ao robô.

A base mecânica do robô é uma caminhonete de controle remoto, onde estão instalados todos os sensores, atuadores e os microcontroladores. A Figura 18 ilustra uma imagem do robô explorador.

Figura 18 – Imagem do robô explorador

O robô possui dois motores, um motor traseiro DC (*Direct Current*), que é responsável pelo controle de velocidade e um motor de passo na dianteira que controla a direção do robô.

A Figura 19 ilustra o modelo cinemático do robô explorador,

onde é possível observar os deslocamentos dos motores traseiro e dianteiro e do servo motor, responsável pelo posicionamento nos eixos x e y da câmera.

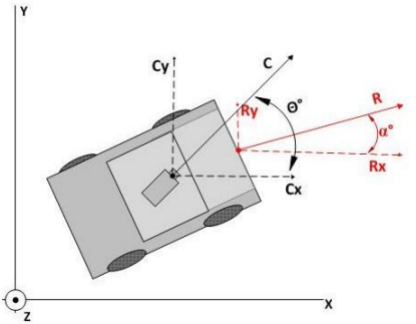

Figura 19 – Modelo cinemático do robô explorador

A navegação do robô é baseada nos dados obtidos através da visão e os dados recebidos dos sensores de ultrassom. O objetivo do sistema de visão é fazer com que o robô consiga seguir um determinado objeto. Nos próximos tópicos serão explicados a Arquitetura de Hardware (Subseção 4.2) e a Arquitetura de Software (Subseção 4.3) do robô explorador.

4.2 ARQUITETURA DE HARDWARE DO ROBÔ EXPLORADOR

A arquitetura de hardware foi dividida em dois módulos, o módulo reativo e o módulo deliberativo. O módulo deliberativo é responsável pelo processamento das imagens recebidas da câmera. O segundo módulo, que é o objetivo deste trabalho, é o módulo reativo, que é responsável pela leitura dos sensores e o controle dos atuadores e efetuadores do robô.

A Figura 20 ilustra a arquitetura de hardware do robô, a qual é possível observar a separação dos módulos reativo e deliberativo.

A seguir são descritos em detalhes os módulos Deliberativo e Reativo.

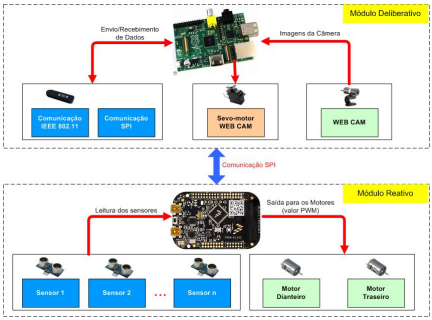

Figura 20 – Arquitetura de hardware do robô explorador

4.2.1 Módulo Deliberativo

O módulo deliberativo é onde acontece todo o processamento de imagem. Este módulo é composto por uma placa do tipo Raspberry Pi modelo B, que é responsável pela execução do sistema de processamento de imagens, o controle do servo motor da câmera e outras funcionalidades de alto nível. A Figura 21 ilustra Raspberry Pi modelo B.

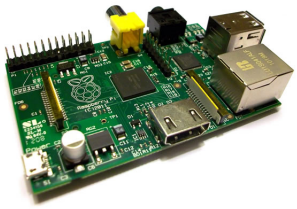

Figura 21 – Raspberry Pi modelo B

O Raspberry Pi é um computador de tamanho reduzido que possui um processador ARM (*Advanced RISC Machine*) 11 de 32 bits com clock de 700MHz, 512MB (*Megabytes*) de memória RAM, além de interfaces RCA (*Radio Corporation of America*), HDMI (*High-Definition Multimedia Interface*), Ethernet e uma interface para cartão do tipo SD (*Secure Digital*), possibilitando a instalação de um sistema operacional. A placa também possui portas de entrada e saída, GPIO (*General*

Purpose Input/Output).

4.2.2 Módulo Reativo

O módulo reativo é composto por uma placa do tipo Freedom Board, modelo KL25Z (ilustrada na Figura 22), onde serão executados os sistemas operacionais MQX Lite e FreeRTOS. Este módulo é responsável por todo o processamento de baixo nível, ou seja, a leitura dos sensores e envio de informações para os atuadores e efetuadores.

Figura 22 – Freedom Board modelo KL25Z

Para que o robô consiga movimentar-se sem colidir em nenhum objeto foram utilizados sensores de ultrassom, que detectam um objeto à uma distância máxima de 4,5 metros fazendo assim com que o robô consiga mudar a rota antes mesmo de colidir com este objeto.

O módulo reativo controla os dois motores presentes no robô, um motor de corrente contínua responsável tração do robô (velocidade), e um motor de passo responsável pela direção do robô.

É importante salientar que com a divisão ilustrada na Figura 20 é necessário que haja uma comunicação entre os dois módulos. A comunicação entre os módulos reativo e deliberativo é realizada via protocolo SPI (*Serial Peripheral Interface*), que é um protocolo fio a fio implementado nas portas de entrada e saída do Raspberry Pi e da Freedom Board.

4.3 ARQUITETURA DE SOFTWARE DO ROBÔ EXPLORADOR

Como descrito na arquitetura de hardware, o robô explorador foi dividido em duas partes, a parte de alto nível composta por uma

placa do tipo Raspberry Pi modelo B (Módulo Deliberativo) e a parte de baixo nível composta por uma placa do tipo Freedom Board da Freescale (Módulo Reativo). Desta forma o sistema de controle também foi separado, conforme ilustra a Figura 23.

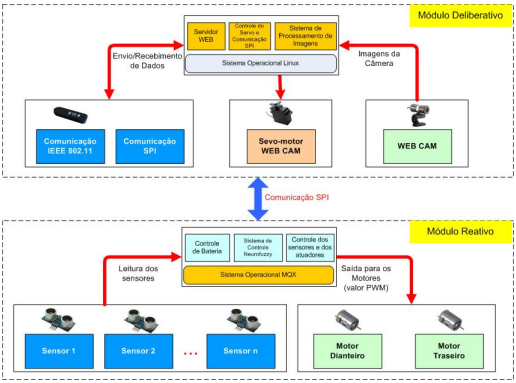

Figura 23 – Arquitetura de software do robô explorador

Na próxima subseção será descrito o módulo Reativo, devido a este se tratar da parte onde serão avaliados os sistemas operacionais MQX Lite e FreeRTOS, objetivos deste trabalho.

4.3.1 Módulo Reativo

O testes com o módulo reativo foram realizados com o sistema operacional MQX Lite e FreeRTOS. Devido a complexidade deste módulo ele foi dividido em três tarefas:

- Controle neurofuzzy: baseado em uma rede neural do tipo Perceptron de Múltiplas Camadas com o ajuste de sua saída baseado em um sistema Fuzzy. Esta tarefa irá receber dados via protocolo SPI do módulo Deliberativo, bem como informações das outras duas tarefas “controle de bateria” e “controle dos sensores e atuadores”.

- Controle de bateria: é responsável por monitorar a carga da bateria do robô, dependendo da carga da bateria, algumas funcionalidades do robô poderão ser desativadas.

- Controle dos sensores e atuadores: é responsável por ler dados dos sensores ultrassom e repassar para o controle neurofuzzy, que irá processar e gerar uma saída para os atuadores. Que os dados resultantes serão a direção e a velocidade do robô.

Das tarefas listadas, somente as tarefas responsáveis pelos sensores e atuadores foram implementadas neste trabalho, nos SOEs FreeRTOS e MQX Lite, e os resultados da avaliação de ambos os sistemas são descritos no próximo capítulo.

5 AVALIAÇÃO DOS SISTEMAS OPERACIONAIS EMBARCADOS MQX LITE E FREERTOS

Neste capítulo são apresentados os resultados da avaliação dos sistemas operacionais MQX Lite e FreeRTOS, no gerenciamento de duas tarefas do módulo Reativo do robô explorador.

5.1 DESCRIÇÃO DOS EXPERIMENTOS REALIZADOS

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. Os experimentos realizados foram: entrada das tarefas em execução (Subseção 5.1.1); execução das tarefas manipulando os atuadores (Subseção 5.1.2); tempo relacionado a troca de contexto (Subseção 5.1.3), e tempo gasto na execução de uma seção crítica (Subseção 5.1.4).

Em todos os experimentos foi atribuída uma prioridade padrão para as tarefas, ou seja, 8 (oito) no MQX Lite e 0 (zero) no FreeRTOS. Os gráficos apresentados em cada experimento foram obtidos com o auxílio de um osciloscópio.

5.1.1 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 24 e na Figura 25 são ilustrados os comportamentos dos dois sistemas operacionais, respectivamente.

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. A atividade das tarefas é sincronizada com o uso de um semáforo binário, onde a tarefa sensores obtém o semáforo, faz a leitura dos sensores dianteiros e libera o semáforo para que a tarefa dos motores possa executar. Neste experimento a tarefa dos motores apenas faz o teste dos valores obtidos pelos sensores.

Ao final da execuçã cada tarefa invoca a chamada de sistema *yield()* para liberar o processador para que a outra tarefa possa entrar em execução. Para melhor visualização foi necessário adicionar um *delay* de 150 ms (millisegundos) em cada uma das tarefas.

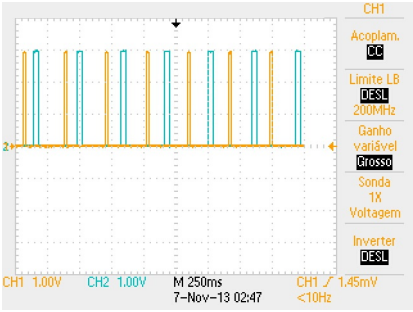

Figura 24 – Resultado obtido com o sistema operacional MQX Lite

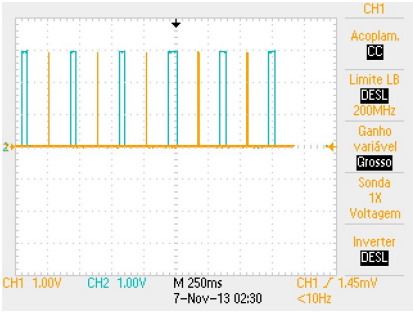

Figura 25 – Resultado obtido com o sistema operacional FreeRTOS

Com os resultados obtidos é possível perceber que ambos os sistemas operacionais obtiveram desempenho semelhante.

5.1.2 Experimento 2 - Execução das Tarefas Manipulando os Atuadores

O objetivo deste experimento é avaliar o desempenho das tarefas na manipulação dos atuadores. Na Figura 26 e na Figura 24 são apresentados os gráficos de desempenho dos dois SOEs.

Para avaliação foi utilizado os sensores dianteiros e traseiros. Neste caso quando um sensor dianteiro acusasse a presença de um objeto a menos de 35 cm, imediatamente o sentido do motor traseiro era alterado, e então era dado inicio ao monitoramento dos sensores trasei-

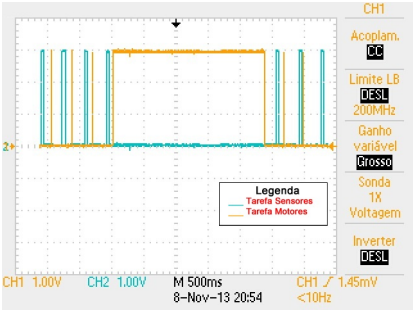

Figura 26 – Resultado obtido com o sistema operacional FreeRTOS na manipulação dos atuadores

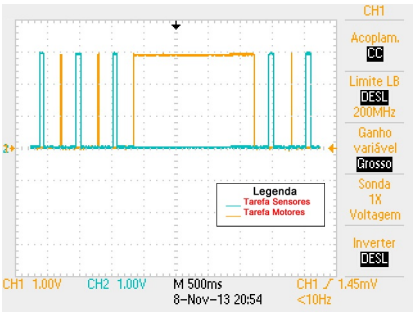

Figura 27 – Resultado obtido com o sistema operacional MQX Lite na manipulação dos atuadores

ros, até que os sensores dianteiros acusassem uma distância considerável do objeto para que o robô pudesse ir novamente para a frente.

Neste experimento é possível verificar que o sistema MQX Lite realizou a operação de mudar o sentido do motor traseiro de maneira mais eficiente enquanto o FreeRTOS teve melhor desempenho na leitura dos sensores.

Vale ressaltar que mesmo o robô possuindo 6 sensores (3 dianteiros, 3 traseiros) neste experimento foram utilizados somente um sensor dianteiro e um sensor traseiro.

5.1.3 Experimento 3 - Tempo relacionado a troca de contexto

O processador pode ser ocupado por uma tarefa por vez. Para que todas as tarefas entrem em execução o escalonador deve realizar a troca de contexto, selecionando a próxima tarefa que irá ocupar o processador.

Para que os dados de cada tarefa não sejam perdidos, cada tarefa é representada po uma PCB (Process Control Block), que geralmente contém as seguintes informações:

- Identificador do processo;
- Prioridade do processo;
- Estado do processo;
- Ponteiro para a próxima PCB;

Para selecionar a próxima tarefa da fila de aptos para ser executada, o escalonador precisa realizar uma interrupção, salvar os dados da PCB e retirar o processo que está em execução, restaurar os dados da tarefa que entrará em execução. O tempo em que o processador substitui as tarefas é chamado de latência de despacho. A Figura 28 ilustra como é realizado a troca de contexto.

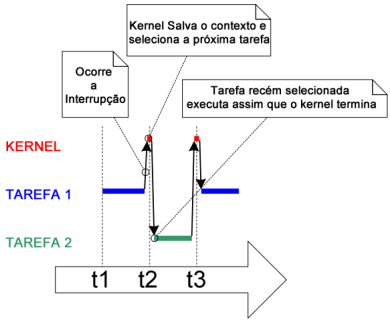

Figura 28 – Troca de Contexto
Extraído e adaptado de: (BARRY, 2007)

Neste experimento serão descritos e avaliados a troca de contexto dos SORT em questão. A Figura 29 ilustra a troca de contexto no FreeRTOS.

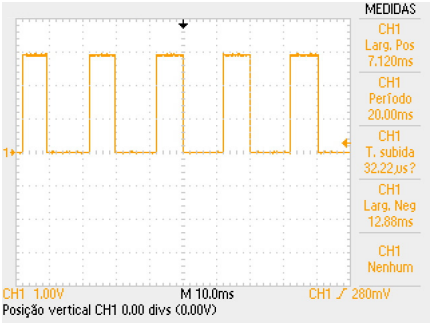

Figura 29 – Troca de Contexto no FreeRTOS

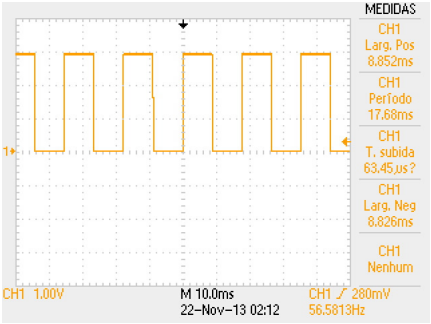

Figura 30 – Troca de Contexto no FreeRTOS

A Figura 30 ilustra a troca de contexto no SOE MQX Lite.

Após concluído o experimento constatou-se que o SOE FreeRTOS, realizou a troca de contexto em 7.120 ms (milisegundos) e teve um desempenho superior ao MQX Lite, que realizou a troca de contexto em 8.852 ms. Mesmo com um desempenho inferior, o MQX Lite realizou a troca de contexto em um período satisfatório.

5.1.4 Experimento 4 - Tempo gasto na execução de uma seção crítica

Uma seção crítica pode ser definida como uma área de memória ou arquivo que tenha o acesso compartilhado entre dois ou mais proces-

sos/tarefas. O problema do produtor e o consumidor é um exemplo de seção crítica, onde o produtor escreve dados em um buffer e o produtor lê dados desse buffer. A Figura 31 ilustra um exemplo do produtor e do consumidor.

Figura 31 – Exemplo de produtor e consumidor

Como ilustrado na Figura 31, a sessão crítica pode ser representado por uma variável compartilhada, por exemplo:

```
1  int dado;
```

O processo 1 (P1) realiza uma operação de escrita:

```
1  dado = 5;
```

E o processor 2 (P2) realiza uma operação de leitura:

Antes de realizar qualquer operação na seção crítica o processo precisa solicitar permissão para acessá-la. Quando um processo está manipulando o dado na seção crítica, nenhum outro processo pode realizar qualquer operação sobre aquela seção crítica. Para garantir a exclusão mútua é possível utilizar um mecanismo de sincronização de processos chamado semáforo.

Ambos os sistemas operacionais embarcados utilizados nos experimentos dão suporte a manipulação de semáforos.

Neste experimento, a tarefa sensores realiza a leitura dos 3 (três) sensores frontal da caminhonete e grava os dados lidos em três variáveis. A tarefa motores faz a verificação destes valores.

A seção crítica são as 3 (três) variáveis que representam os três sensores, o processo 1, é o processo de sensores que realiza a operação de escrita e o processo 2 é o processo dos motores que realiza a operação de leitura dos valores. As Figuras 32 e Figura 33 ilustram os resultados do experimento com os sistemas FreeRTOS e MQX Lite em execução, respectivamente.

É possível perceber que o MQX Lite demorou em média 22.40 ms para executar a tarefa de leitura dos sensores e 3.6 ms para executar

```
1    int dado2;
2    dado2 = dado;
```

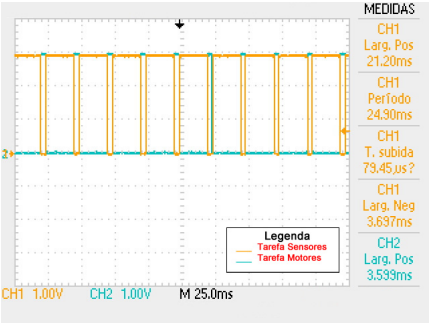

Figura 32 – Tempo que cada tarefa ficou na seção crítica no FreeRTOS

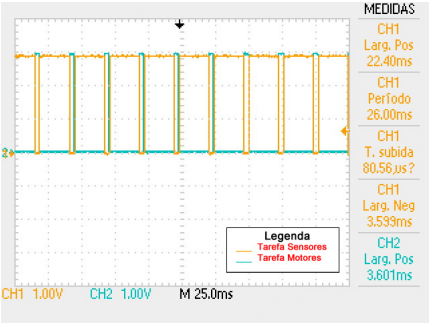

Figura 33 – Tempo que cada tarefa ficou na seção crítica no MQX Lite

o teste dos valores na tarefa motores, enquanto o FreeRTOS demorou em média 21.20 ms e aproximadamente 3.6 ms para executar o teste dos valores na tarefa motores.

5.2 CONSIDERAÇÕES SOBRE OS EXPERIMENTOS COM O MQX LITE E O FREERTOS

A implementação em ambos os SOEs foi de maneira muito semelhante. Destaque para o MQX Lite, onde as configurações das tarefas e do SOE pode ser feita de forma mais fácil com o auxílio de uma interface de configurações.

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

Com os resultados obtidos é possível observar que o uso de um SOE foi fundamental para organização do código do sistema de controle do robô, devido a facilidade na criação e no desenvolvimento das tarefas.

Com os resultados obtidos, é possível observar que o FreeRTOS teve um desempenho superior ao MQX Lite nos experimentos realizados. Mesmo com um tempo superior nos experimentos o MQX Lite se portou de maneira satisfatória, executando dentro de um prazo aceitável.

6 CONSIDERAÇÕES FINAIS E PROPOSTAS PARA TRABALHOS FUTUROS

O uso da robótica em diversas áreas tem trazido benefícios e na maioria das vezes, os robôs realizam tarefas perigosas e repetitivas para o ser humano. Entretanto, com atividades cada vez mais complexas o uso de sistemas operacionais embarcados permite um melhor gerenciamento dos recursos do robô, facilitando assim o controle do hardware.

Afim de comprovar e avaliar os benefícios do uso de um sistema operacional embarcado, este trabalho apresentou uma avaliação dos SOEs MQX Lite e FreeRTOS em aplicações de robótica móvel. Para realizar a avaliação foram realizados alguns experimentos com ambos os SOEs.

Para avaliar a eficiência e a confiabilidade dos sistemas operacionais embarcados foi construído um robô explorador, contendo dois motores e seis sensores ultrassônicos. A partir do uso de sensores é possível identificar obstáculos e com os atuadores desviar o percurso para que o robô não colida. Assim o sistema foi dividido em duas tarefas, uma para os sensores e outra para os motores.

Com os resultados dos experimentos constatou-se que ambos os sistemas se portaram de maneira satisfatória. Mesmo com informações recebida de 6 (seis) sensores, o microprocessador do robô respondeu a tempo suficiente para executar a leitura dos sensores e o controle dos 2 (dois) atuadores em ambos os SOEs. Fazendo assim com o que robô conseguisse se adaptar as mudanças no ambiente.

Por fim, concluiu-se que em algumas aplicações, como as de robótica móvel, o uso de sistemas operacionais embarcados é indispensável, devido a facilidade para gerenciamento de recursos e controle sobre o hardware.

6.1 PROPOSTA PARA TRABALHOS FUTUROS

Nesta seção são listadas algumas propostas para trabalhos futuros:

1. Fazer a integração com o módulo Deliberativo do robô explorador e fazer com que o robô navegue no ambiente conforme os dados recebidos do sistema de processamento de imagens;

2. Adicionar outros tipos de sensores e utilizar o acelerômetro pre-

sente na Freedom Board para a realização de outros experimentos;

3. Realizar os experimentos com e sem SOE na Freedom Board e compara os resultados;
4. Realizar experimentos em um ambiente outdoor.

REFERÊNCIAS

AL-ALI, A.; AL-ROUSAN, M. Java-based home automation system. *Consumer Electronics, IEEE Transactions on*, v. 50, n. 2, p. 498–504, 2004. ISSN 0098-3063.

AL-JARRAH, R.; ROTH, H. Design blimp robot based on embedded system and software architecture with high level communication and fuzzy logic. *Mechatronics and its Applications (ISMA), 2013 9th International Symposium on*, v. 1, n. 1, p. 1–6, 2013.

AMINE, C. M. E.; MOHAMED, O. A localization and an identification system of personnel in areas at risk using a wireless sensor networ. *Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 International Conference on IEEE*, v. 5, 2013.

ARKIN, R. C. *Behavior-based robotics [electronic resource]*. [S.l.]: MIT press, 1998.

BAJD, T. et al. *Robotics (Intelligent Systems, Control and Automation: Science and Engineering)*. Springer. [S.l.]: Springer, 2010. 152 p.

BARAN, M.; MAHAJAN, N. Dc distribution for industrial systems: opportunities and challenges. *Industry Applications, IEEE Transactions on*, v. 39, n. 6, p. 1596–1601, 2003. ISSN 0093-9994.

BARRY, R. *FreeRTOS User Manual*. 1. ed. [S.l.], 2007.

BISHOP, R. C. D. R. H. *Sistemas de Controle Modernos*. [S.l.: s.n.], 2009.

BRASIL, R. Tipos de relógios. p. Disponível em: ¡http://professor–rogerio.blogspot.com.br/2011/03/tipos–de–relogios.html¿ Acesso em: 10 maio 2013., 2013.

BRAUNL, T. *Embedded robotics: mobile robot design and applications with embedded systems*. [S.l.]: Springer, 2008. 541 p. (1, 1).

CHALLACOMBE, B. J. et al. The history of robotics in urology. *World journal of urology*, Springer, v. 24, n. 2, p. 120–127, 2006.

CRAIG, I. *Formal models of operating systems kernels*. [S.l.]: Springer, 2007.

ENTSFELLNER, K. et al. Development of universal gripping adapters: Sterile coupling of medical devices and robots using robotic fingers. In: *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*. [S.l.: s.n.], 2013. p. 1464–1469. ISSN 2159-6247.

FREESCALE. *Processor Expert and Device Initialization - User Guide*. 1. ed. [S.l.], 2010.

FREESCALE. Freescale mqx real-time operating system - users guide. Acesso em: 06, set de 2013, 2013.

FREESCALE. *Freescale MQX RTOS - Reference Manual*. 1. ed. [S.l.], 08 2013.

HAMBLEN, J.; BEKKUM, G. van. An embedded systems laboratory to support rapid prototyping of robotics and the internet of things. *Education, IEEE Transactions on*, v. 56, n. 1, p. 121–128, 2013. ISSN 0018-9359.

HAMBLEN, J. O. A tutorial approach with laboratory exercises for windows embedded ce 6.0 and the ebox 2300 soc embedded computer. *Georgia: Georgiainstitute Oftechnology*, 2007.

INAM, R. et al. Support for hierarchical scheduling in freertos. *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, p. 10, 2011.

JUNIOR, P. J. *Notas sobre Sistemas Operacionais*. [S.l.]: Apostila, 2004.

KLIPP, T. dos S. Proposta de uma arquitetura para alocação de tarefas em grupos de robôs móveis baseada em acordo bizantino. *Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina*, v. 1, p. 73, 2013.

KURFESS, T. R. *Robotics amd automation handbook*. [S.l.: s.n.], 2005.

LICENSE, G. G. P. Disponível em:¡ http://www. gnu. org/licenses/gpl. html¿. *Acesso em: 13 de novembro*, 2013.

LUO, J.; JHA, N. K. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In: IEEE PRESS. *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*. [S.l.], 2000. p. 357–364.

MARCONDES, H. et al. Epos: Um sistema operacional portável para sistemas profundamente embarcados. *WORKSHOP DE SISTEMAS OPERACIONAIS, Campo Grande*, v. 3, p. 31 – 45, 2006.

MISTRY, J. *FreeRTOS and multicore*. Tese (Doutorado) — MSc Thesis, 2011.

NASA. *www.nasa.gov*, Acesso em: 23 de ago., 2013.

NAVET, N. et al. Trends in automotive communication systems. *Proceedings of the IEEE*, v. 93, n. 6, p. 1204–1223, 2005. ISSN 0018-9219.

NIKU, S. B. *Introduction to robotics: analysis, systems, applications*. [S.l.]: Prentice Hall New Jersey, 2001.

NIU, S. et al. Analysis and implementation of migrating real-time embedded operating system freertos kernel based on s3c44b0 processor. In: *Information Science and Engineering (ISISE), 2012 International Symposium on*. [S.l.: s.n.], 2012. p. 430–433. ISSN 2160-1283.

OLIVEIRA, R. S. de; CARISSIMI, A. da S.; CARISSIMO, A. da S. *Sistemas Operacionais*. [S.l.]: 4º Edição, 2010.

PEREZ, A. L. F. *Extensão da Programação Genética Distribuída para Suportar, Algoritmo da Programaçao Genética*. Tese (Doutorado) — Programa de Pós-Graduação em Engenharia Elétrica - Universidade Federal de Santa Catarina, 2010.

PEREZ, A. L. F. et al. Uso da plataforma arduino para o ensino e o aprendizado de robótica. *ICBL2013: International Conference on Interactive Computer aided Blended Learning*, 2013.

PUMPKIN. *Salvo - The RTOS that runs in tiny places*. [S.l.]: Disponível em: http://www.pumpkininc.com/content/doc/manual/SalvoUserManual.pdf, 2010.

PUNTEL, F. E. et al. Desenvolvimento de um robô explorador para ambientes indoor. *Revista tecnico científica do IF-SC*, v. 3, p. 650–656, 2013.

RIBEIRO, C.; COSTA, A.; ROMERO, R. Robôs móveis inteligentes: Principios e técnicas. In: *Anais do XXI Congresso da Sociedade Brasileira de Computaçao-SBC?2001*. [S.l.: s.n.], 2001. v. 3, p. 258–306.

ROBOTS, A. M. http://www.mobilerobots.com/researchrobots/powerbot.aspx. Acesso em: 28, ago de 2013., 2013.

SAGAR, P.; AGARWAL, V. Embedded operating systems for real-time applications. *Electronic Systems Group, EE Dept, IIT Bombay,*, v. 1, p. 14, 2002.

SICILIANO B.; KHATIB, O. Springer handbook of robotics. heidelberg: Springer-verlag. 2008.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de Sistemas Operacionais*. [S.l.: s.n.], 2010.

SILVA, E. G. da. Desenvolvimento de um sistpar para adaptação de contexto em automação residencial. *Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina*, v. 1, p. 87, 2013.

SQUYRES, S. W. Mars exploration rovers. In: *Bulletin of the American Astronomical Society*. [S.l.: s.n.], 2008. v. 40, p. 500.

VAHID, F.; GIVARGIS, T. Embedded system design: A unified hardware/software approach. *University of California, USA*, v. 103, 1999.

VAISHAK, N.; SHILPA, B. Development of advanced risc microprocessor based embedded system with embedded operating system for robot applications. *Development*, v. 1, n. 1, 2013.

WALTERS, M. et al. Companion robots for elderly people: Using theatre to investigate potential users' views. In: *RO-MAN, 2013 IEEE*. [S.l.: s.n.], 2013. p. 691–696. ISSN 1944-9445.

ZANELATTO, M. S. Robotica educacional nos cursos de ciências da computação. *Trabalho de Conclusão de Curso - Universidade Estadual de Londrina*, p. 80, 2004.

ZANOTTO, D. et al. Effects of complementary auditory feedback in robot-assisted lower extremity motor adaptation. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, v. 21, n. 5, p. 775–786, 2013. ISSN 1534-4320.

ANEXO A – Sistemas Operacionais MQX Lite e FreeRTOS

A.1 MQX LITE

O MQX foi criado pela Dy4 System, uma empresa canadense que foi adquirida pela ARC International em 2000.

O MQX Lite é formado por componentes principais e opcionais. Os principais estão presente em todas as imagens de execução geradas, já os componentes opcionais só serão adicionados conforme necessidade do problema. A Tabela 3 lista todos os componentes e suas características.

Tabela 3 – Componentes do sistema MQX Lite

Extraído de: (FREESCALE, 2013a)

Componente MQX-Lite	**Descrição**	**Tipo**
Inicialização	Inicialização e criação automática da tarefa	Principal
Gerenciamento de tarefas	Criação, gerenciamento e exclusão de tarefas	Principal
Escalonador	Escalonador FIFO	Principal
Semáforo Leve	Semáforos leve e mecanismo de sincronização	Principal
Evento Leve	mecanismo de sincronização de eventos	Opcional
Mutex	Exclusão mútua e mecanismo de sincronização	Opcional
Fila de mensagens leve	Comunicação entre tarefas	Opcional
Interrupção e manipulação de exceção	Manutenção de todas as interrupções	Opcional
Temporizador leve	Mecanismo para chamar funções da aplicação em intervalos periódicos	Opcional
Log do kernel	Atividade do MQX-Lite	Opcional

A.1.1 Descrição dos Componentes

Nas próximas seções os componentes listados na tabela 3 serão explicados de forma mais detalhada.

A.1.1.1 Inicialização

O componente Inicialização é um componente principal. É dividido em duas funções:

```
1 _mqxlite_init():
```

Esta função é responsável por inicializar os dados que o MQX Lite irá usar internamente, incluindo os dados do núcleo, filas de aptos e a pilha de interrupção e também criar um semáforo leve para as necessidades de sincronização internos.

```
1 _mqxlite():
```

Esta função deve ser chamado pelo usuário na aplicação, a partir da chamada:

```
1 PE_low_level_init()
```

Cria a tarefa de espera e iniciar o escalonador de tarefas.

A.1.1.2 Gerenciamento de Tarefas

O MQX Lite não suprota a criação dinâmica de tarefas, por isso os recursos devem ser pré-alocados em tempo de compilação. Apenas uma tarefa pode ocupar o processador por vez, a prioridade da tarefa pode ser alterada em tempo de execução. Para cada tarefa é possível determinar uma função de saída e uma exceção que o MQX Lite chama quando a tarefa concluí ou quando gera uma exceção.

A.1.1.3 Escalonador de Tarefas

O escalonador é utilizado para dividir o tempo em que as tarefas utilizam o processador. Apenas uma tarefa está ativa em um determinado momento. A política de escalonamento utilizada é FIFO (*First*

in first out) (baseada em prioridade). A tarefa que está utilizando o processador é a de maior prioridade, outra tarefa somente obtém o processador quando uma das seguintes situações ocorrerem (FREESCALE, 2013a):

- A tarefa ativa abandona voluntariamente o processador;
- Uma interrupção ocorre;
- Uma tarefa que tem prioridade maior que a tarefa que está ativa fica pronta.

Cada tarefa está em um dos estados lógicos abaixo:

- Bloqueada: a tarefa ainda não está pronta para executar, pois está esperando que uma condição ocorra, quando isso ocorrer, ela estará pronta;
- Pronto: a tarefa está pronta para tornar-se ativa, mas não está ativa, porque possui prioridade igual ou menor que a tarefa que está ativa;
- Ativa: tarefa em execução;
- Encerrada: tarefa foi destruída, pois já executou o que tempo que precisava.

Para cada tipo de prioridade existe um fila com as tarefas prontas. A Figura 34 ilustra o diagrama de estados das tarefas.

A.1.1.4 Semáforo Leve

Semáforo leve é utilizado para sincronizar o acesso a recursos compartilhados entre tarefas. Semáforos leves exigem uma quantidade mínima de memória e executam rapidamente. Esse tipo de semáforo pode ser utilizado para:

- Controlar o acesso a recursos compartilhados (exclusão mútua);
- Sinalizar a ocorrência de um evento;
- Permitir que duas tarefas sincronizem suas atividades.

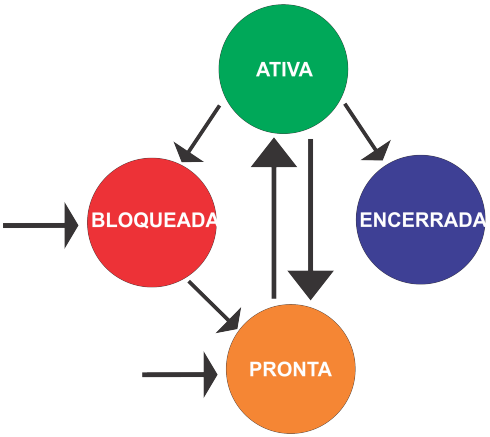

Figura 34 – Diagrama de transição estados das tarefas no sistema operacional MQX Lite
Extraído e Adaptado de: (FREESCALE, 2013a)

A.1.1.5 Evento Leve

Um evento leve pode ser usado para sincronizar tarefas ou sincronizar tarefas com uma ISR (*Interrupt Service Routine*). Os componentes de eventos são um agrupamento de bits. Qualquer tarefa pode esperar um evento que poderá ser gerado por uma outra tarefa ou pelo próprio sistema operacional.

A.1.1.6 Mutex

Um mutex fornece exclusão mútua entre as tarefas, quando acessar um recurso compartilhado, tais como um dispositivo. Semáforos, são estritos, isso é uma tarefa não pode desbloquear um Mutex.

A.1.1.7 Fila de Mensagens Leve

A troca de mensagens é a melhor forma das tarefas estabelecerem comunicação entre si. Mensagens leves são mais simples, de custo computacional menor comparado com o sistema de mensagens imple-

mentadas MQX. Uma mensagem deve ter tamanho fixo, ou seja um valor múltiplo de 32 bits.

A.1.1.8 Interrupção

O MQX Lite controla todas as interrupções de hardware. Proporciona um primeiro nível de ISR (kernel ISR), que é escrito em linguagem de programação assembly. Este ISR é executado antes de qualquer outro ISR e é responsável pelas seguintes tarefas:

- Conservar o contexto da tarefa ativa;
- Controlar a pilha de execução;
- Chamar a aplicação apropriada;
- Rastaurar o contexto da tarefa pronta com maior prioridade.

A.1.1.9 Temporizador Leve

Componente opcional que proporciona comunicações periódicas com a aplicação. Quando uma fila é criada um temporizador é criado junto e é expirado a partir de um determinado momento. É utilizado para que funções atualizem seu tempo.

A.1.1.10 Log do Kernel

É um componente opcional que permite a gravação das atividades do MQX Lite, pode ser configurado para gravar todas as chamadas de sistemas, tais como, mudança de contexto e de serviço de interrupção.

A.1.2 Manipulação de Tarefas

As próximas seções descrevem como projetar e manipular de tarefas no MQX Lite.

A.1.2.1 Criação de Tarefas

Para utilização do sistema operacional embarcado MQX Lite é possível utilizar um IDE fornecido pela própria Freescale, o CodeWarrior. A Figura 35 ilustra os passos para criação de tarefas neste ambiente.

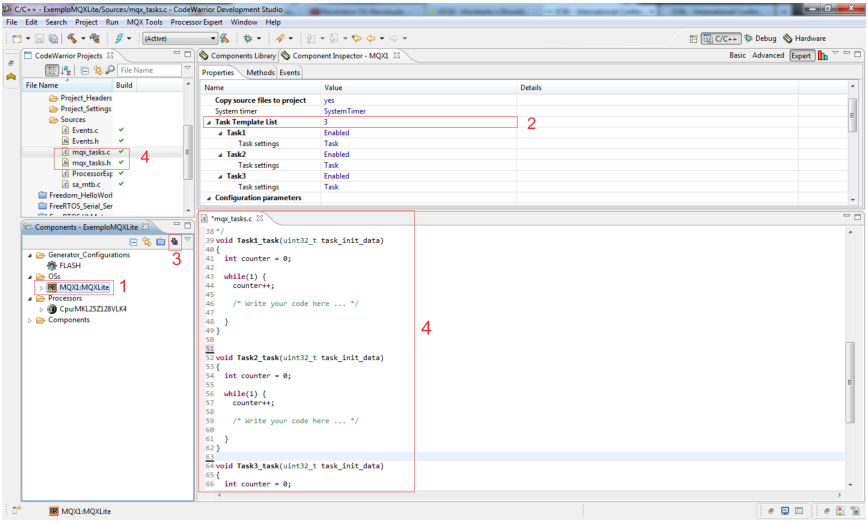

Figura 35 – Exemplo de criação de uma tarefa MQX Lite

Após a criação das tarefas a programação pode ser realizada na parte demarcada como */* Write your code here ... */*. Também é possível alterar o nome da tarefa (1), o nome da função da tarefa (2), o tamanho da pilha (3) e a prioridade (4). Na Figura 36 é possível visualizar como essas propriedades podem ser alteradas.

A.2 FREERTOS

Desenvolvido por volta dos anos 2000, é um sistema operacional embarcado *open source*, totalmente escrito na linguagem C. Possui uma licença GPL(*General Public License*) (LICENSE, 2013), que permite o uso deste em aplicações comerciais.

Nos próximos tópicos serão descritos como é feita a manipulação

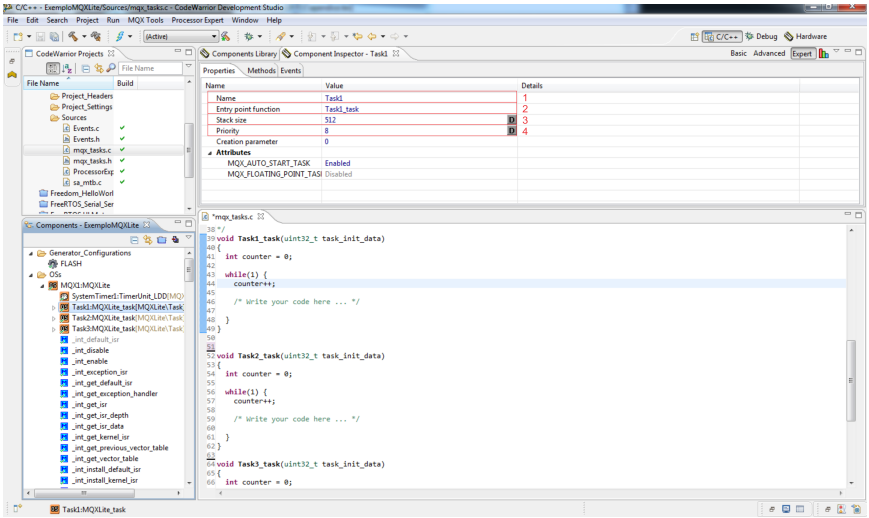

Figura 36 – Configurações de uma tarefa.

de tarefas, o controle do kernel, gerenciamento de filas e o uso de semáforos.

A.2.1 Manipulação de Tarefas

As próximas seções descrevem a maneira com que as tarefas são manipuladas no FreeRTOS.

A.2.1.1 Criação e Exclusão de Tarefas

A criação e exclusão de tarefas é feita por duas funções: xTaskCreate e vTaskDelete. Para tanto, é necessário adicionar a biblioteca *task.h* ao projeto.

Após sua criação, a tarefa irá entrar na lista de tarefas prontas para execução:

Os parâmetros da função xTaskCreate são:

- *pvTaskCode*: ponteiro para a função que implementa a tarefa.
- *pcName*: nome da tarefa. Este nome é utilizado para facilitar a

```
portBASE_TYPE xTaskCreate(
        pdTASK_CODE pvTaskCode,
        const portCHAR * const pcName,
        unsigned portSHORT usStackDepth,
        void *pvParameters,
        unsigned portBASE_TYPE uxPriority,
        xTaskHandle *pvCreatedTask
);
```

depuração.

- *usStackDepth*: tamanho da pilha de dados da tarefa.
- *pvParameters*: parâmetro que será passado para a tarefa.
- *uxPriority*: prioridade da tarefa.
- *pvCreatedTask*: variável que será utilizada para manipular a tarefa

Para excluir uma tarefa é preciso apenas utilizar a variável do tipo *xTaskHandle*, caso seja passado *NULL* a tarefa que chamar a função *delete* será excluída:

```
void vTaskDelete( xTaskHandle pxTask );
```

É importante que seja liberado a memória alocado a esta tarefa antes que ela seja excluída.

A.2.2 Semáforos

A criação e manipulação de semáforos é feita por 6 (seis) funções, que serão explicadas nas próximas subseções.

A.2.2.1 Criação de Semáforo Binário

Este tipo de semáforo é utilizado quando é preciso que ocorra exclusão mútua. Para o controle deste acesso é utilizado o mecanismo de fila, ou seja, a primeira tarefa que solicitar o semáforo será a primeira a obter acesso. Este semáforo só utiliza valores entre 0 e 1.

Semáforo binário é muito parecido com um mutex, a diferença é que o mutex inclue um controle de prioridade da tarefa e o semáforo binário não. Isso faz com que semáforos binários sejam melhores para sincronização de execução e mutex para implementação com tarefas com prioridade (BARRY, 2007).

O semáforo deve ser do tipo *xSemaphoreHandle*. O código abaixo exemplifica o uso de um semáforo binário.

```
1   xSemaphoreHandle xSemaphore; // Declaração do semáforo
2
3   void vATask( void * pvParameters )
4   {
5       vSemaphoreCreateBinary( xSemaphore ); // Criação do semáforo
6       if( xSemaphore != NULL )
7       {
8           // O semáforo foi criado com sucesso e pode ser utilizado.
9       }
10  }
```

A.2.2.2 Criação de um Mutex

A criação de um mutex está disponível a partir da versão 4.5.0. A função para criar um mutex é:

```
1   semphr.h
2
3   xSemaphoreHandle xQueueCreateMutex (void);
```

Assim como o semáforo binário, o mutex também utiliza o mecanismo de fila, porém mutex utiliza o mecanismo de prioridade. Neste caso pode acontecer da tarefa com maior prioridade nunca liberar o mutex, pois sempre estará na frente na fila para obter o mutex, por isso é importante que a tarefa utilize a seguinte chamada de sistema para liberar o mutex (BARRY, 2007):

```
1   xSemaphoreTake();
```

A.2.2.3 Obter o Semáforo

Antes de obter o semáforo é importante verificar se o semáforo foi criado corretamente. O código abaixo ilustra a chamada de sistema para obter o semáforo:

```
1  xSemaphoreTake(
2      xSemaphoreHandle xSemaphore,
3      portTickType xBlockTime
4  )
```

Os parâmetros da função são:

- ***xSemaphoreHandle xSemaphore***: identificador do semáforo, a variável *xSemaphoreHandle*.
- ***portTickType xBlockTime***: tempo de espera para o semáforo ficar disponível. Se o parâmetro for *NULL* a tarefa ficará aguardando até que o semáforo seja liberado.

A.2.2.4 Liberação de um Semáforo

Após realizar todas as operações na seção crítica é necessário liberar o semáforo para que outras tarefas tenham acesso. Abaixo o exemplo para liberar o semáforo:

```
1  xSemaphoreGive( xSemaphoreHandle xSemaphore )
```

O único parâmetro é o identificar do semáforo.