PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
POST GRADUATE PROGRAM IN COMPUTER SCIENCE

COOPERATIVE FAULT TOLERANCE IN MULTIAGENT ROBOTIC SYSTEMS

MÁRCIO GODOY MORAIS

This monograph has the intent of presenting the research plan for the master of science degree in computer science.

supervised by Prof. Dr. Alexandre Amory

PORTO ALEGRE
2014

ABSTRACT: With the increasing complexity of autonomous systems and the implementation of increasingly sophisticated missions involving complex, open, dynamic and unstructured environments and the interaction with humans, it creates the need for mechanisms to ensure the reliability of such systems. Fault tolerance is a potential solution to achieve such reliability and obtain reliable autonomous systems. The focus of this work is the proposition of cooperative methods of fault tolerance in multi-robot systems from the integration of robotic systems and multi-agent systems. Therefore, this study use a heterogeneous network of robots available at the Autonomous Systems Laboratory at FACIN. Thus, this monograph aims to present the research plan of this work.

Palavras-chave: Mobile robots, multiple robots,fault tolerance, multi-agent systems.

SUMARY

1. Introduction

With advances in technology, and with the increasing availability of hardware and software that can be applied in robotics, interest in the development of robots is increasing around the world. For decades, the industry uses stationary robots on assembly lines of major automotive companies and other environments where such machines perform tasks with speed and accuracy, increasing productivity and minimizing the possibility of errors caused by human failures.

However, the use of robots is not limited to the industry. These technological advances allow robots to perform more complex tasks in less controlled environments. Currently robots are present in precision agriculture through seeders robots[1], hasverting fruits[2] and UAVs (Unmanned Aerial Vehicle), in the application of fertilizers, pesticides and herbicides and pest detection and control of agricultural production. In livestock, milking robots that in addition to the milking itself, are able to detect and identify changes in temperature, quantify the production and the quality of the milk[3]. In the entertainment field, there are robots acting as guides in museums and exhibitions[4] [5] or just like robotic pets directed to entertain people[6].

No campo da pesquisa aeroespacial, a agência espacial norte-americana (NASA) vem utilizando robôs para a análise do clima e da geologia do planeta Marte e, em parceria com a General Motors Company, desenvolveram e levaram o robô humanóide Robonaut 2 à estação espacial internacional, com o intuito de continuamente desenvolver o robô e avaliar a possibilidade do mesmo substituir astronautas em tarefas arriscadas ou simplesmente assumir tarefas simples de forma a possibilitar aos astronautas a execução de outras atividades [7][8].

Na área militar, onde grandes investimentos são feitos no campo da robótica, existem robôs auxiliando na logística e movimentação de tropas [9], no reconhecimento e mapeamento de áreas de conflito e borbardeios de precisão[10], entre outros.

Ainda existem inúmeras possibilidades de utilização de robôs em substituição ao homem em tarefas que venham a oferecer risco ou que o exponham a condições insalubres.

Apesar do uso amplo e diversificado, robôs são construídos a partir de necessidades específicas, fazendo com que o custo de produção de um robô seja alto. Desta forma, um robô que venha a apresentar falha pode, além de comprometer a execução das tarefas, trazer prejuízos para os seus usuários.

No caso de robôs estacionários utilizados em linhas de produção, um robô com falha pode interferir no funcionamento de toda uma linha de produção, comprometendo o cumprimento das suas tarefas, das tarefas de outros robôs, e de possíveis funcionários envolvidos na produção, trazendo grande prejuízo. Além disso, robôs que interagem com humanos ou dividem o ambiente com eles, caso apresentem falhas, podem trazer riscos a integridade física do humano.

De forma a evitar esses efeitos das falhas em robôs, um sistema robótico tolerante a falhas possibilita a análise de uma determinada situação ou ação, permitindo a reconsideração de suas ações e planos. Sistemas robóticos multiagentes possuem muitas vantages adicionais em relação a um sistema robótico de um único agente como, por exemplo, a redução do tempo necessário para atingir um determinado objetivo através do uso de paralelismo ou propciar soluções de custo mais baixo para aplicações mais complexas através do uso de uma rede de diferentes robôs especializados ao invés de um único robô capaz de executar todas as tarefas de um objetivo. No caso de tolerância a falhas, um robô pode assumir as responsabilidades de outro robô, garantindo a execução das tarefas.Este trabalho se propõe ao monitoramento dos sistemas de software sintetizando informações que possibilitem o tratamento e, por consequência, a tolerância à falhas num nível mais alto, abstraindo informações relativas ao hardware propriamente dito. Com a utilização de um sistema robótico multiagente este trabalho visa obter um aumento de robustez e confiança no sistema através do uso da informação de diferentes agentes como forma de redundância cooperativa dentro do sistema.

2. Fundamentação Teórica

O presente capítulo aborda de forma breve, os conceitos cruciais no desenvolvimento deste trabalho como organização de software em robótica, dependabilidade e agentes e multiagentes.

2.1. Organização do Software em Robótica

Com a crescente complexidade das aplicações dos sistemas robóticos, cresce também a complexidade da arquitetura da sua software em sistemas robóticos. Com isso, torna-se necessário aprimorar atributos do software tais como modularidade, escalabilidade, reusabilidade, eficiência e tolerância a falhas.Sendo assim, as principais características de um sistema de software robótico são:

- Arquitetura distribuída e concorrente: necessária para tal arquitetura estar apta a utilizar todos os recursos oferecidos pelos processadores, multiprocessadores e microcontroladores, de forma a cobrir todo o custo computacional necessário á um sistema robótico complexo.
- Modularidade: o sistema é formado de vários módulos com alta coesão e baixo acoplamento de forma a garantir uma mínima interdependência e possibilitar mantenabilidade, escalablidade e uma arquitetura reusável. Isto é especialmente importante na robótica devido à falta de padrão e uma relação muito próxima com o hardware, que tende a gerar sistemas não reusáveis.
- Robustez e tolerante a falhas: O malfuncionamento de um componente não deve comprometer o funcionamento de todo o sistema e em contrapartida, o sistema deve ser capaz de continuar sua execução tão bem quanto possível com os recursos disponíveis na condição de falha. Os demais componentes devem ser capaz de agir por iniciativa própria e tomar suas próprias decisões para superar este tipo de situação. Tais decisões devem ser tomadas baseadas na cooperação com outros agentes do sistema ou em suas próprias informações específicas.
- Eficiência e tempo real: Grande parte dos sistemas robóticos tem algum tipo de restrição de tempo real e tais restrições são problemáticas em arquiteturas de software distribuído. Devido a isso, o projeto da arquitetura deve levar em consideração o uso de software,

hardware, protocolos e mecanismos de comunicação que garantam a observância e de tais restrições.

Um dos métodos de desenvolvimento mais aplicados no desenvolvimento de software de robôs é o arranjo em camadas, pois, devido ao alto grau de complexidade envolvido no desenvolvimento de software para robôs, o arranjo em camadas contribui para a separação de responsabilidades, baixo acoplamento e alta coesão em um sistema.

Tal arranjo é tipicamente considerado através da distribuição do sistema em três camadas: funcional, execução e decisão[11].

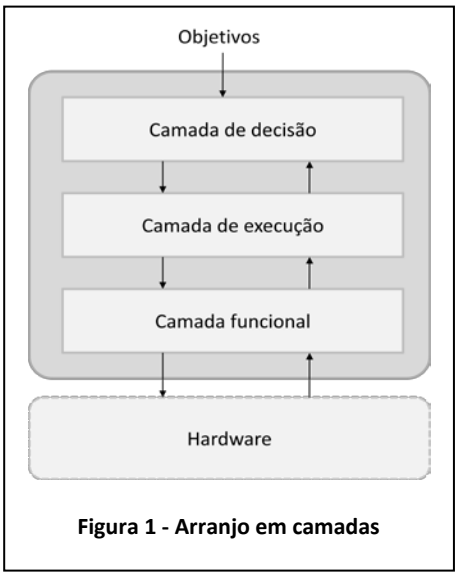

Figura 1 - Arranjo em camadas

A camada funcional é a camada de mais baixo nível dentro do arranjo em camadas. Ela é responsável por implementar uma interface entre as camadas de mais alto nível e o hardware através de operações elementares efetuadas sobre sensores e atuadores. Nesta camada não existe a preocupação com o sistema ou com os demais módulos que o compõe, mas um forte comprometimento com a extração de dados de sensores e envio de instruções para atuadores, de forma a controlar o hardware eficientemente.

A camada de execução é responsável pela definição de uma infraestrutura de software através da qual a troca de informações entre camadas ou módulos do sistema será efetuada. A camada de execução é responsável pela interconexão entre a camada funcional e a camada de decisão através de duas operações principais: decomposição de ações e síntese de informações. A operação de decomposição subdivide ações da camada de decisão em

operações elementares para a camada funcional. A operação de síntese das informações extrai informações úteis da camada funcional e as transforma em percepções para a camada de decisão.

A camada de decisão é responsável pela definição dos objetivos, execução de ações e avaliação da possibilidade de sucesso ou falha quanto à execução das ações necessárias para que os objetivos sejam alcançados. Esta camada é responsável por adicionar “raciocínio” ao robô e é totalmente dependente da entrega correta de resultados por parte dos módulos ou serviços que compõe as camadas mais baixas.

Como forma de interconectar as camadas de software e permitir a comunicação entre elas, através dos processos que compõe cada uma, são propostos os middlewares [12], que são softwares com o potencial de gerenciar a heterogeneidade do hardware, melhorar a qualidade da aplicação de software, simplificar o projeto de software. O desenvolvedor necessita somente implementar a lógica ou algoritmo como um componente do middleware.

Ayssam e Tarek [12] avaliam os principais middlewares utilizados no desenvolvimento do software de robôs, sua arquitetura, seus objetivos, e avalia cada um de acordo com um conjunto de atributos tais como arquitetura, ambiente de simulação, padrões e tecnologias, suporte a ambiente distribuído, detecção e recuperação de falhas. Os middlewares analisados foram: Player, CLARAty, ORCA, MIRO, UPNP, RT-Middleware, ASEBA, MARIE, RSCA, OPRoS, ROS, MRDS, OROCOS, SmartSoft, ERSP, Skilligent, Webots, Irobotaware, Pyro, Carmen, and RoboFrame.

Tendo em vista o contexto deste trabalho, considera-se como atributos relevantes a capacidade de tolerância a falhas e o suporte a ambiente distribuído. A capacidade de tolerância a falhas é essencial a partir do momento que o robô passa a enfrentar os desafios da vida real, fora dos laboratórios e ambientes de testes controlados, e experimenta situações críticas reais que, em caso de falhas, podem causar danos ao robô ou a terceiros. O suporte ao ambiente distribuído passa a ter um papel chave, pois permite a execução de processos monitores bem como a extração e avaliação de dados de forma remota. Além disso, a utilização de sistemas distribuídos permite uma fácil integração numa rede de robôs, seja ela heterogênea ou não.

Dentre os middlewares analisados, a maior parte não possui qualquer implementação de tolerância a falhas (ASEBA, MARIE, RSCA, ERSP, Webots, Irobotaware, Carmen, RoboFrame, Pyro) ou esta não está implementada de forma explícita (Player, ORCA, MIRO, ROS); Uma parcela menor está em fase de desenvolvimento (OPRoS, SmartSoft); E uma pequena parte possui implementação de sistema tolerante a falhas (CLARAty, RT-Middleware, Skilligent).

Tolerância a falhas não explicita refere-se a recursos do middleware que oferecem algum tipo de informação que pode ser utilizado com o intuito de detectar um comportamento fora do esperado. Podemos tomar como exemplo o projeto Player, que possui uma lista de exceções que podem ser interpretadas como falhas. ROS, com a aplicação do conceito de modularização, permite o isolamento da falha em um único módulo (RosNode), evitando a propagação da falha para o restante do sistema.

Com relação à capacidade de processamento distribuído quase todos os frameworks analisados possuem tal atributo, com exceção de OROCOS, Pyro, Webots e ERSP.

2.2. Conceitos Básicos de Dependabilidade

2.2.1. Tipos de falhas

Segundo [13], um sistema é toda e qualquer entidade capaz de interagir com outras entidades, isto é, outros sistemas, sejam eles software, hardware, seres humanos e o ambiente físico e seus fenômenos naturais.

Quando um sistema ou serviço em execução entrega um resultado diferente daqueles esperados, desviando da especificação, dizemos que ocorreu um defeito (do inglês, *failure*) [14][11][1]. O sistema está em estado de erro (em inglês, *error*) se este estado levar a um defeito. O erro é causado por uma falha (em inglês, *fault*), seja ela física ou algorítmica.

Quanto ao estado das falhas, podemos dizer que elas podem ser ativas, quando um determinado sistema ou serviço produz resultados indesejados, ou dormentes, quando a falha está presente, mas uma ação ou situação chave para tornar a falha ativa ainda não ocorreu.

[1] O significado dos termos ‘fault’, ‘error’, ‘failure’ são bem definidos na língua Inglesa. Entretanto, na língua Portuguesa não existem termos universalmente aceitos e muitas vezes os termos são utilizados como sinônimo. Para fins de coerência adotou-se os termos definidos por Weber [12].

Um exemplo de falha dormente seria a utilização, por parte de um módulo de software, de uma área de memória que foi previamente alocada dinamicamente sem que fosse feita a verificação de sucesso da operação de alocação. Provavelmente a execução e os testes do módulo de software ocorreriam dentro da normalidade até que ocorresse um episódio de exaustão de memória.

```
{
        char *pszText = malloc(32);
        strcpy( pszText, “Hello world!” );
}
```

Dentro do contexto da robótica, as falhas podem ser agrupadas da seguinte forma: Falhas físicas, falhas de projeto e falhas de interação.

2.2.1.1. Falhas físicas

Falhas físicas são decorrentes do desgaste natural dos componentes de hardware ou de fatores externos tais como, por exemplo, interferência eletromagnética ou temperaturas extremas. Podemos tomar como exemplo alguns incidentes ocorridos em indústrias no Japão, no final da década de 80, quando operários morreram atingidos ou arremessados por robôs que efetuaram movimentos inesperados, causados por interferência eletromagnética[15].

2.2.1.2. Falhas de projeto

Falhas de projeto podem ocorrer durante toda a fase de desenvolvimento, desde a concepção do projeto até a aprovação – aceitação de que o software está pronto para o uso. Durante o desenvolvimento, o módulo desenvolvido interage com o ambiente de desenvolvimento e falhas podem ser introduzidas através de ferramentas de desenvolvimento ou dos próprios desenvolvedores, introduzindo falhas com objetivos “maliciosos” [13] ou de forma não intencional, por falta de competência [13][11].

2.2.1.3. Falhas de interação

Falhas de interação são falhas externas causadas por elementos do ambiente interagindo com o sistema. Considera-se falha de interação todo evento externo que, devido à dinamicidade do ambiente, impeça o robô de atingir um objetivo ou de interagir com o meio.

Como exemplo, podemos citar um evento ocorrido na competição de carros autônomos DARPA Urban Challenge, com os carros autônomos da Universidade de Cornell e do Instituto de Tecnologia de Massachusetts[16]. Estes carros tiveram a execução de suas tarefas interrompidas por fiscais da prova devido ao fato que o carro de Cornell ficou relutante em decidir se deveria andar ou não e o carro do MIT, que estava logo atrás, decidiu pela ultrapassagem no exato momento em que o carro da Cornell tomou a decisão de andar. Com os dois carros em curso de colisão, os fiscais foram obrigados a interromper a execução sob pena dos carros virem a colidir causando danos irreparáveis, impedindo a cumprimento dos objetivos de ambos os carros.

2.2.2. Atributos de dependabilidade

Dependabilidade de um sistema é a abilidade de evitar falhas mais freqüentes e severas do que o aceitável [13]. O termo dependabilidade integra os seguinte atributos, tais como confiabilidade, disponibilidade, integridade, segurança e mantenabilidade[13]. A variação na ênfase do projeto influencia diretamente no balanceamento desses atributos.

- *Confiabilidade*: capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período;
- *Disponibilidade*: é uma medidade de probabilidade do sistema estar operacional num instante de tempo determinado; alternância de períodos de funcionamento e reparo;
- *Integridade:* ausência de alterações impróprias no sistema;
- *Segurança*: probabilidade do sistema ou estar operacional e executar sua função corretamente ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam;
- *Mantenabilidade:* capacidade de aceitar modificações e reparos, manutenção.

2.2.3. Meios para atingir dependabilidade

No desenvolvimento de um sistema onde exista preocupação em atingir os atributos de dependabilidade, as seguintes técnicas podem ser empregadas: prevenção de falhas, tolerância à falhas, remoção de falhas e previsão de falhas [13].

2.2.3.1. Previsão de falhas

A previsão de falhas tem como objetivo estimar o número atual de ocorrências, a incidência futura e as prováveis consequências das falhas através da avaliação do comportamento do sistema. A avaliação de tal comportamento se dá através de dois aspectos:

- Qualitativo – Tem por objetivo identificar e classificar os modos de falhas ou combinações de eventos que levem à falhas de sistema;
- Quantitativo – Tem por objetivo avaliar em termos de probabilidades na medida em que alguns dos atributos estão satisfeitos, esses atributos são então vistos como medidas.

2.2.3.2. Prevenção de falhas

Trata dos meios de previnir que ocorram falhas ou que elas sejam introduzidas no sistema. . Um exemplo de método de prevenção de falhas é a melhoria dos processos de desenvolvimento de forma a reduzir o número de falhas introduzidas nos sistemas. No caso de falhas de projeto de software, ou *bugs* de software, são utilizadas técnicas e práticas que minimizam a inserção de bugs, como por exemplo, uso de controle de versão, registro de bugs, padrões de codificação.

2.2.3.3. Remoção de falhas

Remoção de falhas trata dos métodos para reduzir o número ou a severidade das falhas. A remoção de falhas pode ocorrer em dois momentos distintos: durante a fase de desenvolvimento ou durante seu uso.

Durante a fase de desenvolvimento de um sistema, a remoção consiste de três etapas: verificação, diagnóstico e correção. Durante a etapa de verificação, são feito testes de forma a avaliar se o sistema está de acordo com um determinado conjunto de condições e especificações. Em caso negativo, o diagnóstico das falhas deve ser realizado, seguindo da sua correção. Após a correção, a etapa de verificação deve ser refeita de forma a assegurar que a remoção da falha não causa consequências indesejadas ou outras falhas. O ato de verificar o preenchimento de um conjunto de requisitos e condições é usualmente denominado validação.

Durante o uso de um sistema, a remoção de falhas consiste de manutenção preventiva ou corretiva. Manutenção corretiva foca em remover falhas que produziram um ou mais erros e foram reportados (detectados) enquanto a manutenção preventiva visa descobrir e eliminar falhas antes que elas possam causar erros durante a operação normal.

2.2.3.4. Tolerância a falhas

Um sistema tolerante a falhas deve estar apto a detectar a ocorrência de um ou mais erros (e possivelmente falhas) e recuperar-se dos mesmos de forma a entregar resultados corretos e isolar tais ocorrências, evitando que os erros detectados voltem a ocorrer.

Para que a ação de recuperação seja bem sucedida, o sistema de tolerância a falhas deve ser capaz de gerenciar tanto a ocorrência de erros quanto de falhas.

No caso do gerenciamento dos erros detectados – remoção dos erros de um estado do sistema através de pontos de salvamento de estado, isto é alcançado através das ações de *rollback*, *rollforward* e *compensação*. *Rollback* consiste em levar o sistema de volta ao último estado salvo (checkpoint) antes da ocorrência de erro; *Rollforward* consiste em colocar o sistema em um estado alcançável e sabidamente sem erros; *Compensação* é o uso de recursos redundantes ou da distribuição da carga de trabalho sobre recursos sem erros, de forma a possibilitar o mascaramento do erro.

Com relação ao gerenciamento de falhas, cujo objetivo é evitar que uma falha volte a ocorrer, quatro passos são necessários para tal: Diagnóstico, Isolamento, Reconfiguração e Reinicialização. O diagnóstico identifica o tipo e localização do erro que ocasionou a falha. A partir daí é necessário o isolamento da origem da falha através da exclusão dos componentes ou dispositivos com. Tendo em vista que pelo menos um módulo ou dispositivo foi excluído, pode ser necessário realizar a reconfiguração do sistema. A reconfiguração trata de associar a tarefa do componente em falha para um componente redundante ou distribuir a tarefa entre outros componentes sem falhas. Feito isso, a informação sobre a nova configuração é registrada e o sistema volta a operar.

2.3. Agentes e Sistema Multiagentes

Um agente é “componente de software autônomo proativo e social”[17]. Agentes podem reagir a eventos externos ou mensagens, mas podem também ser proativos tomando a iniciativa e mudando seu comportamento de forma a atingir seus objetivos.

Um ponto chave na descrição e no entendimento de agentes foi a introdução da arquitetura BDI (Belief, Desires, Intentions). Os elementos básicos de um agente são os objetivos e as informações que o agente possui e algumas técnicas de deliberação que possibilitam a troca de objetivos, baseada em tal informação[18]. A arquitetura BDI refina um pouco tal conceito e incorpora:

- Crenças (*Beliefs*) - descrevendo a visão do agente sobre ele mesmo, sobre outros agentes e sobre o ambiente que o cerca;
- Desejos (*Desires*) – descrevendo os objetivos à longo prazo;
- Intenções (Intentions) – descrevendo os obetivos que foram selecionados e estão sendo perseguidos.

Embora o conceito de agente como uma entidade autônoma seja muito importante, o paradigma de agente só é utilizando em sua plenitude quando aplicado à ações combinadas de múltiplos agentes[18]. Eventualmente o agente enfrenta situações ou possui objetivos complexos onde não basta proatividade e conhecimento sobre o ambiente. E nessas situações, talvez um grupo de agentes, capazes de comunicar-se através de um padrão de comunicação estabelecido, esteja apto a atingir tais objetivos complexos.

Do ponto de vista da robótica, sistemas multiagentes são compostos de múltiplos agentes distribuidos em forma de diversos servidores em uma rede. Eles são sociáveis, ou seja, eles necessitam interagir uns com os outros de forma a atingir os objetivos do sistema.

3. Estado da Arte

Carrasco et al. [19] proprõe um método de detecção e isolamento de falhas cooperativo em redes homogêneas de robôs baseado nas capacidades e recursos dos mesmos. Tal método combina técnicas de detecção e isolamento de falhas já empregadas em um único robô com ideias presentes em sistemas cooperativos de robôs de forma que a informação local de um robô não é utilizada unicamente para o seu sistema de detecção e isolamento de falhas mas como um mecanismo de redundância de dispositivos para o tratamento de falhas em qualquer robô dentro do grupo.

No método proposto em [19], as falhas são pré-definidas e estão divididas em dois grupos:

- Falhas que podem ser detectadas por um único robô – deslizamento de uma das rodas, uma das rodas fca presa, ambas as rodas ficam presas e falha de um dos encoders;
- Falhas que necessitam cooperação entre robôs para que sejam detectadas – falha aditiva no sonar, sonar retornando valor fixo, falha aditiva na bússola e bússola retornando valor fixo.

As falhas que podem ser detectadas por um único robô são determinadas a partir do cálculo da probabilidade da hipótese de ocorrência da falha ser verdadeira. Tal probabilidade é calculada com base na modelagem das falhas utilizando filtro de Kalman de forma a obter uma estimativa ótima do estado e dos vetores de medida de cada modelo. Entretando, a aplicabilidade do método fica limitada apenas às falhas que podem ser modeladas dentro de um filtro de Kalman, caso contrário, é necessário utilizar outro método de isolamento de falhas.

No caso das falhas que dependem de cooperação, elas somente podem ser verificadas se houver proximidade entre os robôs e pelo menos três medidas válidas. Quando dois robôs estão próximos, amostras dos valores dos sensores são confrontadas e, caso possuam valores com diferença acima de um determinado limite, os dois robôs consideram situação de falha. Não podendo isolar a falha visto que não se pode determinar qual dos robôs possui um leitura de valor válido, um terceiro robô é necessário para determinar qual robô realmente se

encontra em situação de falha. A partir da comparação da análise dos valores lidos dos sensores dos três robôs, a falha é isolada levando em consideração o valor discrepante entre os três valores informados.

Devido as características do modelo, ele é focado unicamente em falhas físicas dos robôs, considerando falhas de interação com o ambiente como “falso positivo”. Por exemplo, se um robô colide com um obstáculo e uma roda fica presa, o modelo considera a situação um alarme falso já que não existe defeito no atuador da roda. O potencial de cooperatividade que uma rede de robôs pode vir a oferecer não é explorado em sua totalidade, a coperação está limitada ao uso da informação provida pelos robôs próximos ao robô com falha em forma de redundância de sensores e parâmetros de comparação do comportamento de tais dispositivos. É importante ressaltar ainda que nenhum tipo de sistema multiagente é integrado no modelo cooperativo proposto em tal trabalho.

Com relação à integração de frameworks robóticos e sistemas multiagentes, P. Iñigo-Blasco et al. [17] fazem uma revisão dos principais aspectos de sistemas robóticos multiagentes, analisando as características comuns aos frameworks robóticos na atualidade e suas diferenças e similaridades em relação aos sistemas multiagentes.

Frameworks de sistemas multiagentes e middlewares empregados na robótica possuem ferramentas e oferecem soluções muito semelhantes em muitos aspectos principalmente naqueles focados em arquitetura de comunicação distribuída. A infraestutura de software essencial para sistemas multiagentes já é atualmente implementada por alguns middlewares: suporte ao desenvolvimento de arquiteturas distribuídas, métodos de troca de mensagens entre agentes e arquitetura orientada a serviços (SOA).

Mas a maioria dos sistemas robóticos multiagentes baseia a comunicação entre os agentes desenvolvendo software customizado ou middleware de propósito geral “reinventado a roda” ao invés de utilizar um framework de sistema multiagentes, o que é o mais apropriado no desenvolvimento de sistemas robóticos multiagentes. Uma das principais razões pela qual a utilização de sistemas multiagentes é uma boa escolha para arquitetura de software em robótica é que, quando utilizando esta abordagem, o software resultante é mais reusável, escalável e flexível ao mesmo tempo em que são mantidos a robutez e os requisitos de modularidade [17].

Um método de integração entre frameworks robóticos e sistemas multiagentes é proposto por Rockel et al. [20] através da implementação de uma camada intermediária entre ambos denominada *camada de abstração de sistema robótico - RSAL*. Está camada intermediária incorpora componentes de dados, robô, dispsitivos e comportamento.

- O componente de dados implementa a classe responsável pelos principais tipos de dados

Existem middlewares de robótica que são suficientemente adequados para a implementação de sistemas robóticos multiagentes. Isto porque, apesar de algumas deficiências no que diz respeito aos conceitos de multiagentes, eles oferecem a infraestrutura e as ferramentas necessárias à integração com sistemas multiagentes através da criação e implantação de arquiteturas distribuídas, onde eles executam componentes que correspondem à definição de agente (autonomia, sociabilidade e proatividade).

Crestani e Godary-Dejean [21] apresentam uma visão geral sobre a questão de tolerância à falhas em robótica com foco na arquitetura de controle. Além de analisar as necessidades para desenvolver metodologias de projeto para detecção, isolamento e remoção de falhas, o artigo endereça uma série de questões em aberto no que diz respeito à tolerância a falhas em sistemas robóticos. O artigo destaca que os métodos utilizados atualmente são eficientes porém insuficientes tendo em vista que não estão aptos a endereçar todas as questões que devem ser consideradas:

É difícil garantir que todas as abordagens propostas sempre satisfaçam os requisitos de tempo real quando na ocorrência de uma falha. Este é um ponto importante para a robótica móvel quando em um ambiente dinâmico;

- Rotinas de controle tolerante a falhas que integram sistemas de diagnóstico e detecção de falhas e mecanismos de recuperação de falhas são eficientes enquanto atuam sobre sensores e atuadores mas não estão aptos a considerar falhas relacionadas ao conhecimento de alto nível;
- Apesar da eficiência dos mecanismos de recuperação, as soluções propostas não são flexíveis o suficiente para gerenciar situações experimentadas durante missões complexas.

O artigo conclui afirmando que, nos dias de hoje, os princípios de tolerância a falhas são negligenciados nos projetos de arquitetura software de controle e que um esforço deve ser realizado para a integração de tais princípios.

4. Plano de Pesquisa

4.1. Motivação e Objetivos

A robótica tem se tornado mais importante com o passar dos tempos visto que se observa um aumento do uso de robôs em diversas atividades do nosso dia a dia. Desta forma, é criada uma expectativa quanto ao funcionamento destes robôs. Além disso, estes robôs funcionam em ambiente real, dinâmico, e muitas vezes imprevisível, o que pode levar a situações de falhas. Por outro lado, na realidade, o tempo médio entre falhas típico de um robô em campo é de 20 horas ou menos[19]. Estes reparos consomem tempo e recursos, o que pode vir a inviabilizar o uso do sistema. Os sistemas robóticos precisam gerar confiança nos usuários de forma que estes tenham garantias mínimas de que os comandos e missões serão executados de forma satisfatória.

Sistemas baseados em múltiplos robôs, além de viabilizarem a cooperação entre os agentes robóticos de forma a perseguir objetivos mais complexos, oferecem uma robustez adicional em função da redundância dos robôs. Por exemplo, se um robô falhar, outro pode, caso necessário, cooperar para a detecção de uma possível falha, contribuir na recuperação da falha ou até mesmo assumir as suas funções quando a mesma está relacionada à interação com o ambiente. Entretanto, dada a complexidade de desenvolvimento de sistemas robóticos multiagentes, é necessário aumentar a abstração desta programação buscando maior produtividade e menor complexidade para o desenvolvimento. Sendo assim, o objetivo deste trabalho é a *integração de um sistema multiagente a um middleware robótico* de forma a permitir um desenvolvimento de software mais estruturado e modular e que permita abstrair detalhes de mais baixo nível do robô. Esta infraestrutura poderá ser usada tanto para descrever aplicações típicas de robótica (e.g. mapeamento ou missões) quanto para descrever processos de detecção e recuperação de falhas.

Objetivos secundários

- Propor processos de software que possibilitem o trânsito de informação entre as camadas de software do sistema por meio da síntese de dados (Figura 3). Por exemplo, dados coletados de um ou mais sensores e atuadores servem como origem da percepção do agente em relação ao ambiente ou ao seu estado. Propor processos de

software de decomposição de ações (Figura 4). Por exemplo, uma ação ou intenção das camadas superiores é transformada em um grupo ou sequência de comandos para que a camada de baixo nível transmita ao hardware;

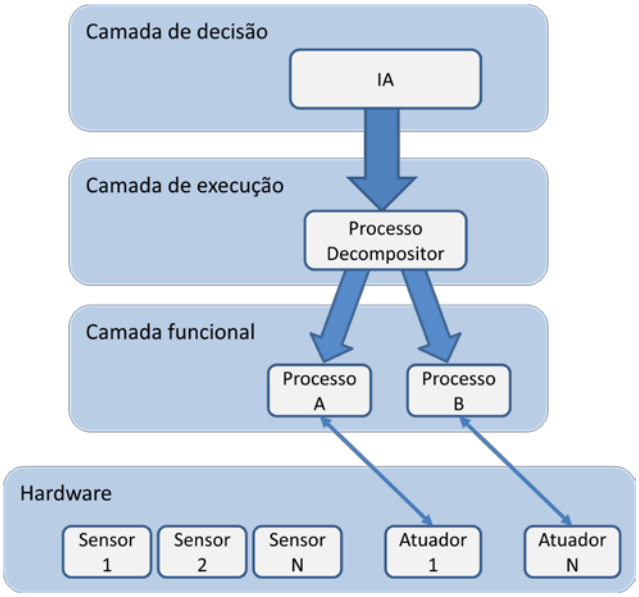

Figura 3 - Decomposição de ações de um processo do middleware executando na camada de decisão em comandos para atuadores do robô cuja interface com o hardware é implementada através de processos do middleware na camada funcional.

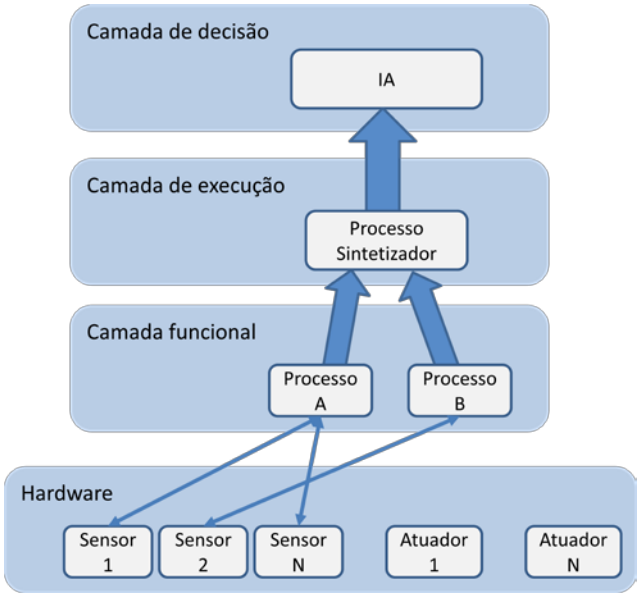

Figura 4 - Síntese de dados recebidos de um ou mais sensores, gerando uma percepção para o processo do middleware executando na camada de decisão.

- Propor uma infraestrutura para permitir a comunicação entre agentes através de troca de mensagens entre a camada de alto nível dos robôs, de forma a integrar middleware e sistema multiagente;
- Propor técnicas de detecção e recuperação de falhas em um ou múltiplos robôs utilizando sistema multiagente;
- Validar o sistema proposto tanto em ambiente simulado, através do uso de simuladores compatíveis com os middlewares disponíveis atualmente, quanto em um sistema robótico multiagente, embarcando o sistema em múltiplos robôs.

4.2. Atividades e Cronograma

Considerando os objetivos citados anteriormente, planeja-se a realização das seguintes tarefas de acordo com o cronograma apresentado na Tabela 1:

1. Estudo do middleware – Estudo das principais características e recursos de comunicação oferecidos pelo middleware de forma a verificar se todas as necessidades da integração com o sistema multiagentes são satisfeitas tendo em vista a comunicação entre diferentes robôs. Ao final desta etapa espera-se ter conhecimento detalhado sobre o middleware e os pontos estratégicos a serem trabalhados para viabilizar o método proposto;
2. Estudo do sistema multiagentes – Estudo das características do sistema multiagente, da sua infraestrutura e sistema de troca de mensagens entre agentes focando na integração com o mddleware. Ao final desta etapa espera-se ter o conhecimento necessário para efetuar as mudanças necessárias à integração;
3. Integração do middleware com sistema multiagente - Baseado nos estudos das duas atividades anteriores, realizar a integração middleware com o sistema multiagente e desenvolver uma pequena aplicação multiagente para validar a implementação e avaliação de desempenho. Esta etapa visa o desenvolvimento de uma API de integração de forma a tornar o sistema multiagente parte de um processo de alto nível do middleware, executando na camada de decisão do mesmo e utilizando todos os recursos de comunicação oferecidos pelo middleware para estabelecer a comunicação e a troca de mensagens entre agentes bem como a troca de mensagens entre o sistema multiagentes e as camadas inferiores do middleware;
4. Definição de situações de falhas – a partir de uma análise das características dos diferentes tipos de robôs disponíveis para este estudo, definir um conjunto de possíveis falhas, sejam elas físicas ou de interação, que serão monitoradas em cada tipo de robô utilizado nos testes. Uma vez selecionada as situações de falhas, desenvolver um método cooperativo de detecção, isolamento, e recuperação de falhas entre múltiplos robôs;
5. Desenvolvimento de processos sintetizadores – Uma vez determinadas as falhas a seres modeladas e seus respectivos métodos de recuperação, desenvolver os processos da camada de execução capazes que receber informações de sensores por meio de mensagens oriundas de um ou mais processos da camada funcional. Isto envolve ***sintetizar*** tais informações gerando um percepção sobre si mesmo ou sobre o

ambiente e enviar, também por meio de mensagem, para o processo na camada de decisão no qual o sistema multiagente está sendo executado, permitindo que o sistema multiagente delibere sobre as ações a serem tomadas de acoro com o cenário corrente;

6. Desenvolvimento de processos decompositores – Assim como na atividade anterior, desenvolver os processos da camada de execução capazes de receber ações por meio de mensagens oriundas da camada de decisão e ***decompor*** tal ação em um conjunto ou sequência de comandos, posteriormente enviados através de mensagens, para os processos da camada funcional, responsáveis pela comunicação com os sensores e atuadores envolvidos na ação desejada;
7. Desenvolvimento de estudo de caso - Desenvolvimento de estudo de caso para validação e avaliação dos métodos de tolerância a falhas em múltiplos robôs;
8. Revisão do Estado da Arte – Revisão, de forma contínua, de publicações científicas relacionadas com este trabalho;
9. Seminário de Andamento – Revisão das atividades listadas e apresentação sobre o andamento de tais atividades e pesquisas relacionadas;
10. Escrita de Artigos – Escrita de um ou mais artigos relacionados aos métodos, algoritmos e técnicas empregados no trabalho proposto;
11. Escrita da Dissertação;
12. Defesa da Dissertação.

Tabela 1 – Cronograma de atividades

ID	Atividade	2014 Jan	2014 Fev	2014 Mar	2014 Abr	2014 Mai	2014 Jun	2014 Jul	2014 Ago	2014 Set	2014 Out	2014 Nov	2014 Dez	2015 Jan
1	Estudo do middleware	X	X											
2	Estudo do sistema multiagentes	X	X											
3	Integração do middleware com sistema multiagente		X	X										
4	Definição de situações de falhas			X										
5	Desenvolvimento de processos sintetizadores				X	X	X							
6	Desenvolvimento de processos decompositores				X	X	X							
7	Desenvolvimento de estudo de caso								X	X				
8	Revisão do Estado da Arte	X	X	X	X	X	X	X	X	X	X	X	X	
9	Seminário de Andamento						X	X						
10	Escrita de Artigos										X	X		
11	Escrita da Dissertação											X	X	X
12	Defesa da Dissertação													X

5. Referências

[1] R. RN Jorgensen, C. Sorensen, J. Maagaard, I. Havn, K. Jensen, H. Sogaard e L. Sorensen, “Hortibot: A system design of a robotic tool carrier for high-tech plant nursing,” em *International Commission of Agricultural Engineering*, 2007.

[2] N. Kondo, M. Monta e T. Fujiura, “Fruit harvesting robots in Japan,” 1996.

[3] DeLaval, “VMS DeLaval em detalhes,” [Online]. Available: http://www.delaval.com.br/-/Product-Information1/Milking/Systems/Voluntaria/DeLaval-VMS-in-detail/. [Acesso em 17 12 2013].

[4] M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte e D. Schulz, “MINERVA: a second-generation museum tour-guide robot,” em *In Proceedings of IEEE International Conference on Robotics and Automation*, Detroit, 1999.

[5] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki e K. Fujimura, “The intelligent ASIMO: system overview and integration,” em *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, 2002.

[6] Zoomerpup, “Zoomer - your real best friend,” Zoomerpup, 2013. [Online]. Available: http://zoomerpup.com/. [Acesso em 17 12 2013].

[7] M. Diftler, J. Mehling, M. Abdallah, N. Radford, L. Bridgwater, A. Sanders, R. Askew, D. Linn, J. Yamokoski, F. Permenter, B. Hargrave, R. Platt, R. Savely e R. Ambrose, “Robonaut 2 - The first humanoid robot in space,” em *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[8] NASA, “The Next Step : Robonaut 2 At Work,” [Online]. Available: http://www.nasa.gov/mission_pages/station/main/robonaut.html. [Acesso em 18 Agosto 2013].

[9] M. Raibert, K. Blankespoor, G. Nelson e R. Playter, “Bigdog, the rough-terrain quadruped robot,” em *In Proceedings of the 17th World Congress*, 2008.

[10] G. Atomics, “Gray Eagle UAS,” General Atomics, 2013. [Online]. Available: http://www.ga-asi.com/products/aircraft/gray_eagle.php. [Acesso em 17 12 2013].

[11] B. Lussier, R. Chatila, F. Ingrand, M.-O. Killijian e D. Powell, “On Fault Tolerance and Robustness in Autonomous Systems,” em *In Proceedings of the Third IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human*

Environments, 2004.

[12] A. Elkady e T. Sobh, “Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography,” em *Journal of Robotics*, 2012.

[13] A. Avizienis, K. L. Vytautas Magnus Univ., J.-C. Laprie, B. Randell e C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” em *IEEE Transactions on Dependable and Secure Computing*, 2004.

[14] T. S. Weber, I. E. S. Jansch-Pôrto e R. F. Weber, *Fundamentos de tolerância a falhas,* Editora SBC, 1990, p. 75.

[15] P. D'Amaro, “Baderna Eletromagnética,” *Superinteressante,* n. 78, Março 1994.

[16] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, A. Nathan e F.-R. Kline, “The MIT – Cornell Collision and Why It Happened,” em *The DARPA Urban Challenge*, 2009, pp. 509-548.

[17] P. Iñigo-Blasco, F. Diaz-del-Rio, M. C. Romero-Ternero, D. Cagigas-Muñiz e S. Vicente-Diaz, “Robotics software frameworks for multi-agent robotic systems development,” *Robotics and Autonomous Systems,* vol. 60, pp. 803-821, 2012.

[18] M. F. Jürgen Dix, “Where logic and agents meet,” *Annals of Mathematics and Artificial Intelligence,* vol. 61, pp. 15-28, 2011.

[19] R. A. Carrasco, F. Núñez e A. Cipriano, “Fault detection and isolation in cooperative mobile robots using multilayer architecture and dynamic observers,” *Robotica,* pp. 555-562, 2011.

[20] S. Rockel, D. Klimentjew e J. Zhang, “A multi-robot platform for mobile robots — A novel evaluation and development approach with multi-agent technology,” *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI),* pp. 470-477, 2012.

[21] D. Crestani e K. Godary-Dejean, “Fault Tolerance in Control Architectures for Mobile Robots: Fantasy or Reality?,” em *7th National Conference on Control Architectures of Robots*, 2012.