Sandro Battistella

# CONTROLE DE MISSÃO BASEADO NA TEORIA DE CONTROLE SUPERVISÓRIO COM APLICAÇÃO A VEÍCULOS SUBAQUÁTICOS AUTÔNOMOS

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas.
Orientador: Prof. Dr. Max Hering de Queiroz

Florianópolis
2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Battistella, Sandro
Controle de missão baseado na teoria de controle supervisório com aplicação a veículos subaquáticos autônomos / Sandro Battistella ; orientador, Max Hering de Queiroz - Florianópolis, SC, 2015.
266 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Controle Supervisório Modular Local. 3. Robótica Móvel. 4. Sistemas a Eventos Discretos. I. Queiroz, Max Hering de. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Sandro Battistella

**CONTROLE DE MISSÃO BASEADO NA TEORIA DE CONTROLE SUPERVISÓRIO COM APLICAÇÃO A VEÍCULOS SUBAQUÁTICOS AUTÔNOMOS**

Esta Tese foi julgada adequada para obtenção do Título de "Doutor em Engenharia de Automação e Sistemas", e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis, 11 de junho de 2015.

_____
Rômulo Silva de Oliveira, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

_____
Max Hering de Queiroz, Dr.
Orientador

**Banca Examinadora:**

_____
Max Hering de Queiroz, Dr.
Presidente
DAS/UFSC

_____
Sebastião Cícero Pinheiro Gomes, Dr.
IMEF/FURG

_____
Antônio Eduardo Carrilho da Cunha, Dr.
SE-3/IME

_____
Jomi Fred Hübner, Dr.
DAS/UFSC

_____
Ubirajara Franco Moreno, Dr.
DAS/UFSC

_____
Leandro Buss Becker, Dr.
DAS/UFSC

# AGRADECIMENTOS

À minha querida esposa *Laura Sánchez* pelo seu constante carinho, amor, incentivo incondicional e paciência a mim dedicados durante a realização desse trabalho.

Aos meus pais *Ary Battistella* e *Josina Santana Battistella* que sempre me motivaram e incentivaram a estudar e, com isso, a crescer na vida.

Aos meus irmãos *Saray Battistella* e *Paulo Eduardo Battistella* pelo convívio e incentivo em todas as fases da minha vida.

Ao orientador *Max Hering de Queiroz* pelo seu apoio, entusiasmo e dedicação mostrados durante todo o desenvolvimento dessa pesquisa, e também pelas inúmeras e frutíferas conversas, sugestões e críticas que foram essenciais para o andamento e conclusão da tese.

Ao colega *Breno Carneiro Pinheiro* pela amizade e apoio e também pelas diversas conversas e discussões relacionadas ao tema da robótica durante toda a realização do doutorado.

Ao professor *Carlos Henrique Farias dos Santos* pelo convite a participar do projeto original que desencadeou a presente tese de doutorado.

Ao aluno *Giovani Rodrigo Glitz* pela pesquisa e desenvolvimento dos algoritmos relacionados com a detecção e desvio de obstáculos.

À *UFSC* e, em especial, ao *Departamento de Automação e Sistemas* e seus diversos professores, funcionários e colaboradores pelas inúmeras oportunidades de estudo, aperfeiçoamento e crescimento acadêmico e profissional oferecidas desde os tempos da graduação e do mestrado.

À *Fundação Parque Tecnológico ITAIPU* (PTI) pelo apoio financeiro oferecido durante todo o desenvolvimento deste trabalho.

## RESUMO

Veículo subaquático autônomo (AUV, do inglês *autonomous underwater vehicle*) é uma classe de dispositivo robótico que se move sob a água, controlado pelo seu próprio sistema embarcado, acionado por um sistema de propulsão adequado e com fonte autônoma de energia. O Sistema de Controle de Missão (SCM) é o elemento do sistema embarcado de um veículo autônomo responsável em coordenar as ações realizadas pelos demais subsistemas, guiando o veículo durante todas as fases da missão, com base em um plano previamente elaborado e no comportamento discreto dos diversos componentes do veículo.

Esta tese propõe uma arquitetura para o SCM baseado em dois componentes principais: uma estrutura de controle supervisório e um gerenciador de missão. O primeiro componente é baseado na Teoria de Controle Supervisório (TCS). A TCS é neste caso usada para a modelagem dos vários subsistemas e restrições relacionadas com a realização de missões de AUVs em ambientes não-estruturados, e para a síntese de supervisores responsáveis em garantir que especificações de segurança e operação sejam atendidas para qualquer missão do veículo. O segundo componente é responsável pela execução de um plano de missão, escolhendo a melhor sequência de eventos habilitada pelo controle supervisório segundo um critério de otimização baseado em algoritmos de planejamento e busca.

Para validação da arquitetura proposta, o SCM é implementado empregando o ROS (*robot operating system*) com a estrutura de controle supervisório integrada mediante geração automática de código. O teste do SCM proposto é realizado em um ambiente para simulação do comportamento dinâmico contínuo e dirigido a eventos de todo o sistema embarcado de um AUV. Os resultados demonstram que o SCM proposto é capaz de garantir a realização de missões em ambientes não-estruturados, atendendo a critérios de segurança especificados pelos modelos formais da TCS. Ao mesmo tempo, o SCM permite o replanejamento de missões ao gerar um plano de missão alternativo possibilitando o tratamento de diversas situações não previstas no plano original. Além disso, a arquitetura proposta para o SCM combina ações deliberativas, que envolvem planejamento, com ações reativas sem necessidade de planejamento e com tempos de execuções relativamente pequenos.

**Palavras-chave:** Controle Supervisório Modular Local, AUV, Controle de Missão, Robótica Móvel, Sistemas a Eventos Discretos

## ABSTRACT

Autonomous underwater vehicle (AUV) is a class of robotic device that moves beneath the water, it is controlled by its own embedded system, triggered by a suitable propulsion system and with an autonomous source of energy. The Mission Control System (MCS) is the element of the embedded system of an autonomous vehicle responsible for coordinating the actions conducted by several subsystems, driving the vehicle during all phases of the mission based on a previously elaborated plan and on the discrete behavior of the vehicle remaining components.

This thesis proposes an MCS architecture based on two main components: a supervisory control structure and a mission manager. The supervisory structure is based on Supervisory Control Theory (SCT) and it is used for modelling the several subsystems and constraints related to the AUV missions in unstructured environments, as well as for the synthesis of supervisores responsible for ensuring that safety and operation specifications are met for any kind of vehicle mission. The second component, the mission manager, is responsabile for carrying out a mission plan, choosing the best sequence of events enabled by the supervisory control according to an optimization criterion based on planning and search algorithms. To validate the proposed architecture, the MCS is implemented using ROS (robot operating system) with the supervisory control models integrated through automatic code generation.

The test of the MCS is performed in a simulation environment that emulates the whole AUV continous and event-driven dynamics. The results demonstrate that the proposed MCS is capable of performing missions in unstructured environments, meeting safety criteria specified by the formal models of the CST. In the meantime, the MCS allows the mission replanning by generating an alternative mission enabling the treatment of several situtations unforeseen in the original plan. Moreover, the proposed architecture for the MCS combines deliberative actions, related to planning, with reactive actions without planning and with relatively small execution times.



**Keywords:** Local Modular Supervisory Control, AUV, Mission Control, Mobile Robotics, Discrete Event Systems

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ADCP | *Acoustic Doppler Current Profiler* |
| AUV | *Autonomous Underwater Vehicle* |
| CS | Controle Supervisório |
| CSM | Controle Supervisório Modular |
| CSML | Controle Supervisório Modular Local |
| GM | Gerenciador de Missão |
| GPS | *Global Positioning System* |
| HIL | *Hardware-In-the-Loop* |
| IA | Inteligência Artificial |
| IHM | Interface Homem-Máquina |
| IP | *Internet Protocol* |
| POO | Programação Orientada a Objetos |
| ROS | *Robot Operating System* |
| ROV | *Remotely Operated Vehicle* |
| RTOS | *Real-time Operating System* |
| SCM | Sistema de Controle de Missão |
| SED | Sistema a Eventos Discretos |
| SM | Supervisores Modulares |
| SO | Sequências Operacionais |
| SP | Sistema-Produto |
| TCH | Teoria de Controle Híbrido |
| TCP | *Transmission Control Protocol* |
| TCS | Teoria de Controle Supervisório |
| TSH | Teoria de Sistemas Híbridos |
| UAV | *Unmanned Aerial Vehicle* |
| UDP | *User Datagram Protocol* |
| UGV | *Unmanned Ground Vehicle* |
| UML | *Unified Modeling Language* |
| UV | *Unmanned Vehicle* |
| VANT | Veículo Aéreo Não-tripulado |
| WCET | *Worst Case Execution Time* |
| XML | *Extensible Markup Language* |

## LISTA DE SÍMBOLOS

| | |
|---|---|
| $E$ | Especificação |
| $E_x$ | Especificação relacionada a um subsistema $x$ |
| $G$ | Autômato |
| $G_1 \| \| G_2$ | Produto síncrono dos autômatos $G_1$ e $G_2$ |
| $K, L$ | Linguagens |
| $\bar{L}$ | Prefixo-fechamento da linguagem $L$ |
| $L(G)$ | Linguagem gerada por $G$ |
| $L(S/G)$ | Linguagem do autômato $G$ sob ação do supervisor $S$ |
| $Lm(G)$ | Linguagem marcada do autômato G |
| $L_m(S/G)$ | Linguagem marcada do autômato G sob ação do supervisor S |
| $\overline{L_m(G)}$ | Prefixo-fechamento da linguagem marcada de G |
| $Q$ | Conjunto de estados |
| $q_0$ | Estado inicial |
| $Q_m$ | Conjunto de estados marcados |
| $S$ | Supervisor |
| $S/G$ | Supervisor $S$ controlando $G$ |
| $S_x$ | Supervisor modular local |
| $SupC(K, G)$ | Suprema sublinguagem de $K$ controlável em relação à $G$ |
| $u, v$ | Cadeias de uma linguagem |
| $\varepsilon$ | Cadeia ou evento vazio |
| $\delta$ | Função parcial de transição |
| $\sigma$ | Evento |
| $\Gamma$ | Função de eventos ativos |
| $\varnothing$ | Conjunto vazio |
| $\Sigma$ | Alfabeto ou conjunto de eventos |
| $\Sigma^*$ | Conjunto de todas as cadeias finitas de $\Sigma$, incluindo $\varepsilon$ |
| $\Sigma_c$ | Conjunto dos eventos controláveis |
| $\Sigma_u$ | Conjunto dos eventos não-controláveis |

# ÍNDICE

# 1. INTRODUÇÃO

A robótica é o ramo da engenharia dedicado à pesquisa, desenvolvimento e construção de dispositivos eletromecânicos computacionalmente controlados capazes de perceber e manipular o mundo físico (SELIG, 1992; KURFESS, 2005), através da integração e coordenação de diversas habilidades manipulativas, perceptivas, comunicativas e cognitivas (XIE, 2003).

Diferente dos robôs manipuladores industriais, que carecem de mobilidade, os robôs móveis, por não estarem presos a elementos físicos, operam com autonomia em seu ambiente a partir da inteligência embarcada em sua própria arquitetura de controle, responsável por processar os dados sensoriais provenientes do meio e decidir quais ações devem ser tomadas com a mínima, ou nenhuma, interferência de um operador humano (MURPHY, 2000; SIEGWART e NOURBAKHSH, 2004). A robótica móvel vem sendo empregada em uma ampla gama de setores, tais como: exploração de áreas inacessíveis ou perigosas aos seres humanos, a exemplo da robótica espacial ou da subaquática a grandes profundidades; aplicações domésticas em geral; veículos autônomos de transporte (KURFESS, 2005). De acordo com o tipo de ambiente onde atuam, os robôs móveis não-tripulados, ou *unmanned vehicles* (UVs), podem ser classificados como: veículos terrestres não-tripulados ou *unmanned ground vehicles* (UGVs); veículos aéreos não-tripulados (VANTs) ou *unmanned aerial vehicles* (UAVs); e veículos subaquáticos não-tripulados ou *unmanned underwater vehicles* (UUVs).

Motivado pela exploração de grandes fontes de recursos naturais, pela descoberta de ampla biodiversidade marinha, bem como pela operação de estruturas construídas sob os oceanos e mares, a robótica subaquática vem se desenvolvendo consideravelmente (YUH, 2000). Apesar de mais de 70% da superfície do planeta estar coberta por água, a exploração do ambiente marinho ainda encontra barreiras devido a dificuldades inerentes ao meio, como por exemplo, a presença de correntes e turbilhões na água, topografia desconhecida, altas pressões em grandes profundidades, corrosividade, opacidade e turbidez que dificultam a leitura de sensores que, na maioria das vezes, possuem dados de interpretação dificultosa devido à presença de ruídos (BLIDBERG *et al.*, 1991). Atualmente, é possível encontrar aplicações

da robótica subaquática em vários campos: pesquisa oceanográfica (mapeamento submarino, monitoramento de ecossistemas); militar (desativação de minas); indústria petrolífera e mineração oceânica (prospecção de recursos marinhos); construção e manutenção de estruturas submarinas (barragens, turbinas, plataformas de petróleo); instalação e inspeção de cabos submarinos para comunicação; inspeção de navios e portos; recuperação de *transponders*; entre outras (VALAVANIS *et al.* 1997; YUH, 2000; XU, ZHANG e FENG, 2004b; MCGANN *et al.* 2008). No Brasil, a robótica subaquática tem se desenvolvido principalmente impulsionada pela demanda por operações nos campos de prospecção e extração de petróleo (AGUILAR, 2007). Entretanto, os robôs subaquáticos em 2008 ainda não eram produzidos comercialmente no Brasil, sendo necessária a importação ou aluguel de tais equipamentos, o que justifica, em parte, a pesquisa e desenvolvimento dedicados neste setor (FERNANDES, 2008). Os robôs subaquáticos não-tripulados são classificados de acordo com o grau de autonomia em: veículos operados remotamente ou *remotely operated vehicles* (ROVs), dependentes de comandos da estação base; e veículos submarinos autônomos ou *autonomous underwater vehicles* (AUVs), que funcionam sem a necessidade de um operador humano. As figuras 1.1 e 1.2 apresentam dois tipos distintos de AUVs, o primeiro de uso comercial e o seguinte, utilizado em pesquisas oceânicas.

**Figura 1.1:** Exemplo de AUV comercial do tipo torpedo e com um propulsor (BRÄUNL, 2008)

**Figura 1.2:** AUV desenvolvido para pesquisas acadêmicas (ARANDA, 2005)

Para que um AUV atenda a missões predeterminadas, sem a necessidade de interferência de um operador humano, o mesmo deve possuir nível suficiente de inteligência embarcada em seus sistemas computacionais, sensoriais e de controle, de modo a conferir ao veículo as competências e capacidades necessárias à navegação autônoma em ambientes parcialmente conhecidos, ou até mesmo totalmente desconhecidos (XU, ZHANG e FENG, 2004b; MCGANN, 2008).

## 1.1. AMBIENTES NÃO-ESTRUTURADOS

Os ambientes estruturados possuem geometrias relativamente simples e regulares, como aquelas encontradas em ambientes de escritório ou armazéns, formado pelos corredores, esquinas e cruzamentos, facilitando a atividade de representação do ambiente em um mapa de duas dimensões (2D) que conterá o plano de movimento do veículo. Entretanto, diferente da maioria das aplicações para veículos terrestres, o ambiente subaquático em que missões de AUVs são realizadas caracteriza-se por ser não-estruturado, ou seja, composto por superfícies irregulares e de geometria complexa, e que pode ser parcial ou totalmente desconhecido, devido a, por exemplo, modificações no seu relevo. A presença de obstáculos, fixos ou móveis, conhecidos ou desconhecidos, e a complexidade associada às tarefas de mapeamento do ambiente em três dimensões (3D) também são elementos que tornam mais difícil a operação de AUVs nesse tipo de ambiente. Tradicionalmente, ROVs são empregados na realização das mais variadas operações em lagos de barragens, como por exemplo, a inspeção visual de superfícies de barragens e estruturas submersas através do uso de câmeras (STANCEL *et al.*, 2006; CROTEAU e DUGUAY, 2010; JIMENEZ *et al.*, 2010). Contudo, com a evolução da tecnologia de veículos subaquáticos, AUVs vêm substituindo ROVs em atividades que exigem tempo de operação relativamente maior, como monitoramento e acompanhamento de ecossistemas ou realização de coleta de dados, em tarefas a profundidades elevadas devido à ausência de cabos, ou ainda, em atividades que exigem precisões maiores quanto ao seu posicionamento e controle.

Em particular, ambientes de lagos de barragens de hidrelétricas, caso inicial que motivou o desenvolvimento desse trabalho, constituem-se em exemplos típicos de ambientes não-estruturados e parcialmente conhecidos, com presença de obstáculos, muitas vezes desconhecidos. A programação e execução de missões de AUVs neste tipo de ambiente

deve considerar inúmeras adversidades. Dentre os exemplos mais significativos de adversidades e peculiaridades desse tipo de ambientes citam-se: presença de áreas de vegetação, muitas vezes desconhecidas e até difíceis de serem detectadas, onde o veículo possa ficar preso; regiões com gradientes de velocidade elevados, como proximidades às tomadas de água das turbinas; presença de correntes em várias profundidades, diferente da maioria dos ambientes marinhos, onde as correntes são do tipo fluxo laminar; opacidade devido à suspensão de partículas, necessitando câmaras mais potentes ou aproximações mais precisas para aquisição de imagens e monitoramento; existência de obstáculos móveis de tamanhos relativamente grandes, como pedaços de madeiras; presença de obstáculos fixos, mas de posição muitas vezes desconhecidas, como os troncos das árvores que compunham a floresta nativa inundada sob o lago; e topografia variada do fundo do leito.

Em comparação com a maioria dos ambientes marinhos aos quais se destina o uso de veículos subaquáticos, onde as correntes são constantes e consideradas baixas, a opacidade da água é mínima e a ausência de materiais em suspensão não implica em danos ao veículo, os ambientes de lagos de barragens impõem dificuldades extras àquelas já encontradas na robótica móvel, aumentando, portanto, a complexidade no projeto desse tipo de sistemas. Além disso, a grande maioria das pesquisas realizadas na área da robótica subaquática, seja a teleoperada (operação remota) ou a autônoma, refere-se ao seu uso em ambientes marinhos, fato este evidenciado pela relativa escassez de publicações fora da área marinha.

A programação da sequência de atividades a serem realizadas pelo AUV em vários pontos de um ambiente similar ao encontrado em lagos de barragens de hidrelétricas necessita que especificações de segurança sejam satisfeitas durante todo o período de realização da missão, mesmo em estados de erros. Assim, a execução de missões de AUV nesse tipo de ambiente envolve, além da sequência de objetivos e trajetórias contida em um arquivo definido pelo usuário, especificações e requisitos de operação e segurança quem devem ser garantidos durante toda a missão, independentemente do seu tipo, tempo de duração ou região de operação. Através de métodos da área de sistemas a eventos discretos (SED), é possível compor modelos que capturam as diversas ocorrências deterministas e não-deterministas do ambiente e da operação do veículo. Tais modelos permitem descrever a missão em termos do comportamento discreto ou dirigido a eventos, fornecendo um arcabouço formal para a modelagem e execução de missões e para o

projeto e desenvolvimento de sistemas de controle de missão (SCM), conforme será visto na seção 1.4.

## 1.2. ARQUITETURA ROBÓTICA

A organização das diversas funcionalidades do veículo com base em princípios específicos permite compor o sistema a partir de componentes de software e hardware, de acordo com critérios de desempenho, processamento e comunicação, estruturando o sistema embarcado do veículo em vários níveis lógicos de abstração, o que constitui a arquitetura robótica específica para cada veículo (MURPHY, 2000).

O uso de modelos no desenvolvimento de sistemas embarcados vem aumentando consideravelmente, pois auxiliam o entendimento de problemas complexos e apontam soluções potenciais através de níveis elevados de abstração (BECKER e PEREIRA, 2002; WEHRMEISTER, 2009), favorecendo assim o desenvolvimento e implementação das arquiteturas robóticas. A abstração, considerada uma das mais importantes características de uso de modelos em engenharia, consiste em remover ou ocultar detalhes que são irrelevantes para certo ponto de vista, permitindo entender melhor o problema, pelo menos sob uma particular perspectiva (SELIC, 2003). Outro aspecto a ser considerado no uso de modelos em desenvolvimento de sistemas de software e sistemas embarcados consiste na correspondência acurada entre a representação expressa no modelo e a sua implementação. Nesse sentido, a geração automática de software permite manter a correspondência a mais fiel possível entre o modelo e a implementação, não dependendo de interpretações dos desenvolvedores e minimizando erros devido à codificação manual.

Em robótica móvel o uso de modelos encontra-se intrinsicamente associado ao desenvolvimento dos seus diversos componentes presentes no sistema embarcado, geralmente implementado em uma estrutura distribuída. Como será visto no capítulo 2, os diversos tipos de problemas que necessitam ser solucionados em direção à operação autônoma do veículo exigem o emprego de abordagens diferentes para cada tipo de subsistema, como do ponto de vista da comunicação, dos componentes de hardware, dos algoritmos de navegação e controle, e da natureza tempo-real do sistema. Portanto, a adoção de diversas visões, favorece o entendimento do sistema, a integração dos diversos

componentes e o desenvolvimento de uma arquitetura de software e hardware.

Nesse sentido, a grande maioria dos sistemas embarcados de robôs móveis dos trabalhos analisados durante a etapa de revisão bibliográfica desta tese, incluindo AUVs, possui uma arquitetura dividida em níveis classificados em dois grupos principais: baixo e alto nível. O baixo nível é responsável pelo controle do veículo e tipicamente encontra-se baseado em modelos dinâmicos que representam o movimento do veículo, com restrições temporais no seu processamento (tempo-real). O alto nível é encarregado do controle de missão, possuindo nível de abstração maior com respeito ao funcionamento dos componentes que compõem o veículo, de modo a permitir a modelagem e resolução de problemas associados à aprendizagem, segurança, tratamento de falhas, realização de planos de missões, replanejamento, entre outros (TANGIRALA *et al.*, 2005).

Do ponto de vista dos modelos que representam a dinâmica do AUV dois tipos são encontrados: a dinâmica contínua ou dirigida pelo tempo, e a discreta ou dirigida a eventos, caracterizando, portanto, o AUV como um sistema híbrido (TANGIRALA *et al.*, 2005). Modelos contínuos são empregados para representar as dinâmicas correspondentes ao controle do movimento, em várias técnicas de tolerância a falhas e em muitos algoritmos de localização e navegação. Por sua vez, modelos advindos das teorias de Sistemas a Eventos Discretos (SEDs) podem ser usados para a descrição do comportamento lógico de componentes e a interação entre os mesmos, para a descrição de modelos lógicos de ambientes ou ainda para a especificação e controle da evolução de missões (CHRISTENSEN e PIRJANIAN, 1997; BIAN *et al.*, 2005).

A execução de missões exige configuração, coordenação, controle e supervisão dos diversos componentes da arquitetura do sistema embarcado do AUV, componentes estes cujos domínios de especificação, modelagem e implementação englobam áreas do conhecimento diferentes entre si. Tal diversidade aponta para a necessidade de um mecanismo de representação e execução de missões que seja, idealmente, independente do tipo de arquitetura do veículo e que também atenda a requisitos de segurança e operação do veículo independentemente do tipo de missão a realizar.

## 1.3. CONTROLE DE MISSÃO

O termo missão é usualmente empregado para descrever a sequência de operações a serem realizadas pelo dispositivo robótico em uma determinada região durante um período de tempo, com objetivos específicos definidos previamente (GARRIDO, 2009). Ainda que um percentual relativamente elevado das decisões em um sistema autônomo possa ser tomada com base em uma estrutura pré-estabelecida, como uma sequência de comandos (*script*), para que o veículo possa reagir adequadamente a situações não planejadas ou esperadas (não-determinismo), o mesmo deve possuir também algum processo autônomo de tomada de decisão (INSAURRALDE e LANE, 2012).

Situado no mais alto-nível da hierarquia do sistema de controle embarcado do AUV, o Sistema de Controle de Missão (SCM) corresponde à parte da arquitetura encarregada de garantir, a partir dos dados de missão definidos pelo usuário, a realização correta e segura das atividades do AUV (XU, ZHANG e FENG, 2004a; PALOMERAS *et al.*, 2009). O SCM é responsável em transformar a representação da missão definida pelo usuário em uma sequência de tarefas compostas por operações pré-configuradas, que terá como resultado a sequência de comandos no nível dos atuadores que permitirão o veículo alcançar os objetivos específicos de cada etapa da missão (CHRISTENSEN e PIRJANIAN, 1997). Coordenação de tarefas e fases da missão, ativação de comportamentos, tratamento de exceções, inicialização e finalização de atividades, são decisões a serem tomadas durante a realização das missões e que dependem diretamente da capacidade cognitiva do robô para manipular e representar informações de modo a permitir análise e raciocínio (GE e LEWIS, 2006).

Uma das dificuldades existentes no projeto de sistemas de controle de missão de AUVs consiste em encontrar um mapeamento para descrição de alto-nível da missão em termos das ações e funções de cada um dos blocos dos níveis inferiores da arquitetura. Tal dificuldade é decorrente, entre outros motivos, da dinâmica híbrida contínuo-discreta dos veículos usados em aplicações subaquáticas e que pode refletir no sistema embarcado do veículo e, em particular, no SCM (KAPELLOS *et al.*, 1995). Por exemplo, ao passo que a lei de controle, que especifica a ação sobre os atuadores em função das referências determinadas pelas trajetórias e leituras sensoriais, possua dinâmica contínua, nos níveis superiores da arquitetura torna-se necessário a tomada de decisão e a coordenação de tarefas, como a necessidade de replanejamento de objetivos ou cancelamento de atividades,

representados, muitas vezes, por sistemas de transição de natureza discreta. Portanto, o problema de missão envolve tanto os mecanismos de representação da tomada de decisão em termos de funcionalidades do AUV (modelagem, análise), bem como a transferência e execução desses mecanismos para o sistema embarcado do veículo (implementação).

Atualmente, apesar da existência de uma ampla gama de AUV e respectivos sistemas embarcados desenvolvidos, a revisão bibliográfica aponta que a quase totalidade das implementações apresenta o SCM dependente diretamente das particularidades de cada AUV, tais como tipo de sensores, hardware, subsistemas envolvidos, dificultando assim, a definição e programação de missões (PALOMERAS *et al.*, 2009). Frequentemente, uma arquitetura é especializada para a resolução de um conjunto de tarefas específicas, não permitindo o tratamento de missões para as quais o AUV não tenha sido previamente projetado, diferindo entre si no modo de interação entre as funcionalidades básicas e as tarefas de planejamento e gerenciamento de missão (ALBIEZ *et al.*, 2010). Nesse sentido, o desenvolvimento de uma abordagem para a definição de missões que seja a mais independente possível das particularidades de cada AUV torna-se desejável, facilitando a programação de missões e propiciando também que profissionais que não sejam especialistas em AUVs possam definir missões com base em aspectos científicos, como topografia do ambiente, tipo de sensores a serem empregados, sem preocuparem-se com os aspectos intrínsecos de funcionamento do veículo.

Segundo Ge e Lewis (2006), as estratégias empregadas na tomada de decisão em um robô autônomo, incluindo aquelas específicas do seu SCM, podem ser classificadas em quatro tipos principais:

- **Controle em malha-fechada:** algoritmos baseados nas teorias de controle para a coordenação do movimento, sendo responsáveis em manter valores de referência desejados, entretanto, dependendo de recursos externos, como trajetórias pré-determinadas ou um sistema de planejamento, para criar tais valores de referência.
- **Estratégias baseadas em custo:** através de representações do espaço de trabalho, empregam algoritmos para otimizar algum custo ou benefício, porém, dependem diretamente do modelo empregado para representação, como por exemplo, os algoritmos empregados em técnicas de navegação e geração de trajetórias baseados em mapas.

- **Estratégias baseadas em máquinas de estado:** estruturas baseadas em grafos que representam a evolução dos estados do sistema mediante a detecção de eventos obtidos a partir de dados sensoriais disponíveis no veículo.
- **Sistemas baseados em regras:** ações são realizadas a partir de representações do conhecimento e da tomada de decisão, como o existente em sistemas especialistas onde, de acordo com uma base de conhecimento e em um conjunto de regras heurísticas, ações são determinadas (PERRIER e KALWA, 2005).

A realização de missões complexas, por exigir a coordenação de componentes situados em diversos níveis lógicos da arquitetura do sistema embarcado, pode combinar várias abordagens para tomada de decisão descritas anteriormente. O uso de máquinas de estados permite abstrair os diversos comportamentos lógicos expressos no domínio contínuo ou discreto, definindo uma semântica comum baseada em estados e eventos para representação e análise de tais comportamentos. A adoção de uma semântica comum a todo o sistema permite o planejamento da tomada de decisão no domínio de especialistas e não no domínio dos projetistas do robô. Através de máquinas de estados também é possível representar a ocorrência não-determinista de eventos relacionadas ao sistema embarcado do AUV, como falhas de equipamentos ou erros em componentes de software, ou ao ambiente em que atua, como presença de obstáculos ou correntes, favorecendo a aplicação de estratégias para tomada de decisão. Contudo, conforme apontado por Ge e Lewis (2006), a abordagem baseada em máquinas de estados exige o conhecimento *a priori* de todos os estados possíveis do dispositivo. Tal conhecimento pode implicar em modelos complexos, com problemas de explosão combinatória para aqueles sistemas com muitos estados. Esse tipo de dificuldade pode ser contornada pela adoção de estratégias que levam em consideração a natureza modular do sistema, como por exemplo o Controle Supervisório Modular Local (CSML) (QUEIROZ e CURY, 2002). Na abordagem do CSML os modelos baseados em máquinas de estados são modulares, diminuindo consideravelmente a complexidade na análise e implementação dos sistemas envolvidos. Particularmente, esta tese trata de estratégias para a tomada de decisão para realização de missões em veículos subaquáticos autônomos baseadas em máquinas de estado, mais precisamente, no uso da Teoria de Controle Supervisório (TCS) (RAMADGE e WONHAM, 1989). Esse trabalho também aborda estratégias complementares

baseadas em controle de malha-fechada e em custo usadas para a simulação dos demais componentes do sistema embarcado do veículo.

Os formalismos mais empregados para representação e especificação de missões são (CHRISTENSEN e PIRJANIAN, 1997; GE e LEWIS, 2006; PERDOMO *et al.*, 2010): tarefas descritas no domínio específico da arquitetura, o que dificulta a portabilidade da linguagem; estruturas de transição advindas da área de SEDs, como redes de Petri ou autômatos, favorecendo a verificação formal, execução e portabilidade de missões; e comportamentos, que se constituem em mecanismos de cooperação de funcionalidades coordenados, típicas de arquiteturas robóticas reativas ou híbridas (seção 2.3).

O emprego de um arcabouço teórico-formal possibilita a modelagem, especificação e verificação de missões, permitindo a validação da corretude e conformidade com respeito aos requisitos da missão (COSTA, 2008). Nesse sentido, teorias de SEDs, como Álgebra de Processos, Teoria de Controle Supervisório (TCS) ou Teoria de Sistemas Híbridos (TSH), vêm sendo sistematicamente empregadas para especificar a sequência de atividades de um robô móvel e o subsequente mapeamento com respeito às funcionalidades implementadas pela arquitetura embarcada do mesmo. Assim, nessas abordagens são utilizadas estruturas baseadas em sistema de transição para descrever a evolução do veículo através de etapas. Métodos formais de modelagem de missões permitem a garantia, por projeto, de propriedades desejáveis, como ausência de *deadlocks* e *livelocks*, o atendimento de especificações na realização de tarefas, além de possibilitar a verificação formal da missão antes da mesma de ser enviada ao veículo.

## 1.4. TRABALHOS RELACIONADOS

Aplicações de SEDs em robótica móvel inicialmente são encontradas em maior número para veículos terrestres. Porém, na medida em que as pesquisas nessa área foram direcionadas para as outras classes de veículos, aéreos e subaquáticos, é possível também encontrar uso de SEDs para esses veículos. A revisão do estado da arte feita para o desenvolvimento desta tese aponta para os seguintes tipos mais comuns de aplicações SEDs em robótica móvel (WANG *et al.*, 1991; CHRISTENSEN e PIRJANIAN, 1997; CHEN *et al.*, 2002; XU, ZHANG e FENG, 2004a; DIAS *et al*., 2006a; GE e LEWIS, 2006; PALOMERAS *et al.*, 2006; GORYCA e HILL, 2013): descrição da evolução da missão através de fases; modelagem e coordenação de

componentes, tarefas ou manobras; modelos para representação híbrida (contínua e discreta) do veículo; e modelagem de missões para sistemas multi-robôs.

Na maioria dos trabalhos, adota-se a decomposição da arquitetura em níveis, agrupadas em duas principais: alto nível, denominada geralmente de controle de missão, e baixo nível, denominada de controle de veículo. Por exemplo, o trabalho de Wang *et al.* (1991) propõe um nível de tradução de linguagens e ativação de tarefas através de redes de Petri, permitindo a decomposição da representação de alto nível da missão em direção aos comandos dos níveis inferiores da arquitetura, porém, sem discutir os aspectos inerentes à implementação e execução da arquitetura. Christensen e Pirjanian (1997) procuram explicar como um sistema baseado em comportamentos (veículo autônomo) pode ser controlado por um sistema baseado em regras, empregando a TCS para descrever e controlar a interação entre componentes e comportamentos de um robô móvel. Uma abordagem similar também é realizada por Chen *et al.* (2002) e Feng *et al.* (2004), ao empregar a TCS para derivar supervisores que irão controlar os vários modos de operação dos diversos componentes da arquitetura do veículo. Porém, como em Christensen e Pirjanian (1997), não são discutidos os problemas de implementação dos modelos formais na arquitetura do veículo, além de fixar a realização de missões ao impor a escolha de uma única sequência de fases da missão.

Molina *et al.* (2010), também empregam a TCS para o controle de movimento de um UGV em um ambiente estruturado. Porém, tal abordagem reduz o problema de navegação de robôs à escolha de um subconjunto de comandos e operações, como girar 90°, avançar 2m, avançar 5m, etc., sendo necessário o conhecimento *a priori* do ambiente para definição de tais operações. Por sua vez, Liu e Darabi (2002) usam a TCS para evitar colisões em uma aplicação com dois UGVs atuando em um ambiente conhecido e estruturado. O trabalho aborda principalmente as limitações e dificuldades no emprego da TCS em plantas reais, mais precisamente, a dificuldade em implementar os supervisores e traduzir os modelos baseados em autômatos para o sistema real. Como será mostrado no capítulo 6, tal dificuldade é contornada nesta tese pelo uso dos modelos baseados em autômatos no próprio sistema embarcado do veículo, através da arquitetura de implementação do Controle Supervisório Modular Local (CSML) (QUEIROZ e CURY, 2002) e pela tradução automática de código.

Outro exemplo de aplicação de SEDs em UGVs consiste no trabalho de Ji e Sarkar (2007), onde é apresentada uma abordagem

baseada na Teoria de Controle Supervisório Híbrido para detecção e modificação de modos de falha do robô. Nesse trabalho, a ação de um supervisor é responsável pela modificação da configuração de controladores de movimento ante a ocorrência de falhas, definindo tipos diferentes de trajetórias conforme o sistema se encontra em modo livre de falha ou em estado de falha. Tal abordagem, contudo, não trata do problema de missões complexas, atendo-se somente ao problema de detecção e acomodação de faltas. Também é possível encontrar abordagens baseada no Controle Híbrido, como em Wang, Mallapragada e Ray (2005), onde a dinâmica contínua do veículo, relativa ao movimento e ao seguimento de trajetória, é desacoplada da dinâmica discreta, responsável pelo planejamento de missão. Ao invés de usar a abordagem de Ramadge-Wonham (RAMADGE e WONHAM, 1989) para síntese do supervisor, os autores empregam a medida de desempenho de sublinguagens como método de síntese de supervisores.

Na área de AUVs a aplicação de SEDs é feita de modo similar, uma vez que várias características e tipos de problemas encontrados em veículos terrestre também se fazem presentes. Por exemplo, em Busquets *et al.* (2012) emprega-se uma estratégia simples baseada em máquina de estados finitos para descrever os possíveis estados de um AUV durante a realização de missões. A cada estado da missão é associado um conjunto de operações que deve ser efetuado pelo AUV, como emersão e correção por GPS ou navegação com sensores ativados. Assim, a missão resulta na execução sequencial de diversas operações descritas pelas máquinas de estados. Palomeras *et al.* (2006) utilizam uma rede de Petri para descrever uma missão com base em comportamentos do veículo, como manter distância, manter profundidade, gravar dados, etc. A missão, nessa abordagem, consiste na combinação de vários comportamentos, uma vez que o formalismo de redes de Petri mostra-se adequado para a descrição da evolução paralela de funcionalidades. Outra vantagem da arquitetura proposta consiste na presença de um jogador de redes de Petri, responsável em traduzir a missão, especificada diretamente por uma rede de Petri, nos comandos de baixo nível do veículo. Porém, o problema de replanejamento de missão ou a otimização de uso dos recursos não é tratado pelo referido SCM. Por sua vez, em Bian *et al.* (2009a) é apresentado um sistema embarcado distribuído para um AUV, que inclui diversas funcionalidades, como planejamento de trajetória e sistema de mapeamento, onde um modelo baseado em uma rede de Petri hierárquica é usado para descrever e compor a missão a partir de suas

fases essenciais, também descritas por redes de Petri. Uma abordagem similar também é empregada em Chen, Yan e Zhao (2011).

Uma aplicação específica da TCS ao problema de missão de AUVs pode ser encontrada em Xu, Zhang e Feng (2004a). Nesse trabalho, uma estrutura hierárquica baseada em três camadas lógicas decompõe a missão descrita por um autômato em atividades e estas, por sua vez, em comandos para o nível contínuo do veículo. As maiores desvantagens da abordagem adotada pelos autores consistem na execução sequencial de missões, advinda da habilitação de um único evento controlável em cada estado da missão, e no cancelamento de missão, sem opção de replanejamento *on-line*, ante a ocorrência de falhas; Além disso, os autores restringem-se a uma abordagem exploratória e inicial, sem efetivamente estabelecer ou sistematizar um método para o uso da TCS na resolução do problema de missões de AUVs.

Outro trabalho que emprega autômatos híbridos para a modelagem e execução de missões de AUVs é apresentado em Dias *et al.* (2006a). Nessa proposta, o plano de missão é modelado por um autômato híbrido, constituído de estados. Cada estado corresponde a uma unidade de movimento, denominada de manobra, e através da combinação de vários estados e transições é possível especificar a lógica de diversos tipos de operações. As manobras, por sua vez, também são representadas por autômatos híbridos, onde seus estados encontram-se associados a configurações no nível dos controladores de movimento, constituindo-se, portanto, de uma abordagem baseada em Controle Híbrido. A abordagem baseada em sistemas híbridos para AUVs também é usada no trabalho de Bhattacharyya *et al.* (2006). No artigo, as tarefas, usadas para descrever fases da missão, são representadas por autômatos híbridos que, por sua vez, são agrupados em modos de operação, também descritos por autômatos híbridos. Os sinais desses autômatos são trocados com os níveis inferiores que implementam a dinâmica contínua do veículo através de vários níveis intermediários, dispostos em uma estrutura hierárquica.

Em uma estratégia diferente, Brito e Griffiths (2011) empregam um diagrama de transições baseado em cadeias de Markov para estimar e identificar estados e transições com elevado grau de risco, com o objetivo de analisar o risco de realizações de missões baseado nos vários tipos de fases e operações que o veículo pode desempenhar. As diferentes fases de uma missão são associados a um estado do modelo markoviano, e através do uso de métodos *bayseanos*, as probabilidades de sucesso ou risco inicialmente conhecidos com base em experimentos

anteriores são então estimadas e atualizadas. Porém, tal trabalho se restringe à análise de risco associado às transições, sendo mais empregado como método de detecção e acomodação de faltas.

Do mesmo modo que UGVs e AUVs, abordagens baseadas em SEDs também são aplicadas aos UAVs. Seibel (2000) utiliza autômatos híbridos para modelar diversos aspectos do problema de execução de missão de UAVs, entre eles, os modelo do estado interno do veículo, do ambiente e do plano de missão, bem como um controlador de voo reativo, responsável pela execução das missões. De modo similar, a abordagem híbrida para UGVs também é aplicada por Costa (2008), onde o planejamento de missões é obtido através da verificação de modelos dinâmicos híbridos do veículo, do ambiente e da missão em si. E em Garrido (2009), a TCS de sistemas híbridos é aplicada em UAVs, onde um modelo híbrido da missão, contendo as trajetórias desejadas, o consumo de combustível e as condições do vento, é descrito por um autômato híbrido na forma de um autômato condição / evento. Esse autômato é empregado para verificação formal da exequibilidade da missão e consequente síntese do supervisor de missão. Outra abordagem da aplicação de sistemas híbridos em UAVs consiste no trabalho de Alves e Cunha (2010), onde vários controladores contínuos, previamente sintonizados, são escolhidos mediante uma máquina de estados que acompanha a evolução da missão do veículo. A adoção de autômatos híbridos em sistemas embarcados exigiriam uma maior capacidade e tempo de processamento, principalmente se alguma etapa de verificação de modelos é usada para o planejamento de missões devido à complexidade matemática inerente aos modelos adotados. Além disso, em ambientes não-estruturados, parcial ou totalmente desconhecidos e com presença de obstáculos e correntes, a representação precisa de um modelo para o ambiente poderia tornar-se uma atividade complexa e com custo computacional elevado, dificultando a sua implementação em um sistema embarcado.

Finalmente, modelos de SEDs também vêm sendo utilizados no planejamento e coordenação de missões de times de robôs móveis, compostos por UGVs, UAVs ou AUVs, como por exemplo, Redes de Petri (EBADI *et al.*, 2010), Teoria de Controle Supervisório (GAMAGE *et al.*, 2009; PAVEI, 2011; GORYCA e HILL, 2013), Teoria de Controle Supervisório e Agentes (CAPKOVIC, 2010) ou Autômatos Híbridos (XIANG *et al.*, 2007). Contudo, tal discussão foge ao escopo desse trabalho e, por esse motivo, não serão apresentados.

Assim, as diversas abordagens da área de SEDs são empregadas para a resolução dos seguintes tipos de problemas relacionados a missões em veículos robóticos autônomos:

- **Representação:** base teórica para descrição lógica de missões, através da representação de eventos de início, fim e erro de execução de operações, manobras, comportamentos ou fases da missão. Os modelos formais são empregados para a descrição da evolução dos estados de componentes individuais situados na arquitetura do AUV, onde a interação de tais componentes determina o funcionamento global do controle de missão, além da representação de ambientes ou mesmo de sistemas multi-veículos.
- **Verificação:** arcabouço teórico para validação de propriedades lógicas do sistema. Tal finalidade é, em parte, derivada da própria estrutura de representação adotada.
- **Coordenação:** *framework* para coordenação de tarefas, comportamentos, etc., onde cada unidade atômica de operação é representadas por máquinas de estados cujas transições são do tipo início, fim e erro de operação.
- **Composição hierárquica:** método para composição hierárquica das várias funcionalidades do veículo, organizada em níveis lógicos de abstrações distintos, incluindo a descrição de componentes híbridos, com dinâmicas contínuas e discretas.
- **Execução:** emprego da própria estrutura de representação e definição de missão como elemento a ser incluído na arquitetura embarcada do veículo, fornecendo a informação necessária para cada uma das etapas da missão. Tal finalidade, entretanto, não foi encontrada na maioria dos artigos analisados, apesar de constituir-se problema importante na implementação de controladores de missão.

O emprego de uma arquitetura que favoreça a análise formal e a garantia por projeto de determinadas propriedades, ao mesmo tempo que possibilite a execução da missão através de uma estrutura de representação, pode trazer como vantagens: o atendimento a propriedades desejáveis para missões, como ausência de bloqueios, cuprimento de especificações, garantia de segurança e operação do veículo; a correspondência lógica entre especificações para descrição de missões e a realização em si da missão; a aplicação de métodos de

verificação formal para a comprovação lógica de missões; ou a diminuição de erros na programação e execução de missões.

A natureza modular do sistema embarcado do AUV, baseado em subsistemas e níveis, possibilita a aplicação do controle supervisório modular, em particular, o Controle Supervisório Modular Local (CSML) (QUEIROZ e CURY, 2000), onde os modelos da planta e especificações desejadas para o sistema são representadas por autômatos modulares. A modularidade empregada no projeto e desenvolvimento do SCM de AUVs possibilita também a incorporação de novos subsistemas e adaptação do SCM a novas restrições para o veículo. Particularmente, a abordagem modular do controle supervisório reduz a complexidade dos modelos e o custo para obtenção de estruturas de controle, denominadas de supervisores.

Na abordagem da TCS, a missão é tratatada como um problema de controle supervisório onde uma lógica para representação e execução de missões é definida em termos do comportamento discreto em malha-aberta da planta sob ação dos supervisores, que impõem restrições a tal comportamento com objetivo de atender especificações. Nesse sentido, a definição de um método para a caracterização do problema de controle de missões como um problema de controle supervisório torna-se desejável e constitui-se em uma das contribuições deste trabalho. A TCS emprega modelos baseados em autômatos e linguagens para descrever o comportamento do sistema e especificações de segurança e operação do sistema, classificando os eventos discretos em controláveis ou não-controláveis. Os eventos controláveis podem ser impedidos de ocorrer pela ação de um agente externo ou supervisor. A partir desses modelos são derivados os supervisores, que são estruturas de controle responsáveis por observar a evolução da planta e restringir o seu funcionamento de acordo com as especificações. Além disso, o processo de síntese de supervisores proposto pela TCS garante a ação minimamente restritiva e não-bloqueante sobre o sistema. Isso significa que a ação de controle é ótima no sentido que somente são impedidos de ocorrer os eventos controláveis que possam desencadear um caminho ou sequência de eventos que não satisfaça às especificações ou que leve o sistema a uma situação de bloqueio. Tal característica confere flexibilidade ao SCM na medida em que vários são os caminhos possíveis para a realização de uma missão, ao mesmo tempo em que garante a segurança do sistema ao satisfazer as especificações.

O formalismo da TCS também é empregado para separar a representação contínua do veículo e do ambiente da dinâmica discreta, reduzindo a complexidade em todas as etapas do desenvolvimento do

sistema, incluindo a sua análise e implementação. Essa é uma importante diferença em relação às abordagens híbridas, onde o custo computacional para representação e obtenção de estruturas de controle pode ser proibitivo em robótica móvel, principalmente naquelas plataformas com reduzida capacidade de processamento. Além disso, a possibilidade da representação da missão no domínio discreto reduz consideravelmente o problema de planejamento ao empregar somente o vocabulário de ações e respostas neste domínio discreto. Contudo, tal restrição pode trazer como limitação a dificuldade na inclusão de trajetórias contínuas para o planejamento de ações do veículo.

O comportamento do AUV necessita daquelas ações que consideram os objetivos a longo prazo do veículo e que, por tal motivo, requerem o planejamento da tomada de decisões neste horizonte de tempo. Porém, ante as diversas ocorrências não-deterministas, típicas em ambientes não-estruturados, o AUV também deve ser capaz de reagir de modo rápido, muitas vezes desconsiderando os objetivos a longo prazo da missão e visando a sua segurança e integridade. Tal capacidade de reação é implementada através de comportamentos reativos, cujos tempos de resposta rápidos são adequados para àquelas situações inesperadas e que possam prejudicar a integridade do veículo. Portanto, o comportamento final do AUV consiste na combinação de diversos comportamentos deliberativos, que empregam planejamento, e reativos implementados através dos diversos componentes do sistema embarcado do AUV. Nesse sentido, o SCM, mesmo aqueles baseados em métodos formais, deve considerar os dois tipos de comportamentos, deliberativo e reativo, visando a completude da missão ao mesmo tempo que garante a segurança e integridade do veículo.

## 1.5. OBJETIVOS DA TESE

O objetivo principal deste trabalho consiste na proposição e desenvolvimento de uma arquitetura para Sistemas de Controle de Missão (SCM) que explore o uso de um método formal baseado em máquinas de estados finitos para garantir o atendimento de propriedades operacionais e de segurança na execução de missões de AUVs em ambientes não-estruturados. Com ações deliberativas e reativas, o SCM confere ao AUV as habilidades de planejamento a longo prazo visando os objetivos da missão, ao mesmo tempo que a segurança do veículo é garantida a cada instante da missão.

Para a representação da lógica de missões, propõe-se o uso da Teoria de Controle Supervisório (TCS) (RAMADGE e WONHAM, 1989) que posteriormente será utilizada como base para o desenvolvimento do SCM. Particularmente, emprega-se como método formal uma extensão da TCS, o Controle Supervisório Modular Local (CSML) (QUEIROZ e CURY, 2000). O problema de controle de missão de AUVs é assim caracterizado como um problema de controle supervisório, no sentido em que o CSML é empregado para descrever as missões do AUV em termos de uma representação discreta e modular de seus componentes. Através desta abordagem é possível: separar a dinâmica contínua dos níveis inferiores do sistema embarcado do veículo da dinâmica discreta, empregada para a representação e execução de missões; empregar o alfabeto de eventos dos modelos formais para o planejamento, reduzindo assim a complexidade de execução do planejamento; e garantir, por projeto, que propriedades desejáveis para a missão, como segurança e ausência de bloqueios, sejam atendidas.

A ação reativa do CSML consiste em impedir que aqueles caminhos, ou sequências de eventos, considerados inseguros, contudo, permitindo todos os demais caminhos. A ação de planejamento, ou deliberativa do SCM, e que complementa a ação do CSML, é implementado através de um segundo componente, denominado de Gerenciador de Missão (GM), cuja responsabilidade consiste em planejar a sequência de ações do AUV em direção aos seus objetivos, explorando os diversos caminhos habilitados pelo CSML. A organização destes componentes e suas interações definem a arquitetura proposta para o SCM.

A arquitetura do SCM é então implementada por meio do *middleware* ROS (*robot operating system*) (ROS, 2015) e testada em um ambiente de simulação desenvolvido para este trabalho. Os modelos baseados em autômatos do CSML são diretamente implementados no SCM do AUV por meio da arquitetura de implementação do CSML (QUEIROZ e CURY, 2000). O GM também é desenvolvido para o ROS, tendo como principais responsabilidades, a execução da missão em si, além das ações de planejamento e replanejamento.

O objetivo principal desta tese pode ser divido nos seguintes objetivos mais específicos:

**Modelagem.** Desenvolver modelos de autômatos para diversos componentes do sistema embarcado de um AUV, bem como as especificações para o seu funcionamento e segurança, segundo o ponto de vista de realização de missões em

ambientes não-estruturados. Tal modelagem visa à descrição da missão com base na evolução de eventos discretos desses componentes, abstraindo o comportamento de baixo nível do AUV.

**Síntese.** Utilizar métodos de síntese de supervisores da abordagem de CSML da TCS para derivar estruturas de controle supervisório que irão delimitar o funcionamento do AUV, impedindo todos os caminhos que possam transgredir as especificações de operação e de segurança, e habilitando todas as sequências de eventos seguras e não bloqueantes.

**Arquitetura.** Propor e desenvolver uma arquitetura de SCM baseado no CSML para AUVs atuando em ambientes não-estruturados.

**Planejamento.** Aplicar métodos e heurísticas de planejamento e replanejamento de missões integrados ao SCM, de modo a possibilitar a escolha do melhor caminho baseado em critérios de otimização.

**Método.** Propor um método de desenvolvimento de SCM para AUVs baseada na TCS.

**Simulador.** Desenvolver um ambiente de simulação de AUVs com os diversos componentes da sua arquitetura (dinâmicas contínuas e discretas) que permita a implementação e execução de cenários de teste para a arquitetura de SCM proposta.

**Implementação.** Desenvolver e implementar um SCM para um cenário de AUV atuando em ambientes não-estruturados, inspirados em ambientes de lagos de barragens de hidrelétricas, com objetivo de testar o uso da arquitetura e do método proposto através de simulação.

## 1.6. RESUMO DAS CONTRIBUIÇÕES

As principais contribuições desta tese são listadas a seguir:

- Proposta de uma arquitetura para Sistemas de Controle de Missão (SCM) de AUVs baseada na Teoria de Controle Supervisório (TCS).
- Desenvolvimento de um componente deliberativo, denominado de Gerenciador de Missão (GM), para planejamento e replanejamento de missões e para execução de missões visando os objetivos de missão para o AUV.

- Caracterização do problema de missão como um problema de controle supervisório através da proposição e desenvolvimento de um componente reativo para garantia de especificações de segurança e operação do AUV com base no Controle Supervisório Modular Local (CSML).
- Implementação da arquitetura proposta para o SCM através do *middleware* ROS (*robot operating system*) e posterior simulação em um ambiente desenvolvido com esse objetivo.
- Proposição de um método para o desenvolvimento de SCM para AUVs baseado na TCS.
- Revisão sistemática das principais funcionalidades, componentes e problemas encontrados na área de robótica móvel, bem como das principais aplicações e abordagens da área de Sistemas a Eventos Discretos (SED) em robótica móvel.

## 1.7. ORGANIZAÇÃO DA TESE

Este trabalho está organizado conforme exposto à continuação.

O capítulo 2 traz uma visão geral do comportamento dinâmico e discreto do AUV, descrevendo as principais capacidades e funcionalidades relevantes para a modelagem e execução de missões. Uma introdução ao planejamento de missões em robótica móvel também é feito neste capítulo. Ao final, uma breve descrição das principais arquiteturas robóticas, que descrevem e organizam os componentes e suas interações, é mostrada.

No capítulo 3 é apresentado o problema de missão de robôs autônomos, em particular AUVs, e as principais estratégias de controle de missão baseadas em SEDs. Uma discussão a respeito da viabilidade, vantagens e limitações no emprego de SEDs em SCM é exposta ao final do capítulo.

O capítulo 4 introduz a Teoria de Controle Supervisório (TCS), bem como o Controle Supervisório Modular Local (CSML). A aplicação da TCS ao problema de missões de AUVs é feita à continuação, derivando modelos baseados em autômatos para descrever a dinâmica baseada em eventos do AUV, segundo o ponto de vista de missões, e obtendo as estruturas de supervisão, denominadas de supervisores, que irão garantir atendimento de especificações de segurança e operação do veículo em qualquer tipo de missão.

O capítulo 5 apresenta a proposta de uma arquitetura para o Sistema de Controle de Missão (SCM), com os seus componentes: a estrutura de Controle Supervisório; e o sistema de planejamento e gerenciamento de missão encarregado de traduzir o arquivo de missão em um plano de missão, e pela escolha do melhor sequência de eventos habilitada pelos supervisores, empregando um critério de otimização baseado em algoritmos de busca. Nesse capítulo também é mostrado o método empregado para o desenvolvimento da arquitetura.

No capítulo 6 mostra-se o desenvolvimento do SCM proposto através do *middleware* ROS (*robot operating system*). Visando testes do SCM, apresenta-se também o ambiente de simulação com a dinâmica contínua, implementada no software Matlab/Simulink, e a discreta emuladas por uma Interface Homem-Máquina (IHM) desenvolvida em linguagem Java.

No capitulo 7 são realizadas simulações do SCM integrado ao ambiente de simulação, onde são mostrados vários cenários de missões de AUV considerando diversas características encontradas em ambientes não-estruturados, como presença de erros, obstáculos e consumo inesperado de bateria.

O capítulo 8 traz as conclusões, principais contribuições e resultados da tese, bem como as perspectivas para continuidade do trabalho.

O apêndice A apresenta as especificações usadas nas etapas de modelagem e síntese do CSML não mostradas no capítulo 4. A inclusão da maios parte das especificações em um apêndice teve como objetivo tornar o texto mais sintético e didático ao leitor no sentido de apresentar somente os principais conceitos referente à modelagem das especificações e seu papel no processo de obtenção dos supervisores modulares.

O apêndice B mostra os diversos processos desenvolvidos em linguagem C/C++ para o *middleware* ROS e que fazem parte do SCM implementado.

No apêndice C é apresentado o processo de geração automática de código dos modelos modulares da planta e dos supervisores modulares. A tradução entre os eventos controláveis e não-controláveis e os respectivos comandos e sinais do AUV é também mostrada nesse apêndice.

A interface homem-máquina (IHM) pertencente ao ambiente de simulação do AUV e utilizada para a emulação do comportamento discreto do veículo é brevemente descrita no apêndice D.

Finalmente, o apêndice E ilustra o uso dos mecanismos de comunicação do tipo *publisher* / *subscriber* e cliente / servidor disponíveis no ROS, bem como a definição dos diversos tipos de mensagens usados no desenvolvimento do SCM.

# 2. SISTEMA DE CONTROLE DE MISSÃO DE AUVS

Os objetivos principais desse capítulo consistem na introdução dos problemas essenciais a serem resolvidos pelos sistemas de controle de missão (SCM) de AUVs, de modo a garantir a navegação correta e segura por ambientes parcialmente conhecidos. Ainda que o sistema proposto nesta tese possa ser aplicado, com ou sem adaptações, a outros tipos de veículos robóticos, o presente trabalha fixa seus objetivos em veículos subaquáticos delimitando-se, assim, a apresentação neste capítulo dos problemas relacionados ao controle de missões de AUVs. Inicialmente, a seção 2.1 expõe as principais categorias de funcionalidades e subsistemas, e respectivas dificuldades, encontradas em sistemas embarcados de AUVs em geral, incluindo o problema de controle de missão. Uma breve introdução ao problema de planejamento aplicado a robôs móveis é mostrado na seção 2.2. Os principais tipos de arquiteturas empregadas no desenvolvimento de sistemas robóticos são comentadas na seção 2.3. Ao final, um breve resumo do capítulo é mostrado na seção 2.4.

## 2.1. VEÍCULOS SUBAQUÁTICOS AUTÔNOMOS

O AUV é considerado um dispositivo robótico autônomo que se desloca na água propelido por um sistema de propulsão, controlado e pilotado por sua própria arquitetura de controle embarcada, desprovido de cabo, com fonte própria de energia e com possibilidade de movimento nas três dimensões e orientações (VALAVANIS *et al.*, 1997; XU, ZHANG e FENG, 2004a). Alguns AUVs foram desenvolvidos com o objetivo de funcionarem como bancadas de teste para tecnologias subaquáticas, enquanto que outros, com finalidades específicas de realização de atividades submarinas. Independente da finalidade do veículo subaquático, o projeto e construção deste tipo de dispositivos robóticos envolve a integração de vários campos do conhecimento científico, a exemplo de: posicionamento e navegação; sensoriamento visual; manipulação autônoma; planejamento automatizado de tarefas; aquisição e representação de conhecimento; interface homem-máquina; sistemas de energia; sensores e

processamento sensorial; comunicação; ferramentas de simulação e desenvolvimento; confiabilidade (BLIDBERG *et al.*, 1991).

De acordo com o tempo de autonomia e distância de operação, os AUVs podem ser divididos em veículos de longa ou curta distância (BLIDBERG *et al.*, 1991). Na primeira categoria, enquadram-se os veículos que podem atuar a mais de mil quilômetros entre o seu ponto de lançamento e ponto de retorno, utilizado para, por exemplo, levantamento topográfico de regiões da zona bêntica de oceanos. Por sua vez, a utilização dos AUVs de curta distância está limitada a missões de até 10h de duração, cobrindo pequenas áreas, mas com tarefas diversas bem definidas, como a inspeção de estruturas subaquáticas.

Do ponto de vista da maneabilidade, os AUVs podem ser divididos em veículos de cruzeiro ou de operação (VALAVANIS *et al.*, 1997). Os veículos de cruzeiro, usados para inspeção, pesquisa, localização de objetivos, estão em movimento durante todas as fases da missão, e geralmente requerem controle de apenas três graus de liberdade: velocidade, *pitch* e *yaw*. AUVs de operação, entretanto, podem exigir controle dos seis graus de liberdade, tipicamente necessários para a realização de atividades que exigem a inspeção ou manipulação de estruturas e objetivos em várias posições e orientações, seja com o veículo em movimento ou parado.

Os elementos que o compõem a estrutura física de um AUV podem ser classificados em três tipos principais: *mecânicos* – armação (sustentação), flutuadores, compartimentos herméticos para instrumentação, braço manipulador; *eletromecânicos* – lemes e propulsores; e *eletrônicos* – sensores, placas de aquisição de dados, conversores, processadores, modems, dispositivos e redes de comunicação, baterias. Os sensores são elementos fundamentais para o processo de determinação mais apurada do posicionamento do robô no ambiente, uma vez que todo o funcionamento autônomo do AUV depende das percepções provenientes de sensores que se encontram na própria estrutura do veículo, sendo esses, portanto, os responsáveis em fornecer toda a informação disponível, necessária aos cálculos e estimativas a respeito do estado do ambiente e do veículo. O AUV necessita de um sistema autônomo de células de energia para o acionamento dos propulsores e demais equipamentos embarcados, limitando, assim, o seu período de funcionamento. O movimento do AUV é realizado geralmente por meio de propulsores a hélice, acionados por motores elétricos ou hidráulicos, e por lemes direcionais dispostos ao longo do veículo (TAVARES, 2003). O número e a

distribuição física dos propulsores e dos lemes na estrutura do AUV define o número de graus de liberdades controlados do veículo, sendo necessários seis graus de liberdade para que o veículo possua mobilidade total nas três direções e três orientações.

Responsável em conferir ao AUV a inteligência necessária para atuar na ausência de operador humano, o sistema embarcado do veículo encontra-se composto pelos diversos subsistemas, classificados de acordo com a natureza principal das suas funcionalidades: energético (baterias); atuação (sistema de propulsão); sensoriamento (aquisição e processamento de dados sensoriais); suporte (monitoramento, diagnóstico e acomodação de falhas); localização (posição e orientação); navegação (mapeamento e geração de trajetórias); pilotagem e controle (leis de controle em malha fechada); comunicação (entre os diversos componentes ou entre sistemas externos ao AUV); interface homem-máquina (programação de missões, atualizações, descarga de dados); gerenciamento de missão e tarefa, entre outros (AGUILAR, 2007). O sistema embarcado, composto por tais subsistemas, encontra-se baseado em uma arquitetura robótica, que descreve as estruturas e componentes básicos de todo o sistema, bem como os mecanismos de comunicação entre si (SHAW e GARLAN, 1995; PRESSMAN, 2001).

Portanto, além das dificuldades inerentes à própria dinâmica não-linear, acoplada e sujeita a incertezas e perturbações do AUV, a implementação do sistema de controle através de um sistema distribuído e de natureza tempo-real constitui um importante fator responsável pela complexidade final envolvida no projeto desse tipo de dispositivos robóticos, exigindo uma análise prévia, separada e detalhada de cada um dos principais subproblemas a serem tratados. Com esse objetivo, apresenta-se brevemente nas próximas subseções um resumo destas principais funcionalidades.

### 2.1.1. Controle de Movimento

O problema do controle de movimento do AUV consiste no estudo da controlabilidade de um corpo rígido atuando em um espaço de seis graus de liberdade (ANTONELLI, 2004). Essencialmente, o controlador baseia-se nos modelos contínuos cinemático e dinâmico do AUV, e várias são as técnicas empregadas para o projeto do controlador: PID, controle robusto, controle adaptativo, controle não-linear, controle ótimo, lógica *fuzzy*, redes neurais artificiais, algoritmos genéticos,

controle híbrido, dentre outras (FOSSEN, 1994; YUH, 2000; ANTONELLI, 2004; ROBERTS e SUTTON, 2006).

As equações que regem o comportamento do veículo subaquático encontram-se consolidadas e amplamente difundidas na literatura. O modelo que representa a dinâmica contínua do AUV é altamente não-linear, sujeito a incertezas paramétricas e perturbações externas, e, na medida em que as velocidades de operação do veículo incrementam-se, o acoplamento entre as variáveis do modelo aumenta significativamente (FOSSEN, 1994; SOUZA, 2003; TAVARES, 2003). O modelo contínuo do AUV é descrito pelas equações cinemáticas, que descrevem os aspectos geométricos do movimento, e dinâmicas, resultantes da análise das forças e momentos envolvidos no contexto dos movimentos produzidos (ROBERTS e SUTTON, 2006).

Acoplado ao sistema de controle, pode existir um sistema de pilotagem, responsável por integrar ao controle de movimento a geração de pontos ou trajetórias de referência, levando em consideração as limitações ao movimento do veículo. Algumas estratégias empregam o denominado recurso *line of sight* (LOS), para gerar pontos intermediários para as referências dos controladores de malha-fechada nas rotas planejadas, de modo a reduzir problemas no posicionamento do veículo. Um exemplo desse tipo de configuração é apresentado na figura 2.1, onde à malha de navegação é incluída uma segunda malha de controle para o algoritmo de LOS (CHEN, KOUH e TSAI, 2013).

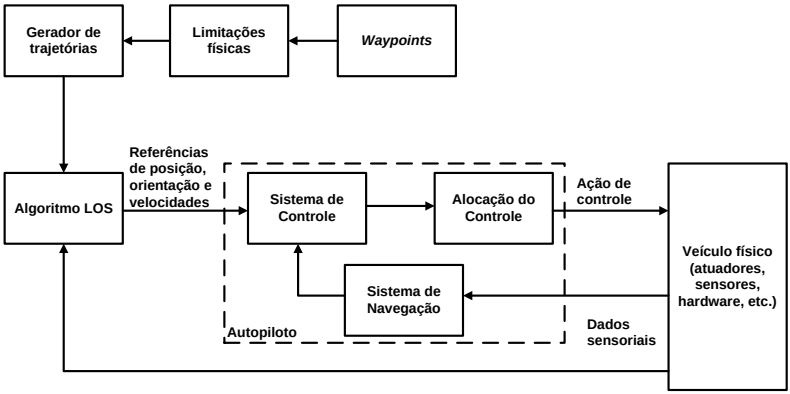


**Figura 2.1:** Sistema integrado de controle e geração de trajetória. Adaptado de Chen, Kouh e Tsai (2013).

A resolução do problema do controle de movimento, portanto, é essencial a qualquer tipo de AUV, e qualquer veículo robótico

autônomo em geral, independente da estratégia empregada para a execução de missões complexas, podendo ser, ou não, explicitamente incorporada à representação de missões.

### 2.1.2. Localização

Dentre as várias competências necessárias à autonomia de um robô móvel, a capacidade de determinar sua localização (posição e orientação) é fundamental, e constitui-se o problema perceptual mais básico em robótica móvel (BAILEY, 2002; THRUN, BURGARD e FOX, 2006). O conhecimento da localização do veículo, bem como áreas, obstáculos e elementos de interesse para as missões, constitui-se em um dos fundamentos básicos para as atividades dos níveis superiores da arquitetura do veículo, tais como planejamento de tarefas, mapeamento ou geração de trajetórias.

Um dos grandes problemas na determinação da localização consiste no fato da maioria, ou às vezes, a totalidade da informação ser proveniente dos sensores embarcados no próprio veículo, medidas estas sempre sujeitas a ruídos e imprecisões (*sensor noise*) e ambiguidades de leituras (*sensor aliasing*) (SIEGWART e NOURBAKHSH, 2004). Sensores são dispositivos imperfeitos com erros de natureza estática e randômica, sendo que esse último tipo de erro não pode ser corrigido, sendo necessário a sua representação estatística (GE e LEWIS, 2006). A informação a respeito da posição e orientação não pode ser medida diretamente, mas sim inferida a partir de dados sensoriais, a exemplo da posição nos eixos $x$ e $y$, uma vez que não existe um sistema inercial capaz de fornecer diretamente o valor dessas grandezas, exigindo processamento adicional, como técnicas de fusão de sensores, com o objetivo de melhor integrar as diversas informações sensoriais (YUH, 2000; THRUN, BURGARD e FOX, 2006). Outro aspecto consiste no fato de que, diferente da robótica terrestre e aérea, a robótica subaquática apresenta o grande inconveniente de não permitir a utilização direta do sistema de posicionamento global (GPS), devido à impossibilidade de transmissão eletromagnética subaquática nas frequências usuais do GPS, e de outros dispositivos sem fio, dificultando ainda mais a tarefa de localização do AUV (ANTONELLI, 2004).

Dentre os vários tipos de sensores existentes, os principais encontrados na maioria dos AUVs correspondem a (SOUZA, 2003; ANTONELLI, 2004; SIEGWART e NOURBAKHSK, 2004; PAULL *et*

*al.*, 2014): unidade de medição inercial ou *inertial measure unit* (IMU), composta por giroscópios e acelerômetros, empregados para a determinação de acelerações e velocidades; sonares diversos para medição de profundidades, velocidades relativas ou detecção de obstáculos, como, por exemplo, *acoustic doppler current profiler* (ADCP), *doppler velocity logger* (DVL), *side scan sonar* (SSS) ou *obstacle avoidance sonar* (OAS); medidor de pressão, a partir da qual é possível obter-se a profundidade; inclinômetros para medida de *roll* (ângulo em torno ao eixo *x*) e *pitch* (ângulo em torno ao eixo *y*); câmeras diversas; modems acústicos; sistemas de localização baseados em *transponders*; GPS; sensores CTD (condutividade, temperatura e profundidade); medidor de PH da água e condutividade elétrica.

Ainda que seja uma atividade fundamental para o veículo móvel, a sua relevância para a missão está mais associada à confiabilidade associada em determinar a posição do veículo no ambiente do que a funcionalidade em si desse subsistema. Por exemplo, é mais importante, do ponto de vista da missão, conhecer se o erro associado ao posicionamento do veículo no ambiente é relativamente elevado a saber o funcionamento interno dos módulos, algoritmos e subsistemas de localização. Assim, como na quase totalidade dos trabalhos científicos analisados durante a revisão do estado da arte, essa tese limita-se a modelar eventos de erros ou de pouca confiabilidade do subsistema de localização, não se ocupando dos aspectos intrínsecos do desenvolvimentos dos algoritmos desse subsistema.

### 2.1.3. Navegação

O problema de navegação em robótica móvel consiste em, a partir do conhecimento parcial sobre o ambiente e posições objetivos, tomar as decisões suficientes para guiar o veículo baseadas nas informações sensoriais, para alcançar de modo seguro e mais eficiente possível as posições objetivos (SIEGWART e NOURBAKHSH, 2004). Nesse sentido, a habilidade de navegação consiste na capacidade de encontrar um caminho entre a posição atual, determinada pelos algoritmos de localização e mapeamento, e a posição final desejada, definida pelos algoritmos de planejamento (MURPHY, 2000). A geração de trajetória constitui-se, portanto, em uma das competências importantes para a autonomia do robô, uma vez que este deve ser capaz de tomar decisões a longo prazo para alcançar seus objetivos.

A navegação também é responsável por gerar as referências para os algoritmos de controle de movimento, sendo usual classificar as trajetórias em dois tipos de curvas: seguimento de trajetória parametrizada no espaço e tempo – *trajectory tracking*, e seguimento de caminho parametrizado somente no espaço, onde o robô deve, a partir de um ponto inicial, alcançar assintoticamente a posição final – *path following* ou *posture regulation* (ENCARNAÇÃO, 2002; SICILIANO *et al.*, 2009). Uma técnica que vem se destacando consideravelmente na última década, consiste na realização simultânea de localização e do mapeamento ou *simultaneous localization and mapping* (SLAM), também conhecida anteriormente como *concurrent mapping and localization* (CML) (BAILEY, 2002; THRUN, BURGARD e FOX, 2006; ROMAGÓS, 2008).

O problema de desvio de obstáculos também se inclui como uma competência de navegação e várias são as estratégias empregadas. Entre as estratégias mais utilizadas, citam-se: algoritmo *bug*, um dos mais simples e consiste em seguir o contorno do objeto até circum-navegá-lo; histograma de campo vetorial, com as probabilidades e possibilidades de existir obstáculos em várias direções; *bubble band*, que define a máxima região sem obstáculos ao redor do veículo; técnicas de curvatura de velocidade, baseadas em restrições cinemáticas e dinâmicas; além de outros tipos (SIEGWART e NOURBAKHSH, 2004; ROBERTS e SUTTON, 2006).

O tipo de navegação – com ou sem mapeamento, navegação por pontos ou baseada em trajetórias, algoritmo de desvio – possui estreita relação com a missão do veículo. Como será visto na seção 2.1.5, uma das atribuições do SCM é de traduzir o arquivo de missão em um plano de missão, contendo os parâmetros associados a cada tarefa ou objetivo, como as referências de posição e orientação ou parâmetros da trajetória desejada, além de estado dos sensores e *timeouts* associados a manobras. O método de desvio de obstáculos, também influi no desenvolvimento e implementação do SCM, pois é necessário combinar as trajetórias de cada etapa da missão com as rotas de desvios.

### 2.1.4. Diagnóstico e Monitoramento

O emprego do AUV em missões de longo período de duração, como o monitoramento de vastas áreas oceânicas ou o acompanhamento prolongado de ecossistemas, exige que o veículo seja capaz de funcionar mesmo ante a presença de falhas inesperadas (RANGANATHAN *et al.*,

2001). Desse modo, durante a operação do AUV existe a possibilidade que seus sistemas operativos apresentem erros, devido à falha elétrica ou mau funcionamento de algum sensor, ou até mesmo devido a alguma perturbação externa do ambiente, como o choque com algum obstáculo não detectado pelo sistema sensorial do veículo. As falhas podem afetar os sensores, gerando erros consideráveis em suas leituras, bem como os atuadores, afetando a influência destes sobre a planta, e as próprias relações entrada-saída dos sistemas envolvidos (LEFEBVRE e LECLERCQ, 2011). Sob essas condições, o sistema de controle embarcado deve ser suficientemente inteligente para detectar tais situações, e ser capaz de encontrar alternativas que possibilitem o veículo a completar a totalidade ou um percentual das tarefas pendentes da sua missão atual, no caso de falhas toleráveis, e, ante falhas graves, comandar que o mesmo retorne à superfície, evitando assim a perda do veículo (PERRAULT e NAHON, 1998; YANG, YUH e CHOI, 1998; PERRIER e KALWA, 2005; ALTAMIRANDA e COLINA, 2007).

Independente das técnicas empregadas para a detecção, isolamento e acomodação de falhas, o sistema de monitoramento e diagnósticos é fundamental para a realização das operações do AUV, principalmente em missões de longo-curso, onde falhas eventuais devem ser contornadas de modo a não prejudicar a realização da missão, ou mesmo a perda do veículo. Sob a perspectiva do controle de missão, a detecção e tratamento de falhas pode ser tratada internamente ao subsistema que se encontra relacionado, quando a gravidade da mesma é considerada baixa ou até irrelevante para o funcionamento do AUV como um todo. Porém, quando tal ocorrência interfere na realização da missão, mesmo sendo possível a correção da falha, essa informação deve ser repassada ao SCM de modo que o mesmo possa decidir pelo replanejamento ou até pelo eventual cancelamento da missão.

### 2.1.5. Controle de Missão

Tradicionalmente, os primeiros AUVs evoluíram de sistemas exclusivamente com controle de movimento, responsáveis por manter o movimento segundo uma trajetória previamente fixada, crescendo em complexidade na medida em que maior inteligência era agregada às competências do veículo, neste caso, localização e mapeamento, até alcançarem os níveis dos atuais veículos com sistemas de controle de missão, que dispõem de ferramentas auxiliares para o planejamento,

visualização, simulação, verificação formal e correção de missões, bem como a sua modificação *on the fly* (DIAS *et al.*, 2006a).

Assim, a grande maioria de AUVs usados para a realização de missões mais complexas – vários objetivos, diversidade nas manobras e funcionalidades, períodos de operação prolongados, ambientes com topologia muito variada ou atuação em times de veículos autônomos – possui algum tipo de estrutura de controle de missão embarcado em seu sistema que permite a codificação e gerenciamento da sequência de operações a serem realizadas durante a missão do veículo. Ao conjunto de estruturas, usualmente divididas em subníveis, responsáveis diretamente pela execução de uma missão denomina-se Sistema de Controle de Missão (SCM).

Basicamente, o ciclo de vida de realização de missões pode ser didaticamente dividido nas seguintes etapas (PINTO *et al.*, 2006; PERDOMO *et al.*, 2010): representação, planejamento, verificação, execução e análise. Na etapa de representação são definidas e codificadas as estruturas que permitem a especificação dos objetivos e ações que irão compor um plano de missão, empregando abstrações das funcionalidades sem descrever como as mesmas são implementadas através da arquitetura do sistema do veículo. O planejamento de missão, geralmente realizado através de ferramentas computacionais, consiste na especificação da sequência de passos e trajetórias de uma missão em particular, descrita em termos das coordenadas do ambiente e de modo compreensível para o usuário. A etapa de simulação e / ou verificação permite, através da análise de simulações ou de métodos de verificação formal, detectar inconsistências e erros na programação da missão. Finalmente, a etapa de execução consiste no envio do arquivo de missão para o veículo, a sua tradução para as operações dos níveis inferiores do sistema embarcado, de modo a gerar os comandos adequados para os atuadores e demais equipamentos do veículo. Posteriormente, a recuperação de registros de eventos e dados de missões permite a análise de resultados obtidos durante a realização de missões.

Nos primeiros veículos robóticos móveis, a missão era descrita diretamente pelas trajetórias ou sequência de pontos a serem seguidos pelo veículo, muitas vezes incluídas como linhas de código do programa em execução no robô. Com a evolução da tecnologia em robótica móvel tornou-se possível incluir informações além da trajetória, como a ativação de equipamentos e comportamentos específicos em determinados pontos da missão, o tratamento de erros, o replanejamento de missões, ou ainda o mapeamento de ambientes desconhecidos.

O planejamento e execução de missões passou a exigir que a organização e a codificação dos objetivos, muitas vezes definidos em termos das competências implementadas pelo veículo, fossem descritas em termos de um vocabulário de ações a serem empregados para a modelagem formal de missões (PERRIER e COSTE-MANIÈRE, 1994; PEBODY, 2007; PERDOMO, 2010). Tal evolução possibilitou o emprego de sintaxes mais complexas, baseadas em pseudo-linguagens, *scripts* ou estruturas de transição, que, contidas em um arquivo denominado de plano de missão, podiam ser enviadas ao AUV através de um canal de comunicação específico, como um cabo *ethernet*, um modem acústico ou até mesmo um *link* com satélite (DIAS *et al.*, 2006a; PERDOMO, 2010).

Assim, vários são os modos de implementação de controladores de missões: missões mais simples podem ser realizadas seguindo-se uma trajetória previamente estabelecida, ao passo que missões mais complexas podem ser coordenadas através de um elemento central tolerante a falhas ou uma estrutura dirigida a eventos responsável pela integridade do veículo, além da realização da trajetória desejada (MARTIN *et al.*, 2006). O SCM pode ser implementado como parte do baixo-nível responsável direto pelo controle de movimento, quando o vocabulário de ações da missão é expresso diretamente em termos de trajetórias ou pontos de navegação e não inclui elementos semânticos relacionados aos demais subsistemas, ou como processo independente que se comunica com os níveis inferiores da arquitetura, quando há a necessidade de representação e coordenação de informações dos diversos módulos, típico das informações de maior nível de complexidade.

Os vários subsistemas envolvidos no funcionamento do veículo podem possuir mecanismos próprios para garantir a robustez perante a ocorrência de eventos não previstos na missão. Porém, muitas vezes tais mecanismos estão circunscritos aos próprios subsistemas que pertencem, não existindo um elemento responsável em coordenar a sequência completa de atividades do veículo Portanto, uma das vantagens da utilização de um SCM reside na possibilidade de definir a ação a ser executada por um módulo em função do estado de um terceiro, como por exemplo, o replanejamento *on-line* de missões ou reações ante eventos não-deterministas, evitando o cancelamento de missões ao mesmo tempo em que a integridade do veículo é garantida.

## 2.2. PLANEJAMENTO DE MISSÕES EM DOMÍNIOS NÃO-ESTRUTURADOS

Um conceito chave no funcionamento autônomo de veículos robóticos consiste na habilidade de planejar o seu próprio movimento em um ambiente conhecido ou não, com o objetivo de realizar alguma atividade (GE e LEWIS, 2006). Por planejamento entende-se desde a geração e seguimento de trajetórias consideradas seguras e livre de obstáculos, até a ativação coordenada de diversos equipamentos embarcados do veículo, como sensores, câmeras, ou ainda, o funcionamento integrado de times de robôs.

O problema de planejamento envolve o emprego de técnicas de três áreas do conhecimento: Inteligência Artificial (IA), Sistemas de Controle e Robótica (LAVALLE, 2006). Na robótica, o termo planejamento está muito ligado ao problema de gerar movimentos – trajetórias – a partir de modelos geométricos complexos. Na IA, o foco consiste no projeto de sistemas que usam modelos teóricos de decisão empregados para o cálculos de ações apropriadas. E na área de controle, por sua vez, o enfoque do planejamento está relacionado ao cálculo de trajetórias fisicamente realizáveis ao mesmo tempo em que aspectos de desempenho e otimização sejam atendidos.

Devido à multidisciplinaridade de áreas científicas associadas à robótica, não há um consenso quanto ao termo planejamento, assim, propõe-se, com finalidades exclusivamente didáticas, abordar o problema de planejamento de missões sob dois pontos de vistas diferentes: planejamento do movimento e planejamento de tarefas.

### 2.2.1. Planejamento de Movimento

O planejamento do movimento, consiste na escolha de trajetórias, caminhos ou pontos (*waypoints*) adequados, gerando-se referências para as malhas-fechada de controle nos níveis inferiores do veículo, empregando ou não, um sistema de mapeamento. Nesse tipo de problema, o robô alcança os seus objetivos através da realização de movimentos, geralmente sob restrições do ambiente e obstáculos, mas também sob limitações inerentes às próprias características do veículo, denominadas de restrições não-holonômicas (GE e LEWIS, 2006). Assim, dado uma geometria do ambiente e as características do veículo, o problema de planejamento consiste em encontrar um caminho (não parametrizado no tempo) ou uma trajetória (parametrizada no tempo),

livre de colisões e que guie o veículo a partir do seu ponto inicial em direção ao ponto final.

Nessa abordagem, a modelagem do problema está diretamente relacionada com as equações dinâmicas do movimento em si do veículo, das características do ambiente, como presença de obstáculos, irregularidades no ambiente ou correntes (robôs subaquáticos), e são fundamentadas na área de Sistemas Dinâmicos e Sistemas de Controle.

Tradicionalmente, muitos sistemas de controle de trajetória para AUVs são divididos funcionalmente em três subsistemas (ENCARNAÇÃO, 2002): *localização* – estimação da posição e orientação do veículo; *guiagem* – gerar as trajetórias de referências para os controladores; *controle* – para gerar as ações de controle dos atuadores com base nas referências desejadas. O desenvolvimento de tais subsistemas pode ser feito separadamente, contudo, abordagens mais sofisticadas e complexas vem possibilitando o desenvolvimento em conjunto dos três subsistemas.

Outro enfoque ao problema de planejamento de trajetórias consiste no emprego de mapas discretos do ambiente e geração de caminhos a partir desses nós, o que é classificado na literatura como sendo um problema de navegação (GE e LEWIS, 2006). Nesse enfoque, a abordagem mais comum consiste em transformar a representação contínua do ambiente em um mapa discreto, a partir do qual é gerado um grafo de conectividade entre os pontos livres de obstáculo, e com base nesse grafo, aplica-se um algoritmo de busca para obtenção de um caminho em direção ao estado final desejado. Inúmeras estratégias são empregadas nesse tipo de planejamento de trajetórias, entre elas: mapas de rotas, decomposição em células e campos potenciais (MURPHY, 2000; SIEGWART e NOURBAKHSH, 2004). Uma grande vantagem desse enfoque de planejamento de trajetória consiste na possibilidade do uso de vários algoritmos de busca baseado em grafos da IA clássica, como a busca em largura, em profundidade, A*, entre outros. Contudo, tal enfoque ignora completamente as características cinemáticas e dinâmicas do veículo autônomo em questão.

### 2.2.2. Planejamento de Tarefas

O planejamento de tarefas refere-se à representação do problema de tomada das várias decisões relacionadas com a operação do veículo robótico, contudo, não se restringindo única e exclusivamente ao movimento do veículo, ainda que a possa incluir. Exemplo desse tipo de

decisões consiste nas trajetórias do veículo, mas também em aspectos envolvendo a segurança do veículo, a otimização de custos, a coleta de dados, a exploração de áreas e as atividades de comunicação. Devido à inexistência de um consenso sobre o termo que descreve o conjunto de ações definidos nos níveis mais elevados de uma arquitetura robótica, esta tese opta pelo termo tarefa para designar esse conjunto de ações.

Ao conjunto de tarefas, especificado pelo usuário e usualmente armazenado em um arquivo de dados, denomina-se de missão, e é decodificado por algum método de planejamento localizado no SCM em um plano de missão. Essa decodificação ocorre através do mapeamento do vocabulário de ações de alto-nível, com os objetivos da missão, em sentenças e comandos entendíveis pelos níveis inferiores do veículo. O planejamento sob essa perspectiva centra-se, assim, no emprego da representação e nos algoritmos para escolha de ações, como ligar ou desligar um sensor, realizar uma unidade de movimento (translação ou rotação), ativar ou desativar um modo de operação, e também podem incluir a geração de trajetórias.

A diversidade de operações em missões complexas aponta para a necessidade de uma linguagem de missões que permite a especificação da missão em um arquivo e posterior envio ao veículo robótico, antes ou durante a realização de missões, e que no caso contrário, teria que ser programado diretamente no código-fonte do sistema do robô (KIM *et al.*, 2010). Particularmente em ambientes não-estruturados parcial ou totalmente desconhecidos, com ocorrência não-determinista de eventos, como falhas, ou presença de obstáculos, o veículo necessita ser mantido em segurança em qualquer ponto da missão, contudo devido ao não-determinismo nem sempre é possível incluir todas as possibilidades de ocorrências e situações em um arquivo de missão. Tais características exigem capacidades de reatividade e deliberação, como o planejamento e o replanejamento das ações do veículo. Assim, um método para especificação, validação e garantia de execução de um plano de missão pré-estabelecido, mas flexível, torna-se uma vantagem importante e desejável para os sistemas de operação de AUVs. Usualmente, empregam-se *scripts* sequenciais, pseudo-linguagens e estruturas de transição como recursos para representação de missões, o que permite a descrição da missão em termos de conhecimentos específicos dos usuários, como sistemas de coordenadas, valores de velocidade, ativação de operações, sem que o usuário, obrigatoriamente, conheça os detalhes de implementação do veículo.

Contudo, torna-se necessário traduzir os objetivos e parâmetros da missão em termos dos componentes de software e hardware

distribuídos nos diversos níveis que compõem o sistema de controle do veículo. Tal tradução é possível através da organização das operações e subsistemas do dispositivo robótico em níveis. Desse modo, as camadas inferiores de controle do veículo proveem um vocabulário semântico aos níveis superiores, servindo de base para a descrição e planejamento de missões do AUV, diminuindo, assim, a complexidade na programação de missões em função da abstração que o sistema fornece (PEBODY, 2007). Assim, é possível empregar um mecanismo de abstração – a linguagem de missão – para representar as operações definidas em termos das capacidades e habilidades do veículo, expressas em termos dos níveis mais elevados das arquiteturas robóticas.

### 2.2.3. Planejamento baseado em Algoritmos de Busca

Uma abordagem fundamental para resolução do problema do planejamento de tarefas consiste no uso de técnicas da Inteligência Artificial (IA) clássica, mais especificamente de resolução de problemas (*problem solving*) e algoritmos de busca (*search*). O termo planejamento em IA significa a ação de procurar por uma sequência lógica de operadores (ações) que irão transformar um estado inicial do mundo em um estado final desejado (RUSSELL e NORVIG, 2003; LAVALLE, 2006). O agente é o elemento, geralmente um processo que implementa um algoritmo, que mapeia as percepções do ambiente, bem como de seus estados internos, em ações, e é especializado na resolução de um determinado tipo de problema.

Um exemplo típico dessa abordagem aplicada à navegação de veículos robóticos móveis pode ser visto na figura 2.2. Nela, o ambiente em duas dimensões é transformado em um mapa discreto (figura 2.2-a), onde células brancas indicam caminhos livres de obstáculos e células negras, a presença de obstáculos. Em cada célula, o robô pode escolher uma entre oitos direções possíveis (figura 2.2-b). Abstrai-se aqui o método de construção ou obtenção do mapa (*off-line*, quando o mapa é conhecido *a priori*, ou *on-line*, quando o mesmo é obtido a partir das leituras sensoriais). A árvore de busca, representando o espaço de estados ou o problema é então construído (figura 2.2-c), e a busca pela solução consiste em encontrar o menor caminho até a célula objetivo. Vários algoritmos de busca podem então ser aplicados sobre o grafo obtido, entre eles: busca em profundidade, busca em largura, Dijkstra, A* e seus variantes (RUSSELL e NORVIG, 2003; LAVALLE, 2006).

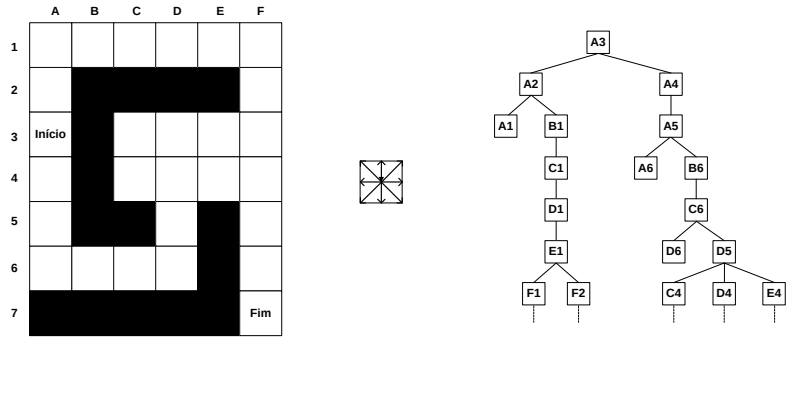


**Figura 2.2:** Mapa em duas dimensões discretizado, direções permitidas de movimento e árvore de busca. Adaptado de (LIKHACHEV *et al.*, 2005).

Contudo, um aspecto essencial, relacionado com as limitações de processamento do hardware, consiste no tempo para execução dos algoritmos de planejamento. Muitas vezes a adoção de uma algoritmo que encontre uma solução ótima ao problema nem sempre é viável, devido às limitações de tempo impostas, sendo necessário definir um compromisso entre a qualidade da solução e as especificações de tempo-real relacionadas às aplicações robóticas. Além disso, a dificuldade em se obter um mapa relativamente preciso, e com uma grande quantidade de dados em três dimensões, como os encontrados na robótica subaquática, acaba restringindo a validade da solução ótima, sendo mais um motivo para emprego de algoritmos mais eficientes mas com soluções não-ótimas. Assim, vários trabalhos propõem o uso de variantes do clássico algoritmo A*, como D* e variantes (STENTZ, 1995), *Anytime Dynamic* A* (LIKHACHEV *et al.*, 2005), *Tree Adaptive* A* (HERNÁNDEZ *et al.*, 2011), de modo a relaxar as condições ótimas do algoritmo porém aumentando o seu desempenho.

Inúmeras abordagens, algumas baseadas em outras áreas da IA, como redes neurais artificiais, lógica *fuzzy* ou algoritmos genéticos, também são aplicados ao problema de planejamento de trajetórias, porém fogem ao escopo desse trabalho.

## 2.3. ARQUITETURAS ROBÓTICAS

A arquitetura representa a estrutura de dados e os componentes de software, suas propriedades e forma de interação e comunicação entre tais componentes, necessários para o desenvolvimento de um sistema computacional, sem caracterizar, entretanto, as operações do programa em si ou sua implementação (SHAW e GARLAN, 1995; PRESSMAN, 2001; OREBÄCK e CHRISTENSEN, 2003). Uma arquitetura determina o modo de composição do sistema a partir de elementos menores, sejam eles programas, módulos ou subsistemas. A adoção de uma arquitetura permite definir critérios claros para a construção de um sistema complexo de natureza tempo-real e distribuído, garantindo aspectos como modularidade, portabilidade e robustez para o sistema (MURPHY, 2000). A arquitetura determinando a sequência de execução dos componentes individuais e o fluxo de informação entre eles (XUEMEI, 2007). Nesse sentido, a arquitetura permite aos projetistas e desenvolvedores uma análise mais criteriosa da eficiência do projeto em atender às especificações, reduzindo os riscos associados à implementação do sistema.

Vários critérios podem ser empregados na escolha de uma arquitetura robótica mais adequada às necessidades de cada sistema, estando os mesmos, geralmente, associados a requisitos básicos de controle e desempenho de execução, bem como aspectos de engenharia de software (BROOKS, 1986; VALAVANIS *et al.*, 1997; MURPHY, 2000; OREBÄCK e CHRISTENSEN, 2003; LIN *et al.*, 2010). Desse modo, considera-se como características desejáveis a uma arquitetura de controle de robôs móveis: abstração do sistema com respeito ao hardware do robô; extensibilidade e escalonabilidade permitindo a inclusão de novas funcionalidades, módulos e equipamentos; desempenho adequado com respeito às imposições de natureza tempo-real; modelos adequados para dados sensoriais e atuadores, propiciando previsibilidade no movimento e funcionamento do robô; robustez perante incertezas e situações inesperadas; tolerância a falhas; modularidade e reutilização de código; ferramentas adequadas para o seu desenvolvimento; dentre outras (MURPHY, 2000; OREBÄCK e CHRISTENSEN, 2003).

A organização do sistema de controle em camadas permite a combinação de várias formas de representação do conhecimento, onde cada nível pode possuir um nível de abstração distinto, aumentando em complexidade e em capacidade de processamento na medida em que se avança dos níveis inferiores em direção aos níveis superiores, ao mesmo

tempo em que os tempos de resposta tornam-se maiores e menos restritivos (GE e LEWIS, 2006). As funções que exigem ações extremamente rápidas requerem processadores dedicados de modo a garantir os períodos de amostragem relativamente pequenos. As tarefas que exigem maior capacidade de processamento e maior quantidade de dados, por sua vez, representam modificações no estado do veículo em um horizonte de tempo relativamente maior, o que permite relaxar as restrições temporais associadas a tais tarefas.

Em robótica móvel, as arquiteturas seguem princípios básicos e um conjunto de técnicas que permitem a sua classificação de acordo com as relações entre as três primitivas da robótica e pelo modo em que os dados sensoriais são tratados e redistribuídos através do sistema (MURPHY, 2000). As três primitivas da robótica classificam as funcionalidades ou operações do robô em: medir / sensoriamento (*sense*) – transformação das leituras sensoriais em dados para as demais operações do robô; planejar / planejamento (*plan*) – transformação da informação e posterior produção de tarefas a serem realizadas; e agir / ação (*act*) – geração de comandos para os motores e atuadores do veículo. Atualmente, as arquiteturas em robótica móvel são classificadas em três tipos principais, sendo elas: hierárquica ou deliberativa, reativa ou comportamental e híbrida deliberativo-reativa (BROOKS, 1986; VALAVANIS *et al.*, 1997; MURPHY, 2000; OREBÄCK e CHRISTENSEN, 2003; SIEGWART e NOURBAKHSH, 2004; LIN *et al.*, 2010).

A arquitetura hierárquica, ou deliberativa, primeira arquitetura robótica móvel desenvolvida, baseia-se em uma abordagem *top-down*, inicialmente com foco no processamento monolítico e sequencial da informação, além de dividir o sistema em níveis (MURPHY, 2000). Nesta arquitetura, os níveis superiores estão encarregados do controle global da missão, delegando funcionalidades e tarefas específicas aos níveis inferiores, onde a comunicação entre as diversas camadas somente está permitida entre dois níveis adjacentes da arquitetura (VALAVANIS *et al.*, 1997). Na camada inferior, geralmente encontra-se a implementação das malhas-fechadas de controle, aquisição de dados e suporte ao processamento tempo-real.

A figura 2.3 apresenta a estrutura geral da arquitetura hierárquica. Cada nível recebe como entrada os dados processados ou os comandos da camada anterior, com exceção da primeira camada que capta os dados dos sensores. A partir de um conjunto de leituras provenientes dos sensores (primitiva *sense*), os dados são processados e ações são tomadas (primitiva *plan*) e enviados através de cada um dos níveis até

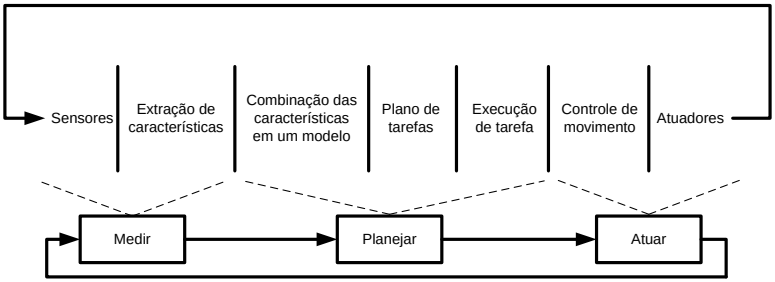


**Figura 2.3:** Decomposição sequencial de tarefas da arquitetura hierárquica. Adaptado de Murphy (2000).

chegarem aos atuadores (primitiva *act*), portanto, as operações de transformação sobre os dados ocorrem de modo sequencial.

Os três principais níveis ou componentes usualmente encontrados na arquitetura hierárquica correspondem ao planejador de missão, geralmente baseado em autômatos ou redes de Petri, o sistema de navegação e localização, ou navegador, e, no nível mais inferior, o piloto responsável pelo controle de movimento (MURPHY, 2000). Este tipo de arquitetura apresenta como grande desvantagem o tempo relativamente grande para tomada de decisão, pois toda ação é obtida a partir de todos os dados obtidos e processados de modo sequencial pelos demais níveis de arquitetura (VALAVANIS *et al.*, 1997).

A arquitetura reativa, ou comportamental, consiste na ação combinada de vários comportamentos em paralelo sem, entretanto, possuir um supervisor de nível superior (LIN *et al.*, 2010). Arquiteturas reativas variam desde simples redes de controle, correspondendo a robôs móveis com capacidades relativamente restritas, como por exemplo aqueles que mimetizam o comportamento de insetos, tornando-se mais complexa, na medida em que mais camadas de comportamentos vão sendo incluídas no sistema de controle (LIMSOONTHRAKUL, 2009). Nesse tipo de esquema, a inteligência global do robô encontra-se determinada pelo nível de competência expresso pelos diversos comportamentos que o dispositivo apresenta. Logo, quanto maior for a complexidade exigida, maior será o número de comportamentos que o robô deverá implementar.

No paradigma reativo, qualquer ação realiza-se através de um ou mais comportamentos. Um comportamento pode ser entendido como o mapeamento direto entre as entradas sensoriais, a primitiva *sense*, e as ações, primitiva *act*, que o robô deve realizar para alcançar uma tarefa ou objetivo específico, cujo funcionamento é totalmente independente

dos demais comportamentos. Funcionalidades como “desviar obstáculo”, “seguir uma parede”, “manter distância” e “gravar vídeo” são exemplos de possíveis comportamentos em um veículo móvel não tripulado. Uma das arquiteturas reativas mais conhecidas é a subsunção (BROOKS, 1986) e, muitas vezes, é confundida como sinônimo de arquitetura reativa. Nessa arquitetura, comportamentos são agrupados em módulos descritos, muitas vezes, por máquinas de estados finitos, e representam níveis de competência, onde, os comportamentos situados nos módulos de maior nível podem sobrescrever, ou subsumir, a saída dos comportamentos dos níveis inferiores.

Uma das principais características, e vantagens, da arquitetura reativa consiste na efetiva rapidez de resposta, ou reatividade, do sistema frente aos diversos estímulos provenientes do ambiente, sem a necessidade de guardar e processar uma representação global para a evolução dos estados do ambiente. Entretanto, essa limitação na representação do conhecimento possui como desvantagem a inépcia do robô em planejar tarefas a médio e longo prazo, uma vez que algoritmos inteligentes exigem processamento elevado e acesso a grande quantidade de dados globais, falhando, desse modo, na obtenção de objetivos não triviais (LIN *et al.*, 2010).

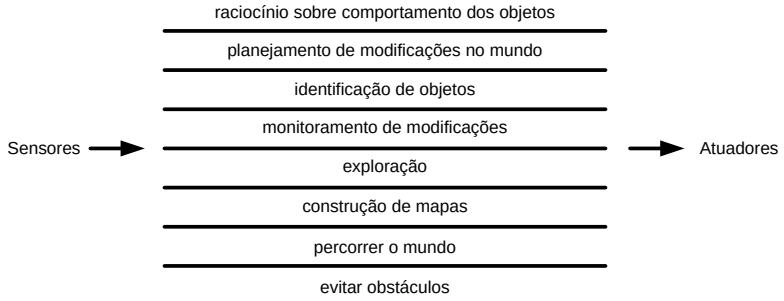


**Figura 2.4:** Decomposição baseada em comportamentos em uma arquitetura reativa. Adaptado de Brooks (1986).

A figura 2.4 apresenta uma decomposição, encontrada em arquiteturas reativas, das tarefas de um sistema de controle de um robô móvel segundo os vários comportamentos que o mesmo é capaz de realizar. Cada nível corresponde a um mapeamento direto entre a primitiva de sensoriamento de dados (*sense*) e a de ação sobre os atuadores (*act*), não dependendo do funcionamento e dados dos demais níveis.

Resultado da combinação das arquiteturas hierárquica e reativa, a arquitetura híbrida, ou deliberativo-reativa, é reconhecida como sendo a mais adequada e também a mais empregada em veículos não tripulados (OREBÄCK e CHRISTENSEN, 2003; LIN *et al.*, 2010). Nessa abordagem, as atividades como planejamento de trajetória, mapeamento de ambientes, tomada de decisão, tipicamente deliberativas, que exigem elevada capacidade de processamento, são implementadas sem interferir nos comportamentos reativos, cujos tempos de resposta são mais rápidos devido a sua natureza tempo-real.

Arquiteturas híbridas encontram-se também divididas em camadas, usualmente, em duas ou três, compreendendo as atividades de nível inferior, o controle de movimento, e, nos níveis superiores, as partes envolvendo tanto funcionalidades reativas e quanto deliberativas. Entre as funcionalidades reativas e deliberativas, existe um componente atuando ao modo de sequenciador, coordenador ou supervisor, responsável em intermediar os dois tipos de operações (MURPHY, 2000; OREBÄCK e CHRISTENSEN, 2003). Os comportamentos reativos são executados até que o plano ou a etapa da missão seja completado e o sequenciador, ou um planejador, volte a ativar um novo conjunto de comportamentos. Desse modo é possível identificar como componentes comuns nas arquiteturas híbridas: planejador de missão, encarregado de coordenar a atividade global do veículo em função das várias etapas da missão; sequenciador, responsável por gerar o conjunto de comportamentos ativos a partir do plano geral da missão; gerenciador de recursos; cartógrafo, responsável por criar, armazenar e atualizar a representação do mundo; e monitoramento de desempenho e solucionador de problemas.

Uma das grandes vantagens encontradas na arquitetura híbrida reside na possibilidade de realização de tarefas complexas, que exigem maior capacidade de processamento, mantendo, contudo, ao mesmo tempo, a flexibilidade exigida pelas mudanças dinâmicas em ambientes através dos comportamentos reativos (VALAVANIS *et al.*, 1997). O emprego de técnicas de processamento assíncrono permite, assim, a realização de tarefas deliberativas independentes dos comportamentos reativos (MURPHY, 2000).

A estrutura de componentes distribuídos de software e hardware que comanda o AUV é, portanto, organizada em níveis hierárquicos, denominada de arquitetura robótica, e cada nível compõe um conjunto de funcionalidades, favorecendo o projeto, análise, simulação e implementação dos sistemas embarcados para aplicações robóticas. A composição de funcionalidades em níveis permite a quebra da sequência

de objetivos em tarefas e estas, por sua vez, em unidades mais elementares de ação, geralmente em sequências pré-configuradas de operações, de acordo com o tipo de arquitetura robótica adotada. O SCM deve ser capaz, portanto, de mapear cada uma das sentenças contidas no arquivo de missão em ações correspondentes que irão gerar o comportamento ou funcionamento desejado para o AUV, levando em consideração o tipo de arquitetura implementada, incluindo a natureza híbrida do seu sistema embarcado.

## 2.4. RESUMO

Veículos subaquáticos autônomos vêm sendo empregados em diversas atividades, cada vez mais complexas, exigindo o desenvolvimento de uma estrutura que possibilite a especificação e execução de missões mais sofisticadas, denominada de Sistema de Controle de Missão (SCM). Problemas relacionados à navegação, localização, controle de movimento, monitoramento e diagnóstico de falhas, e controle de missão do AUV são tratados por diversos subsistemas, constituídos por componentes de hardware e/ou software e organizados em uma arquitetura robótica distribuída. Independente do tipo e natureza das funcionalidades implementadas pelo veículo autônomo, a organização das mesmas, combinando ações de planejamento (deliberativas) com processamento mais intenso e tempos de resposta relativamente maiores, e comportamentos reativos, sem planejamento e tempos de execução curtos, define a arquitetura robótica, da qual o SCM é um componente.

A adoção de organização em camadas permite a separação dos aspectos de representação, planejamento e execuções de missões, expressas através de uma semântica de alto nível, dos aspectos relacionados à implementação e funcionamento dos módulos inferiores da arquitetura do veículo. Desse modo, como será visto no capítulo seguinte, é possível empregar SEDs como formalismo para a composição de arquiteturas de robôs autônomos, fornecendo métodos e técnicas para a especificação, análise e execução de missões de AUVS atuando em ambientes não-estruturados.

# 3. ARQUITETURAS DE SCM BASEADAS EM SEDS

Ainda que o emprego de modelos formais baseados em SEDs em missões de AUVs já venha sendo feito desde a década de 1990, como em Wang *et al.* (1991), observa-se a não existência de um método bem definido ou estabelecido referente ao problema de missões em AUVs, ou, eventualmente, outros tipos de veículos robóticos autônomos, haja vista as inúmeras e diferentes propostas encontradas na literatura científica. Também se constata a relativa pouca quantidade de artigos sobre o problema de controle de missão de AUVs, se comparado aos problemas de construção, controle de movimento, localização, navegação e detecção de falhas. Tal escassez de material pode ser atribuída, por hipótese, a três principais fatores: a evolução natural dos sistemas dos AUVs a partir do problema de modelagem e controle em direção aos demais problemas; a complexidade na implementação dos sistemas embarcados, notadamente os problemas de tratamento sensorial e processamento tempo-real; e na dificuldade inerente na modelagem e implementação de sistemas híbridos.

Na sequência são apresentados as principais utilizações encontradas na literatura para a resolução de problemas de missão em veículos robóticos móveis, incluindo AUVs, baseados em SEDs. Tais artigos apresentam a organização hierárquica dos sistemas de controle propostos e/ou implementados, porém a maioria das funcionalidades são abstraídas e, por isso, não descritas em detalhes, quando não completamente omitidas nos trabalhos. Assim, optou-se pela apresentação das principais características de cada proposta, como tipo de arquitetura, tipo de SEDs empregado, sem aprofundar os detalhes de implementação e de algoritmos específicos dos demais subsistemas do veículo robótico móvel.

A seção 3.1 apresenta diversas arquiteturas que empregam a Teoria de Controle Supervisório (TCS) para tratar o problema de controle de missão. Na sequência, a seção 3.2 mostra algumas soluções baseadas no uso de redes de Petri para modelagem e execução de missões. Na seção 3.3, estratégias baseadas em Sistemas Híbridos incorporadas ao controle de missão são apresentadas. Na sequência, na seção 3.4, com intuito de identificar e evidenciar as principais vantagens e limitações no uso de SEDs em veículos subaquáticos, uma análise das

diferentes abordagens é realizada. Ao final do capítulo, um resumo com os pontos essenciais é apresentado.

## 3.1. ARQUITETURAS BASEADAS NA TEORIA DE CONTROLE SUPERVISÓRIO

Na Teoria de Controle Supervisório (TCS) são empregados modelos formais baseados em autômatos e linguagens para descrever o comportamento em malha aberta, ou planta, e as especificações para o seu funcionamento (RAMADGE e WONHAM, 1989). Um autômato pode ser representado por um grafo, onde os vértices correspondem a estados, e os arcos orientados entre os vértices, transições. O supervisor é o elemento responsável em observar a evolução da planta e, através da habilitação e desabilitação de eventos, garantir que as especificações pré-estabelecidas sejam satisfeitas. No próximo capítulo, a TCS será formalmente apresentada, incluindo conceitos importantes como a controlabilidade.

Em Xu, Zhang e Feng (2004a) e Xu, Zhang e Feng (2004b) uma estrutura de controle supervisório, baseada na TCS, organizada em níveis hierárquicos é apresentada. Em Zhang *et al.* (2006) uma abordagem simular é aplicada a um veículo subaquático planador (*glider*). Através de uma arquitetura de duas camadas, as operações do veículo são divididas em controle de movimento, correspondendo ao nível inferior da arquitetura, e em controle de missão. O controle de missão, apresentado na figura 3.1, é organizado, por sua vez, em três subníveis, cada um descrito por um ou mais autômatos específicos: comportamento, planejamento de tarefa e planejamento de missão.

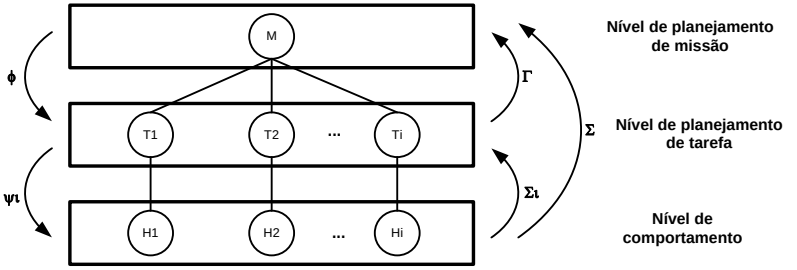


**Figura 3.1:** Organização lógica da camada superior de planejamento. Adaptado de Xu, Zhang e Feng (2004a).

O nível de comportamento é modelado pelos autômatos $H_i$ e representam funcionalidades abstraídas do veículo, em função das operações do nível inferior de controle de movimento, como por exemplo ligar ou desligar sensores, detectar se o AUV está na água ou não, funcionamento do GPS, movimento em direção a um ponto no plano *X*-*Y*, ir a uma profundidade específica, etc. O nível de planejamento de tarefa, composto por vários supervisores $T_i$ independentes entre si, são responsáveis pelo gerenciamento dos comportamentos $H_i$. Finalmente, o supervisor global *M* do nível de planejamento de missão é responsável pela coordenação dos vários supervisores $T_i$. Na camada de planejamento são implementados nove supervisores, cada um correspondendo a um estado no supervisor da camada de missão, que define a sequência de ativação de cada uma das tarefas.

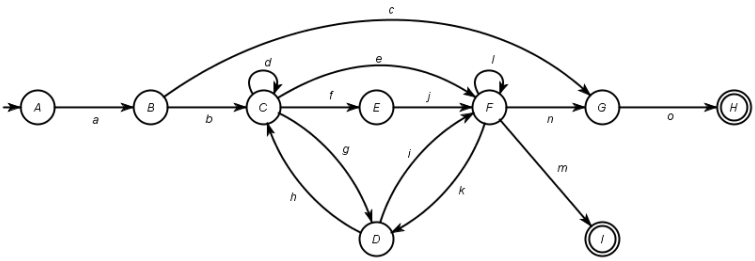


**Figura 3.2:** Autômato do supervisor de planejamento de missão. Adaptado de Xu, Zhang e Feng (2004b).

A figura 3.2 apresenta o supervisor global *M* de missão, onde cada um dos nove estados corresponde a uma tarefa específica, como lançamento ou retorno do veículo, navegação em direção à área objetivo e ajuste de posição por GPS, e as transições correspondem a eventos do tipo início, fim e erro de operação. Cada estado, por sua vez, também é descrito por um autômato $T_i$. Assim, a missão é criada pela composição hierárquica das tarefas implementadas pelo veículo. Como vantagens do uso da TCS, segundo os autores, citam-se: obtenção de um modelo para o sistema de controle e seus subsistemas em múltiplos níveis, a modularidade do sistema e o aumento da precisão no nível lógico. Entretanto, os autores não sistematizam um método para o uso da TCS, não explicando a obtenção dos autômatos de missão e de cada uma das tarefas que o compõe, e como tais autômatos são implementados através da arquitetura, ainda que exemplos de simulação e implementação tenham sido apresentados. Assim, uma abordagem ou metodologia mais

sistemática para o desenvolvimento de SCM baseada na TCS é ainda uma questão em aberto.

O artigo de Molina *et al.* (2010) aborda o problema de navegação de um robô móvel terrestre em um ambiente estruturado e totalmente conhecido. O ambiente em que o veículo navega é apresentado na figura 3.3. Observa-se que os modelos possuem como estado a posição do veículo dentro do ambiente, porém não qualquer posição, mas somente aquelas que correspondem a unidades de movimento entre os corredores. Tal recurso reduz a complexidade da representação do ambiente, porém fixa o posicionamento e o deslocamento do veículo somente àquelas pré-estabelecidas.

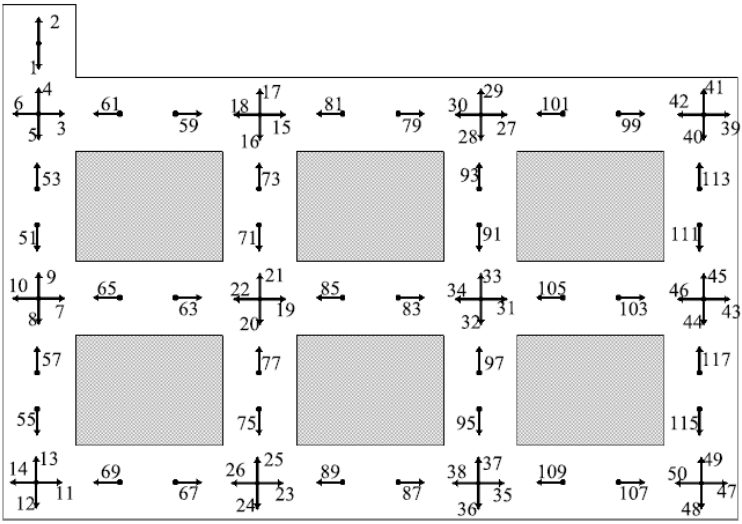


**Figura 3.3:** Representação da posição do robô no ambiente (MOLINA *et al.*, 2010)

A arquitetura do SCM é do tipo hierárquica e dividida em três camadas: planejador, supervisor e robô-ambiente. Comportamentos básicos, como rotação e translação desacoplados, replanejamento e inicialização, são descritos por autômatos, e com tais modelos é possível representar todas as combinações possíveis de movimento do veículo dentro do ambiente. Os eventos representam a detecção de obstáculos e os movimentos possíveis de serem realizados, como avançar 2 ou 5 metros, girar -90 ou +90º, e as especificações, as limitações do movimento do robô com respeito à planta. A planta é descrita pelo pro-

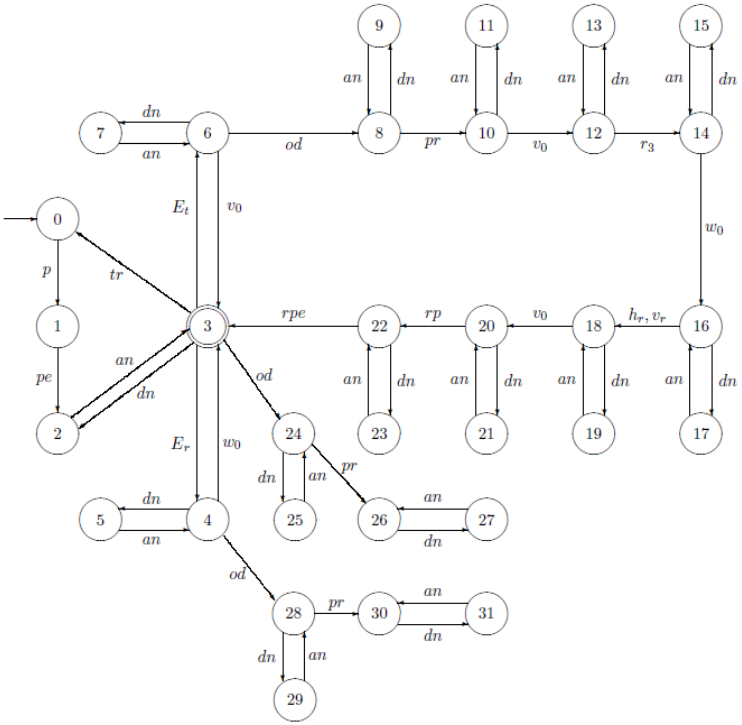


**Figura 3.4:** Produto síncrono de todos os comportamentos possíveis (MOLINA *et al.*, 2010)

duto síncrono de todos os comportamentos básicos e é mostrado na figura 3.4. O planejador é o responsável em escolher a melhor trajetória que leva o robô de um estado (posição inicial) até outro estado (posição de destino), através de um algoritmo de ordenação de tarefas. O nível de supervisão é responsável pela execução de sequências de comportamentos permitidos, e o nível robô-ambiente, omitido no artigo, corresponde ao controle do veículo. Uma desvantagem dessa abordagem, consiste na necessidade do conhecimento *a priori* completo do ambiente, sendo necessário realizar modificações na modelagem e no sistema a cada modificação do ambiente. O sistema é testado mediante simulações com software MobileSim do fabricante de robô P3-DX.

O trabalho de Liu e Darabi (2002) apresentam uma proposta para evitar colisões entre dois robôs terrestres movendo-se em um mesmo ambiente estruturado e totalmente conhecido. Cada robô possui um sensor de tato que é empregado para detectar colisões com o ambiente e

com o outro robô. Através dos eventos de início e parada de movimento do veículo e de detecção de colisão, são criados os modelos que representam a atuação livre dos dois robôs no ambiente. Por meio de especificações, que definem a condição de parada dos veículos ante colisões, são obtidos os supervisores, e estes implementados em um computador do tipo PC que constantemente comunica-se com os robôs móveis do tipo Arrick Robotics©. Ainda que apresente aspectos de implementação e das plataformas empregadas, o artigo não descreve como é feito o planejamento e a execução de um plano de missão. O artigo também comenta as principais vantagens na implementação de sistemas de supervisão baseados na TCS, como a rapidez e eficiência na execução das estruturas de controle supervisório. O trabalho também aborda os principais obstáculos na implementação da TCS, notadamente, a dificuldade em mapear os modelos formais do supervisor e planta para o sistema real.

## 3.2. ARQUITETURAS BASEADAS EM REDES DE PETRI

Redes de Petri são modelos formais que vêm sendo empregados no problema de missões de AUVs de vários modos. As redes de Petri possuem uma representação gráfica que faz uso de lugares, arcos, transições e fichas para representar de modo bastante intuitivo o comportamento de SEDs (CASSANDRAS e LAFORTUNE, 2008). Além disso, há uma série de métodos formais bem estabelecidos para a modelagem e análise de propriedades em redes de Petri.

A seguir são brevemente apresentados os principais tipos de usos de redes de Petri em AUVs, como: representação e coordenação paralela de atividades ou componentes do veículo; composição de comportamentos, operações e tarefas; tradução entre as linguagens de eventos entre os vários submódulos do sistema; e descrição de missões em termos das capacidades do veículo.

Em Chang *et al.* (2005) apresenta-se uma arquitetura híbrida para o sistema de controle de missão de AUVs baseado no trabalho de Wang *et al.* (1991), originalmente descrito para robôs móveis. A ênfase nesse trabalho consiste no emprego de redes de Petri para a coordenação de tarefas. Baseadas na teoria de Controle Inteligente Hierárquico, cuja hierarquia de componentes é mostrada na figura 3.5, as redes de Petri são usadas para transladar um determinado plano de tarefa de entrada em um plano de tarefa de saída, funcionando como tradutores de linguagens.

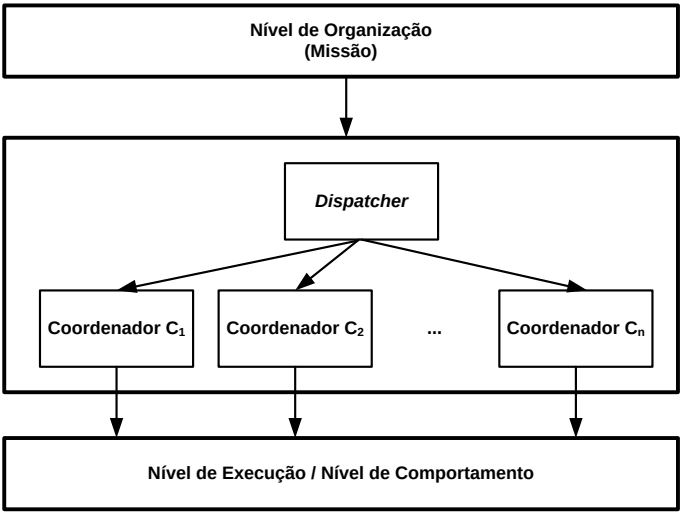


**Figura 3.5:** Arquitetura híbrida baseada no controle inteligente hierárquico. Adaptado de Wang *et al.* (1991) e Chang *et al.* (2005).

O nível de missão, ou de organização, é encarregado pelo planejamento e replanejamento, e pelo cálculo de trajetórias que levem em consideração o desvio de obstáculos e a minimização de critérios de energia e duração de missão. O nível de tarefa, ou de coordenação, compõe-se de um *dispatcher* e vários coordenadores. O *dispatcher* recebe a sequência de tarefas do nível de missão, supervisiona os estados do sistema e decide pela ativação ou desativação, através do envio de mensagens aos coordenadores. A estes, por sua vez, corresponde a coordenação de uma tarefa específica, traduzindo-a em termos de comandos básicos de veículo, disponíveis ao modo de um servidor de serviços. Navegação de longo alcance, aquisição de dados de terreno, ajuste por GPS, desvio de obstáculos, lançamento de veículo são exemplos de tarefas específicas. Tanto o *dispatcher* como os coordenadores possuem seu comportamento descrito por redes de Petri. O *dispatcher* identifica a linguagem de entrada gerada pelos estados do sistema e os coordenadores geram a lista de eventos de saída, sendo que os eventos controláveis[1] são encaminhados pelo *dispatcher* aos

[1] Os eventos controláveis correspondem àqueles eventos possíveis de serem impedidos por um agente externo ao sistema (supervisor). Por sua vez, os

respectivos coordenadores, enquanto que os eventos não-controláveis são repassados para o módulo de controle de missão. O módulo de coordenação de tarefas, portanto, é realizado pela integração das diversas redes de Petri, através de pontos de conexão formalmente definidos.

O trabalho de Palomeras *et al.* (2006) apresenta a modelagem e implementação de um SCM baseado em redes de Petri para a combinação de comportamentos que irão compor o plano de missão. A arquitetura empregada, mostrada na figura 3.6, é dividida em três camadas: controle de missão, controle de tarefas e controle de veículo. A geração de trajetória baseada em critérios de otimização não é implementada mas sim uma estratégia baseada na reatividade do robô, empregando combinação de comportamentos do tipo "manter distância", "manter profundidade", dentre outros.

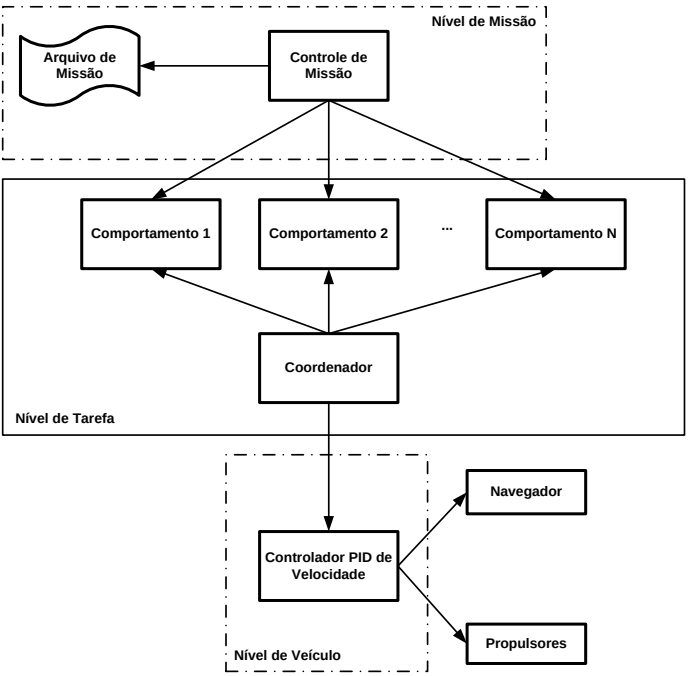


**Figura 3.6:** Arquitetura para um SCM baseado em redes de Petri. Adaptado de Palomeras *et al.* (2006).

eventos não-controláveis não podem ter sua ocorrência impedida. Tais conceitos serão apresentados no capítulo 4.

Para evitar a dependência do controlador de missão com configurações específicas de AUV, emprega-se uma camada de abstração da arquitetura, responsável por gerenciar a comunicação entre o SCM e a arquitetura do veículo, incluindo os módulos de percepção e controle (tarefas e veículo). As missões são executadas pela ativação de tarefas de alto-nível que, por sua vez, são realizadas pela ativação de um ou mais comportamentos do veículo. A missão, portanto, é descrita em termos de um conjunto de tarefas e como os comportamentos devem ser combinados para completar cada uma destas tarefas. Para isso é utilizado o formalismo de redes de Petri, que descreve o conjunto de ações a serem realizadas. A rede de Petri com a representação da missão é posteriormente executada por um jogador de redes de Petri, situado no

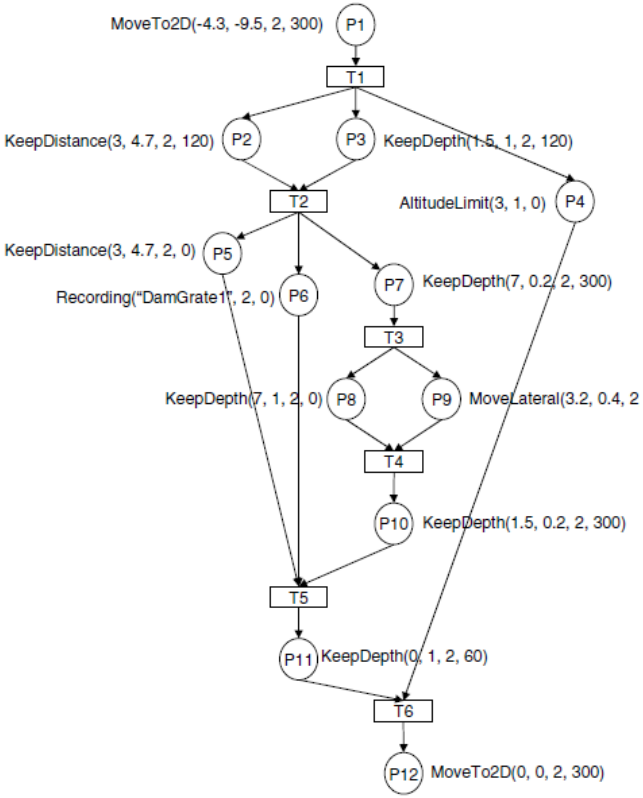


**Figura 3.7:** Rede de Petri de uma missão de inspeção (PALOMERAS *et al.*, 2006)

controle de missão, sendo esse o elemento que ativará ou desativará as tarefas e seus respectivos comportamentos. Posteriormente, em Palomeras *et al.* (2009), os autores estendem o trabalho com a proposição de uma linguagem de alto nível para especificação de missões de AUVs, compilável em uma rede de Petri, que pode ser executado pelo SCM já implementado. A figura 3.7 apresenta um exemplo de missão representada em termos de uma rede de Petri.

Em Bian *et al.* (2009b) apresenta-se uma arquitetura de duas camadas – gerenciamento de missão e controle de movimento – empregando métodos de composição de redes de Petri para descrever e especificar uma missão. O gerenciamento de missão é feito pela integração de diversas estruturas em um modelo global, incluindo o mapa do ambiente, especificação da missão, duração máxima da missão, tipos de operação e sequência de movimentos descritos por estados da missão, além de especificações sobre a segurança do veículo. A geração da trajetória é realizada com critérios de otimização de uso de bateria e restrições à velocidade do AUV.

Composta por fases específicas, como lançamento, navegação a um ponto, operação, retorno ou recuperação, a missão do AUV é programada em uma rede de Petri, posteriormente refinada através de métodos de decomposição de redes de Petri. O diagrama da composição final das várias redes de Petri é mostrada na figura 3.8. Desse modo, as várias etapas da missão são descritas por suas respectivas subredes, combinadas em uma estrutura hierárquica que define a missão.

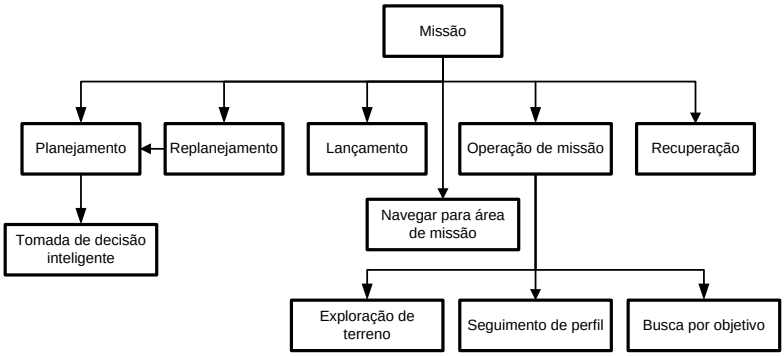


**Figura 3.8:** Hierarquia das redes de Petri. Adaptado de Bian *et al.* (2009b).

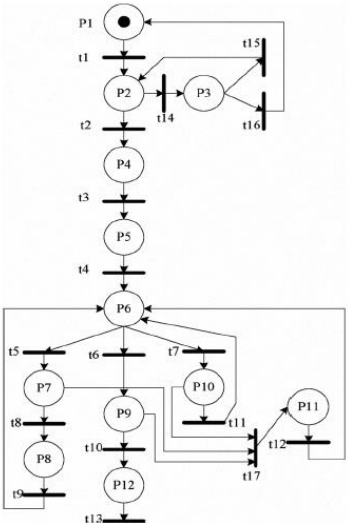


**Figura 3.9:** Rede de Petri de missão (BIAN *et al.*, 2009b)

Devido a natureza hierárquica do sistema de missão, separando as operações em redes específicas, torna possível a fácil inclusão de novas operações. Contudo, o trabalho emprega tal formalismo para organizar e descrever a interação entre os diversos componentes e os modos de operação, mas, não são apresentadas informações a respeito de análises de propriedades do sistema modelado e como as redes de Petri são implementadas ou adaptadas em um sistema embarcado de um AUV real. Na figura 3.9 é possível observar a rede de Petri de missão, onde cada lugar representa uma tarefa descrita por outra rede de Petri específica.

Finalmente, em Barrouil e Lemaire (1999) apresenta-se um sistema de planejamento e execução de missões, em que redes de Petri são empregadas para especificar o fluxo de dados, descrevendo como a missão é modificada em termos dos eventos que ocorrem, porém, sem realizar o tratamento de dados e informações. Comportamentos e funcionalidades são implementados como servidores em uma rede local, e a execução global do sistema é obtida por um jogador de rede de Petri, de acordo com o plano de missão, composto por um conjunto ordenado de macro-funções. Os comportamentos são implementados por funcionalidades, que consistem de código compilável e respondem a petições e geram eventos. O sistema possui vários módulos, compostos por vários comportamentos, e entre eles incluem-se o planejador de

trajetória, o replanejador de missão, o controlador de movimento, o monitor e estimador de estados do sistema. Uma rede de Petri é definida por um arquivo de texto e carregada por um jogador de redes de Petri. A figura 3.10 ilustra a rede de Petri do planejador de missão, onde o sistema realiza uma trajetória de navegação entre os pontos atual e desejado, até que todos os pontos da missão tenham sido percorridos ou caso alguma falha ocorra.

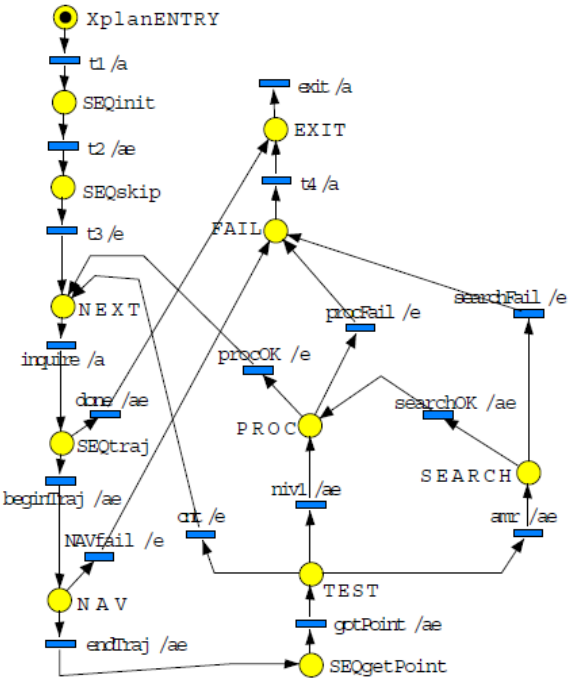


**Figura 3.10:** Rede de Petri do planejador de missão (BARROUIL e LEMAIRE, 1999)

## 3.3. ARQUITETURAS BASEADAS EM SISTEMAS HÍBRIDOS

Assim como as redes de Petri, existem relativamente mais artigos referentes ao uso de sistemas híbridos que abordagens baseadas em TCS na resolução de problemas relacionados à missão de robôs móveis. Na abordagem de sistemas híbridos, são considerados os modelos dinâmicos contínuos e discretos do veículo para a composição e execução de missões. Diversas são as abordagens para a modelagem e

controle de sistemas híbridos (LEAL, 2005), como por exemplo: controle para sistemas chaveados, controle supervisório de sistemas híbridos; autômatos temporizados, autômatos híbridos; redes de Petri temporizadas; lógica temporal; dentre outras. Em missão de AUVs, tais abordagens podem ser empregadas para descrever os modelos de consumo de energia do sistema, chaveamento de configurações pré-definidas de controladores de movimento ou configurações de movimento (manobras). A seguir, são mostrados alguns exemplos de trabalhos que empregam abordagens híbridas na resolução do problema de controle e execuções de missões.

Em Dias *et al.* (2006a), Dias *et al.* (2006b), Marques *et al.* (2006) e Pinto *et al.* (2006) são apresentados um conjunto de *frameworks*, *middlewares*, ferramentas e técnicas para planejamento, verificação e execução de missões para ROVs, AUVs e ASVs (*autonomous surface vehicles*). Particularmente em Dias *et al.* (2006a) descreve-se com mais detalhes a modelagem da missão em termos de sistemas híbridos. Nesse trabalho, o plano de missão é modelado por um conjunto de autômatos híbridos que representam tarefas. Em cada um desses autômatos os estados corresponde a unidades de movimento (manobras) e as transições, à lógica de cada manobra. As manobras também são modeladas por autômatos híbridos, descritas por estruturas de representação em XML, cujos exemplos típicos são: "Ir para", "Ajuste de GPS", "Seguir trajetória", "Seguir parede", "Tele-operação", "Submergir". Uma missão é composta por um mapa de missão, com a representação digital do ambiente, e pela sequência de tarefas a serem executadas que, por sua vez, são compostas por um conjunto específico de manobras.

A arquitetura de controle encontra-se dividida em duas estruturas principais: a embarcada no veículo e a parte em execução nos módulos de monitoramento e acompanhamento de missões, na estação base, e que pode ser vista na figura 3.11. Os controladores de baixo-nível permitem abstrair a interação com os equipamentos existentes no veículo. Por exemplo, o autômato da manobra "Ir para", apresentado na figura 3.11, recebe comandos de início do supervisor do veículo e, segundo a sua evolução, envia eventos de finalização de manobra ou falhas à este supervisor do veículo. Cada um dos estados do autômato da manobra, representam um particular conjunto de operações, e suas respectivas configurações no nível dos controladores de baixo nível. O supervisor de veículo, por sua vez, também descrito por um autômato híbrido, recebe a especificação de manobra do nível superior da arquitetura, criando um controlador de manobra, responsável pelo acom-

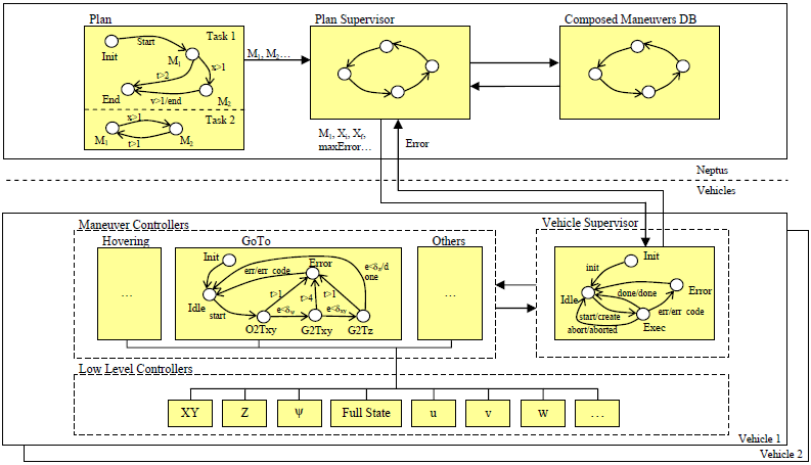


**Figura 3.11:** Arquitetura de controle de missão com a manobra "Ir Para" ou "*Go To*" (DIAS *et al.*, 2006a)

panhamento da evolução individual das manobras.

No topo desta arquitetura, encontra-se a parte de planejamento de missão e o supervisor de plano de missão. O planejador é responsável pelo comando e controle da missão, enviando as especificações de cada manobra ao supervisor de cada veículo e recebendo os eventos destes, na medida em que a manobra é executada. Ao final de uma manobra, o planejador envia a próxima manobra ao veículo. Contudo, os autores concentram-se na exposição da estrutura do sistema e em aspectos de implementação, não apresentando informações sobre os métodos de projeto, análise ou verificação formal dos autômatos híbridos ou sobre os sistemas de planejamento e o tratamento de exceções e falhas. Ainda assim, observa-se que os modelos representam um conjunto de atividades sequenciais e temporizadas para todo o conjunto do veículo e não para os diversos componentes individuais que fazem parte da arquitetura do veículo. Tal recurso delega a complexidade da representação e execução da missão aos modelos individuais de cada manobra, descritos por sequências de comandos, e para os controladores de baixo nível, requerendo uma análise mais criteriosa e formal do comportamento dinâmico contínuo e discreto do AUV ao realizar as missões. Outro aspecto, relacionado ao sistema de comunicação, consiste na ausência de um sistema de planejamento local a cada veículo prejudicando a realização de missões em casos que ocorram falhas de comunicação, pois, não fica claro nos trabalhos, se os eventos de falha e

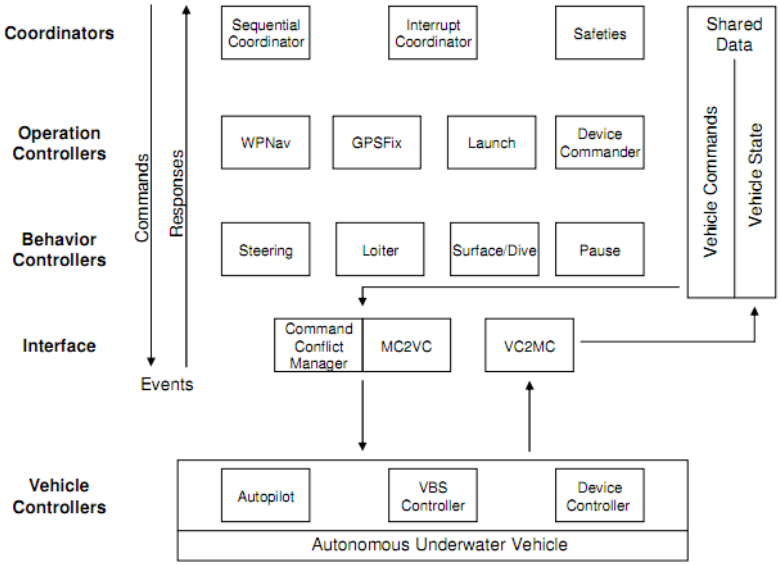


**Figura 3.12:** Arquitetura hierárquica baseada em sistemas híbridos (TANGIRALA *et al.*, 2005)

erro na realização de manobras são tratados a nível de missão pelo veículo ou pelo planejador.

Através de uma arquitetura dividida em camadas, mostrada na figura 3.12, em Tangirala *et al.* (2005) emprega-se a teoria de sistemas híbridos para compor o controlador de missão de um AUV. O sistema é dividido em três camadas: controle de veículo, interface e controle de missão. O controle de veículo provê vários comandos e parâmetros configuráveis para o controle de sensores e do movimento do AUV, como por exemplo, comandos para controle de atitude, profundidade ou velocidade. O nível de interface é responsável pelo intercâmbio de comandos e mensagens através de um mecanismo de sincronização de eventos e compartilhamento de dados, além de possuir uma estrutura para seleção de comando de veículos, caso ocorram conflitos, uma vez que mais de um comando pode estar ativo ao mesmo momento. Finalmente, o nível de controle de missão transforma uma missão em uma sequência coordenada de operações que, por sua vez, constituem-se de um conjunto de comportamentos e comandos de veículo. O funcionamento global do nível de controle é dado pela interação dos vários autômatos híbridos que modelam os diversos módulos existentes

em cada camada. O sistema completo é modelado através de ferramentas da área de sistemas discretos, como Teja e Uppaal, sendo esta última usada também para realizar a verificação formal dos modelos. A geração automática de código traduz os modelos para *scripts* de animação / simulação gráfica em OpenGL.

## 3.4. DISCUSSÃO

A seguir são apresentadas algumas observações sobre as arquiteturas baseadas em SEDs e empregadas no planejamento e execução de missões e na implementação de SCM de veículos robóticos móveis, notadamente AUVs.

A adoção de uma arquitetura para a organização lógica do sistema em camadas ou componentes favorece a aplicação de abordagens baseadas em SEDS, pois permite tratar separadamente a dinâmica dirigida pelo tempo e aspectos relacionados com a implementação do sistema da dinâmica dirigida a eventos usada para modelagem das missões. Tal abordagem facilita a modelagem e análise das dinâmicas contínuas e discretas que podem ser tratadas em conjunto. A divisão em níveis permite abstrair alguns aspectos ou características não necessários à especificação de missões. Questões como o tipo de estrutura de controle em malha-fechada, os algoritmos de localização e navegação, os mecanismos de sincronização de mensagens e compartilhamento de dados, ou ainda os métodos de detecção e correção de falhas, podem ser omitidos no nível de controle de missão. Assim, o nível inferior do veículo pode ser modelado com um gerador de eventos ou provedor de serviços, onde somente a informação relevante para a representação e execução de missões é tratada pelo SCM. Tal característica é desejável, pois torna o SCM independente da arquitetura de baixo nível do veículo, mesmo sendo necessário, como observado em vários trabalhos, o emprego de camadas intermediárias explicitamente criadas para esse propósito.

Talvez como maior desvantagem (GE e LEWIS, 2006), o emprego de representações baseadas em estados pré-configurados e fixos descritos por SEDs exige que todo o planejamento seja feito *a priori*, no domínio do especialista, antes da primeira operação do veículo, exigindo que o projetista seja capaz de antecipar todas as combinações possíveis de eventos que possam ocorrer durante a operação do sistema. Porém, tal limitação pode ser contornada pelos

sistemas de replanejamento e geração de trajetórias, ao permitir ao veículo escolher novas sequências de ações e trajetórias alternativas.

O problema de geração de trajetórias e de controle de movimento, na maioria das estratégias analisadas, é tratado como sendo adstrito aos respectivos módulos de natureza contínua, não sendo explicitamente tratado pelos módulos de natureza discreta, mesmo nas abordagens híbridas analisadas. Tal abordagem é possível devido à composição do sistema em níveis e ao uso de modelos específicos em cada nível, possibilitando abstrair o funcionamento de um nível com respeito aos níveis inferiores. O replanejamento também pode ser tratado pelo SCM, quando um sistema de custo é usado para encontrar um novo caminho ante a modificação do ambiente ou quando o sistema necessita decidir quais tarefas devem descartadas. Sem o replanejamento de trajetórias ou tarefas, o veículo robótico pode ficar preso a uma estratégia fixa de comandos, sendo possível tratar com erros, falhas e outros eventos não-deterministas, porém sem flexibilidade na tomada de decisão. Nem sempre o replanejamento é implementado em um AUV, pois tais algoritmos devem ser eficientes ao processar uma quantidade relativamente elevada de dados e serem realizadas em períodos de tempos muito curtos, uma vez que, por definição, o replanejamento é executado de modo *on*-line com o veículo em operação.

O uso de SED possibilita separar a modelagem e análise de missões da sua execução pelo SCM, ainda que este aspecto seja muitas vezes omitido na literatura analisada durante a revisão bibliográfica desse trabalho. Tal separação propicia o uso de duas visões complementares durante o projeto e implementação do SCM: a visão lógica e a visão de implementação do SCM. Na primeira visão, são definidas as estruturas de transição usadas para representar missões, permitindo a análise e validação de propriedades desejadas para o sistema. A visão de implementação, por sua vez, trata dos aspectos relacionados à execução das missões e como as estruturas de representação são usadas para integrar, coordenar e comandar as diversas funcionalidades do veículo pelo SCM.

Nesse sentido, na maioria da bibliografia revisada, as abordagens baseadas em autômatos, redes de Petri e outros modelos de área de SEDs são empregadas, principalmente, como ferramentas para descrever a sequência, a interação e a coordenação de tarefas, operações, etc., do ponto de vista lógico, permitindo, assim, o desenvolvimento de um *framework* para análise formal e representação de missões. Tais abordagens são também justificadas pela capacidade de oferecer métodos formais para a verificação de propriedades da missão, como

ausência de *deadlocks* ou inconsistências, ainda que nem todos os trabalhos apresentem exemplos ou análises de tais propriedades. Contudo, somente nos trabalhos mais amadurecidos, como em Palomeras *et al.* (2006) e Dias *et al.* (2009a) em que protótipos são empregados para comprovação das técnicas desenvolvidas, observa-se a preocupação em traduzir os modelos formais de representação lógica de missão em estruturas que possam ser incluídas e executadas pela arquitetura de controle embarcada dos veículos. Assim, muitas vezes, questões como a integração do SCM com as estratégias de controle, guiagem e pilotagem, os algoritmos de localização e navegação, o problema de sincronização de mensagens e dados e aspectos relacionados à natureza tempo-real do robô, são omitidas nos trabalhos, sendo muitas vezes abstraídos pelas funcionalidades disponíveis na sua arquitetura de controle e pelas operações unitárias implementadas pelo veículo.

A revisão bibliográfica, incluindo trabalhos além dos mostrados nas seções anteriores, sugere também a inexistência de uma metodologia estabelecida para a modelagem e implementação de sistemas de controle de missão de AUVs, principalmente no sentido de aliar questões teóricas a respeito de modelagens com aspectos práticos de implementação. Enquanto que alguns trabalhos possuem como foco principal questões teóricas relacionadas à resolução de um tipo específico de problema, como decomposição hierárquica de missões (WANG *et al.*, 1991; XU, ZHANG e FENG, 2004a; XU, ZHANG e FENG, 2004b; CHANG *et al.*, 2005), composição de missão a partir de manobras ou comportamentos (PALOMERAS *et al.*, 2006; BIAN *et al.*, 2009b), controle híbrido do movimento de robôs (TANGIRALA *et al.*, 2005), outros artigos apresentam ênfase maior nos aspectos práticos da resolução de problemas inerentes a sistemas embarcados e de tempo-real, omitindo questões pertinentes aos modelos e teorias empregadas (BARROUIL e LEMAIRE, 1999; BIAN *et al.*, 2005; DIAS *et al.*, 2006a; KIM *et al.*, 2010; LIN *et al.*, 2010).

Nesse sentido, o emprego de uma abordagem formal para a resolução mais ampla do problema de controle de missão, que envolva aspectos tanto teóricos como práticos, como apresentado em Palomeras *et al.* (2009) e também, mas com menor nível de detalhe, em Barrouil e Lemaire (1999), Dias *et al.* (2006a) e Dias *et al.* (2006b), torna-se desejável. Por exemplo, em Palomeras *et al.* (2009), o emprego de um compilador *off-line* que traduz um *script* de missão em uma rede de Petri e um jogador de rede de Petri no sistema embarcado do veículo, permite o emprego de verificação formal ao mesmo tempo em que a

estrutura de representação é utilizada diretamente para a execução da missão, evitando erros decorrentes da tradução manual de um modelo formal em código executável.

O emprego da TCS em controle de missão de veículos autônomos é realizado, de acordo com os trabalhos analisados, através da restrição do funcionamento livre da planta, conforme a determinação de especificações de segurança e sequenciamento de operações. Por sua vez, a capacidade de representação paralela de atividades das redes de Petri é empregada para a composição de missão a partir da combinação de comportamentos ou manobras. Em ambos os modelos, TCS e redes de Petri, a composição hierárquica é um recurso frequentemente empregado e permite expressar as operações de um nível com respeito a uma abstração das operações implementadas no nível inferior. Além disso, também em ambas as abordagens, o funcionamento do veículo é abstraído por um gerador de eventos ou um servidor de serviços, representados por um conjunto de ações e respostas, porém, adstritos à representação dos níveis superiores da arquitetura.

Os trabalhos de Xu, Zhang e Feng (2004a), Xu, Zhang e Feng (2004b), e, de modo similar, porém aplicado a planadores subaquáticos em Zhang *et al* (2006), empregam a TCS para a resolução do problema de missão de veículos subaquáticos. Porém, tal aplicação é realizada de modo exploratório, sem a sistematização ou emprego de um método. Além disso, os trabalhos não discutem aspectos relacionados à implementação dos modelos baseados em autômatos no sistema embarcado do veículo.

SCM baseados em sistemas híbridos, por sua vez, permitem incluir nos modelos discretos o comportamento temporal da dinâmica do sistema. Ainda que tal comportamento, na fase de execução de missão seja também abstraído por um provedor de serviços ou gerador de eventos, a evolução da dinâmica do AUV é explicitamente levada em consideração durante o planejamento de missões. Contudo, devido à necessidade de gerar planos de missão baseados em modelos híbridos através de ciclos de análise-verificação-compilação-tradução, frequentemente a missão é fixada a uma sequência pré-determinada de ações. Assim, o custo computacional envolvendo o emprego de sistemas híbridos pode se tornar proibitivo em plataformas com recursos limitados, principalmente se a presença de ocorrências não-deterministas exigir a necessidade de geração constante de planos de missão.

Como resumo das principais abordagens estudadas, apresenta-se na tabela 3.1 as principais características encontradas na bibliografia revisada. A coluna *veículo* indica as classes de veículos autônomos

**Tabela 3.1:** Comparação uso de SEDs em robótica móvel

| Artigo | Veículo | Nº. ca-mad | Ambi-ente | Pla nej. | Re-plan. | Método Formal | Especificação | Teste | Imple-ment. |
|---|---|---|---|---|---|---|---|---|---|
| Xu, Zhang e Feng, 2004a/2004b | AUV | 2 | Não-estrut. | Sim | Sim | Autômatos (TCS) | Modelagem, Análise, Síntese | Simulaç. | Indireta |
| Molina *et al*., 2010 | UGV | 3 | Estru-turado | Não | Não | Autômatos (TCS) | Modelagem, Síntese | Simulaç. | Não |
| Liu e Darabi, 2002 | UGV | 2 | Estru-turado | Não | Não | Autômatos (TCS) | Modelagem, Síntese | Não | Indireta |
| Wang *et al*., 1991 / Chang *et al.*, 2005 | UGV, AUV | 3 | Não-estrut. | Sim | Sim | Redes de Petri | Análise | Simulaç. | Não |
| Palomeras *et al*., 2006 | AUV | 3 | Não-estrut. | Sim | Não | Redes de Petri | Modelagem, Análise | Verificaç. | Direta |
| Bian *et al.*, 2009b | AUV | 2 | Não-estrut. | Sim | Sim | Redes de Petri | Modelagem | Não | Indireta |
| Barrouil e Lemaire, 1999 | AUV | 2 | Não-estrut. | Sim | Sim | Redes de Petri | Modelagem | Não | Direta |
| Dias *et al*., 2006a/2006b | ROV, AUV, ASV | 3 | Não-estrut. | Sim | Sim | Autômatos Híbridos | Análise | Simulaç. | Indireta |
| Tangirala *et al.*, 2005 | AUV | 3 | Não-estrut. | Sim | Sim | Autômatos Híbridos | Análise | Simulaç., Verificaç. | Não |

abordadas no trabalho (terrestre – *UGV*; aéreo – *UAV*; subaquático autônomo – *AUV*; subaquático teleoperado – *ROV*; veículo de superfície autônomo – *ASV*). O item *nº. de camad.* (número de camadas) identifica o número de níveis empregado na arquitetura robótica. A atuação do veículo em ambiente estruturado e não-estruturado é mostrado na coluna seguinte. O item *planej.* (planejamento) indica se a solução permite o planejamento de missões genéricas, sem a necessidade de recompilar código ou traduzir manualmente a estrutura de representação de uma missão. A coluna *replan.* (replanejamento) representa a capacidade do sistema de gerar novos planos de missão *on-line*, ou seja, durante a missão em curso. O termo *método formal* expressa o formalismo de SEDs empregado na resolução do problema de controle de missão. A coluna *especificaão* (especificação da lógica) apresenta como o formalismo de SEDs é usado no problema de missão: *modelagem* – entendimento do sistema, definição de comportamentos discretos do sistema, representação da missão em si, especificações de operação e segurança; *análise* – propriedades formais da missão representada pelo SED, como controlabilidade ou ausência de bloqueios; *síntese* – supervisores para garantia de especificações. O item *teste* (teste da lógica) apresenta como é realizada a comprovação do uso do método formal na resolução do problema de controle de missão: *não* – não há informação sobre a realização de testes; *simulaç.* (simulação) – emulação do comportamento discreto em conjunto com a simulação de modelo dinâmico do AUV; *verificaç.* (verificação) – emprego de algum método de verificação formal. Finalmente a coluna *implemente.* (implementação) indica: *não* – o sistema não foi implementado em veículo real; *indireta* – se o trabalho apresenta alguma implementação em um veículo real, porém os modelos são adaptados ou traduzidos para o sistema embarcado; *direta* – se além de implementar o sistema em um veículo formal, os modelos formais usados durante as etapas de modelagem, análise e testes são diretamente incluídos no sistema embarcado.

## 3.5. RESUMO

O emprego de teorias da área de SEDs vem contribuindo na resolução dos diversos problemas associados à modelagem e execução de missões em AUVs, servindo como formalismo para a representação e validação de missões, mas também como modelos para coordenação e

execução das mesmas. Assim, diferentes estratégias baseadas em TCS, redes de Petri ou Sistemas Híbridos foram apresentadas com o intuito de ilustrar as vantagens e limitações no emprego de cada uma dessas abordagens.

A adoção de uma arquitetura robótica, necessária para o planejamento e desenvolvimento do sistema embarcado de robôs autônomos, favorece o uso de abordagens baseadas em SEDs ao dividir o sistema em níveis, permitindo o emprego de modelos formais para representação, análise, verificação e execução de missões, diminuindo a presença de erros de projeto ou mesmo durante a realização das missões.

No próximo capítulo, apresenta-se a aplicação de uma abordagem modular da TCS, denominada de Controle Supervisório Modular Local – CSML (QUEIROZ e CURY, 2002) aplicado ao problema de controle de missão de AUVs. O CSML é empregado para derivar uma especificação para a lógica de representação e execução de missões. Tal recurso permite a análise de propriedades desejáveis para o sistema, como ausência de bloqueios e garantia de atendimento a especificações operacionais e de segurança.

# 4. MODELAGEM E SÍNTESE DE CONTROLE SUPERVISÓRIO PARA MISSÕES DE AUVS

Este capítulo apresenta a aplicação da Teoria de Controle Supervisório (TCS) (RAMADGE e WONHAM, 1989) ao problema de modelagem, síntese de supervisor e realização de missões de AUVs. Particularmente, propõe-se o emprego da abordagem modular do controle supervisório denominada de Controle Supervisório Modular Local (CSML) (QUEIROZ e CURY, 2002) para a modelagem dos subsistemas do AUV, das especificações de segurança e operação do veículo. A realização de missões é descrita pela evolução de vários subsistemas do AUV, representados por autômatos que compõem o modelo formal da planta, de modo que uma missão concreta corresponde a uma sequência particular de eventos gerados pelo modelo formal. Eventos controláveis são empregados para representar ações que o SCM pode decidir quando realizar, enquanto que eventos não-controláveis correspondem a ocorrências provenientes do ambiente e do funcionamento do veículo e que não podem ser impedidas. Entretanto, várias sequências geradas pela planta são consideradas indesejadas, por levar o veículo a estados que possam comprometer a sua integridade ou que não estejam de acordo com objetivos da missão e, portanto, necessitam ter a sua ocorrência impedida. Com esse objetivo, especificações de segurança e operação do veículo são definidas e usadas para a síntese de estruturas de controle, denominadas de supervisores, responsáveis por garantir o cumprimento de tais especificações, limitando a dinâmica da planta àquelas sequências controláveis e não-bloqueantes consideradas desejáveis para qualquer tipo de missão.

Assim, a seção 4.1 apresenta um resumo da TCS e do CSML. A seção 4.2 aborda o problema de realização de missões de AUVs em ambientes não-estruturados. A modelagem da planta é realizado na seção 4.3, e a modelagem das especificações, na seção 4.4. O processo de síntese dos supervisores é comentado na seção 4.5. Na seção 4.6 são apresentadas algumas simulações visando a validação inicial dos modelos de planta e especificações, bem como os supervisores obtidos para o cenário de missões considerado. Ao final do capítulo, são

apresentadas algumas considerações a respeito da modelagem proposta, suas vantagens e limitações.

## 4.1. TEORIA DE CONTROLE SUPERVISÓRIO

### 4.1.1. Modelagem de Sistemas a Eventos Discretos

Sistemas a Eventos Discretos (SEDs) correspondem a sistemas que evoluem através da ocorrência abrupta de eventos, caracterizando-se por possuir um conjunto discreto como espaço de estados e com mecanismo de transição de estado dirigido a eventos discretos. O conceito de evento está associado a uma ocorrência instantânea no sistema, ou no ambiente em que este se encontra inserido, e que provoca uma modificação no mesmo, conduzindo-o a uma nova configuração ou estado. O estado do sistema é modificado em pontos específicos de tempo e que correspondem, fisicamente, à ocorrência assíncrona de transições discretas, ou seja, a dinâmica que rege o comportamento de um SED é determinada pela ocorrência de eventos (CASSANDRAS e LAFORTUNE, 2008).

Uma das formas de representar o comportamento dirigido a eventos de um sistema é por meio de uso de linguagens, que são conjuntos de cadeias ou sequências de eventos (HOPCROFT, MOTWANI e ULLMAN, 2001). Uma transição ou evento da dinâmica discreta em malha-aberta do sistema, denominada de planta, pode ser associada a um símbolo. Uma sequência de símbolos, ou cadeia, é denominado de palavra. O conjunto de todas as sequências possíveis de símbolos de eventos gerados pelo sistema é representada por uma linguagem $L$. Como nas demais abordagens de SEDs, a TCS não admite a ocorrência simultânea de dois ou mais eventos no sistema, devido à instantaneidade dos eventos, assumindo que os sistemas modelados apresentam uma evolução sequencial da sua dinâmica discreta.

Formalmente, uma linguagem $L$ sobre um conjunto finito de eventos (alfabeto) $\Sigma$ é um conjunto tal que $L \subseteq \Sigma^*$, onde $\Sigma^*$ representa o conjunto de todas as cadeias finitas formadas por elementos de $\Sigma$, incluindo a cadeia vazia $\varepsilon$. O prefixo-fechamento de uma linguagem $L$ consiste no conjunto de todas as cadeias de $\Sigma^*$ que são cadeias incompletas de $L$, expresso formalmente pela seguinte expressão:

$$\bar{L} = \{u: \exists v \in \Sigma^* \wedge uv \in L\} \tag{4.1}$$

O comportamento apresentado pela planta pode ser descrito por linguagens, que, por sua vez, podem ser representadas por autômatos. Um autômato determinístico é formalmente definido pela quíntupla

$$G = (Q, \Sigma, \delta, q_0, Q_m) \tag{4.2}$$

onde os elementos são dados por:

- $\boldsymbol{Q}$**:** o conjunto finito de estados
- $\boldsymbol{\Sigma}$**:** o alfabeto de eventos
- $\boldsymbol{\delta}$**:** $Q \times \Sigma \rightarrow Q$ a função de transição parcial
- $\boldsymbol{q_0 \in Q}$**:** o estado inicial
- $\boldsymbol{Q_m \subseteq Q}$**:** os estados marcados

Os eventos ativos em um determinado estado são definidos através de uma função $\Gamma : Q \rightarrow 2^{\Sigma}$, onde $\Gamma(q)$ representa o conjunto de todos os eventos $e$ para o qual $\delta(q, e)$ está definida. Os estados marcados representam cadeias que completam tarefas do sistema.

É possível visualizar o autômato por meio de grafos, onde os vértices são conectados entre si por arcos em flechas. Os vértices simbolizam os estados, e os arcos, os eventos ou transições. Um estado é representado por um nó com um círculo, os estados marcados, por nós com círculos duplos e o estado inicial é indicado por uma seta. Uma transição pode ser representada pela tupla $(q, \sigma, q')$, onde o evento $\sigma$ provoca a modificação do estado $q$ do sistema para $q'$. Assim a transição $\sigma$ é representada pelo arco orientado partindo do vértice do estado $q$ em direção ao estado $q'$. Eventos controláveis (ver seção 4.1.2) são indicados por um arco com traço. A figura 4.1 ilustra um exemplo de autômato, onde: $\{1\}$ é o estado inicial e também o estado final; $\Sigma = \{a, b, c\}$ é o alfabeto de eventos; e $(1,a,2)$, $(1,c,3)$, $(2,b,1)$, $(3,a,2)$ e $(3,b,1)$ as transições.

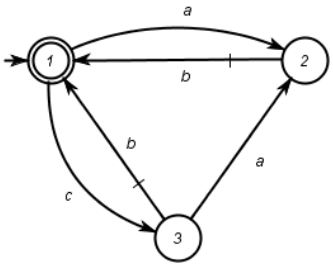


**Figura 4.1:** Exemplo de autômato

Um estado $Q$ de um autômato $G$ é dito acessível se o mesmo pode ser alcançado pela função de transição através de alguma sequência possível de eventos originada a partir do estado inicial $q_0$. Quando todos os estados são acessíveis, o autômato $G$ é acessível. Caso seja sempre possível a partir de um estado $q \in Q$ alcançar um estado marcado $q_m \in Q_m$ então o estado $q$ é denominado co-acessível. Um autômato é dito ser co-acessível caso todos os seus estados sejam co-acessíveis.

O autômato $G$ reconhece duas linguagens: a *linguagem gerada*, representada por $L(G)$, e a *linguagem marcada*, denotada de $L_m(G)$, tal que $L_m(G) \subseteq L(G) \subseteq \Sigma^*$. $L(G)$ é o conjunto de todas as possíveis sequências de eventos produzidas pelo sistema, e $L_m(G)$ um subconjunto com as sequências geradas pelo sistema e que representam sequências com conteúdo semântico particular, como por exemplo, a realização de tarefas completas.

Quando a evolução do autômato restringe-se a um conjunto de estados nos quais não há possibilidade de alcançar um estado marcado então o sistema encontra-se em bloqueio, e por permitir a ocorrência de vários eventos ou alcançar alguns estados não marcados, essa condição é denominado de *livelock*. A condição de bloqueio denominada de *deadlock* ocorre quando a sequência de eventos leva o sistema a um único estado não marcado em que nenhuma transição de saída exista. Sempre que todos os estados acessíveis de um autômato forem co-acessíveis, o autômato é não-bloqueante. Formalmente, um autômato é dito não bloqueante caso a linguagem gerada $L(G)$ seja igual ao prefixo-fechamento da sua linguagem marcada $\overline{L_m(G)}$, expressa na seguinte condição:

$$\overline{L_m(G)} = L(G) \tag{4.3}$$

Sistemas complexos podem ser compreendidos pelo funcionamento concorrente dos vários subsistemas que o constituem e que interagem entre si (WILLNER e HEYMANN, 1991). Nesses casos, é possível representar a combinação dos comportamentos dinâmicos dos subsistemas através da operação de composição ou produto síncrono entre os autômatos que representam tais subsistemas (CASSANDRAS e LAFORTUNE, 2008). O produto síncrono de dois autômatos $G_1$ e $G_2$, representado por $G_1 \| G_2$, é obtido pela execução concorrente dos mesmos, onde os eventos compartilhados são sincronizados, ou seja, somente podem ser executados se ambos os autômatos os executem

simultaneamente, e os eventos não compartilhados podem ser executados sempre que estiverem ativos no respectivo autômato.

### 4.1.2. Controle Supervisório

A Teoria de Controle Supervisório (TCS) propõe um método para o desenvolvimento de sistemas de controle de SEDs baseados em modelos formais de autômatos e linguagens, adotando uma distinção entre o sistema a ser controlado, denominado de planta, e a estrutura que a controla, chamada de supervisor (RAMADGE e WONHAM, 1989; CASSANDRAS e LAFORTUNE, 2008).

A planta, representada pelo autômato *G*, corresponde ao comportamento em malha-aberta fisicamente possível, resultante dos vários subsistemas que a compõem. Especificações representam imposições sobre o comportamento discreto em malha-aberta do sistema. A planta e as especificações são modeladas por autômatos. Na TCS os eventos são espontaneamente gerados pela planta, sendo possível evitar a ocorrência de alguns destes eventos. O papel do supervisor *S* consiste em observar a evolução dos eventos gerados pela planta e realizar uma ação sobre a mesma restringindo o seu comportamento de modo a atender especificações. Assim, o alfabeto $\Sigma$ é particionado em dois subconjuntos: eventos controláveis $\Sigma_c$, que podem ser diretamente desabilitados por um agente externo ou supervisor, e os eventos não-controláveis $\Sigma_u$, cuja ocorrência não pode ser impedida.

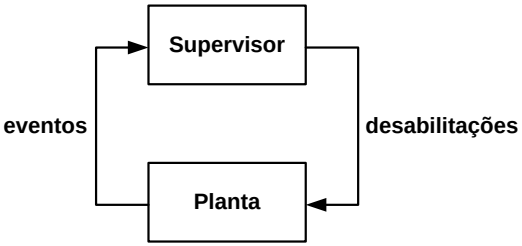


**Figura 4.2:** Estrutura em malha-fechada do controle supervisório

O processo de desenvolvimento de um supervisor para uma planta consiste em, basicamente, representar o modelo do autômato *G* da planta, os modelos dos autômatos das especificações e, através de um processo de síntese, encontrar o supervisor *S* que irá confinar as operações da planta àqueles estados desejados. Através da observação

da evolução da planta *G*, o supervisor *S* se encarrega de habilitar e desabilitar eventos controláveis da planta, exercendo, portanto, uma ação restritiva sobre a mesma. O conjunto dos eventos desabilitados pelo supervisor em um determinado instante define a ação de controle sobre a planta. Um supervisor *S* é dito admissível caso não desabilite eventos não-controláveis. A figura 4.2 ilustra a ação do supervisor sobre a planta através de uma malha-fechada.

O sistema em malha-fechada resultante da ação do supervisor *S* sobre a planta *G* é representado por *S/G*, com $L(S/G) \subseteq L(G)$ e $L_m(S/G) \subseteq L_m(G)$. Além disso, o supervisor *S* é dito não bloqueante caso as sequências de eventos geradas pela planta sob supervisão sempre permitam concluir uma tarefa considerada completa, ou seja:

$$\overline{L_m(S/G)} = L(S/G) \tag{4.4}$$

O supervisor pode ser representado na forma de um autômato que associa a cada estado um conjunto de eventos desabilitados. Como o comportamento em malha-aberta da planta *G* pode apresentar algumas sequências de eventos não-desejáveis e possíveis estados de bloqueio, são criadas especificações genéricas de controle $E_i$ e que correspondem a linguagens sobre conjuntos de eventos relevantes da planta, onde $i \in \{1, 2, ..., n\}$. As especificações genéricas $E_i$ podem ser também representadas por autômatos.

O supervisor *S* é obtido a partir de uma especificação desejada para o comportamento em malha-fechada, denominada de *K* ou também de linguagem alvo, que é obtida pelo produto síncrono da planta *G* e cada uma das especificações genéricas $E_i$.

Uma linguagem $K \subseteq L(G) \subseteq \Sigma^*$ é denominada controlável em relação a *L(G)*, caso a ocorrência de qualquer evento não-controlável e prevista pelo autômato G (evento fisicamente possível) após uma cadeia que pertença ao prefixo-fechamento de *K* mantenha a sequência dentro do prefixo-fechamento de *K*. Tal condição é expressa na seguinte condição:

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K} \tag{4.5}$$

A condição necessária e suficiente para a existência do supervisor não-bloqueante *S*, que atenda uma especificação $K = L_m(S/G) \subseteq L_m(G)$, é a controlabilidade de *K* em relação a *G*, e corresponde à condição 4.5. Entretanto, quando a especificação não for controlável com relação a *G*

é possível calcular o supervisor minimamente restritivo tal que $L_m(S/G) = SupC(K, G) \subseteq L_m(G)$, onde $SupC(K,G)$ corresponde à máxima linguagem controlável (RAMADGE e WONHAM, 1989). Assim, o supervisor não-bloqueante que implementa a máxima linguagem controlável é denominado supervisor ótimo. O algoritmo de Su e Wonham (2001) permite reduzir o número de estados do autômato que representa o supervisor ótimo, de modo a se obter um supervisor reduzido cuja ação de controle sobre a planta seja também não-bloqueante e minimamente restritiva.

A síntese dos supervisores empregando a TCS pode ser resumida nas seguintes etapas:

- Modelagem da planta: obtenção do modelo do autômato G da planta; particionamento do conjunto de eventos em controláveis e não-controláveis; caso a planta seja composta por vários modelos, obtenção do modelo global da planta através da composição entre os modelos individuais.
- Obtenção de um modelo *E* para as especificações; caso sejam empregadas especificações locais $E_i$ aos componentes do sistemas, estas são combinadas para obtenção da especificação global.
- Síntese de uma lógica de controle não bloqueante e ótima.

### 4.1.3. Controle Supervisório Modular Local

Na abordagem monolítica, um supervisor único e global é encontrado a partir do conjunto de todas as especificações. Entretanto, devido ao crescimento excessivo do número de estados para modelos complexos, o esforço computacional para encontrar o supervisor monolítico pode ser inviável e sua implementação em um sistema microprocessado ser difícil. O Controle Supervisório Modular (CSM) (RAMADGE e WONHAM, 1989) explora características modulares do sistema físico e das especificações para a obtenção dos supervisores. Cada supervisor é responsável por uma especificação, e o funcionamento global do sistema é determinado pelo conjunto das ações individuais de cada supervisor, como pode ser visto na figura 4.3.

Ainda que supervisores modulares possam ser encontrados, ainda é necessário o cálculo da planta a partir dos modelos dos subsistemas que a compõem, uma vez que as especificações baseiam-se no modelo global do sistema. Nesse sentido, o Controle Supervisório Modular Local (CSML) proposto em (QUEIROZ e CURY, 2000), prescinde des-

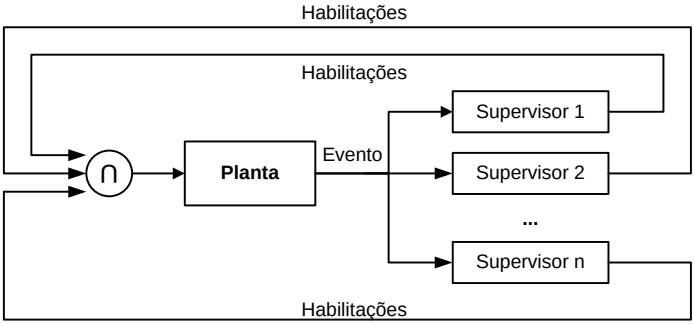


**Figura 4.3:** Controle supervisório modular. Adaptado de Queiroz (2004).

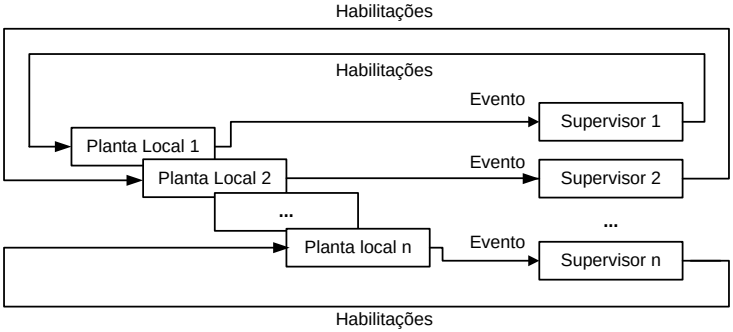


**Figura 4.4:** Controle supervisório modular local. Adaptado de Queiroz (2004).

sa necessidade ao derivar modelos modulares também para os subsistemas, além das especificações e supervisores. A figura 4.4 ilustra o diagrama básico da configuração de uma estrutura de controle supervisório modular local.

As especificações locais genéricas $E_x$ representam o comportamento desejável para o sistema, onde $x \in \{1, 2, ..., m\}$ é o conjunto de índices para as especificações e com $\Sigma_{Ex} \subseteq \Sigma$. O sistema global está composto por vários subsistemas $G_k$ que são modelados por autômatos assíncronos (sem eventos em comum), onde $k \in \{1, 2, ..., n\}$ representa o conjunto de índices para os subsistemas locais e com $\Sigma_k \subseteq \Sigma$. Porém, cada especificação é formulada em termos de eventos que afetam um ou mais subsistemas e não pelo modelo global da planta, como no CSM. Para a síntese dos supervisores locais, portanto, é necessário o cálculo das plantas locais $G_{locx}$ referentes às especificações $E_x$. A planta local é encontrada pelo produto síncrono dos subsistemas

$G_k$ relacionados à especificação $E_x$, ou seja $G_{locx} = ||_j G_j$ para $j \in \{k \in \{1, 2, ..., n\} \mid \Sigma_k \cap \Sigma_{Ex} \neq \varnothing\}$.

Por sua vez, a especificação local $K_{locx}$ é definida pelo produto síncrono entre a planta local e a especificação local genérica, ou seja, por $L_m(G_{locx}) \parallel E_x$. Os supervisores não-bloqueantes $S_x$ são encontrados a partir das especificações locais $K_{locx}$, e são tais que $L_m(S_x/G_{locx}) = SupC(K_{locx}, G_{locx}) \subseteq K_{locx}$. A ação concorrente e conjunta dos supervisores modulares sobre o sistema deve apresentar o mesmo comportamento do supervisor monolítico, sendo necessário assegurar o não-bloqueio ou a modularidade local dos supervisores modulares. Assim, tanto na abordagem CSM como na CSML, além da condição de controlabilidade, faz-se necessária a comprovação do não-conflito entre os vários supervisores locais (QUEIROZ e CURY, 2000), garantindo o comportamento global não-bloqueante e minimamente restritivo, expresso por:

$$||_{i=1}^{n} \overline{SupC(K_{locx}, G_{locx})} = \overline{||_{i=1}^{n} SupC(K_{locx}, G_{locx})} \tag{4.6}$$

## 4.2. APLICAÇÃO DO CSML AO PROBLEMA DE MISSÃO DE AUVS

A missão contém os objetivos de operação do veículo que usualmente são expressos em termos de atividades básicas de navegação como manobras, funcionamento de sensores e primitivas de comunicação. No nível de missão, é possível representar as operações de cada subsistema do AUV em termos de uma representação de alto-nível como um SED, abstraindo detalhes de implementação de cada subsistema. A aplicação da TCS ao problema de missões consiste em encontrar uma representação para a missão que irá definir todos os comportamentos discretos considerados seguros, ou seja, que atendem as especificações. Ao final, obtém-se uma lógica discreta para representação de missões que será empregada também para a execução de missões.

Para a caracterização do problema de missão de AUVs como um problema de controle supervisório, identifica-se, primeiramente, quais serão os diversos subsistemas e funcionalidades essenciais e que devem estar representadas pelo SED. Tais subsistemas e funcionalidades irão compor a planta. Na sequência, definem-se as restrições para o

comportamento em malha-aberta do sistema, representado pela planta. Na TCS, conforme já comentado na seção 4.1, a especificação é usada para representar uma restrição sobre a planta. Finalmente o comportamento desejado é garantido pela ação de supervisores sobre a planta, obtidos a partir das especificações pelo processo de síntese da TCS.

Na sequência será mostrada a aplicação da abordagem do CSML ao problema de missões de um veículo subaquático autônomo. Inicialmente apresenta-se a descrição geral do problema, e, na sequência, os passos de modelagem da planta e especificações e a síntese dos supervisores são mostrados em detalhes. Ao final, algumas emulações do sistema modelado sob ação dos supervisores realizadas com a ferramenta Supremica (AKENSON, 2002) são apresentadas.

Basicamente o problema de aplicação do CSML no problema de missão em AUVs pode ser resumido nas seguintes etapas:

1. Encontrar os modelos de representação discreta dos diversos componentes de software e hardware que compõem o veículo, sob o ponto de vista da realização de missões, porém do modo mais independente possível dos tipos de missões a realizar e dos aspectos de implementação do AUV.
2. A partir das condições do ambiente não-estruturado e de requisitos para funcionamento do veículo, derivar especificações de segurança e operação que devem ser atendidas em todos os tipos de missões.
3. Obtenção das estruturas de controle supervisório que irão garantir o atendimento às especificações.

Como na TCS considera-se que os eventos são gerados espontaneamente pela planta, a ação dos supervisores consiste em habilitar e desabilitar eventos controláveis e, portanto, não lhe corresponde a decisão de quais serão os eventos controláveis que devem ser executados pela planta de modo a alcançar os objetivos da missão. Assim, no próximo capítulo propõe-se uma arquitetura geral para um SCM com base no CSML e com um segundo componente, denominado de Gerenciador de Missão (GM), responsável pela escolha de eventos controláveis, entre outras funções.

Para modelagem da planta e especificações, síntese de supervisores, comprovação de propriedades de controlabilidade de especificações e não conflito dos supervisores, e simulação do sistema foi empregada a ferramenta Supremica (AKESSON *et al.*, 2006).

### 4.2.1. Cenário de Missão

O cenário de missão considerado para este trabalho consiste na aquisição de batimetria (dados topográficos do fundo do leito), coleta de dados de condutividade, temperatura e profundidade a partir da pressão (sensor CTD) e aquisição visual de estruturas submersas mediante uma câmera. Considera-se também que o veículo dispõe de um dispositivo de posicionamento global (GPS) que, apesar de não funcionar sob a água, é empregado na superfície para correções de posicionamento. Ao conjunto de sensores empregados exclusivamente para atividades de coleta de informações, não usados para a navegação do veículo, denomina-se carga útil (*payload*).

Em uma missão o veículo pode ser programado para navegar em diferentes áreas objetivos e usar conjuntos diferentes de sensores, de acordo com a natureza da atividade a ser realizada. A carga útil somente é ativada quando o veículo já se encontrar na região destinada à coleta de dados, com o objetivo de economizar a potência fornecida pelas baterias. Portanto, durante os deslocamentos entre as áreas objetivos, os sensores não empregados devem ser desligados. Também, devido ao mesmo motivo, quando o veículo estiver em modo de falha, deseja-se que a carga útil seja desligada. A figura 4.5 apresenta um exemplo de missão com duas áreas objetivos e pontos de lançamento e recuperação do veículo.

Também é considerado que, durante a realização de missões, obstáculos (fixos ou móveis) desconhecidos e de geometrias irregulares podem estar presentes, o que exige o desvio dos mesmos, além da existência de áreas de exclusão, como zonas com vegetação submersa ou locais com velocidades elevadas, como por exemplo, as próximas às eclusas de água, que devem ser evitadas pelo veículo (ITAIPU, 2009; ITAIPU, 2014).

Assim, através da TCS deseja-se, do ponto de vista formal, especificar uma lógica para representação de missões como uma sequência de eventos que satisfaçam, para qualquer tipo de missão, requisitos de operação e segurança do AUV. Desse modo, é possível representar a execução de missões em termos de um SED representado pelos diversos autômatos dos subsistemas sob ação dos supervisores modulares. Tal representação apresenta propriedades desejáveis como ausência de *deadlocks* e *livelocks* e garantia de especificações, além de servir de base para o desenvolvimento do SCM.


**Figura 4.5:** Exemplo de cenário de missão em ambiente não-estruturado.Adaptado de ITAIPU (2014).

### 4.2.2. Subsistemas e Funcionalidades do AUV

O AUV apresenta diversas capacidades e funcionalidades que se encontram organizadas em vários subsistemas, distribuídos em uma arquitetura de controle hierárquica, reativa ou híbrida (seção 2.3). Cada subsistema, por sua vez, é formado por vários componentes de hardware e software. Conforme já comentado no início dessa seção, do ponto de vista da missão, somente uma parcela da informação sobre o sistema é necessária. Assim, a operação do veículo é abstraída em um nível de representação que permita a tomada de decisão baseada no plano de missão, estado do veículo e condições do ambiente. Desse modo, sinais são recebidos por uma camada de abstração, como a indicação de finalização de tarefa ou situação de alarme, etc., e comandos são enviados pela mesma ao veículo, indicando o início de realização de alguma atividade ou ação a ser desempenhada. Particularmente na TCS é possível e adequado representar ocorrências geradas pelo ambiente ou pelo AUV através de eventos não-controláveis e comandos passíveis de serem ativados pelo AUV por meio de eventos controláveis.

Além dos sensores de carga útil e GPS, devem ser considerados para a tomada de decisão no nível de missão o nível das baterias, falhas

gerais na instrumentação do veículo, informações a respeito dos movimentos realizados pelo veículo (manobras, desvio de obstáculos, erros de posicionamento). Assim, cada aspecto relevante para o nível de missão é capturado por modelos de alto nível (autômatos) que abstraem os vários subsistemas reais que serão implementados pelo AUV.

Para a realização de missões no cenário previsto foram considerados os seguintes subsistemas, mostrados na figura 4.6. Na sequência, tais subsistemas serão brevemente apresentados.

**Subsistema de Potência:** o subsistema de potência é encarregado pela distribuição da energia fornecida pela bateria aos demais subsistemas, além de informar o nível de carga disponível, e pode ser visualizado na figura 4.6.a.

**Subsistema de Carga Útil:** este subsistema é responsável pelo gerenciamento dos sensores de carga útil (sensores CTD, sonar de batimetria e câmera). Os dados coletados pelos sensores são processados, filtrados e armazenados em arquivos para posterior recuperação. A figura 4.6.b apresenta esse subsistema.

**Subsistema de Controle do GPS:** o GPS (*global positioning system*) permite a determinação da posição do veículo através de satélites, porém, somente quando o mesmo encontra-se na superfície. Os dados lidos pelo GPS são empregados em ajustes na navegação do AUV e podem ser armazenados em um arquivo de dados. O GPS pode também ser considerado um sensor de carga útil (*payload*). Esse subsistema é mostrado na figura 4.6.c.

**Subsistema de Detecção de Obstáculos:** baseado em um *forward looking sonar*, o subsistema de detecção capta os obstáculos existentes à frente do movimento do veículo, estimando sua posição, dimensões e velocidades, e tornando esses dados disponíveis ao sistema de navegação. O diagrama desse subsistema pode ser visto na figura 4.6.d.

**Subsistema de Localização:** o subsistema de localização emprega métodos e algoritmos para estimar a posição, orientação e velocidade do AUV. O conjunto de sensores empregados na localização depende do tipo do veículo. Exemplos típicos de sensores para o cenário proposto são: a *inertial measure unit* (IMU) que fornece acelerações para o cálculo das velocidades e posições; sistema de posicionamento do tipo *long base line* (LBL), que usa a triangulação de *transponders* para medição de posição relativa; sensor de pressão para o cálculo da profundidade; e *doppler velocity logger* (DVL), tipo de sonar empregado para determinar e velocidade do veículo com respeito ao fundo do leito. Os dados sensoriais usualmente são combinados empre-

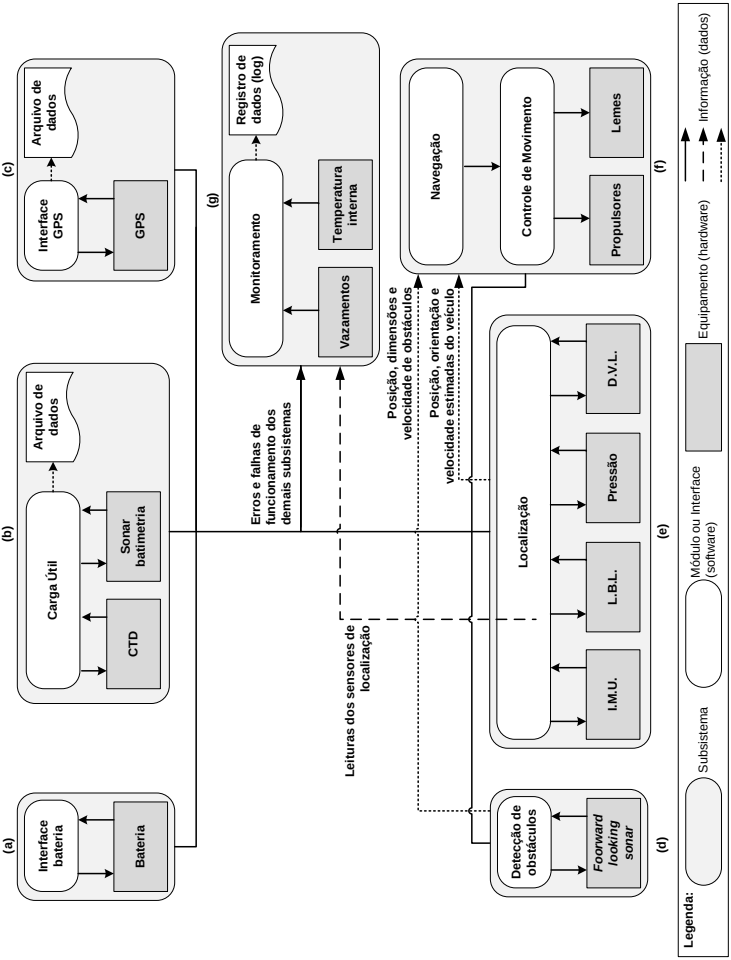


**Figura 4.6:** Subsistemas e funcionalidades do AUV

gando técnicas de fusão de dados, como filtros de Kalman. A figura 4.6.e apresenta tal subsistema.

**Subsistema de Navegação:** o subsistema de navegação, mostrado na figura 4.6.f, é encarregado de gerar as trajetórias de referência para os controladores de malha-fechada e pelo controle do movimento em si do AUV. Os algoritmos de desvio de obstáculos também podem ser implementados neste subsistema. O controle de movimento, ou seja, a ação de controle gerada para os atuadores em função das trajetórias desejadas e das posições, orientações e velocidades estimadas são implementadas neste subsistema, por meio de leis de controle em malha-fechada, conforme já comentado no capítulo 2.

**Subsistema de Monitoramento e Diagnóstico:** este subsistema, apresentado na figura 4.6.g, permite a detecção e previsão de ocorrência de falhas no AUV. Com base nas leituras de sensores de temperatura do cilindro de instrumentação, detectores de vazamento, falhas de funcionamento em sensores e demais subsistemas, é possível detectar, estimar e isolar falhas. O emprego de modelos permite também comparar os valores de localização estimados com as leituras reais dos sensores de navegação de modo a identificar possíveis erros de localização.

### 4.2.3. Requisitos Gerais de Operação dos Subsistemas

Independente do tipo de missão, alguns requisitos são fundamentais para a segurança e integridade do veículo. No cenário de missão considerado são estabelecidos requisitos visando a possibilidade de recuperação do veículo ante situações não previstas, a coordenação de operações e a economia no uso de energia disponível. Assim, os principais requisitos para a operação do veículo considerados para esta tese são:

- Evitar que manobras, por erro de programação, por exemplo, sejam realizadas em paralelo.
- Permitir o cancelamento de missão e retorno com segurança ao ponto de recuperação ante a ocorrência de erros ou devido a níveis insuficientes de bateria para completar a missão.
- Garantir o não-bloqueio do sistema (ausência de *deadlocks* e *livelocks*).
- Possibilitar a emersão para correção da posição mediante o emprego de GPS.

- Ligar e desligar sensores a fim de economizar energia.
- Evitar que o GPS seja ligado com o veículo submerso.
- Desligar com segurança o veículo nas situações em que o mesmo venha a colidir com algum obstáculo, na presença de erros críticos, em níveis muito baixos da bateria.
- Garantir que a carga útil seja ligada somente em fases da missão em que está prevista a aquisição de dados por meio de tais sensores.

A partir dos requisitos gerais é possível identificar restrições ao comportamento em malha-aberta do sistema, descrito pelas plantas modulares, e definir assim as especificações que também serão representadas por autômatos. Os supervisores modulares que garantem o atendimento a tais restrição são então obtidos pelo processo de síntese da TCS a partir das especificações.

## 4.3. MODELAGEM DA PLANTA

O comportamento em malha-aberta usado para descrever a missão do AUV consiste na interação dos vários subsistemas apresentados na seção 4.2.2, modelados através de autômatos de acordo com a TCS. Eventos não-controláveis são usados para mapear ocorrências geradas pelo veículo, como falhas, fim de manobras, ao passo que eventos controláveis representam ações que o veículo deve realizar para completar a missão, como ligar ou desligar equipamento, iniciar ou suspender uma manobra, etc., permitindo a representação de missões pela sequência de comandos (eventos controláveis) e respostas (eventos não-controláveis) do sistema. Assim, é possível representar a missão como a evolução dos modelos lógicos (sequência de eventos) que descrevem os componentes individuais de software e hardware, abstraídos das suas funcionalidades de baixo-nível de arquitetura e considerando somente aqueles aspectos importantes para a coordenação e execução da missão.

Desse modo, neste trabalho foram usados 12 modelos para os seguintes subsistemas e/ou funcionalidades: nível de carga da bateria; quatro tipos de sensores de carga útil; submersão do veículo; quatro tipo de manobras; colisões; e falhas. Tais modelos são apresentados detalhadamente a seguir.

**Subsistema 1: Nível de Carga da Bateria**

A carga disponível da bateria é empregada para indicar níveis compatíveis com a missão a ser realizada, ou então para decidir o cancelamento da missão a fim de preservar o veículo, evitando maiores danos ou perda do equipamento. Três níveis são considerados: normal, baixo (*low*) e crítico (*critic*). As transições entre tais estados é representada pela ocorrência de dois eventos não-controláveis. O primeiro caso é indicado pelo nível de carga baixa, representado pelo evento não-controlável *wrn_btr*, e o segundo caso, ocorre quando o nível de bateria é extremamente baixo, indicado pela ocorrência do evento não-controlável *ctr_btr*. Os percentuais de bateria que correspondem ao eventos de nível baixo e crítico são configurados na camada de Sequências Operacionais (SO) do CSML (seção 5.1). A figura 4.7 apresenta o autômato $G_{btr}$ que modela o comportamento da bateria, onde está incluindo também o evento controlável *rst_btr* de reinicialização do sistema.

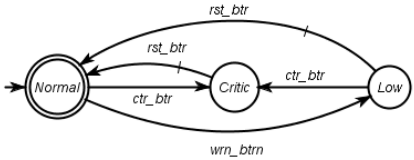


**Figura 4.7:** Autômato $G_{btr}$ do nível de carga da bateria

**Subsistema 2: Sensores de Carga Útil**

Para a realização da missão é necessário indicar a operação dos quatro sensores de carga útil (batimetria, câmera, CTD e GPS) que consiste nas ações de ligar ou desligar o sensor, além da informação da ocorrência de falhas. Assim, os sensores são modelados com operações do tipo ligar sensor (*on_j*), desligar sensor (*off_j*) e reinício de sensor (*rst_j*), onde $j \in \{batimetria, câmera, CTD, GPS\}$. O modelo também prevê a ocorrência de falhas, modelados pelo evento não-controlável *er_j*. A figura 4.8 apresenta o autômato $G_{sj}$ para tais sensores.

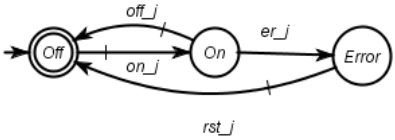


**Figura 4.8:** Autômato $G_{sj}$ dos sensores de carga útil

**Subsistema 3: Submersão**

O autômato $G_{sub}$ da figura 4.9 apresenta o modelo para o estado da submersão do veículo, onde os eventos não-controláveis *in_h2o* e *out_h2o* indicam, respectivamente, que o veículo submergiu e que veículo encontra-se na superfície. O sensor empregado nesse sistema corresponde ao sensor de pressão. A informação da submersão do veículo é empregada posteriormente para coordenar a ativação ou desativação de alguns sensores, como por exemplo, o GPS que somente pode ser usado na superfície.

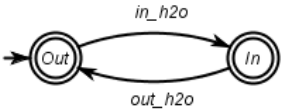


**Figura 4.9:** Autômato $G_{sub}$ do estado de submersão

**Subsistema 4: Manobras**

Do ponto de vista da realização de missão, é possível encontrar várias estratégias para incorporação da dinâmica contínua à definição e execução da missão. Algumas abordagens consideram a missão essencialmente como o seguimento de pontos (*waypoints*) ou o seguimento de trajetórias (ENCARNAÇÃO, 2002). Um segundo enfoque, incorpora a dinâmica contínua do movimento do veículo à dinâmica discreta empregada para descrever a evolução da missão, através de abordagens da área de sistemas híbridos. Assim, alguns trabalhos descrevem a missão como sendo a evolução dos modelos dinâmicos híbridos que descrevem aspectos diferentes do veículo e / ou do ambiente, como quantidade de combustível, presença de correntes (AUVs) ou vento (UAVs), ou a própria missão em si. Outros trabalhos, empregam o Controle Híbrido para o chaveamento ou escolha de um entre vários controladores previamente sintonizados, conforme a missão é realizada. Também existem os trabalhos baseados em autômatos temporizados, empregando tempos mínimos e máximos de manobras para detecção de erros, podendo, ou não, incluir os dois tipos anteriores citados. Finalmente, num outro extremo, existem as abordagens que desconsideram a dinâmica contínua do veículo, ocupando-se do acompanhamento da sequência de etapas da missão e da tomada da decisão perante a ocorrência de eventos cujo significado pode possuir relação direta com a dinâmica contínua, porém, não explicitamente tratada nos modelos.

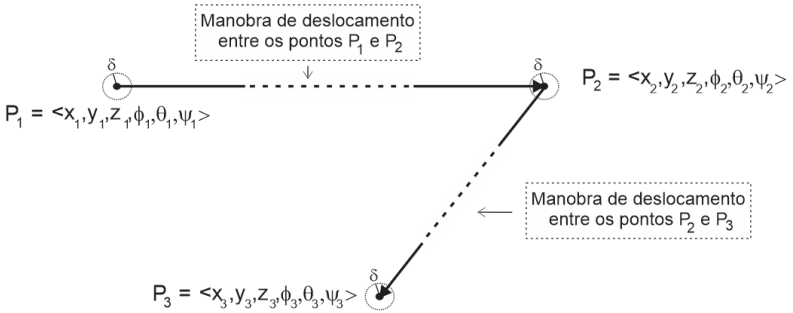


**Figura 4.10:** Exemplo de manobras de deslocamento entre posição inicial e final

Nesse sentido, a dinâmica contínua é abstraída por manobras. O modelo de manobras descreve a realização de um deslocamento entre uma posição inicial e final no ambiente, conforme mostrado na figura 4.10. Tal modelo é usado para abstrair e encapsular os aspectos internos aos subsistemas de navegação e localização. Do ponto de vista do subsistema de localização, que estima a posição e orientação do veículo, somente deseja-se conhecer se o veículo alcançou ou não a posição final da manobra, dentro do período de tempo estipulado pelo arquivo de missão. Por sua vez, o sistema de navegação é responsável por gerar as trajetórias e enviá-las às malhas de controle de movimento. A realização de percursos exige o seguimento de uma trajetória, que pode ser calculada previamente à missão e incluída como dado do plano de missão, ou ainda gerada *on-line*. Contudo, para esse trabalho considera-se, por simplicidade, que o veículo se locomove a uma velocidade constante e segue um trajeto o mais linear possível entre os pontos de início e fim e, portanto, a trajetória completa consiste em um conjunto de pontos (*waypoints*). No nível de SO da arquitetura do CSML (seção 5.1), as referências de posição final são enviadas para o nível do controle do veículo a partir dos dados configurados no arquivo de missão, porém, a estrutura em si dos controladores não é usada nos modelos das manobras. Além disso, em torno aos pontos inicial e final é considerada uma margem de erro $\delta$ de posicionamento, também configurada no arquivo de missão.

Para esse trabalho foram consideradas as seguintes manobras: manobra de navegação, manobra de descida, manobra de retorno e manobra de subida. A manobra de navegação consiste na realização de percursos retilíneos, geralmente entre as áreas objetivos da missão ou durante a aquisição de dados sensoriais e consiste na manobra mais

empregada. A manobra de descida é empregada para representar a submersão do veículo. A manobra de retorno é utilizada para indicar o deslocamento em direção ao ponto de recuperação, permitindo representar na missão a etapa final da mesma. Por fim, a manobra de subida descreve a ação do veículo dirigir-se à superfície na posição imediatamente acima da qual o veículo se encontra.

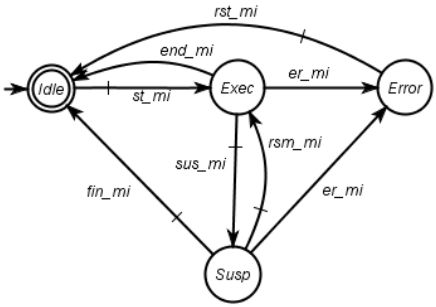


**Figura 4.11:** Autômato $G_{mi}$ das manobras de navegação, descida e retorno

As manobras de navegação, subida e retorno seguem o modelo do autômato $G_{mi}$ da figura 4.11. Nesse modelo, o início da manobra é representado pelo evento controlável *st_mi*, e a finalização com êxito da mesma, pelo evento não-controlável *end_mi*, onde $mi \in \{m, md, mr\}$ representando, respectivamente, as manobras de navegação, descida e retorno. Manobras podem ser suspensas e retomadas posteriormente, indicadas, respectivamente, pelos eventos controláveis *sus_mi* e *rsm_mi*. Manobras suspensas podem ser finalizadas, ocasião em que o evento controlável *fin_mi* ocorre. Finalmente, o evento *er_mi* indica a ocorrência de erros durante a realização da manobra ou ainda, durante o estado de suspensão da mesma. Tipicamente um erro de realização de manobra é gerado no nível SO da arquitetura do CSML por um *timeout*, ou seja, caso o veículo demore muito em alcançar o ponto final do trajeto, condição que pode representar vários contextos, desde a falha em algum propulsor até o fato do veículo ter ficado preso na vegetação submersa.

A manobra de subir à superfície apresenta um modelo diferenciado, conforme pode ser visualizado no autômato $G_{mup}$ da figura 4.12. Como esse tipo de manobra é empregado em condições de falha severa, como a presença de líquido no interior do cilindro de instrumentação, não é permitido que essa manobra seja suspensa. Assim, o modelo consiste basicamente em iniciar e terminar a manobra, além da condição de erro da mesma.

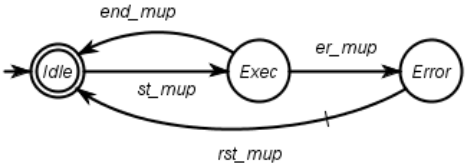


**Figura 4.12:** Autômato $G_{mup}$ da manobra de subir à superfície

**Subsistema 5: Colisões (obstáculos)**

No nível de missão é possível representar os desvios de objetos fixos ou móveis, independente dos algoritmos usados pelos subsistemas de detecção e navegação. A detecção de colisões (obstáculos) é representada pelo evento não-controlável *dt_col*. Após a detecção, o AUV pode conseguir desviar do objeto, indicado pelo evento não-controlável *ok_col*, ou colidir com o mesmo, representado pelo evento não-controlável *er_col*. A opção de reinício do subsistema de colisão é modelada pelo evento controlável *rst_col*. A figura 4.13 apresenta o autômato $G_{col}$ do modelo do veículo para colisões.

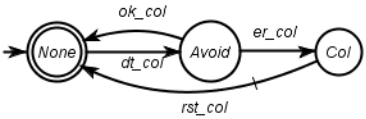


**Figura 4.13:** Autômato $G_{col}$ da detecção e desvio de colisões

**Subsistema 6: Falhas**

O subsistema de monitoramento e diagnóstico, responsável por detectar diversos tipos de erros, como divergências entre modelos e valores reais dos sensores, a presença de água em compartimentos de

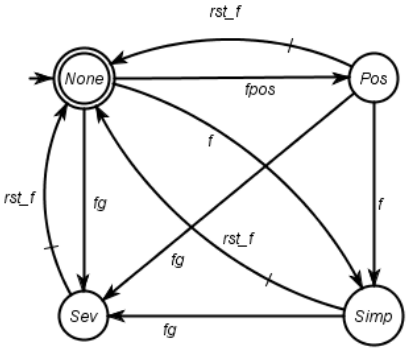


**Figura 4.14:** Autômato $G_f$ das falhas

instrumentação, ou a perda parcial ou total de algum propulsor, são abstraídos pelo modelo geral de falhas apresentado na figura 4.14. Esse modelo incorpora todas as falhas não previstas nos demais modelos, de modo a permitir tratá-las e decidir pelo replanejamento ou cancelamento de missões.

O modelo encapsula todas as falhas em 3 tipos específicos: falhas simples (*f*); falhas críticas ou graves (*fg*); e falhas de posicionamento (*fpos*). As falhas simples (*f*) representam situações em que a missão não pode mais ser executada, porém, o veículo ainda possui capacidade de navegabilidade que o permitem retornar ao ponto de recuperação. As falhas graves (*fg*), por sua vez, indicam ocorrências de erros que podem acarretar no dano permanente do equipamento, sendo desejável que o mesmo seja desligado o mais breve possível. Particularmente supõe-se que o peso negativo do veículo, ou seja, que a força e momento resultantes dos esforços restaurativos (força gravitacional e empuxo hidrostático) sejam negativos de modo que a tendência do veículo seja sempre de subir à superfície. Finalmente, as falhas de posicionamento (*fpos*) constituem-se de situações em que os erros de localização são consideráveis, sendo necessário o retorno do veículo à superfície para calibração com o equipamento GPS. Tais estados de erro são modelados pelo autômato $G_f$ e representam o estado do veículo com respeito ao nível de gravidade de ocorrência de uma falha. A falha de gravidade maior corresponde a falha grave e a de menor gravidade, a falha de posicionamento.

## 4.4. MODELAGEM DAS ESPECIFICAÇÕES

A partir do levantamento dos requisitos gerais para a operação segura do veículo e dos modelos para os subsistemas do veículo, é possível encontrar aquelas restrições que representam sequências de eventos consideradas indesejadas, ou seja, que não atendem aos requisitos gerais. Na TCS, a ação de um supervisor sobre a planta é restritiva, ou seja, desabilita um evento controlável impedindo que uma sequência indesejável ocorra no sistema. Assim, as especificações expressam ações restritivas no sentido de evitar sequências indesejáveis de eventos. Portanto, o processo de modelagem de especificações no âmbito da TCS consiste em encontrar modelos baseados em autômatos que representem imposições e restrições sobre o sistema expressas pelos requisitos gerais.

As especificações indicam, na TCS, restrições ao comportamento em malha-aberta da planta indicado pelos modelos apresentados na seção anterior. As especificações são modeladas em função das restrições apresentadas na listagem da subseção 4.2.3, e com base nos modelos de planta adotados para os subsistemas do veículo. Para esse trabalho foram consideradas 58 especificações, mostradas na tabela 4.1. Por limitações de espaço, serão apresentadas nesta seção 5 classes de especificações. Exemplos das demais especificações usadas para este trabalho podem ser encontradas no apêndice A.

**Tabela 4.1:** Classes de especificações consideradas

| Requisitos | Número de Especificações |
|---|---|
| Falhas simples ou graves e funcionamento dos sensores de carga útil | 4 |
| Falhas de posição e funcionamento dos sensores de carga útil | 3 |
| Falhas simples e realização de manobras | 2 |
| Falhas graves e realização de manobras | 3 |
| Falhas de posição e realização de manobras | 2 |
| Nível crítico de bateria e funcionamento dos sensores de carga útil | 4 |
| Nível crítico de bateria e realização de manobras | 3 |
| Estado de submersão e funcionamento dos sensores de carga útil | 4 |
| Estado de submersão e realização de manobras | 3 |
| Colisões e funcionamento dos sensores de carga útil | 4 |
| Colisões e realização de manobras | 2 |
| Realização simultânea de manobras | 1 |
| Erro em manobras e funcionamento dos sensores de carga útil | 16 |
| Erro em manobras e realização das demais manobras | 7 |
| **Total** | **58** |

**Falhas Simples ou Graves e Sensores de Carga Útil**

A carga útil (sonar de batimetria, câmera e CTD) e o GPS são empregados em condições normais de operação do veículo. Quando ocorrem falhas simples ou graves, representado respectivamente pelos eventos não-controláveis *f* e *fg*, tais sensores não são mais necessários pois não são empregados na navegação do veículo e, portanto, a fim de economizar bateria não é mais permitida a sua ativação (evento controlável *on_s* onde $s \in \{batimetria, câmera, CTD, GPS\}$). Assim, são definidos 4 autômatos, um para cada sensor de carga útil. O evento controlável *rst_f* indica que se, por ventura, uma falha for corrigida, então o sistema pode voltar a funcionar normalmente. A figura 4.15 apresenta o exemplo do autômato para esse tipo de especificação, denominada de $E_{F\_FG\_s}$.

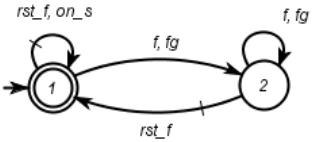


**Figura 4.15:** Especificação $E_{F\_FG\_s}$ de sensores e falhas simples ou graves

**Nível Crítico de Bateria e Sensores Carga Útil**

Como a segurança do veículo é prioritária à realização das aquisições de dados previstas na missão, impede-se a ativação dos sensores de carga útil (evento controlável *on_s* onde $s \in \{batimetria, câmera, CTD, GPS\}$) quando o nível de bateria disponível descender a um patamar crítico (evento não-controlável *ctr_btr*). Tal condição é desejável para os quatros sensores de carga útil, sendo que para cada sensor é definido uma especificação do tipo apresentado pelo autômato $E_{btr\_s}$ da figura 4.16.

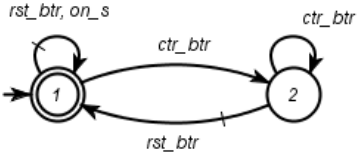


**Figura 4.16:** Especificação $E_{btr\_s}$ para nível crítico de bateria e sensores de carga útil

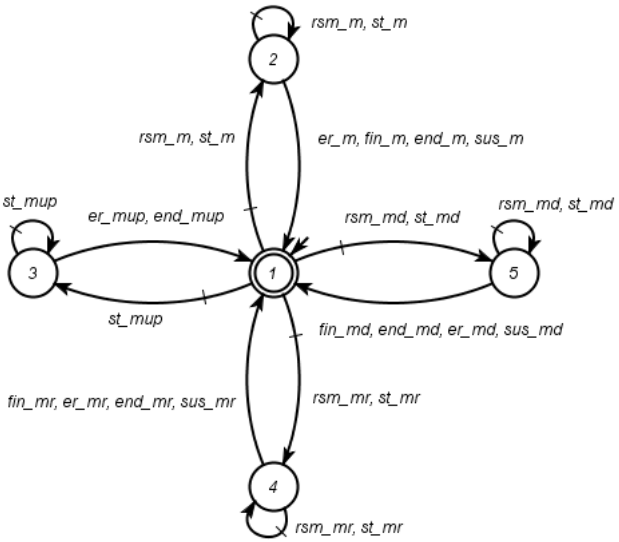


**Figura 4.17:** Especificação $E_m$ do paralelismo de manobras

**Realização Simultânea de Manobras**

Para evitar que duas ou mais manobras sejam realizadas simultaneamente, o que indicaria falha de programação da missão ou mesmo erro na implementação do sistema, foi definida a especificação $E_M$ que impõem que somente uma manobra esteja em curso. Manobras em estado de suspensão podem ser retomadas depois que a manobra em curso terminar ou for finalizada. O autômato da figura 4.17 representa a especificação $E_m$ que modela a restrição de realização de manobras em paralelo para os quatros tipos de manobras. Enquanto uma manobra estiver em realização, outras manobras não podem ser iniciadas (evento controlável *st_mi* onde $m_i \in \{md, m, mr, mup\}$ representando, respectivamente, as manobras de descida, navegação, retorno e subida) ou retomadas (evento controlável *rsm_mi*), até que a manobra atual saia do estado de ativa ou em realização (evento controlável *fin_mi* ou eventos não-controláveis *end_mi* ou *er_mi*).

**Erro em Manobras e Sensores de Carga Útil**

Sempre que uma manobra não puder ser realizada o evento não-controlável do tipo e*r_mi* ocorre, onde $m_i \in \{descida, navegação, retorno, subida\}$. Sob tais condições, assume-se que o veículo encontra-

se em um modo de erro em que não mais será possível continuar com a missão. Assim, não há sentido utilizar os sensores de carga útil, sendo desejável que a sua ativação seja impedida com o objetivo de economizar bateria. Ao total são definidas 16 especificações, uma para cada uma das quatro manobras e para cada um dos quatro sensores de carga útil. O modelo geral dessas especificações segue o padrão do autômato $E_{er\_mi\_s}$ apresentado na figura 4.18, com *mi* indicando o tipo de manobra e *s*, o tipo de sensor.

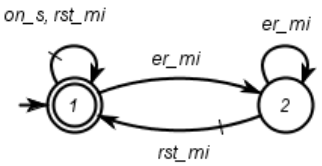


**Figura 4.18:** Especificação $E_{er\_mi\_s}$ para erro em manobras e sensores de carga útil

**Erro em Manobras e Demais Manobras**

Algumas manobras possuem precedência sobre outras, de modo que o erro na execução de uma manobra bloqueia ou impede a realização das demais. Assim, são definidas sete especificações para restringir a ação de início ou retomada das demais manobras ante o estado de falho de outra de maior prioridade. A manobra de subida é a de maior prioridade e, portanto, os erros nas demais não impedem que a mesma seja realizada e, por sua vez, erros na realização da manobra de subida impossibilitam a ativação das manobras restantes. De modo similar, a manobra de retorno possui prioridade sobre as manobras de navegação e descida pois é também empregada para o cancelamento de missões. Finalmente, erros na manobra de navegação impedem o início ou retomada da manobra de descida, e vice-versa. O autômato $E_{er\_mi\_mj}$, onde *mi* significa a manobra que apresentou o erro e *mj* a manobra que é impedida de ocorrer, é apresentado na figura 4.19.

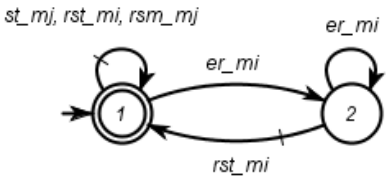


**Figura 4.19:** Especificação $E_{er\_mi\_mj}$ para erro em manobras

## 4.5. SÍNTESE DOS SUPERVISORES

Para a síntese de um supervisor monolítico é necessária a composição síncrona de todos os autômatos da planta, através do modelo global *G* da planta. Entretanto, a ferramenta Supremica não foi capaz de calcular a composição do autômato único, devido à explosão no número de estados[2]. Pelo mesmo motivo, a composição síncrona das especificações locais em uma única especificação global não foi possível. Tal dificuldade expressa a limitação no emprego da abordagem monolítica, útil para plantas relativamente pequenas, porém na medida em que os modelos e especificações crescem, a operação de produto síncrono pode exigir muito esforço computacional ou ainda mesmo não ser possível realizá-la.

Contudo, como na abordagem modular local os supervisores são obtidos a partir dos modelos locais das plantas envolvidos com a especificação local, o número de estados dos modelos diminui consideravelmente e, portanto, o esforço computacional necessário para síntese dos supervisores é muito menor. Inicialmente são obtidas, para cada especificação local genérica $E_x$, as plantas locais $G_{locx}$, através do produto síncrono dos subsistemas $G_x$ relacionados à especificação em questão. Por exemplo, a planta local para a especificação da figura 4.18 é dada pelo produto síncrono entre os autômatos da manobra e do sensor em questão. A especificação local, ou linguagem alvo, $K_{locx}$ é então obtida pelo produto síncrono da especificação genérica $E_x$ e a planta local $G_{locx}$. O supervisor modular local ótimo não-bloqueante pode então ser obtido.

A tabela 4.2 apresenta para cada grupo de especificação, conforme apresentado na seção 4.4, os respectivos estados e transições da especificação genérica $E_x$ em si, da planta local $G_{locx}$, da especificação local $K_{locx}$ e do supervisor modular local $S_x$. Os testes de controlabilidade das especificações e do não conflito dos supervisores são realizados de modo automático pela ferramenta Supremica.

**Tabela 4.2:** Estados e transições dos autômatos do CSML

| **Especificação** | $E_x$ | $G_{locx}$ | $K_{locx}$ | $S_x$ |
|---|---|---|---|---|
| $E_{F\_FG\_bat}$, $E_{F\_FG\_cam}$, $E_{F\_FG\_CTD}$, $E_{F\_FG\_GPS}$ | 2/7[3] | 12/43 | 12/41 | 12/41 |

[2] A ferramenta deixa de funcionar quando o sistema chega próximo a 4 milhões de estados.

[3] Número de estados / número de transições.

| Especificação | $E_x$ | $G_{locx}$ | $K_{locx}$ | $S_x$ |
|---|---|---|---|---|
| $E_{FPOS\_bat}$, $E_{FPOS\_cam}$, $E_{FPOS\_CTD}$ | 2/5 | 18/43 | 18/57 | 18/57 |
| $E_{F\_md}$, $E_{F\_mn}$ | 2/6 | 16/68 | 20/76 | 20/76 |
| $E_{FG\_md}$, $E_{FG\_mn}$, $E_{FG\_mr}$ | 2/6 | 16/68 | 16/66 | 16/66 |
| $E_{FPOS\_mn}$, $E_{FPOS\_mr}$ | 2/6 | 16/68 | 24/90 | 24/90 |
| $E_{btr\_bat}$, $E_{btr\_cam}$, $E_{btr\_CTD}$, $E_{btr\_GPS}$ | 2/5 | 9/27 | 9/26 | 9/26 |
| $E_{btr\_md}$, $E_{btr\_mn}$, $E_{btr\_mr}$ | 2/6 | 12/44 | 12/42 | 12/42 |
| $E_{sub\_bat}$, $E_{sub\_cam}$, $E_{sub\_CTD}$ | 2/5 | 6/20 | 6/19 | 6/19 |
| $E_{sub\_GPS}$ | 2/5 | 6/20 | 6/19 | 6/19 |
| $E_{sub\_mn}$, $E_{sub\_ms}$ | 2/6 | 8/32 | 8/30 | 8/30 |
| $E_{col\_bat}$, $E_{col\_cam}$, $E_{col\_CTD}$, $E_{col\_GPS}$ | 2/5 | 9/24 | 9/23 | 9/23 |
| $E_{col\_md}$, $E_{col\_mn}$ | 2/6 | 12/40 | 12/38 | 12/38 |
| $E_M$ | 5/25 | 192/1408 | 72/308 | 72/308 |
| $E_{er\_md\_bat}$, $E_{er\_md\_cam}$, $E_{er\_md\_CTD}$, $E_{er\_md\_GPS}$, $E_{er\_mn\_bat}$, $E_{er\_mn\_cam}$, $E_{er\_mn\_CTD}$, $E_{er\_mn\_GPS}$, $E_{er\_mr\_bat}$, $E_{er\_mr\_cam}$, $E_{er\_mr\_CTD}$, $E_{er\_mr\_GPS}$ | 2/5 | 12/40 | 12/39 | 12/39 |
| $E_{er\_ms\_bat}$, $E_{er\_ms\_cam}$, $E_{er\_ms\_CTD}$, $E_{er\_ms\_GPS}$ | 2/5 | 9/24 | 9/23 | 9/23 |
| $E_{er\_ms\_md}$, $E_{er\_ms\_mn}$, $E_{er\_ms\_mr}$ | 2/6 | 12/40 | 12/38 | 12/38 |
| $E_{er\_mr\_md}$, $E_{er\_mr\_mn}$, $E_{er\_md\_mn}$, $E_{er\_mn\_md}$ | 2/6 | 16/64 | 16/62 | 16/62 |

Para cada uma das 58 especificações $E_x$ foi encontrado um supervisor $S_x$ ótimo não-bloqueante. Na tabela 4.2 é possível observar os estados de todas as especificações $E_x$, e as respectivas plantas locais $G_{locx}$, especificações locais $K_{locx}$, e supervisores modulares $S_x$. Como todas as especificações são controláveis e não-bloqueantes ($K_{locx} = S_x$) é possível usar o modelo da especificação $E_x$ como supervisor, reduzindo a complexidade na implementação dos supervisores. Com o emprego da

ferramenta Supremica verificou-se que os 58 supervisores modulares $S_x$ são não-conflitantes, o que garante a sua ação conjunta ótima, ou seja, o seu funcionamento conjunto equivale à ação do supervisor monolítico.

Segundo a abordagem monolítica da TCS, não foi possível encontrar, dentro de um tempo hábil, o modelo monolítico da planta, pois com quase 4 milhões de estados computados, a ferramenta Supremica deixa de funcionar. A implementação de tamanha estrutura, ainda que computável, torna-se, na prática, inviável em um programa de computador e, particularmente, em um sistema embarcado com restrições de memória e tempo de execução dos componentes. Portanto, a estrutura do CSML apresenta uma vantagem importante sobre a abordagem monolítica, pois os modelos e supervisores obtidos para o caso do AUV apresentam um número muito menor de estados. Ainda que exista um modelo $G_{locx}$ para uma especificação local com 192 estados, tal modelo é empregado durante a fase de síntese dos supervisores, não sendo empregado para implementação da arquitetura do CSML (supervisores modulares $S_x$ e subplantas $G_{locx}$). Assim, a estrutura com maior número de estados corresponde à especificação $E_M$ (restrição à execução paralela de manobras), com 72 estados. Além disso, como o supervisor modular $S_x$ é igual à linguagem alvo $K_{locx}$ então é possível empregar a especificação $E_x$ como supervisor, reduzindo, para o caso da especificação $E_M$ com restrição à execução paralela de manobras, o número de estados do supervisor de 72 para 5. Desse modo, o número reduzido de estados favorece a implementação da arquitetura CSML em sistemas microprocessados e microcontrolados.

## 4.6. SIMULAÇÃO DO CONTROLE SUPERVISÓRIO

Através da ferramenta Supremica também é possível simular a ocorrência de diversos caminhos ou trajetórias de eventos de modo que o projetista possa comprovar visualmente a ação dos supervisores sobre a planta. Esses testes são muito importantes pois permitem detectar possíveis erros na modelagem da planta e das especificações numa fase do projeto em que as correções ainda não são muito onerosas. No simulador, eventos habilitados ou possíveis de serem ativados são indicados pela cor verde, e eventos que ocorreram no último passo da simulação são indicados no modelo pela cor rosa. Nessa ferramenta, os eventos não-controláveis são representados pelo seu nome escrito com a fonte em itálico, ao passo que eventos controláveis estão com formato

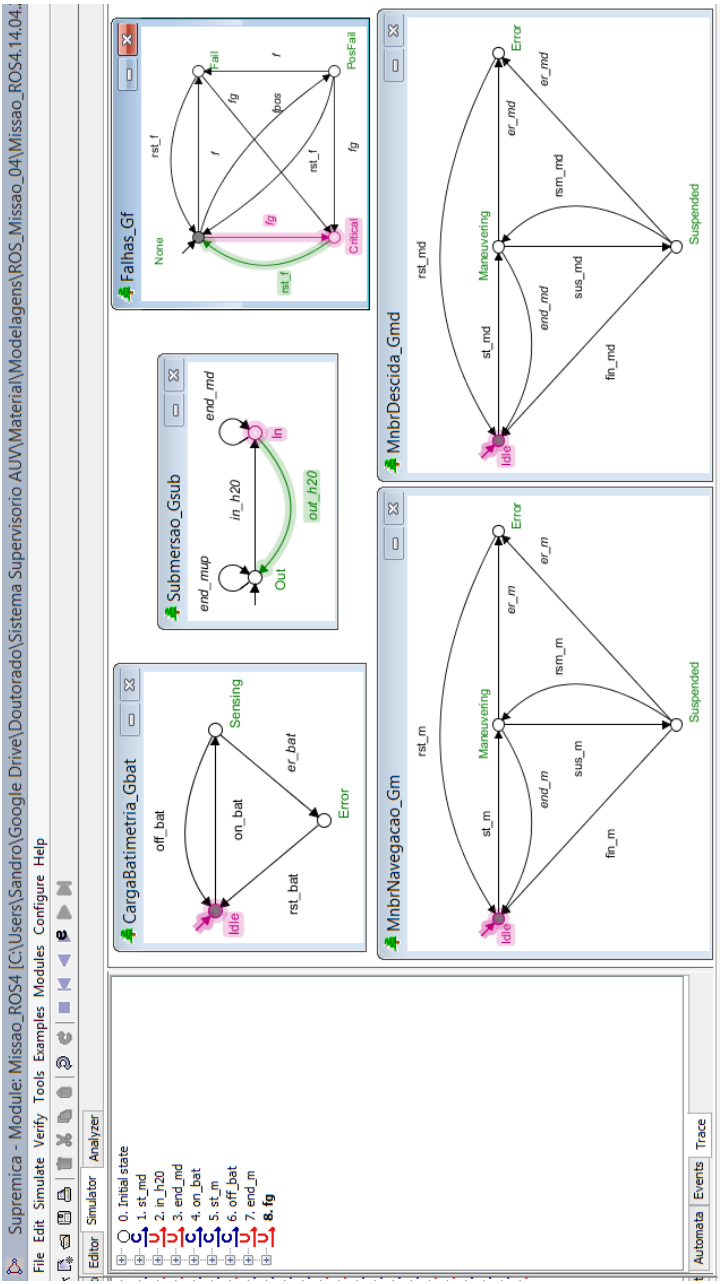


**Figura 4.20:** Tela do programa Supremica com o modelo de falhas e manobra de

normal. O estado também na cor rosa indica o estado atual de um modelo.

A figura 4.20 apresenta uma captura de tela da ferramenta com a simulação da ocorrência de uma falha grave com o veículo submerso. Este caso ilustra a ação do supervisor da especificação que impede que a manobra de navegação seja iniciada ou retomada quando uma falha grave ocorre (especificação $E_{FG\_mn}$) e um sensor de carga útil está ligado (especificação $E_{F\_FG\_bat}$). A sequência de eventos que levam o sistema até o estado mostrado na figura 4.20 é apresentada na tabela 4.3, onde a coluna *Nº* indica o número do passo dentro da sequência de eventos, a coluna *Autômato* denota o nome do modelo descrito na seção 4.3 e nome correspondente entre parêntesis usado no software Supremica, a coluna *Evento* e *Controlável* apresentam, respectivamente, o símbolo do evento e se este é controlável ou não. Finalmente a coluna *Descrição* fornece um breve comentário sobre o passo em questão.

**Tabela 4.3:** Sequência de eventos para caso de falhas grave

| Nº. | Autômato | Evento | Controlável | Descrição |
|---|---|---|---|---|
| 1 | $G_{md}$ (MnbrDescida_$G_{md}$) | *st_md* | sim | Início da manobra de descida |
| 2 | $G_{sub}$ (Submersao_$G_{sub}$) | *in_h2o* | não | Submersão do veículo |
| 3 | $G_{sub}$ (Submersao_$G_{sub}$) | *end_md* | não | Fim da manobra de descida |
| 4 | $G_{bat}$ (CargaBatimetria_$G_{bat}$) | *on_bat* | sim | Ativação do sonar de batimetria |
| 5 | $G_{m}$ (MnbrNavegacao_$G_{m}$) | *st_m* | sim | Início da manobra de navegação |
| 6 | $G_{m}$ (MnbrNavegacao_$G_{m}$) | *end_m* | não | Fim da manobra de navegação |
| 7 | $G_{bat}$ (CargaBatimetria_$G_{bat}$) | *off_bat* | sim | Desligamento do sonar de batimetria |
| 8 | $G_{f}$ (Falhas_$G_{f}$) | *fg* | não | Ocorrência de falha grave |

Basicamente, a sequência descreve a ação de submergir, ligar um sensor de carga útil, no caso, o sonar de batimetria, realizar uma trajetória (manobra de navegação) para aquisição de dados. Ao final da aquisição dos dados a manobra é terminada e o sensor desligado, e na sequência ocorre o evento de falha grave (*fg*). Após o evento de falha, observa-se tanto o impedimento do início da manobra de navegação quanto a ação de ligar o sensor de batimetria, pois os eventos controlá-

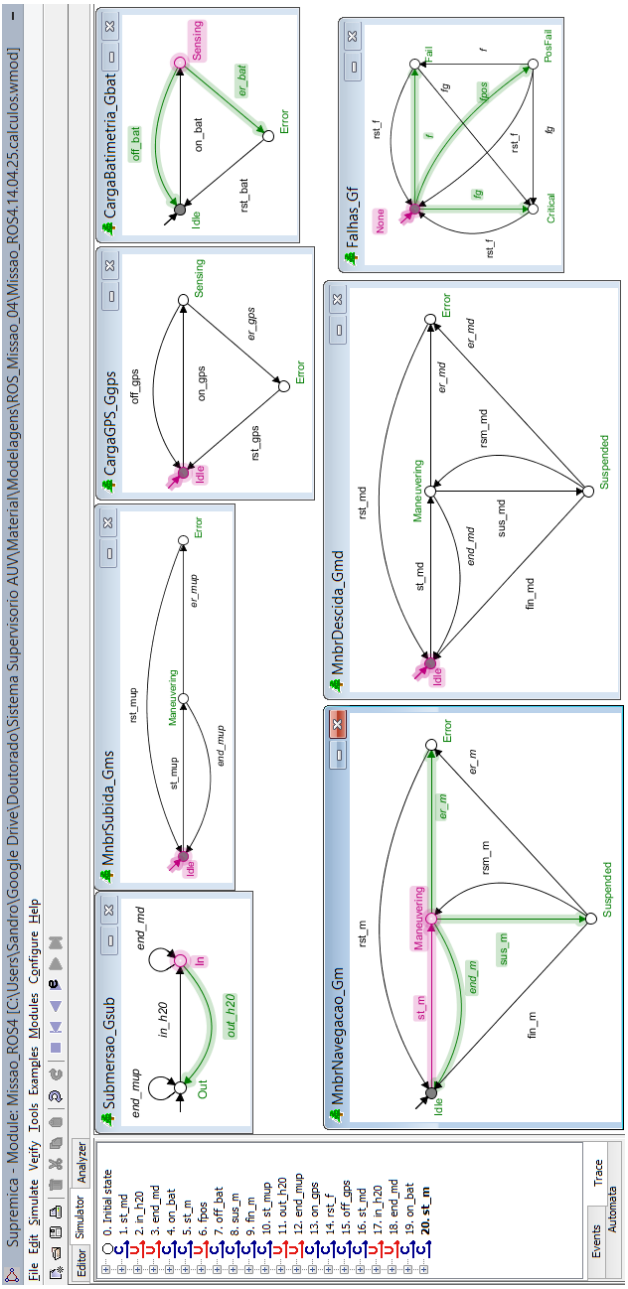


**Figura 4.21:** Tela do programa Supremica apresentando sequência de vários caminhos possíveis após correção de falha de posicionamento.

veis *st_m* do modelo de manobra de navegação $G_m$ e *on_bat* do modelo do sonar de batimetria $G_{bat}$ não estão habilitados.
Outro exemplo da ação dos supervisores pode visualizado na figura 4.21, onde apresenta-se o estado do sistema uma vez o mesmo tenha se recuperado de uma falha de posicionamento. Os modelos da planta considerados foram: falhas ($G_f$), sensor de carga útil GPS ($G_{GPS}$), submersão ($G_{sub}$), e as manobras de descida ($G_{md}$), navegação ($G_m$) e subida ($G_{mup}$). Algumas das especificações diretamente relacionadas com esse exemplo são: impedir que o sensor de batimetria seja ligado em falhas de posicionamento ($E_{FPOS\_bat}$); impedir que a manobra de navegação seja ligada ou retomada em falhas de posicionamento ($E_{FPOS\_mn}$); somente ativar o GPS na superfície ($E_{sub\_GPS}$); ou não ativar a manobra de navegação na superfície ($E_{sub\_mn}$). Inicialmente o veículo submerge por meio da manobra de descida. Ao chegar à profundidade desejada, a manobra de navegação para aquisição de dados via sensor de batimetria é iniciada, com o respectivo sensor ligado. O sistema de diagnóstico detecta então um erro considerável na estimação da localização do sistema (erros nos algoritmos de localização, falha em algum sensor, estimação de erro obtido a partir de um modelo de detecção de falhas, etc.) gerando o evento de falha de posicionamento (*fpos*). A manobra de navegação é então finalizada, e o veículo inicia a manobra de emersão à superfície. Ao chegar à superfície o veículo realiza a calibração do sistema de localização por meio do GPS (*on_gps*). Uma vez terminada a correção, desativa-se o estado de falha do veículo e o GPS é desligado. O veículo retorna ao ponto em que se encontrava inicialmente, e retoma a manobra de navegação com a batimetria ligada. A sequência de eventos simulada é apresentada na tabela 4.4.

**Tabela 4.4:** Sequência de eventos para caso de correção por GPS

| **Nº.** | **Autômato** | **Evento** | **Controlável** | **Descrição** |
|---|---|---|---|---|
| 1 | $G_{md}$ (MnbrDescida_$G_{md}$) | *st_md* | sim | Início da manobra de descida |
| 2 | $G_{sub}$ (Submersao_$G_{sub}$) | *in_h2o* | não | Submersão do veículo |
| 3 | $G_{sub}$ (Submersao_$G_{sub}$) | *end_md* | não | Fim da manobra de descida |

| Nº. | Autômato | Evento | Controlável | Descrição |
|---|---|---|---|---|
| 4 | $G_{bat}$ (CargaBatimetria_$G_{bat}$) | *on_bat* | sim | Ativação do sonar de batimetria |
| 5 | $G_m$ (MnbrNavegacao_$G_m$) | *st_m* | sim | Início da manobra de navegação |
| 6 | $G_f$ (Falhas_$G_f$) | *fpos* | não | Detecção de erro de posicionamento |
| 7 | $G_{bat}$ (CargaBatimetria_$G_{bat}$) | *off_bat* | sim | Desligamento do sonar de batimetria |
| 8 | $G_m$ (MnbrNavegacao_$G_m$) | *sus_m* | sim | Suspensão da manobra de navegação |
| 9 | $G_m$ (MnbrNavegacao_$G_m$) | *fin_m* | sim | Finalização da manobra de navegação |
| 10 | $G_{ms}$ (MnbrSubida_$G_{ms}$) | *st_mup* | sim | Início da manobra de subida |
| 11 | $G_{sub}$ (Submersao_$G_{sub}$) | *out_h2o* | não | Emersão do veículo |
| 12 | $G_{ms}$ (MnbrSubida_$G_{ms}$) | *end_mup* | não | Fim da manobra de subida |
| 13 | $G_{GPS}$ (CargaGPS_$G_{GPS}$) | *on_gps* | sim | Ativação do GPS |
| 14 | $G_f$ (Falhas_$G_f$) | *rst_f* | sim | Correção do erro de posição indicado pelo evento de reinício de falha |
| 15 | $G_{GPS}$ (CargaGPS_$G_{GPS}$) | *off_gps* | sim | Desligamento do GPS |
| 16 | $G_{md}$ (MnbrDescida_$G_{md}$) | *st_md* | sim | Início da manobra de descida |
| 17 | $G_{sub}$ (Submersao_$G_{sub}$) | *in_h2o* | não | Submersão do veículo |
| 18 | $G_{md}$ (MnbrDescida_$G_{md}$) | *end_md* | não | Fim da manobra de descida |
| 19 | $G_{bat}$ (CargaBatimetria_$G_{bat}$) | *on_bat* | sim | Ativação do sonar de batimetria |
| 20 | $G_m$ (MnbrNavegacao_$G_m$) | *st_m* | sim | Início da manobra de navegação para retomada da operação de batimetria |

## 4.7. COMENTÁRIOS

As etapas apresentadas nas seções anteriores mostraram a organização dos subsistemas do AUV em vários componentes essenciais que servem de base para aplicação da TCS, em particular o CSML, na modelagem e síntese de supervisores para missões de AUVs. Assim, foram estabelecidas especificações de operação e intertravamento para um veículo atuando em ambientes não-estruturados com as características encontradas em ambientes de lagos de barragens. Tais especificações restrigem o comportamento em malha-aberta dos modelos baseados em autômatos da planta. Desse modo é possível derivar supervisores que garantam que tais especificações sejam atendidas durante a execução de qualquer missão e durante toda a sua realização.

O resultado final da modelagem e síntese consiste na especificação de uma lógica de controle para representação e execução de missões de AUVs, com propriedades formais bem definidas, como ausência de bloqueios e atendimento de especificações. Além disso, a ação do CSML é garantida ser minimamente restritiva, no sentido de que somente as sequências de eventos consideradas indesejadas são impedidas de ocorrer, permitindo que todas as demais sequências possam ocorrer na planta, evitando, como em algumas abordagens da TCS (XU, ZHANG e FENG, 2004a; XU, ZHANG e FENG, 2004b; MOLINA *et al.*, 2010), limitar a missão a uma única sequência possível de eventos. Outra vantagem no emprego da TCS consiste na abstração do sistema complexo do AUV, constituído por vários elementos de hardware e software, distribuídos em um sistema embarcado distribuído, favorecendo a análise e especificação de missões.

A partir da especificação da lógica formal para representação e execução de missões, é necessário desenvolver um SCM que seja capaz de implementar tal lógica, além de realizar outras funções também de responsabilidade do SCM, a exemplo da tradução do arquivo de missão, da definição de coordenadas e parâmetros de configuração de missão, das operações de planejamento e replanejamento de missões e da escolha ótima de sequência de eventos que realizem a missão. Particularmente a arquitetura de implementação do CSML (QUEIROZ e CURY, 2002), permite empregar os modelos da planta e dos supervisores diretamente no sistema a ser desenvolvido.

Assim, no próximo capítulo é apresentada uma arquitetura para SCM com base na arquitetura de implementação do CSML, porém adaptada ao problema de missão de AUVs. Um segundo componente é

incluído ao SCM, denominado de Gerenciador de Missão (GM), responsável por realizar operações complementares ao CSML, como escolha de eventos, tradução de arquivos, planejamento e replanejamento.

# 5. PROPOSTA DE ARQUITETURA PARA SCM

Este capítulo apresentada a proposta de uma arquitetura híbrida deliberativo-reativa para o Sistema de Controle de Missão (SCM) de um AUV, que incorpora o CSML, mostrado no capítulo anterior, e introduz o Gerenciador de Missão (GM), elemento que complementa a ação do CSML no nível da missão. A missão é descrita pela evolução de modelos baseados em autômatos, descritos pelo CSML e coordenados pelo GM que traduz o arquivo de missão em um plano de missão e realiza a escolha da melhor sequência de eventos que irá atender ao plano original de missão.

De modo a empregar diretamente os modelos formais introduzidos no capítulo anterior, é utilizada a arquitetura de implementação do CSML, mostrada na seção 5.1. Na sequência, a seção 5.2 apresenta uma visão geral da arquitetura proposta para o SCM desenvolvido neste trabalho. As seções 5.3 e 5.4 descrevem os dois principais componentes do SCM proposto, o CSML e o GM, mostrando como os modelos e supervisores baseados em autômatos são traduzidos para a estrutura de controle de missão e como as ações de planejamento e replanejamento são tratadas e incorporadas ao sistema de missão. O método para desenvolvimento e testes do SCM é apresentado na seção 5.5, e algumas observações sobre o SCM proposto são comentadas na seção 5.6. Ao final, um breve resumo do capítulo é mostrado.

## 5.1. ARQUITETURA DE IMPLEMENTAÇÃO DO CSML

O comportamento descrito pelos modelos baseados em autômatos da planta, composta pelos subsistemas, especificações e estruturas de supervisão determinam uma lógica de controle para a realização de missões em AUVs, conforme já apresentado no capítulo anterior. Para que o AUV possa desenvolver missões com as características especificadas por tal lógica torna-se necessário que a mesma seja implementada por alguma estrutura do sistema embarcado do veículo. Nesse sentido, o comportamento descrito pelos modelos baseados em autômatos deve ser mapeado para uma estrutura de controle responsável

por garantir, do ponto de vista lógico, que as missões apresentem as propriedades desejadas para sua segurança e operação.

A arquitetura de implementação do CSML, apresentada em (QUEIROZ e CURY, 2002), permite que os modelos baseados em autômatos empregados para a análise da planta e síntese de supervisores sejam empregados diretamente no sistema a ser controlado, diminuindo os erros oriundos da adaptação dos modelos para uma estrutura lógica equivalente. Assim, a arquitetura do SCM proposta neste trabalho, a ser mostrada a partir da seção 5.2, emprega como base a arquitetura de implementação do CSML.

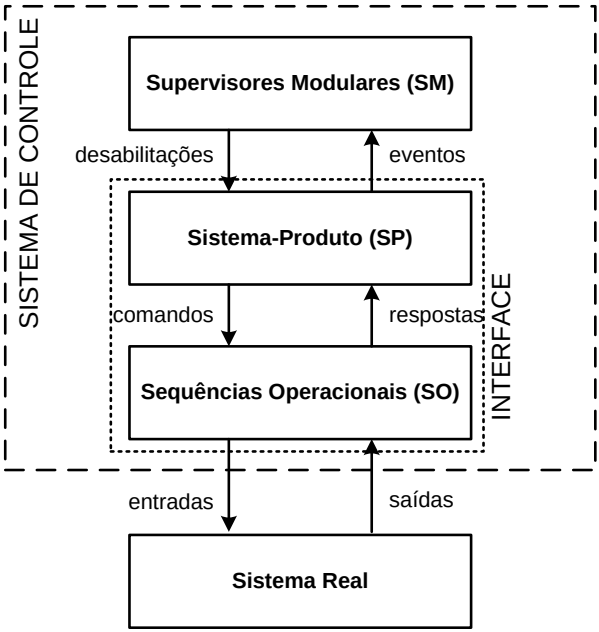


**Figura 5.1:** Arquitetura de implementação do CSML.
Adaptado de Queiroz e Cury (2002).

A implementação do CSML é feita em uma arquitetura hierárquica de 3 camadas, como pode ser vista na figura 5.1, permitindo a implementação direta dos supervisores em uma linguagem de programação (QUEIROZ e CURY, 2002; QUEIROZ, 2004). Uma das críticas existente a respeito do emprego de SEDs, particularmente a TCS (LIU e DARABI, 2002), consiste na dificuldade em implementar ou

traduzir os modelos e resultados teóricos empregados durante a fase de análise e síntese, uma vez que a ação dos supervisores depende dos estados e eventos da planta modelada, nível este de abstração que pode não manter uma correspondência direta com o sistema real. Através da arquitetura de implementação do CSML tal limitação é excluída ao permitir usar diretamente os modelos abstratos na implementação do sistema.

A arquitetura de implementação do CSML evita a necessidade de modificação ou adaptação dos supervisores modulares (SM) ao empregar uma interface com o sistema real. Essa interface está composta pelo sistema-produto (SP) e pelo nível de sequências operacionais (SO). O SP corresponde ao modelo abstrato da planta, que executa os eventos não desabilitados pelo nível de SM, enviando comandos ao nível de SO, e atualizando os modelos de acordo com as respostas do nível inferior. Os eventos controláveis podem ser desabilitados pelos supervisores modulares, contudo, os não-controláveis não podem ter sua ocorrência impedida. Por sua vez, o nível de SO traduz os comandos dos modelos abstratos da planta em sinais de controle, as entradas do sistema real, e lê os sinais de saída do sistema real, gerando as respostas enviadas ao SP. Assim, o SP permite a execução direta do SM através da tradução dos sinais e comandos reais da planta, enviados ou recebidos pelo nível SO, em eventos no nível de SP. Desse modo, a especificação da lógica de controle, descrita pela dinâmica discreta do nível sistema-produto (planta abstrata) sob ação do nível supervisores modulares, corresponde exatamente ao comportamento do SED composto pelo SP e pelo SM. A relação com a planta real é, portanto, mantida através da tradução realizada pelo nível SO.

## 5.2. SCM BASEADO EM CSML

Para organização das funcionalidades lógicas do AUV, divide-se o sistema embarcado do veículo em dois níveis principais: o nível de missão e o nível de controle do veículo. O nível de controle do veículo, compreendido pelos níveis inferiores da arquitetura, contém os diversos componentes de software e hardware responsáveis pelo funcionamento do AUV e que implementam as funcionalidades comentadas na seção 2.1. No nível de missão, encontra-se o SCM proposto neste trabalho. A organização completa do sistema embarcado segue a arquitetura robótica híbrida deliberativa-reativa, combinando comportamentos que

realizam ações sem planejamento (reatividade), como a ação de desvio de obstáculos ou a própria ação dos supervisores modulares, e comportamentos que necessitam tomar decisão baseados em mecanismos de planejamento. Tais ações e mecanismos serão explicados com mais detalhes na seção 5.4.

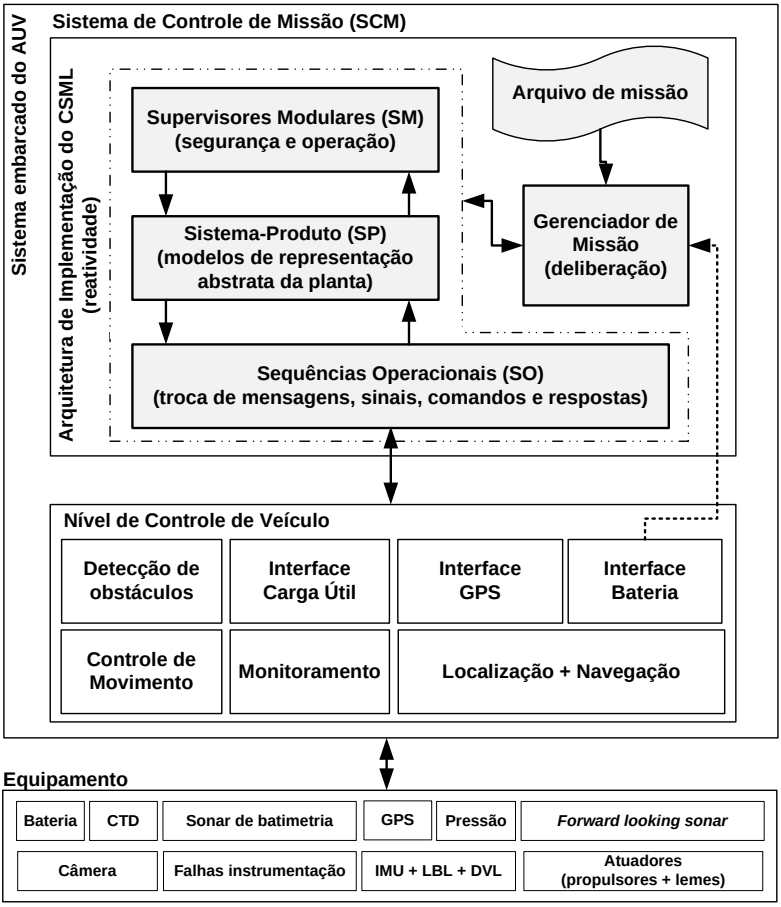


**Figura 5.2:** Organização do sistema embarcado do AUV, incluindo o SCM e subsistemas

O SCM proposto nesse trabalho é apresentado na figura 5.2 e contém dois componentes principais: a arquitetura de implementação do Controle Supervisório Modular Local (CSML) e o Gerenciador de Missão (GM). Essencialmente, o problema de representação e execução

de missão é dividido pelos dois componentes. A ação combinada dos dois componentes propicia ao AUV atuar em ambientes não-estruturados, conferindo reatividade a eventos inesperados ao mesmo tempo em que ações são planejadas de acordo com os objetivos da missão. A ação do controle supervisório é considerada reativa no sentido em que não há planejamento e sim habilitação e desabilitação de eventos controláveis, enquanto que a ação do GM é deliberativa devido a presença do planejamento e replanejamento de missões.

Para a representação da missão, modelos formais são empregados pelo CSML para garantir que requisitos de segurança e operação do veículo sejam atendidos para qualquer tipo de missão. Além disso, a ação do CSML é reativa no sentido em que sua atuação sobre o sistema independe das habilidades de planejamento do veículo, realizando um mapeamento direto entre os eventos que representam as entradas sensoriais (eventos não-controláveis) e os eventos que representam os comandos sobre os atuadores (eventos controláveis), possuindo um tempo relativamente curto de processamento. Esse mapeamento é incluído diretamente no sistema embarcado do veículo através da arquitetura de implementação do CSML, mostrada na seção 5.1. A interface entre o SCM, responsável pelo controle da missão, o nível de controle de veículo é realizada pelo nível SO do CSML.

Por sua vez, o GM possui capacidades deliberativas de planejamento e replanejamento, aplicando um método de escolha da melhor sequência de eventos, entre todas habilitadas pelo CSML, no sentido de guiar o veículo para os objetivos contidos no arquivo de missão, sem necessariamente seguir um caminho sequencial único. O GM também é responsável por traduzir o arquivo de missão em eventos e parâmetros de configuração para os vários subsistemas do veículo, compondo assim o plano de missão.

Para escolha da melhor sequência de eventos é necessário considerar os objetivos da missão, o estado do veículo e do ambiente, e o nível de bateria disponível para, a partir dessas informações, empregar alguma estratégia de otimização segundo critérios como tempos ou distâncias de deslocamentos, consumo de bateria, etc. Contudo, a ocorrência de eventos não-deterministas pode, em algumas situações, impor uma adaptação à missão original, havendo inclusive a necessidade de gerar um novo plano de missão, operação essa denominada de replanejamento.

A execução da missão pelo GM consiste em, basicamente, selecionar e ativar o próximo evento controlável previsto no plano de missão, original (planejamento) ou modificado (replanejamento), ou

aguardar pela ocorrência de próximo evento não-controlável, até que todos os eventos previstos no plano de missão tenham sido ativados ou gerados. Associados a alguns eventos encontram-se parâmetros de missão, contendo informações a respeito da configuração das ações que os componentes no nível de controle de veículo devem executar.

Nas abordagens empregadas na área de Engenharia baseada em Modelos (MDE do inglês *Model-Driven Engineering*) a utilização de diversas visões e modelos favorece a concepção de sistemas muito menos presos a tecnologia de implementação e muito mais próximos ao domínio do problema (SELIC, 2003). Assim, de modo similar a MDE, a adoção da TCS permite a visualização do problema de realização de missões do ponto de vista dos modelos lógicos que representam o problema em si (missões) e suas possíveis soluções, sem, contudo, definir ou impor uma forma de implementação em uma linguagem de programação ou plataforma de hardware. Assim, o SCM desenvolvido apresenta também como vantagem significativa, a relativa independência da plataforma de implementação, pois os modelos empregados para descrever a missão não dependem de características dos elementos de baixo nível do AUV, sendo todos encapsulados no nível SO do CSML. Portanto, é possível, em tese, empregar o SCM proposto em veículos subaquáticos com características distintas aos considerados neste trabalho, e até mesmo, em outros tipos de veículos móveis.

Outro aspecto fundamental da arquitetura proposta consiste no fato do SCM explorar as várias sequências de eventos habilitadas pelo CSML, diferente de abordagens da TCS aplicada a veículos móveis, como em Xu, Zhang e Feng (2004b), não restringindo a missão à execução de um único caminho de eventos. Tal característica permite combinar técnicas de planejamento para escolha dos eventos, conferindo maior flexibilidade na realização de missões, ao mesmo tempo em que permite tratar com eventos não-deterministas comumente encontrados em missões em ambientes não-estruturados.

A seguir são apresentadas as propostas para os dois componentes que formam o SCM, o CSML e o GM, incluindo as responsabilidades de cada elemento para a representação e execução de missão de AUVs em ambientes não-estruturados.

## 5.3. COMPONENTE CONTROLE SUPERVISÓRIO MODULAR LOCAL

O objetivo do CSML no SCM consiste em implementar a estrutura de controle supervisório empregada na representação e execução de missões através da evolução dos vários subsistemas modulares, descritos por autômatos. Os autômatos modulares, cuja modelagem foi apresentada no capítulo anterior, representam o comportamento individual dos vários subsistemas considerados sob o ponto de vista de realização de missão, ou seja, definem uma especificação desejada para a lógica de funcionamento das missões de AUVs. As ações consideradas indesejadas, por não atender a alguma especificação de segurança e operação do veículo são impedidas de ocorrer através da desabilitação dos eventos controláveis pelos supervisores. Assim, para qualquer tipo de missão é possível evitar que tais ações indesejadas ocorram, devido à ação dos supervisores sobre os modelos da planta.

Conforme já mencionado na seção 5.1, a arquitetura de implementação do CSML é dividida em três níveis: SO, SP e SM. O nível SP contém os modelos abstratos, baseados em autômatos, dos subsistemas que descrevem o comportamento discreto em malha-aberta do AUV, segundo o ponto de vista da missão. De acordo com a modelagem e síntese apresentados no capítulo 4, foram considerados 12 autômatos para descrever o comportamento discreto do AUV. Já o nível SM implementa o comportamento concorrente dos autômatos e respectivas ações de desabilitação dos 58 supervisores modulares reduzidos obtidos pela síntese formal.

Finalmente, o nível SO é responsável pela troca de sinais entre o nível de controle de veículo, contendo os diversos subsistemas reais do AUV, e o SCM, e depende diretamente do tipo de veículo e seu sistema embarcado. Basicamente a ação do SO consiste na tradução de comandos, relacionados com os eventos controláveis, em entradas para o AUV e de sinais provenientes do AUV em eventos não-controláveis. Os modelos abstratos do nível SP são atualizados a cada ocorrência de um evento não-controlável ou controlável, e os eventos controláveis são habilitados / desabilitados de acordo com a ação do nível SM.

## 5.4. COMPONENTE GERENCIADOR DE MISSÃO

O papel do CSML consiste em impedir que sequências consideradas indesejáveis sejam realizadas pela planta. O CSML assume que os eventos são gerados espontaneamente pela planta (seção 4.1) e, portanto, não é seu papel definir qual sequência de eventos controláveis deve ser gerada para realizar uma missão. Nesse sentido, o Gerenciador de Missão (GM) é o elemento do SCM responsável em complementar a ação do CSML, escolhendo a melhor sequência de referência para os eventos entre todas as possíveis sequências habilitadas pelo CSML e que realize os objetivos da missão. As principais responsabilidades do GM, incluindo a escolha da melhor sequência de eventos, podem ser agrupadas nos seguintes tipos de funcionalidades:

- **Tradução e representação:** ação de traduzir o arquivo de missão em uma estrutura de dados para representação da missão (plano de missão) contendo cada um dos objetivos de missão e respectivos parâmetros.
- **Planejamento:** ação de escolher a melhor sequência de objetivos de acordo com algum critério de otimização e que ao mesmo tempo corresponda a uma sequência permitida pelo CSML.
- **Replanejamento:** ação de modificar o plano original de missão em função da ocorrência não-determinista de eventos, como falhas e erros, e dos níveis disponíveis de bateria, guiando o veículo para estados seguros que permitam prosseguir a missão, permitindo realizar, quando necessário, um subconjunto de objetivos cancelando algumas etapas da missão ou a própria missão em si.
- **Execução:** ação de decidir qual será a próxima ação que irá guiar o veículo em direção aos objetivos, particularmente, na escolha do próximo evento controlável (comando) que deve ser ativado ou na espera pelo próximo evento não-controlável (sinal) que deve ocorrer no AUV ou no ambiente.

A tradução do arquivo de missão é realizada somente ao início da missão e consiste no mapeamento da sintaxe, linguagem ou forma de representação de missões para estruturas de dados contidas no SCM. Dados como pontos de início e fim de deslocamentos ou trajetórias de referência, tempos máximos para realização dos deslocamentos, instantes em que os sensores de carga útil devem ser ligados ou desligados são exemplos de informações contidas no arquivo de missão.

Essencialmente obtém-se as diversas atividades, as respectivas regiões de operação e os parâmetros de configuração associados.

Outro aspecto a ser considerado pelo GM durante o planejamento consiste na tradução de certos parâmetros da missão em comandos para os demais subsistemas do veículo. Para isso são definidos três tipos de mapeamento. No primeiro caso, encontram-se os eventos controláveis que possuem parâmetros associados e que necessitam ser repassados aos subsistemas correspondentes, porém que não fazem parte da modelagem ou do formalismo que descreve o componente em si. Tal parametrização ocorre com o modelo de manobras, onde ao evento de início de manobra (*st_mi* onde *mi* $\in$ *{descida, navegação, retorno, subida}*) encontra-se associado a posição e orientação final desejados, a margem de erro em torno a esse ponto final para que se considere que o veículo alcançou o destino final, e o *timeout* ou tempo máximo para execução do deslocamento. O segundo caso consiste naqueles eventos controláveis que são mapeados diretamente em sinais ou comandos, como são os eventos de ligar ou desligar sensores, e que não possuem parâmetros associados. Finalmente, no terceiro caso estão os eventos não-controláveis que são associados a diversos sinais de origem. Um exemplo desse último caso de mapeamento ocorre com os eventos de falhas, onde sinais de erros de vários subsistemas possuem a mesma interpretação no nível de missão, gerando, assim, os três tipos de falhas considerados (simples, graves e de posição) no autômato de falhas (seção 4.3).

A partir do conjunto de objetivos, o algoritmo de planejamento busca encontrar a melhor forma de realizar a sequência de operações. Empregando um critério de otimização, como consumo de bateria, deslocamentos entre etapas, é possível ordenar os objetivos em uma ordem ideal de fase. Essa ordem ideal é então codificada em uma estrutura de dados (plano de missão) e que contém a sequência em que os eventos controláveis e não-controláveis devem ocorrer para que o veículo realize a totalidade da missão. Esta sequência de eventos é denominada de sequência de referência pois define qual é o próximo evento controlável que o GM deverá selecionar ou qual o próximo evento não-controlável que o GM deverá aguardar ser gerado pelo nível de veículo do AUV. A ação de criar o plano de missão empregando um critério de otimização e a partir dos dados obtidos do arquivo de missão é denominada aqui de planejamento de missão. Assim, idealmente em condições normais de operação da missão, sem falhas, e com potência suficiente, o veículo deverá seguir este plano de missão.

Contudo, para evitar que o usuário tenha que codificar todas as possibilidades de ocorrência de eventos não-deterministas, o GM deve implementar estratégias que permitam tratar as sequências de eventos que não correspondem ao esperado no plano de missão. Algumas ocorrências podem gerar a suspensão temporária da missão para a realização de alguma correção, como a ação de subir à superfície para executar um ajuste por GPS, o que não implica no cancelamento ou modificação da missão original, mas somente na suspensão da etapa em realização. Tais ocorrências são tratadas pelo GM através de políticas de atribuição de prioridades e modos de operação que permitem a suspensão temporária da execução da missão para tratar uma situação em particular, como a correção por GPS. A missão, neste caso, é retomada assim que a situação, ou problema, não estiver mais presente ou for corrigida. Outras ocorrências podem acarretar na geração de um novo plano de missão com exclusão de alguns objetivos, sendo necessário avaliar a carga disponível de bateria e o tipo de carga útil disponível. Finalmente, ante ocorrências mais graves, visando à segurança do veículo, a missão pode ser cancelada em sua totalidade. Como estratégias para tratar ocorrências não especificadas no arquivo de missão incluem-se: escolha de modos alternativos ao modo normal de operação, como correção por GPS ou cancelamento de missão; e a obtenção de um novo plano de missão (replanejamento). Tais funcionalidades serão explicadas mais adiante.

A partir do plano de missão o GM necessita executar a sequência de referência de eventos, selecionando um evento controlável, como ligar ou desligar um sensor, iniciar ou suspender uma manobra, ou aguardar que um evento não-controlável seja gerado pelo nível de controle de veículo do AUV, como fim de manobra ou algum evento de falha. Portanto, é necessário que o GM implemente uma política para a escolha de eventos levando em consideração o plano de missão, a ocorrência de eventos não-deterministas e os níveis de bateria a mesmo tempo em que são respeitados os estados válidos e seguros dos modelos baseados em autômatos que descrevem a evolução da missão. Além disso, tanto o planejamento e quanto o replanejamento devem gerar sequências de eventos que correspondam a uma sequência $s$ de eventos realizável pelos modelos abstratos do CSML, ou seja, a sequência deve pertencer à linguagem do sistema-produto (plantas modulares) sob ação dos supervisores modulares ($s \in L(S/G)$).

O uso de prioridades é uma das formas de implementar uma política para a seleção de eventos sem infringir os estados válidos do CSML ao mesmo tempo que conduz o veículo para os objetivos da

missão. Desse modo, através da política de atribuição de prioridades aos eventos é possível evitar que a ação do GM em conjunto com o CSML cause uma inconsistência entre o evento escolhido pelo GM e os eventos habilitados pelo CSML. Essa atribuição da prioridade depende da fase em curso da missão, dos estados dos autômatos do CSML e do nível de potência disponível. As prioridades são dinâmicas no sentido em que a cada modificação no sistema (ocorrência de eventos, níveis de bateria) ou na missão pode haver uma alteração nas prioridades dos eventos. Eventos de prioridade baixa correspondem àqueles indesejados ou que não levam necessariamente à realização da missão. Sempre que existir mais de um evento com prioridade máxima, um sorteio é feito de modo a escolher somente um dos eventos.

Com o uso de uma política de atribuição dinâmica de prioridades é possível guiar o veículo em direção aos objetivos da missão, contudo sem necessariamente escolher, forçar ou aguardar a realização de um único evento ou sequência de eventos em particular. Além disso, tal política evita o bloqueio dos modelos baseados em autômatos do CSML pois sempre haverá um ou um conjunto de eventos mais prioritários a ser escolhido (ou aguardado, caso não-controlável) mesmo que, na pior das hipóteses, todos os eventos tenham a mesma prioridade e, nesse caso, a ocorrência de eventos não-controláveis não necessariamente conduza o veículo aos objetivos da missão.

A figura 5.3 apresenta a estrutura geral das responsabilidades e funcionalidades básicas do GM, atuando em conjunto com o CSML para compor o SCM. A ação essencial do SCM é descrita a seguir.

Ação de planejamento, com base no arquivo de missão, gera o plano da missão. O replanejamento, com base no estado atual do sistema e na carga disponível de bateria, pode ser executado gerando um novo plano de missão que substitui o original. Tanto as ações de planejamento quanto de replanejamento dependem dos modelos abstratos baseados em autômatos do SP e SM para comprovar se a sequência de referência de eventos que irá compor o plano de missão pertence à linguagem do sistema sob ação dos supervisores. O modo de seleção de operação, baseado nos estados do nível SP do CSML, determina se o veículo está em modo de operação normal, quando a sequência de eventos é aquela definida pelo plano de missão obtido pelo processo de planejamento ou replanejamento, ou em um modo de operação alternativo. Como exemplos de modos de operação alternativos citam-se os modos de cancelamento de missão ou correção por GPS. O plano de missão para esses modos são pré-fixados por código ou em uma estrutura de dados, e contém a sequência de referência desejada de eventos. A política de atri-

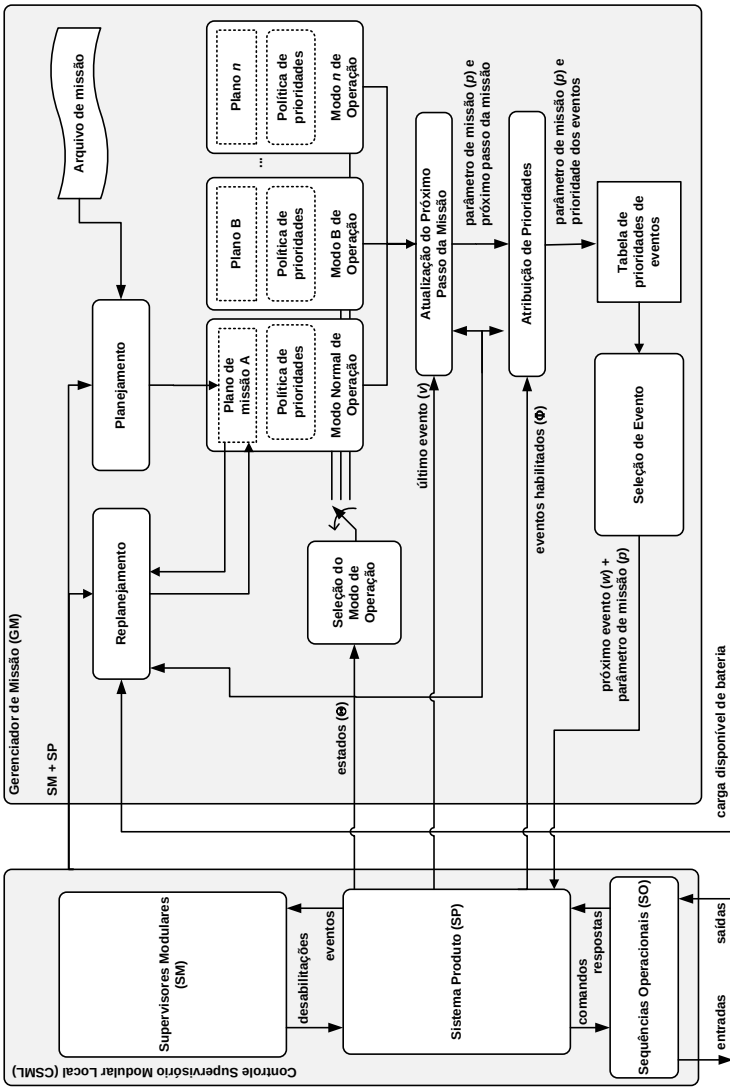


**Figura 5.3:** Estrutura geral das funcionalidades básicas do Gerenciador de Missão em conjunto com o CSML

buição de prioridades depende do estado do sistema (nível SP), do próximo passo da missão, de acordo com o modo de operação selecionado. Ao final, o evento de maior prioridade é selecionado: caso seja um evento controlável, os modelos abstratos do CSML são atualizados e os respectivos comandos são gerados pelo nível SO; e caso seja um evento não-controlável, o SCM aguarda que esse evento seja gerado espontaneamente pela planta. A missão avança para o próximo passo sempre que um evento controlável previsto no plano é selecionado ou quando um evento não-controlável previsto é gerado pela planta. Uma missão termina quando não mais houver eventos previstos no plano de missão.

O fluxograma representando o ciclo de funcionamento do SCM é apresentado na figura 5.4, e é descrito a seguir. A partir do arquivo de missão, o sistema de planejamento cria o plano de missão original. O nível sequências operacionais (SO) traduz os eventos controláveis em comandos e os envia para os respectivos subsistemas do nível de controle do veículo, e recebe sinais do AUV e os traduz em respostas para o nível SP. O nível SP transforma tais respostas em eventos não-controláveis. Os eventos são mantidos em uma fila de eventos para respeitar a ordem de ocorrência dos mesmos e permitir manter a correspondência entre os modelos abstratos baseados em autômatos e a planta real. Enquanto houver eventos, controláveis ou não-controláveis, o SCM irá atualizar os modelos do nível sistema-produto (SP) e supervisores modulares (SM), o modo de operação e também comprovará se a missão avançou. Quando não houver mais eventos na fila de eventos, o SCM verifica se há necessidade de realizar o replanejamento. Na sequência, o SCM atualiza as prioridades dos eventos para selecionar o próximo evento controlável ou não-controlável. Eventos controláveis são "gerados" apenas a partir da seleção de eventos, ou seja, somente depois de uma atualização de prioridades e da seleção do evento mais prioritário. Eventos não-controláveis, por sua vez, são gerados pela planta e recebidos pelo nível SO. Ambos tipos de eventos atualizam os modelos do CSML e o próximo passo da missão. O modo de operação é atualizado sempre que um novo evento modifica os estados dos autômatos do CSML. O laço é realizado enquanto houver eventos no plano de missão. Um modo alternativo de operação termina quando a sua sequência de referências é cumprida ou quando um modo de maior prioridade é selecionado.

Assim, para o desenvolvimento de um SCM baseado na arquitetura proposta nesse trabalho, torna-se necessário definir e resolver alguns pontos relacionados à tradução e representação, planeja-

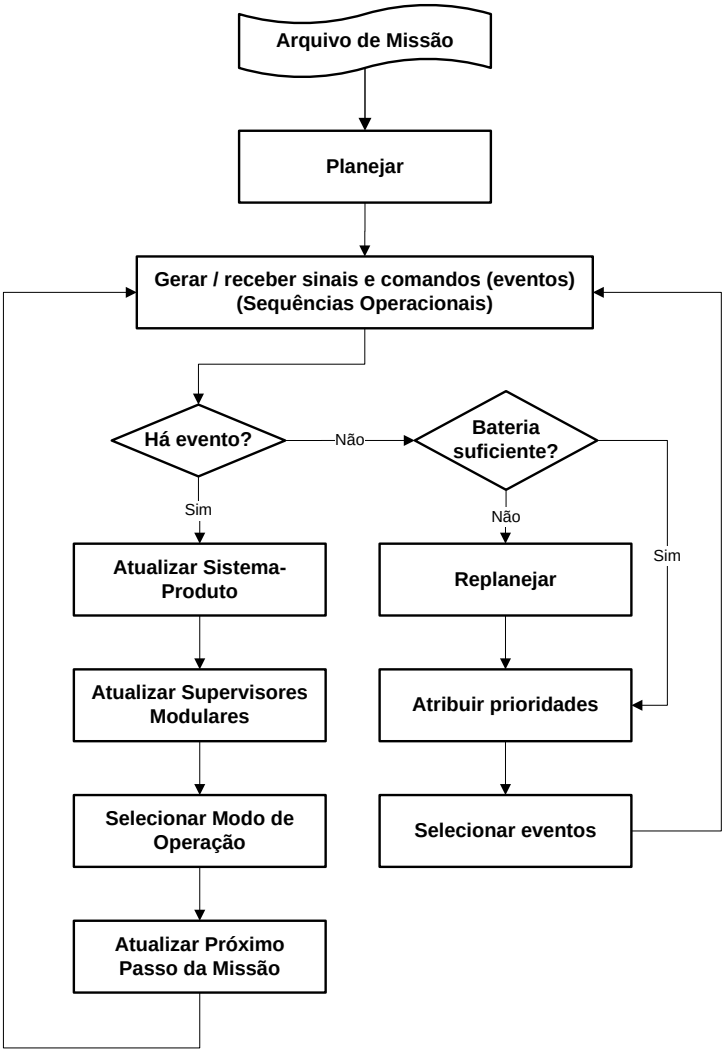


**Figura 5.4:** Fluxograma geral do SCM

mento e replanejamento de missões, bem como na troca de informações entre os modelos formais baseados em autômatos e os algoritmos empregados pelo GM. Tais questões serão discutidas nas seguintes seções.

### 5.4.1. Tradução e Representação de Missão

A missão é um conjunto de fases que podem ser executadas em qualquer ordem. A fase é um conjunto de tarefas que atendem a um determinado objetivo da missão. E uma tarefa está descrita por uma sequência de referência de eventos controláveis e não-controláveis e seus respectivos parâmetros. A missão necessita ser codificada em uma estrutura de dados e transferida para o sistema embarcado do veículo, para posteriormente ser interpretada pelo controlador de missão. Neste trabalho, o termo arquivo de missão é utilizado para designar a estrutura de dados que contém a missão do AUV definida pelo usuário.

No SCM proposto, a ação mais elementar e básica possível de ser representada no nível de missão corresponde a um evento controlável ou não-controlável. Uma tarefa está composta por uma sequência mínima de eventos controláveis e não-controláveis e que indicam a sequência de referência de ações e ocorrências que devem acontecer para completar a tarefa. A sequência de várias tarefas compõe uma fase e representa um conjunto de ações que atende a um objetivo específico da missão. A missão pode ser modelada como um conjunto de fases, que não necessariamente necessitam ocorrer em uma ordem sequencial, porém, cada fase precisa ter sua sequência de tarefas e respectivos eventos executada na sequencia correta para que o objetivo final daquela fase seja alcançado.

No arquivo de missão, propõe-se que uma tarefa σ pertencente a uma fase seja representada pela seguinte tupla:

$$\sigma_{ij} = \langle p_s, p_f, s_{bat}, s_{cam}, s_{CTD}, s_{GPS}, timeout, tipo_{manobra}, \delta \rangle \tag{5.1}$$

onde:

**$j \in \{1, 2, ..., n_t\}$:** índice da tarefa dentro da fase;
**$n_t$:** número de tarefas da fase;
**$i \in \{1, 2, ..., n_f\}$:** índice da fase na missão;
**$n_f$:** número de fases da missão;
**$p_s$:** posição e orientação iniciais da tarefa em coordenadas $<x_s, y_s, z_s, \phi_s, \theta_s, \psi_s>$;
**$p_f$:** posição e orientação finais da tarefa em coordenadas $<x_f, y_f, z_f, \phi_f, \theta_f, \psi_f>$;
**$s_{bat}$:** estado do sensor de batimetria (ligado ou desligado);
**$s_{cam}$:** estado da câmera (ligada ou desligada);
**$s_{CTD}$:** estado do sensor CTD (ligado ou desligado);

**$s_{GPS}$:** estado do GPS (ligado ou desligado);
***timeout*:** tempo máximo em segundos para a realização da manobra ou para alcançar a posição e orientação finais;
**$tipo_{manobra}$:** tipo da manobra a realizar (descida, navegação, retorno ou subida);
**δ:** margem de erro em torno a um ponto e orientação para considerar que o veículo encontra-se naquela posição.

A tarefa representa uma manobra de deslocamento, sendo possível que um ou mais sensores de carga útil estejam ligados, e que, portanto, pode ser diretamente mapeada em uma sequência de eventos controláveis e não-controláveis. Os eventos controláveis de início de manobra estão associados a parâmetros de missão, e todos os demais eventos não possuem parâmetros associados. Por exemplo, a realização da manobra de navegação entre dois pontos $P_1$ e $P_2$ quaisquer em no máximo 20 minutos, com o sensor de batimetria ligado, e com uma margem de erro de 0.5 m é expresso pela seguinte tarefa:

$$\sigma_{11} = \langle p_1, p_2, true, false, false, false, 1200, navegacao, 0.5 \rangle \quad (5.2)$$

**Tabela 5.1:** Exemplo de sequência de referência de eventos para uma tarefa

| Evento | Significado | Início e fim[4] | Batimetria[5] | Câmera | CTD | GPS | Timeout | Manobra | Margem de erro |
|---|---|---|---|---|---|---|---|---|---|
| *on_bat* | Ligar sensor de batimetria | - | *on* | *off* | *off* | *off* | - | - | - |
| *st_m* | Iniciar manobra de navegação | $P_1$, $P_2$ | *on* | *off* | *off* | *off* | 1200 | Nav.[6] | 0.5 |
| *end_m*[7] | Fim da manobra de navegação | - | *on* | *off* | *off* | *off* | - | - | - |

[4] Pontos de início e fim de manobra.
[5] Estado do sensor: *on* – ligado; *off* – desligado.
[6] Manobra de navegação.
[7] Evento sublinhado indica evento não-controlável.

| Evento | Significado | Início e fim[4] | Batimetria[5] | Câmera | CTD | GPS | Timeout | Manobra | Margem de erro |
|---|---|---|---|---|---|---|---|---|---|
| *off_bat* | Desligar sensor de batimetria | - | *off* | *off* | *off* | *off* | - | - | - |

A tarefa é então mapeada pelo GM em uma sequência de eventos controláveis e não-controláveis e respectivos parâmetros de missão. Esse mapeamento é fixo e está associado à sintaxe do arquivo de missão e os eventos dos modelos baseados em autômatos. O exemplo da tarefa mostrada na expressão 5.2 pode ser visualizado na tabela 5.1.

A realização da tarefa ocorre pela ação do GM, que irá selecionar o evento controlável ou aguardar pela ocorrência de um evento não-controlável conforme a sequência obtida a partir do arquivo de missão. Desse modo, a realização de várias manobras em uma região pode ser especificada com uma sequência de tarefas e agrupada em uma fase. Uma fase pode ser cancelada, caso alguma das suas tarefas não possa ser realizada, como por exemplo devido a uma falha ou a pouca quantidade de bateria disponível. Contudo, o cancelamento de uma fase não implica, necessariamente, no cancelamento da missão, ficando a cargo do GM a escolha pela realização ou pelo cancelamento das fases remanescentes.

### 5.4.2. Planejamento de Missão

A partir do arquivo de missão, o GM deve ser capaz de encontrar a sequência de eventos controláveis e não-controláveis que representam a evolução da missão. Para isso, o GM realiza duas ações: *1)* para cada tarefa, é feito o mapeamento direto em uma sequência de de referência de eventos e respectivos parâmetros de missão; *2)* e a escolha da melhor sequência de fases. Como neste trabalho uma fase consiste na sequência de tarefas, esta não pode ser modificada. Contudo, as fases da missão podem ser realizadas em uma ordem que minimize algum critério, como o consumo de bateria. As fases podem então ser realizadas em uma ordem diferente daquela especificada no arquivo de missão.

Como se deseja que cada fase seja executada em uma determinada sequência, cabe ao algoritmo de planejamento escolher a melhor sequência de fases, e os deslocamentos entre elas. Nesse tipo de abordagem, portanto, emprega-se o planejamento de fases, que pode incluir ou não o planejamento do movimento e trajetórias do veículo, conforme já comentado na seção 2.2. Por exemplo, através de algoritmos de busca da IA é possível montar uma árvore de busca onde os nós correspondem às fases da missão e através de um algoritmo, como A* ou busca em largura ou profundidade, é possível encontrar o caminho de menor custo e, portanto, definir a ordem em que as fases serão executadas na missão.

O resultado final do planejamento consiste em um plano de missão, que contém uma sequência de fases, onde cada fase possui uma sequência de eventos controláveis e não-controláveis, com os respectivos parâmetros de missão, possível de ser realizado pela planta representada pelo CSML. Em condições normais de operação, ou seja, sem erros, falhas, variação de consumo, etc., o GM somente necessita seguir a sequência de referência de eventos na medida em que eles são escolhidos (controláveis) ou gerados pelo veículo (não-controláveis).

### 5.4.3. Replanejamento de Missão

A missão especificada em um plano de missão exige que o sistema siga exatamente a sequência de referência de eventos pré-estabelecida. Contudo, para permitir que o veículo se adapte a eventos não-deterministas, é incluída no GM a opção de replanejamento da missão. O replanejamento pode ser realizado de modo similar ao planejamento, porém levando em consideração parâmetros como o consumo estimado para cada etapa da missão, total de bateria disponível, deslocamentos realizados, o tempo de operação do veículo, entre outros. Esse replanejamento é executado de modo *on-line*, enquanto o veículo realiza a tarefa em curso. Um novo plano de missão é então gerado e o anterior substituído sempre que o critério para execução do replanejamento é satisfeito. Assim, o GM atualiza o número de fases possíveis de serem completadas, excluindo algumas delas do plano de missão, dando prioridade para terminar a fase em curso. O replanejamento não modifica a sequência de referência dos eventos em uma fase, mas somente a ordem em que as fases são executadas. Diferente do planejamento, o replanejamento é executado *on-line*, enquanto o veículo está realizando a fase em curso. Ocasiões

em que tal procedimento podem ocorrer são: falhas em sensores de carga útil empregados em fases da missão; nível de potência disponível inferior ao nível necessário para completar a missão; ou impossibilidade de alcançar alguma região objetivo devido à presença de obstáculo.

### 5.4.4. Execução da Missão

O GM deverá escolher sempre qual é o próximo evento controlável, que deverá ser selecionado e traduzido em comando pelo nível SO ou então aguardar pelo próximo evento não-controlável que deverá ser gerado pelo AUV. Para isso propõe-se o uso de uma política de atribuição de prioridades dinâmicas onde o evento de maior prioridade é selecionado. Conforme já comentado no início da seção 5.4, para evitar bloqueios no sistema, o uso de prioridades permite que sempre exista um evento a ser selecionado, mesmo que tal escolha não conduza aos objetivos de missão.

Os modos de operação são empregados para tratar com aquelas intercorrências não previstas explicitamente no plano de missão, seja ele o plano original ou aquele obtido através do replanejamento. Tais modos podem tratar explicitamente de uma situação específica, com planos de missão com sequências de referência pré-estabelecidas de eventos e política específicas de atribuição de prioridades. Como exemplos, já comentados anteriormente, é possível citar: o cancelamento da missão com retorno ao ponto de recuperação; o cancelamento com retorno imediato à superfície; ações para correções de posicionamento, como subir à superfície para calibração por GPS; a realização de manobras específicas em alguma área, como pairar ao redor de um objeto; ou manter o AUV parado enquanto o mesmo realiza alguma operação com um braço mecânico ou enquanto está se comunicando com uma estação base ou outros veículos subaquáticos.

A partir do modo de operação, e respectivas sequências de referência de eventos e política de prioridades, o GM deve escolher sempre o evento mais prioritário e possível de ocorrer no estado atual dos modelos do CSML. A política de atribuição de prioridades deve funcionar em qualquer dos modos de operação. As prioridades, armazenadas em uma tabela, podem ser classificadas em vários níveis. Usualmente é possível atribuir o nível mais elevado de prioridade àqueles eventos controláveis considerados desejados, na maioria das vezes um único evento, e que correspondem ao próximo evento controlável da missão. Um nível de prioridade abaixo deste pode ser

atribuído a todos os eventos não-controláveis. Sempre que o próximo evento da missão corresponder a um evento não-controlável, não haverá nenhum outro evento controlável com prioridade mais alta, logo, o GM aguarda que o mesmo seja gerado pelo AUV para prosseguir com a execução da missão. Assim, os eventos não-controláveis recebem uma prioridade fixa, garantindo que o sistema não entre em bloqueio devido à atribuição de prioridades. Isso ocorre pois o GM não aguarda por um único evento ser gerado pelo AUV (não-controlável) ou selecionado (controlável). Finalmente estão os eventos controláveis que não se deseja que sejam selecionados e, portanto, recebem prioridades inferiores aos eventos não-controláveis de modo que nunca serão ativados. Eventos controláveis e não-controláveis não permitidos e/ou não habilitados pelos modelos do CSML podem receber as prioridades mais baixas do sistema. Com essa política, os eventos não-controláveis possuem prioridade fixa e os eventos controláveis possuem prioridade dinâmica. Ao final, um mecanismo de seleção de eventos opta por "gerar" um evento controlável, quando há eventos controláveis com prioridades mais alta, ou por não fazer nada, o que significa que irá aguardar que um evento não-controlável ocorra. Quando há mais de um evento com prioridade elevada é necessário implementar algum mecanismo de seleção, como por exemplo, sorteio.

No modo de operação normal, onde a sequência de referência dos eventos é obtida a partir do plano de missão e comprovada ser realizável ou permitida pelos modelos formais do CSML, sempre será possível realizar a missão, desde que os eventos não-controláveis assim o permitirem, pois o próximo evento da missão é o mais prioritário. Contudo, nos modos alternativos de operação, onde a ação do AUV é dado por uma sequência de referência fixa ou pré-programada de ações e que pode ser executada em qualquer etapa durante a realização da missão, um problema em aberto consiste em garantir formalmente que a tabela de prioridades irá levar ao cumprimento da missão, sempre que os eventos não-controláveis assim o permitirem.

## 5.5. MÉTODO DE DESENVOLVIMENTO DO SCM

O SCM constitui-se de um dos componentes existentes no sistema de um AUV e, portanto, o desenvolvimento do SCM ocorre dentro do escopo do desenvolvimento do sistema embarcado de todo o veículo. Metodologias de projeto e implementação de sistemas complexos, com componentes distribuídos, limitações de natureza

tempo-real e dependência do ambiente, da plataforma de execução e dos processos computacionais, como aqueles encontrados em sistemas embarcados, vêm sendo aplicadas de modo a favorecer o desenvolvimento desse tipo de aplicações (JENSEN, CHANG e LEE, 2011). Portanto, o método aqui apresentado para o desenvolvimento do SCM deve ser aplicado em conjunto com o desenvolvimento do sistema embarcado do AUV.

Várias são as metodologias de desenvolvimento de sistemas embarcados (WEHRMEISTER, 2009; BECKER *et al.*, 2010; LEE e SESHIA, 2011; JENSEN, CHANG e LEE, 2011) e que são divididas em várias fases concorrentes ou dependentes, por exemplo, levantamento de requisitos, modelagem, análise, testes e implementação. Essencialmente, a partir de uma descrição adequada do problema, são definidas os requisitos de funcionalidade e restrições em um nível alto de abstração, posteriormente traduzidos em uma arquitetura para o sistema, representando a estrutura conceitual e o funcionamento desejado, levando em consideração tanto os aspectos de software (processamento, estrutura de dados e funcionalidades) quanto de hardware (plataforma alvo) (WEHRMEISTER, 2009). Testes são realizadas em várias etapas do desenvolvimento do sistema, de acordo com a metodologia aplicada, no sentido de comprovar se requisitos são satisfeitos e se os modelos derivados atendem às necessidades do problema. A arquitetura obtida é então mapeada para uma implementação em uma plataforma alvo específica.

Nesse contexto, o método para o projeto e implementação do SCM baseado no CSML apresentado neste trabalho consiste na aplicação das etapas apresentadas na figura 5.5. Essencialmente o método compõe-se das seguintes etapas: descrição do problema e análise de requisitos, delimitando o tipo de missões e decorrentes funcionalidades que o AUV necessita realizar para atender tais missões; modelagem formal dos diversos componentes e subsistemas sob o ponto de vista de missão; definição de especificações de segurança e operação do veículo e síntese dos supervisores para garantia dessas especificações; especificação das responsabilidades do gerenciador de missão, incluindo a codificação, o planejamento e replanejamento de missões, e políticas para escolha de eventos; a escolha da plataforma alvo, do ambiente de desenvolvimento; a implementação da estrutura de CSML; a implementação do gerenciador de missão; teste do SCM através de simulações; e a implementação e testes em um veículo real. Observa-se que tais atividades devem ser realizadas concomitante e em conformidade com a metodologia empregada para o desenvolvimento do

sistema embarcado como um todo. Contudo, tal integração foge ao objetivo desse trabalho.

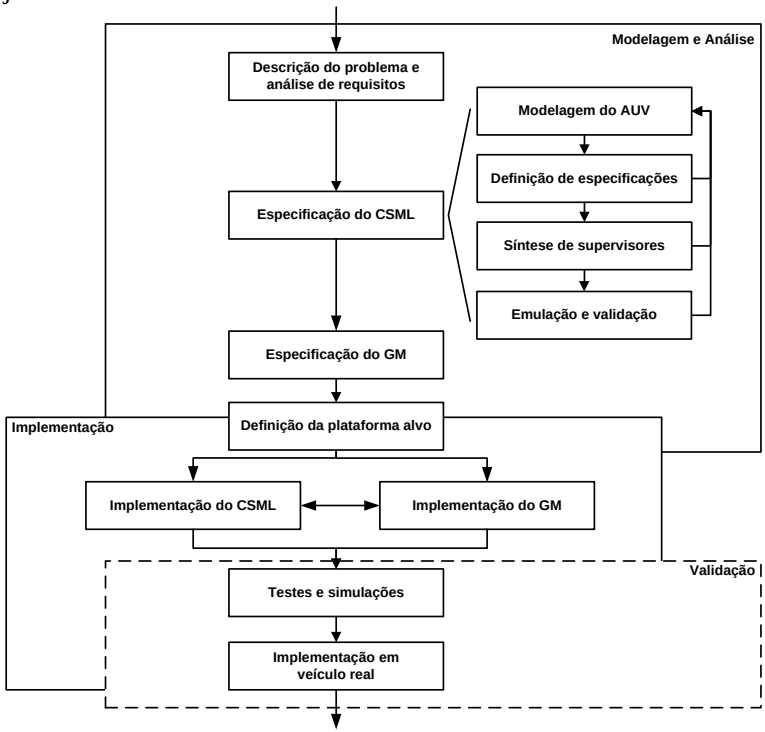


**Figura 5.5:** Método de desenvolvimento do SCM

Assim, o método aplicado para o desenvolvimento e testes do SCM para AUVs atuando em ambiente não-estruturados proposto neste trabalho pode ser didaticamente dividido nas seguintes etapas, a saber:

1. Descrição do problema e análise de requisitos
2. Especificação do Controle Supervisório Modular Local (CSML)
    a. Modelagem do AUV
    b. Definição de especificações
    c. Síntese de supervisores
    d. Emulação e validação
3. Especificação do Gerenciador de Missão (GM)
4. Definição da plataforma alvo
5. Implementação do CSML
6. Implementação do GM

7. Testes do SCM através de simulações
8. Implementação e testes em um veículo real

**Etapa 1: Descrição do Problema e Análise de Requisitos**

Nessa primeira etapa são estabelecidos os requisitos para funcionamento e operação segura do AUV em ambientes não-estruturados com características similares aqueles encontrados em lagos de barragens, como tipo de manobras e comportamento reativos para navegação do veículo, sensores típicos usados nesses ambientes, presença de obstáculos. Uma vez definido tais requisitos, é possível delimitar o tipo de missões bem como os principais componentes de software e hardware que o mesmo deverá dispor ou implementar para viabilizar a realização das missões. Assim são estabelecidos os subsistemas essenciais que deverão existir no sistema embarcado do AUV, contudo sem se preocupar como os mesmos serão implementados.

**Etapa 2: Especificação do Controle Supervisório Modular Local (CSML)**

Esta etapa constitui-se da aplicação da abordagem do CSML ao problema de controle de missão de AUVs (QUEIROZ e CURY, 2002). A modelagem dos componentes e subsistemas do AUV são usados para a descrição da evolução da missão, e especificações de operação e segurança são empregadas para restringir o comportamento dos modelos àqueles considerados seguros. A resolução de problemas empregando a TCS, e também o CSML, envolve a aplicação de basicamente quatro fases: modelagem da planta; modelagem de especificações; síntese de supervisores; e emulação e validação dos modelos e supervisores. Salienta-se que tal processo não implica diretamente na arquitetura de implementação, pois os modelos baseados em autômatos representam um nível de abstração com respeito à implementação das funcionalidades do veículo. Contudo, tais estruturas necessitam ser posteriormente implementadas no sistema embarcado. Além disso, as diversas fases de especificação do CSML podem ser aplicadas sucessivamente até encontrar um conjunto de comportamentos satisfatórios com os requisitos levantados na etapa 1.

**Subetapa 2.1: Modelagem do AUV**

A partir dos requisitos e dos componentes e subsistemas essenciais, são derivados os modelos baseados em autômato que irão descrever o comportamento em malha-aberta do sistema, denominado de planta. Cada componente, comportamento ou operação cujo

significado semântico possui relação com a tomada de decisão no nível de representação e execução de missões é modelado por uma máquina de estados finitos ou autômato.

**Subetapa 2.2: Definição de Especificações**

Quando o comportamento da planta não atende aos requisitos de operação do sistema é necessário impor restrições ao seu funcionamento, através da modelagem de especificações. Descritas também por autômatos, as especificações representam condições que devem ser impedidas, como por exemplo aquelas que irão prejudicar o desempenho do sistema ou até mesmo comprometer a sua integridade física. Assim, nessa etapa são derivadas todas as especificações que irão restringir o funcionamento em malha-aberta do modelo a um subconjunto de comportamentos compatível com a realização de missões.

**Subetapa 2.3: Síntese de Supervisores**

Os supervisores são estruturas de controle que garantem que as especificações desejadas para a planta sejam atendidas durante toda a realização da missão e para qualquer tipo de missão. O método de síntese de supervisores da TCS e do CSML garante que a ação dos mesmos sobre o sistema é ótima (RAMADGE e WONHAM, 1989) no sentido que somente os caminhos considerados indesejáveis sejam impedidos, possuindo, portanto, uma ação minimamente restritiva sobre os modelos da planta.

**Subetapa 2.4: Emulação e Validação**

Ao final do processo de modelagem da planta e especificações e da síntese dos supervisores obtém-se uma estrutura de controle supervisório, similar à apresentada na figura 4.3, que pode ser simulada a modo de comprovar o comportamento desejado, mesmo sem a definição de uma arquitetura de implementação de tais estruturas. Tal procedimento permite a correção e ajuste de aspectos de modelagem e comportamentos não considerados nas fases anteriores. Assim, a etapa de especificação do CSML pode ser sucessivamente realizada até obter-se o comportamento desejado para o sistema de acordo com os requisitos e necessidades levantadas na etapa 1.

**Etapa 3: Especificação do Gerenciador de Missão (GM)**

Uma vez estabelecido o comportamento lógico do sistema com respeito à realização de missões, é necessário definir a estrutura complementar ao CSML, responsável pela tradução, planejamento, replanejamento e execução de missões, denominada de Gerenciador de Missão (GM). O GM é o responsável em traduzir o arquivo de missão

em um plano de missão e em escolher a melhor sequência de eventos habilitadas pelas estruturas do CSML, possibilitando que o AUV possa seguir vários caminhos para realizar uma missão, não delimitando a missão a uma única sequência de eventos.

**Etapa 4: Definição da Plataforma Alvo**

Visando a implementação do SCM, composto pelo CSML e pelo GM, nessa etapa propõem-se a definição da plataforma que irá implementar os diversos algoritmos e processos que irão compor o SCM. Por plataforma entende-se tanto as ferramentas e ambiente de desenvolvimento como de execução do SCM e que devem ser considerados concomitante ao desenvolvimento do sistema embarcado do veículo. Aspectos como tipo arquitetura de hardware (microprocessadores; comunicação entre componentes, sensores e atuadores; requisitos temporais de execução; sistema operacional, etc.), *framework* de desenvolvimento e programação do sistema, ferramentas para simulação e prototipagem, etc. devem ser analisados e especificados no sentido de permitir a implementação do sistema.

**Etapa 5: Implementação do SCM**

A partir da definição da plataforma alvo, arquitetura de hardware / software, ferramentas de desenvolvimento, ambiente de execução, etc., os modelos baseados em autômatos do CSML necessitam ser traduzidos e adaptados para a arquitetura de execução. Nesse sentido, a arquitetura de implementação do CSML, dividida nos três níveis de Sequências Operacionais (SO), Sistema-Produto (SP) e Supervisores Modulares (SM), apresentados na figura 5.1, permite que os modelos abstratos usados na etapa de modelagem sejam diretamente empregados na arquitetura embarcada do sistema. Além disso, como o processo de modelagem das plantas e especificações e a síntese de supervisores da TCS é comumente realizado usando ferramentas computacionais, através de programas adequados é possível traduzir automaticamente tais estruturas formais em código fonte a ser posteriormente integrado ao SCM.

**Etapa 6: Implementação do GM**

Os algoritmos de planejamento e replanejamento, a forma de representação de missões, os mecanismos de tradução de arquivos de missão em plano de missões, as funcionalidades de atribuição de prioridades para escolha de eventos necessitam então ser implementadas pelo GM. O desenvolvimento do SCM e do GM pode ser realizado

concomitante, contudo, deve-se observar o compartilhamento de algumas estrutura de dados, como os eventos e os parâmetros de configuração de algumas tarefas entre o GM e o CSML.

**Etapa 7: Testes do SCM através de Simulações**

Visando o teste do SCM proposto são simuladas várias operações em diversos cenários de missão, baseados em ambientes não-estruturados como lagos de barragens de hidrelétricas, de modo a certificar as várias funcionalidades desenvolvidas, entre elas, o planejamento, o replanejamento, a reatividade a obstáculos e a garantia das especificações de segurança e operação do veículo. Empregando técnicas de simulação baseada em *Hardware-In-the-Loop* (HIL), por exemplo, é possível validar o funcionamento do SCM proposto através da sua integração com um ambiente que simula os demais componentes do veículo.

**Etapa 8: Implementação e Testes em um Veículo Real**

Finalmente, a última etapa do método proposto envolve a integração do SCM no sistema embarcado de um AUV. Tal SCM também pode ser adaptado a um AUV já existente, mediante a integração da comunicação dos módulos já implementados no AUV e o SCM.

## 5.6. RESUMO

Nesse capítulo foi proposta a arquitetura geral de Sistemas de Controle de Missão (SCM). Seguindo uma arquitetura híbrida deliberativo-reativa, o SCM compõe-se de dois componentes principais, o CSML e o GM, que implementam as várias atividades necessárias para a realização da missão, entre elas: tradução do arquivo de missão em um plano de missão; a escolha de uma sequência de fases de missão que otimize a quantidade de bateria disponível; a ação dos supervisores sobre o modelo abstrato do AUV para garantia de propriedades e especificações; a atualização dos modelos abstratos; o replanejamento das missões ante a ocorrência de eventos não-deterministas ou devido a diferenças entre o consumo estimado de bateria e a carga disponível real; e a troca de informações entre o SCM e os diversos processos responsáveis pela aquisição de dados sensoriais, detecção de obstáculos, entre outros.

Os aspectos de uma implementação para o SCM proposto são mostrados no próximo capítulo, onde é empregado o ROS, uma plataforma de desenvolvimento de aplicações robóticas. Para a validação do referido SCM desenvolve-se um ambiente de simulação, que também será apresentado na continuação.

# 6. IMPLEMENTAÇÃO DO SCM E AMBIENTE DE SIMULAÇÃO

Este capítulo apresenta uma breve visão da implementação do SCM no *middleware* ROS e o desenvolvimento do ambiente de simulação criado para validar o SCM proposto. A seção 6.1 discute os principais aspectos relacionados à implementação do SCM no ROS, incluindo a descrição geral do SCM e dos processos adicionais responsáveis pela troca de informação entre o SCM e os demais componentes do AUV. A implementação do CSML é comentada na seção 6.2. Na seção 6.3 é mostrada a implementação do GM no ROS, com as soluções empregadas para representação de missão, algoritmo de planejamento e o modo de execução de missões. De modo a possibilitar a validação do SCM proposto, desenvolve-se um ambiente que simula a dinâmica contínua e dirigida a eventos do AUV, e que é apresentado na seção 6.4. O software Matlab/Simulink é empregado para simular a dinâmica contínua descrita pelos modelos cinemático e dinâmico do AUV, bem como o controle de movimento em malha-fechada. A dinâmica discreta, como ocorrência de eventos de falhas, são gerados por uma interface homem-máquina (IHM), desenvolvida em linguagem Java. Ao final, apresenta-se um resumo do capítulo.

## 6.1. IMPLEMENTAÇÃO DO SCM NO ROS

O ROS (*robot operating system*) é um *middleware* e conjunto de ferramentas, bibliotecas e componentes que permite o desenvolvimento e execução de aplicações robóticas (ROS, 2014). ROS vem sendo empregado pela comunidade acadêmica para desenvolvimento e prototipagem de veículos robóticos autônomos (MARCO, WEST e COLLINS, 2011). A utilização do ROS possibilita ao desenvolvedor o fácil e rápido uso de mecanismos de comunicação, bibliotecas de funções matemáticas, processamento de dados sensoriais e imagens, construção de mapas de ambiente, integração com hardware e *drivers* de diferentes fabricantes, ferramentas de simulação e visualização, permitindo direcionar esforços para resolução de problemas específicos das aplicações robóticas.

O SCM proposto constitui-se em um dos programas a ser embarcado diretamente no AUV. Para tanto, é necessário que o AUV possua um processador capaz de executar o ROS. O SCM proposto não depende, contudo, do ROS ou dos protocolos de comunicação empregados na implementação apresentada neste trabalho. Além do sistema operacional Linux Ubuntu, incluindo suas versões tempo-real, sistema em que o ROS é originalmente desenvolvido, há versões experimentais do ROS para diversos sistemas operacionais de dispositivos embarcados, como o Angström (ANGSTRÖM, 2014) e UbuntuARM (UBUNTUARM, 2014). Desse modo é possível afirmar que há várias opções para emprego do ROS em plataformas robóticas, justificando a escolha do *middleware* para desenvolvimento do SCM, mas também como ambiente de execução do referido sistema. Além do ROS existem outros middlewares e ferramentas para desenvolvimentos de aplicações robóticas, como por exemplo OpenRave (DIANKOV e KUFFNER, 2008), Orocos (OROCOS, 2014) ou Microsoft Robotics (MICROSOFT ROBOTICS, 2013), porém a avaliação e comparação entre tais ferramentas foge ao escopo desse trabalho.

O SCM é implementado através de 10 processos independentes desenvolvidos em linguagem C/C++ e gerenciados pelo ROS. Os modelos abstratos baseados em autômatos do CSML são automaticamente gerados a partir da ferramenta Supremica (AKESSON *et al.*, 2006) usada para modelagem e síntese dos supervisores. Os demais processos são codificados manualmente empregando as ferramentas e bibliotecas disponibilizadas pelo ROS. Para a codificação dos divervos processos e bibliotecas de funções foi empregado o paradigma de programação orientada a objetos (POO).

O SCM proposto também pode ser adaptado a um AUV já existente, mediante a integração da comunicação dos módulos já implementados no AUV e o SCM. O SCM pode ser diretamente implementado sem a necessidade de adaptação, desde que o sistema embarcado possa executar o ROS. Caso não exista a possibilidade de execução no ROS, então a parte referente à comunicação – mecanismos de troca de mensagens – estruturas de dados, funcionalidades para cálculo de trajetórias e transformações de sistemas de coordenadas, além do mecanismo de detecção de obstáculo necessitariam ser adaptados, contudo, a parte de planejamento e replanejamento, a codificação das missões e o SCM em si, com o CSML e o GM, incluindo a geração automática de código, podem ser mantidos com praticamente nenhuma alteração.

O SCM desenvolvido a partir dos modelos baseados em autômatos não é validado através de verificação formal. A arquitetura de implementação do CSML permite que tais modelos sejam diretamente implementados no sistema embarcado do veículo e, portanto, satisfazem, pelo menos nesse nível de abstração, o atendimento a propriedades definidas durante a fase de análise, como ausência de bloqueios e o próprio cumprimento das especificações. Além disso, devido ao GM, a modelagem da missão e a implementação do nível SO da arquitetura do CSML serem todos implementados manualmente não há garantias de funcionamento do sistema devido a erros de codificação ou mesmo erros lógicos de funcionamento dos níveis inferiores da arquitetura. Por esse motivo, desenvolve-se o ambiente de simulação cujo objetivo é validar toda a arquitetura do SCM, incluindo o CSML que já possui garantia de propriedades, de modo a comprovar que o SCM proposto funciona adequadamente para os cenários e requisitos de missão estabelecidos nas fases iniciais do trabalho.

Os eventos gerados pelo GM e permitidos pelos modelos dos níveis SM e SP correspondem aos eventos controláveis e são traduzidos pelo nível SO em entradas para o AUV. Tais entradas são enviadas aos simuladores da parte dinâmica contínua e discreta através dos diversos processos em execução no ROS. Os sinais gerados pela dinâmica contínua e discreta são enviados para o GM e para o CSML, de modo que os modelos do sistema e a missão sejam atualizados. Para isso, as interfaces recebem os sinais do Matlab/Simulink e da IHM e as enviam ao nível SO do CSML que, por sua vez, os traduzem em respostas que posteriormente são mapeadas em eventos não-controláveis, permitindo atualizar a evolução dos modelos baseados em autômatos.

A figura 6.1 apresenta a organização geral doSCM proposto com todos os componentes e processos, e que são descritos a seguir. Na figura, cada processo é representado por um retângulo arredondado e as setas indicam o sentido da troca de mensagens entre os mesmos. A tabela 6.1 apresenta um resumo com os 10 processos ROS que fazem parte do SCM proposto, e que são mostrados com mais detalhes no apêndice B. Os nove tipos de mensagens trocadas entre os processos em execução no ROS, mostrados na figura 6.6 próximos às flechas que conectam os processos, são apresentados no apêndice C e os dois tipos de comunicação (cliente / servidor e *publisher* / *subscriber*) são mostrados no apêndice E.

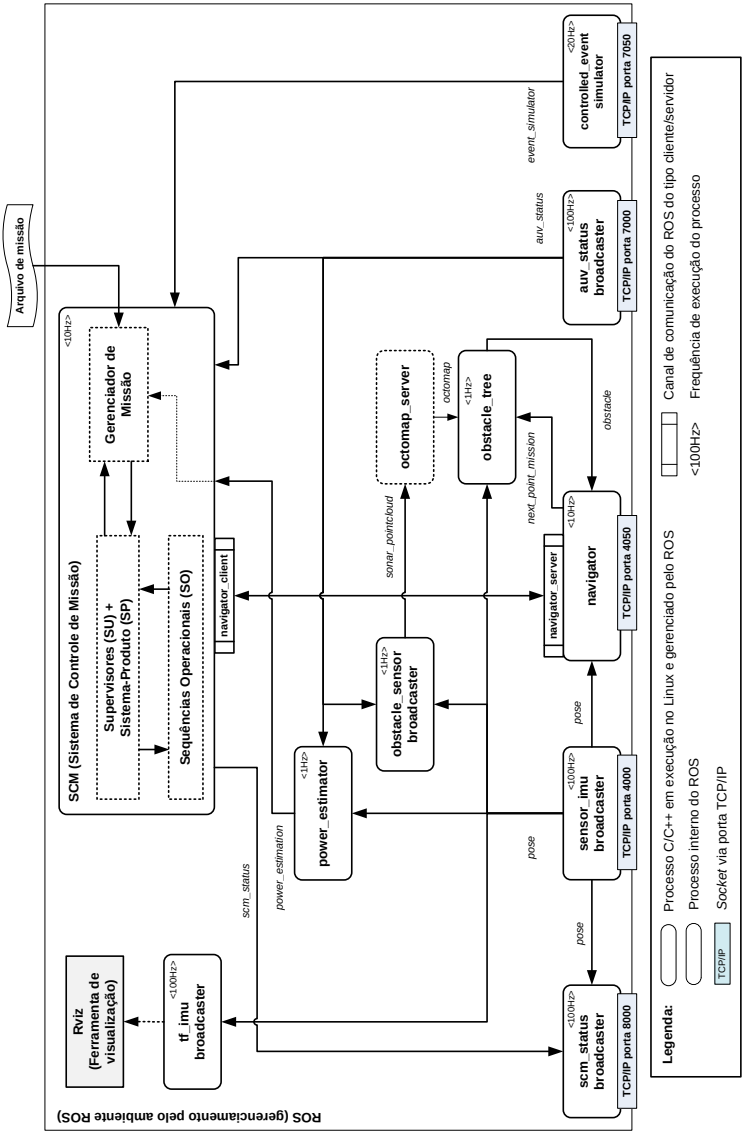


**Figura 6.1:** Organização geral do SCM

**Tabela 6.1:** Processos ROS do SCM

| Processo | Nome | Freq. (Hz)[8] | Descrição |
|---|---|---|---|
| Sistema de Controle de Missão | *SCM* | 10 | Processo principal do SCM. Implementa o CSML e o GM. |
| Aquisição de Posição e Orientação | *sensor_imu_ broadcaster* | 100 | Recebe posição e orientação do movimento do AUV e publica em um canal. |
| Navegação do Veículo | *navigator* | 10 | Gera as referências de posição e orientação para os controladores de malha-fechada. Também implementa o desvio de obstáculos. |
| Estado da Missão e Sistema | *scm_status_ broadcaster* | 100 | Envia para a IHM o estado do veículo (sensores, posição, orientação) e os estados lógicos do CSML. |
| Estado do AUV | *auv_status_ broadcaster* | 100 | Recebe comandos e eventos não-controláveis gerados pela IHM. |
| Eventos Controláveis | *controlled_event _simulator* | 20 | Recebe eventos controláveis gerados pela IHM para operação em modo manual, sem planejamento de missão. |
| Consumo de Bateria | *power_estimator* | 1 | Simula o consumo de bateria a partir dos deslocamentos e estado dos sensores. |
| Detecção de Obstáculos | *obstacle_sensor_ broadcaster* | 1 | Simula a presença de obstáculos e gera mapa de ocupação de obstáculos. |
| Detecção de Colisões | *obstacle_tree* | 1 | Detecta se há obstáculos entre a posição atual do veículo e a posição destino. |

[8] Os valores de frequência foram escolhidos empiricamente, durante a implementação e teste do SCM em conjunto com o ambiente de simulação, contudo uma análise mais criteriosa a respeito da escolha de tais valores ainda é necessária.

| Processo | Nome | Freq. (Hz)[8] | Descrição |
|---|---|---|---|
| Transformação de Coordenadas | *tf_imu_ broadcaster* | 100 | Transforma a posição e orientação do AUV em dados para visualização em ferramenta gráfica. |

## 6.2. IMPLEMENTAÇÃO DO CSML

Uma das dificuldades no desenvolvimento de sistemas supervisórios consiste na etapa de implementação das estruturas de controle (supervisores) na plataforma do sistema (computador, microprocessador, microcontrolador ou controlador lógico programável), pois nem sempre existe uma correspondência direta entre os eventos e estados com os sinais reais da planta (LIU e DARABI, 2002). A arquitetura de implementação do CSML, proposta em (QUEIROZ e CURY, 2002) diminui consideravelmente tal dificuldade, pois permite a utilização direta dos modelos empregados durante a fase de análise do sistema. Particularmente nesse trabalho, o código que implementa tais estruturas foi gerada automaticamente por uma ferramenta desenvolvida com esse propósito, mostrada no apêndice C. A ferramenta Supremica, utilizada para a modelagem e síntese de supervisores codifica os modelos baseados em autômatos em uma estrutura em formato XML. Tal estrutura é então traduzida automaticamente em código C/C++ de modo a ser incluída na biblioteca de classes e funções do SCM, gerando o nível Sistema-Produto (SP) com as plantas modulares e o nível Supervisores Modulares (SM). Assim, o nível SP e SM é gerado automaticamente, contudo o nível SO é codificado manualmente. O funcionamento do controle supervisório corresponde àquele apresentado na seção 5.1. A arquitetura de implementação do CSML é então incluída no processo ROS que executa o SCM e compilada em conjunto com o restante dos componentes do sistema, incluindo o GM.

Os eventos controláveis e não-controláveis correspondem a ocorrências (sinais, comandos) na planta real (subsistemas do AUV) e que são capturados e representados nos modelos formais do CSML. Portanto, durante a implementação do CSML também é necessário realizar a correspondência entre os eventos e os sinais reais do AUV. Esta correspondência também é apresentada no apêndice C.

## 6.3. IMPLEMENTAÇÃO DO GM

### 6.3.1. Algoritmo Básico do GM

Conforme já apresentado na seção 5.4, o GM possui como responsabilidades principais: a tradução da missão em um plano de missão, realizado pela atividade de planejamento; a realização do replanejamento quando a carga de bateria não for suficiente para completar todas as fases da missão; e a escolha do próximo evento que irá guiar o veículo em direção aos objetivos da missão. Tal escolha de eventos está baseada na sequência de referência, nos modos de operação e na política de atribuição de prioridades, conforme discutidos na seção 5.4.4, onde em condições normais de operação o plano de missão original é seguido, mas em estados de falhas impõem-se a realização de um plano pré-estabelecido de recuperação do veículo (modos de operação) ou então um novo plano de missão é gerado (replanejamento). Sempre que o GM decidir pela realização de uma ação, o respectivo evento controlável e seu parâmetro de configuração é então selecionado, de acordo com a política de prioridades implementada. Contudo, caso seja necessário que o veículo termine uma ação ou operação em curso, indicado por um evento não-controlável, o GM opta então por não escolher nenhum evento controlável. De maneira resumida, a tabela 6.2 apresenta o algoritmo básico de execução do SCM, e constitui-se de uma implementação do fluxograma genérico proposto para o SCM mostrado na seção 5.4. O algoritmo é explicado na continuação.

Inicialmente o SCM inicializa variáveis internas, filas e processos de comunicação do ROS (linhas 2 e 3). Comprova-se se o plano de missão, gerado a partir do arquivo de missão, contém uma sequência de referência de eventos realizável pela planta (linhas 4 a 6). Enquanto a missão não chegar ao final, o programa permanece em um laço de repetição que dura até o final da missão (linha 7). De modo a garantir a integridade dos modelos abstratos do CSML e manter os mesmos sincronizados com a planta, os eventos não-controláveis são gerados a partir de uma fila de sinais recebidos pelo nível SO (linha 9). Essa fila mantém a ordem dos eventos conforme são recebidos pelo nível SO. Uma vez recebido o evento não-controlável, os modelos das plantas do nível SP e os supervisores modulares (SM) são atualizados (linhas 10 e 11). Na sequência o plano de missão (linha 12) e o modo de operação (linha 13) são atualizados, e também é realizada a comprovação da necessidade de replanejamento se, por exemplo, um evento não-contro-

**Tabela 6.2:** Algoritmo básico do SCM

```
1   Main
2       Init_variables()
3       Init_ROS_components()
4       Read_mission_file()
5       Generate_mission_plan()
6       Check_mission_plan()
7       while (mission is not at end && ROS_ok)
8           while (SO_level_hasResponses)
9               event_uncontrollable = SO_level.getResponse()
10              SP_level.DoEvent(event_uncontrollable)
11              SM_level.DoEvent(event_uncontrollable)
12              updateMissionPlan(event_uncontrollable)
13              updateMode(SP_level, SM_level)
14              Check_Do_Replanning(event_uncontrollable,
    battery_level)
15          event_controllable = planner_getEvent()
16          if event_controllable is not null
17              SP_level.DoEvent(event_controllable)
18              SM_level.DoEvent(event_controllable)
19              updateMissionPlan(event_controllable)
20              updateMode(SP_level, SM_level)
21              cmd=SO_level.Generate_command(event_controllable)
22          Send_Data2IHM()
```

lável de erro não previsto na sequência de referência de eventos do plano de missão ocorrer (linha 14). Quando não mais houver eventos não-controláveis, o GM comprova a necessidade de execução de algum evento controlável, analisando o plano de missão (linha 15). Caso exista tal evento controlável (linha 16), o nível SP e SM são atualizados (linhas 17 e 18), o plano de missão e o modo de operação também são atualizados (linhas 19 e 20) e o nível SO traduz e envia o comando equivalente, com respectivos parâmetros, para os níveis inferiores da arquitetura (linha 21). As tabelas de prioridades dos eventos são atualizadas na seguintes situações: cada vez que um novo evento não-controlável é gerado pelo sistema; quando um evento controlável é selecionado pelo GM; ou quando ocorre o replanejamento da missão. Finalmente, são enviados para a IHM (linha 22) as informações referentes ao estado da missão, estado dos modelos, eventos habilitados / desabilitados, posição e orientação atuais do veículo.

Para o SCM implementado e simulações realizadas para este trabalho (capítulo 7), em média, um ciclo do algoritmo do SCM (linhas

7 a 22) dura cerca de 100 a 200 ms, levando em consideração que o ROS suspende o processo sempre que o tempo de execução do ciclo for inferior ao tempo estipulado para execução do processo, programado mediante uma diretiva da biblioteca de funções do ROS (10 Hz ou 100 ms). O valor de 100 a 200 ms é obtido a partir dos registros (*logs*) do sistema, contudo, como o ROS e o Ubuntu usados para implementar o SCM não suportam processamento tempo-real, tal valor é somente uma média, não havendo garantias destes tempos de execução. Visando aumentar o tempo de desenvolvimento do SCM sem ocupar-se diretamente com aspectos de comunicação, troca de mensagens, detecção de obstáculos, visualização de dados, cálculos de transformações de coordenadas, optou-se pelo ROS. Contudo, como o número de operações do SCM é fixo, com exceção do planejamento e replanejamento, é razoável assumir que o SCM pode ser implementado em um sistema tempo-real.

A posição e orientação atuais do veículo, enviados pelo Matlab/Simulink, os dados enviados pela IHM e pelos demais processos em C/C++ em execução no ROS são recebidos mediante *callback* síncrono, ou seja, sempre que um dado chega, a execução do laço principal do programa é interrompido e a função corresponde ao tratamento do dado é executada. Semáforos de exclusão mútua (*mutexes* na biblioteca C/C++) são usados para evitar que dois processos (*threads*) manipulem a mesma variável simultaneamente, ou ainda, tentem acessar uma área de memória sendo modificada por outra parte do programa. Ainda que o ROS e bibliotecas da linguagem C/C++ forneçam tais mecanismos é necessário a programação dos mesmos no desenvolvimento do sistema. Contudo, a arquitetura para SCM proposta no capítulo 5 independe do sistema operacional e do tipo de comunicação empregados.

### 6.3.2. Representação de Missões

A codificação do arquivo de missão será ilustrada através de um exemplo, e que posteriormente também será usado para explicar os algoritmos de planejamento e replanejamento. Considera-se, então, a missão mostrada na figura 6.2, onde o veículo deve realizar atividades em quatro regiões distintas, a saber:

- **Fase 1:** realização de batimetria com execução de trajetória composta por quatro segmentos diferentes e aquisição via CTD nos dois segmentos finais;

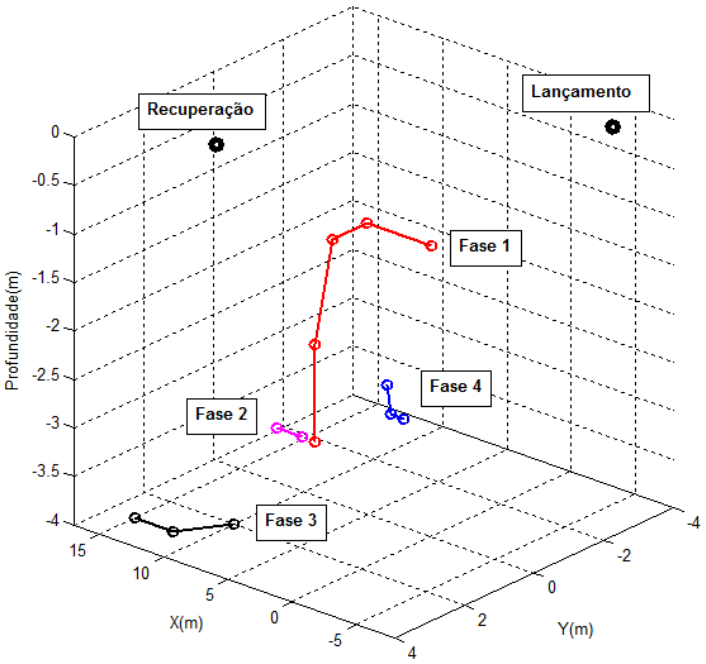


**Figura 6.2:** Fases de missão e respectivas trajetórias

- **Fase 2:** aquisição de dados via CTD e câmera com execução de trajetória com um único segmento;
- **Fase 3:** realização de batimetria, aquisição de dados via CTD e utilização da câmera com realização de trajetória composta por dois segmentos;
- **Fase 4:** realização de batimetria com realização de trajetória composta por dois segmentos.

A missão é codificada em um arquivo no formato TXT (figura 6.3) onde cada uma das fases é codificada pela *tag Phase*, que contém também o número da fase. Na sequência, são especificados a coordenada de início da fase (*tag Start*) e pelas tarefas que compõem a mesma. Uma tarefa contém a informação do ponto final desejado, especificado pelas três coordenadas de posição e três coordenadas de orientação, pelo estado ligado / desligado dos sensores de carga útil, pelo tipo de manobra e pela margem de erro. Essa codificação de tarefa está de acordo com definição de uma tarefa apresentada na seção 5.4.1. Além disso, são também especificados no arquivo as coordenadas do

ponto de partida (*tag launch*) e do ponto de recuperação (*tag recovery*) do veículo ao final da missão.

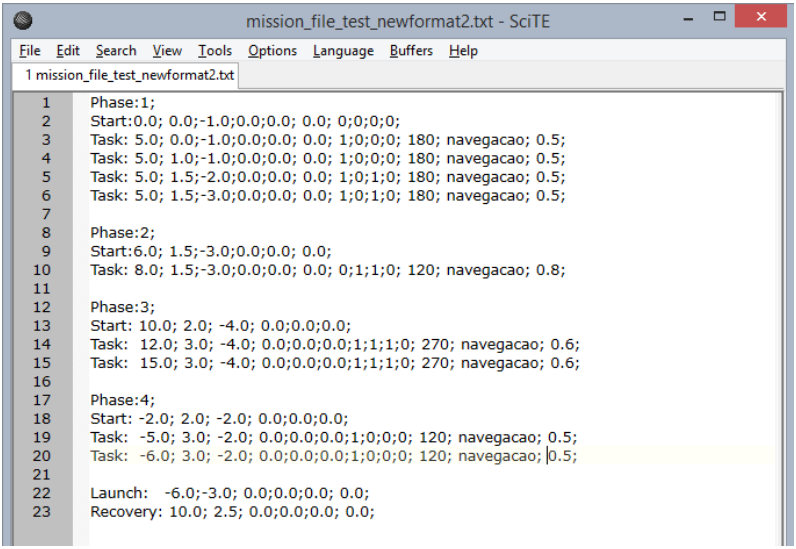


**Figura 6.3:** Exemplo de arquivo de missão em formato TXT

Para cada fases, considera-se o deslocamento entre o ponto inicial e final da mesma, com os respectivos sensores de carga útil ligados. Cada um dos deslocamentos da fase corresponde a uma tarefa, que envolve, além do deslocamento do veículo entre o ponto inicial e final da fase, a ativação e desativação de sensores de carga útil. Por exemplo, na fase 1 especifica-se a realização de batimetria durante toda a fase e aquisição de dados via câmera a partir do terceiro deslocamento. As fases de aproximação entre as fases de aquisição de dados não precisam ser codificadas no arquivo pois durante o planejamento e replanejamento tais aproximações são apropriadamente inseridas entre as fases de aquisição de dados.

### 6.3.3. Planejamento e Replanejamento de Missão

O plano de missão corresponde a estrutura de dados que contém a sequência de referência de eventos, e respectivos parâmetros de missão, e é gerado pela ação do planejamento ou replanejamento. Neste trabalho, o planejamento consiste em encontrar a melhor sequência de

fases que minimiza o consumo da bateria. Como uma fase necessita ser realizada em toda sua totalidade, com seus deslocamentos já estabelecidos e definidos no arquivo de missão, a ação do planejamento consiste, basicamente, em encontrar a melhor ordem das fases. Para isso é empregado uma árvore de busca e onde o menor caminho entre as diversas fases da missão é aquele que será utilizado como plano de missão. Empregando um algoritmo de planejamento baseado em busca em largura, é possível então escolher o caminho de menor custo. Diferente das abordagens apresentadas na seção 2.2.3, que utilizam o planejamento para encontrar um caminho entre o ponto atual e o ponto desejado, neste trabalho o planejamento consiste em encontrar a melhor sequência de fases que minimize o consumo de bateria. Optou-se pela busca em largura devido à facilidade na sua implementação, ainda que, para o tipo de planejamento empregado nesta tese, a busca em largura é menos eficiente que a em profundidade, pois necessita de mais memória para manter toda a árvore de busca. Contudo, devido a relativa baixa complexidade no processamento da busca, baseada na ordem das fases e não na sequência dos eventos, tal limitação não afetou significativamente a performance do planejamento, da ordem de 1 ou 2 segundos conforme as simulações realizadas (capítulo 7).

Como as fases possuem um custo fixo estimado igual ao seu deslocamento mais o consumo dos sensores ligados durante a sua realização, a minimização da melhor sequência consiste em encontrar os menores deslocamentos possíveis entre os pontos de início da fase atual e o ponto final da fase anterior. Considerando também que esses deslocamentos, chamados de manobras de aproximação, são realizados com os sensores de carga útil desligados, e que o veículo desloca-se a uma velocidade constante (sem malha de controle de velocidade), é razoável assumir que o custo da bateria para a manobra de aproximação é igual ou muito próximo ao deslocamento. Assim, é possível montar uma árvore de busca, onde os nós correspondem às fases e os vértices conectando os nós representam o custo entre o ponto final da fase anterior e o ponto inicial da fase atual. O processo de criação da árvore de busca no método de busca em largura consiste em primeiro expandir todos os nós situados em um mesmo nível da árvore, para então expandir todos os nós filhos.

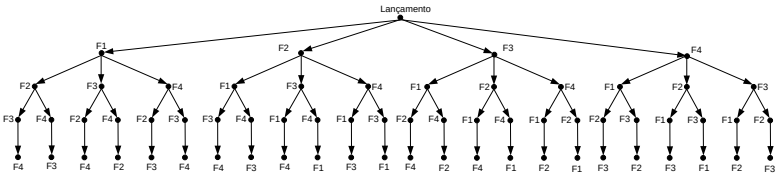


**Figura 6.4:** Árvore de busca para o cenário de missão considerado

Para o caso da missão da figura 6.2 a árvore de busca completa é apresentada na figura 6.4, onde $F_j$ indica a fase. O custo entre cada fase é calculada pela distância absoluta entre as coordenadas de início e fim e um termo de ponderação associada à carga útil ativa, ou seja:

$$c_{mn} = \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2 + (z_m - z_n)^2} + p(sensores) \quad (6.1)$$

onde $c_{mn}$ representa o custo obtido através da distância entre a coordenada do ponto final da fase *m* e coordenada de início da fase de índice *n*, *m* corresponde ao índice da fase anterior, *n* ao índice da fase atual, e $x_j$, $y_j$ e $z_j$ são as coordenadas no referencial inercial. Diferenças na orientação dos pontos inicial e final são desconsideradas. O termo *p(sensores)* representa o fator de ponderação que associa um custo maior ao deslocamento quando os sensores estão ativados. Assim, sensores com maior necessidade de potência, como câmera ou sonar possuem um custo maior que sensores que necessitem menos energia, como CTD. Ao realizar o planejamento no nível das fases, e não no alfabeto de eventos do CSML, o problema de explosão combinatória é reduzido. Além disso, a etapa de planejamento é realizada antes do início da missão com o veículo já posicionado na água.

Uma vez construída a árvore de busca, escolhe-se a sequência de fases com menor custo. A tabela 6.3 apresenta, como exemplo, oito dos 24 caminhos completos possíveis, com os custos individuais de cada nó e o custo total de cada caminho. O caminho completo com menor custo, ou seja, a sequência de fases cujo deslocamento entre as mesmas possui a menor distância é dada por $Fase_4 \rightarrow Fase_1 \rightarrow Fase_2 \rightarrow Fase_3$, com custo total de 23,13701.

Uma vez escolhida a melhor sequência de fases, o GM traduz o caminho selecionado em uma sequência de eventos controláveis e não-controláveis. Para cada tarefa, a sequência de eventos é obtida a partir do mapeamento direto entre a sintaxe que expressa a tarefa e os correspondentes eventos definidos pelos alfabetos dos autômatos modu-

**Tabela 6.3:** Custos individuais e totais para algumas sequências de fases

| Caminho | 1ª fase | Custo | 2º fase | Custo | 3ª fase | Custo | 4ª fase | Custo | Custo total |
|---|---|---|---|---|---|---|---|---|---|
| $F_4 \rightarrow F_2 \rightarrow F_1 \rightarrow F_3$ | 4 | 6,7 | 2 | 12,1 | 1 | 8,2 | 3 | 11,5 | 38,6 |
| $F_4 \rightarrow F_1 \rightarrow F_2 \rightarrow F_3$ | 4 | 6,7 | 1 | 6,71 | 2 | 1,0 | 3 | 8,7 | 23,1 |
| $F_3 \rightarrow F_4 \rightarrow F_2 \rightarrow F_1$ | 3 | 17,2 | 4 | 17,1 | 2 | 12,1 | 1 | 14,1 | 60,6 |
| $F_3 \rightarrow F_1 \rightarrow F_2 \rightarrow F_4$ | 3 | 17,2 | 1 | 15,4 | 2 | 1,0 | 4 | 26,1 | 59,8 |
| $F_2 \rightarrow F_3 \rightarrow F_1 \rightarrow F_4$ | 2 | 13,1 | 3 | 2,2 | 1 | 15,4 | 4 | 23,2 | 54,1 |
| $F_2 \rightarrow F_1 \rightarrow F_3 \rightarrow F_4$ | 2 | 13.1 | 1 | 8,2 | 3 | 5,1 | 4 | 33,2 | 59,8 |
| $F_1 \rightarrow F_3 \rightarrow F_2 \rightarrow F_4$ | 1 | 6,9 | 3 | 5,1 | 2 | 9,1 | 4 | 26,1 | 47,4 |
| $F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow F_4$ | 1 | 6,9 | 2 | 1,0 | 3 | 2,2 | 4 | 33,2 | 43,4 |

lares. Tal mapeamento é fixo e é determinado previamente à implementação do sistema. A tabela 6.4 apresenta o plano final completo de missão obtido para a sequência de fases $F_4 \rightarrow F_1 \rightarrow F_2 \rightarrow F_3$, onde $F_{ap}$ indica a fase de aproximação entre o ponto final da fase anterior e o ponto de início da fase seguinte, $\sigma_{i\text{-}k}$ indica a tarefa *k* da fase *i*, e $P_{k\text{-}q}$ indica o ponto final de número *q* da tarefa *k*. Para a primeira fase, a fase de aproximação, o respectivo consumo é calculado entre o ponto de lançamento do veículo e a coordenada inicial da primeira fase, no caso, a fase de número 4. Do mesmo modo, para a última fase (fase 3), o seu consumo é calculado como sendo o deslocamento entre o ponto final da fase e a coordenada de recuperação do veículo.

**Tabela 6.4:** Plano de missão para o melhor caminho selecionado

| Fase | Tarefa | Evento | Controlável | Ponto Inicial | Ponto Final | Carga Útil[9] | *Timeout* | Manobra | Margem de Erro |
|---|---|---|---|---|---|---|---|---|---|
| $F_{ap1}$ | $\sigma_{ap1}$: aproxi-mação 1 | *st_md* | sim | $P_{launch}$ | $P_{4\text{-}0}$ | 0,0,0,0 | 90 s | Desc.[10] | 0.5 |
| | | *in_h2o* | não | - | - | 0,0,0,0 | - | - | - |
| | | *end_md* | não | - | - | 0,0,0,0 | - | - | - |
| $F_4$ | $\sigma_{4\text{-}1}$: | *on_bat* | sim | - | - | 0,0,0,0 | - | - | - |

[9] Carga útil expressa o estado ligado (1) ou desligado (0) do sonar de batimetria, câmera, sensor de CTD e GPS, respectivamente.

[10] Manobra de descida.

| Fase | Tarefa | Evento | Controlável | Ponto Inicial | Ponto Final | Carga Útil[9] | *Timeout* | Manobra | Margem de Erro |
|---|---|---|---|---|---|---|---|---|---|
| | batimetria - trajeto 1 | *st_m* | sim | $P_{4-0}$ | $P_{4-1}$ | 1,0,0,0 | 120 s | Naveg.[11] | 0.5 |
| | | *end_m* | não | - | - | 1,0,0,0 | - | - | - |
| | $\sigma_{4-2}$: batimetria - trajeto 2 | *st_m* | sim | $P_{4-1}$ | $P_{4-2}$ | 1,0,0,0 | 120 s | Naveg. | 0.5 |
| | | *end_m* | não | - | - | 1,0,0,0 | - | - | - |
| | | *off_bat* | não | - | - | 0,0,0,0 | - | - | - |
| $F_{ap2}$ | $\sigma_{ap2}$: aproximação 2 | *st_m* | sim | $P_{4-2}$ | $P_{1-0}$ | 0,0,0,0 | 90 s | Naveg. | 0.5 |
| | | *end_m* | não | - | - | 0,0,0,0 | - | - | - |
| $F_1$ | $\sigma_{1-1}$: batimetria - trajeto 1 | *on_bat* | sim | - | - | 1,0,0,0 | - | - | - |
| | | *st_m* | sim | $P_{1-0}$ | $P_{1-1}$ | 1,0,0,0 | 180 s | Naveg. | 0.5 |
| | | *end_m* | não | - | - | 1,0,0,0 | - | - | - |
| | $\sigma_{1-2}$: batimetria - trajeto 2 | *st_m* | sim | $P_{1-1}$ | $P_{1-2}$ | 1,0,0,0 | 180 s | Naveg. | 0.5 |
| | | *end_m* | não | - | - | 1,0,0,0 | - | - | - |
| | $\sigma_{1-3}$: batimetria e CTD - trajeto 3 | *on_ctd* | sim | - | - | 1,0,1,0 | - | - | - |
| | | *st_m* | sim | $P_{1-2}$ | $P_{1-3}$ | 1,0,1,0 | - | - | - |
| | | *end_m* | não | - | - | 1,0,1,0 | - | - | - |
| | $\sigma_{1-4}$: batimetria e CTD - trajeto 4 | *st_md* | sim | $P_{1-3}$ | $P_{1-4}$ | 1,0,1,0 | - | - | - |
| | | *end_m* | não | - | - | 1,0,1,0 | - | - | - |
| | | *off_bat* | sim | - | - | 0,0,1,0 | - | - | - |
| | | *off_ctd* | sim | - | - | 0,0,0,0 | - | - | - |
| $F_{ap3}$ | $\sigma_{ap3}$: aproximação 3 | *st_m* | sim | $P_{1-4}$ | $P_{2-0}$ | 0,0,0,0 | 90 s | Naveg. | 0.5 |
| | | *end_m* | não | - | - | 0,0,0,0 | - | - | - |
| $F_2$ | $\sigma_{2-1}$: câmera e CTD - trajeto 1 | *on_cam* | sim | - | - | 0,1,0,0 | - | - | - |
| | | *on_ctd* | sim | - | - | 0,1,1,0 | - | - | - |
| | | *st_m* | sim | $P_{2-0}$ | $P_{2-1}$ | 0,1,1,0 | 120 s | Naveg. | 0.8 |
| | | *end_m* | não | - | - | 0,1,1,0 | - | - | - |
| | | *off_cam* | sim | - | - | 0,0,1,0 | - | - | - |
| | | *off_ctd* | sim | - | - | 0,0,0,0 | - | - | - |
| $F_{ap4}$ | $\sigma_{ap4}$: aproximação 4 | *st_m* | sim | $P_{2-1}$ | $P_{3-0}$ | 0,0,0,0 | 90 s | Naveg. | 0.5 |
| | | *end_m* | não | - | - | 0,0,0,0 | - | - | - |
| $F_3$ | $\sigma_{3-1}$: | *on_bat* | sim | - | - | 1,0,0,0 | - | - | - |

[11] Manobra de navegação.

| Fase | Tarefa | Evento | Controlável | Ponto Inicial | Ponto Final | Carga Útil[9] | *Timeout* | Manobra | Margem de Erro |
|---|---|---|---|---|---|---|---|---|---|
| | batime-tria, câmera e CTD - trajeto 1 | *on_cam* | sim | - | - | 1,1,0,0 | - | - | - |
| | | *on_ctd* | sim | - | - | 1,1,1,0 | - | - | - |
| | | *st_m* | sim | $P_{3\text{-}0}$ | $P_{3\text{-}1}$ | 1,1,1,0 | 270 s | Naveg. | 0.6 |
| | | *end_m* | não | - | - | 1,1,1,0 | - | - | - |
| | $\sigma_{3\text{-}2}$: batime-tria, câmera e CTD - trajeto 2 | *st_m* | sim | $P_{3\text{-}1}$ | $P_{3\text{-}2}$ | 1,1,1,0 | 270 s | Naveg. | 0.6 |
| | | *end_m* | não | - | - | 1,1,1,0 | - | - | - |
| | | *off_bat* | sim | - | - | 0,1,1,0 | - | - | - |
| | | *off_cam* | sim | - | - | 0,0,1,0 | - | - | - |
| | | *off_ctd* | sim | - | - | 0,0,0,0 | - | - | - |
| $F_r$ | $\sigma_r$: retorno | *st_mup* | sim | $P_{3\text{-}2}$ | $P_{recovery}$ | 0,0,0,0 | 90 s | Subida | 0.5 |
| | | *end_mup* | não | - | - | 0,0,0,0 | - | - | - |

A missão é realizada pela sequência de referência de eventos da coluna *evento* da tabela anterior, onde os eventos controláveis são escolhidos pelo GM e traduzidos para as respectivos entradas para o AUV pelo nível SO, com o parâmetro de missão correspondente quando necessário. Quando a sequência contiver um evento não-controlável, o GM irá aguardar que esse evento seja gerado pela planta para prosseguir com a missão. Para que a missão seja realizável pelo sistema, é necessária que a sequência de referência de eventos esteja contida na linguagem da planta sob supervisão, portanto, após encontrar o plano de missão, o GM realiza tal comprovação, empregando os modelos contidos no sistema CSML para efetuar tal cálculo. Nas simulações realizadas neste trabalho, o tempo médio para encontrar o menor caminho foi inferior a 1 s.

O método de replanejamento emprega a mesma estratégia do algoritmo de busca em largura, porém limita a busca ao considerar também o consumo de cada fase contrastando-o com o limite disponível de bateria. Somente as fases que podem ser executadas em sua totalidade são consideradas, deixando de fora as que possuem um consumo demasiado alto. Vários são os critérios que podem ser implementados para escolhas de fases, como por exemplo, a atribuição de número de prioridades de modo que o replanejamento inclua as tarefas mais prioritários, ou então, escolher o maior conjunto possível de tarefas, entre outras.

### 6.3.4. Execução da Missão

O mecanismo de escolha de eventos usa a política de atribuição de prioridades dinâmicas, com quatro níveis de prioridades, conforme já sugerido na seção 5.4.4. Os modos de operação implementados correspondem a quatro estados possíveis da missão, sendo eles: modo normal, quando não há ocorrência de erros; modo de falhas com cancelamento da missão e retorno imediato à superfície; modo de falhas com cancelamento da missão e retorno ao ponto de recuperação; e modo de falha de posição ou localização, ocasião em que o AUV deve realizar uma correção por GPS. Cada modo de operação possui uma política de atribuição de prioridades que pode ser diferente das demais. Além disso, somente o modo normal possui o plano de missão obtido pelo planejamento ou replanejamento como sequência de referência de eventos. Nos demais modos, a sequência de referência é específica a cada modo e é codificada no programa do SCM. Em todos os modos, os eventos não-controláveis possuem o segundo maior nível de prioridade e são mantidos fixos. Assim, na pior das hipóteses, ou seja, quando nenhum evento controlável possui prioridade elevada, o SCM irá aguardar pela ocorrência de um evento não-controlável, o que evita, em princípio, o bloqueio do sistema.

No modo normal, o evento mais prioritário é o próximo evento indicado pelo plano de missão. Caso este evento seja controlável, então a sua prioridade é definida como sendo máxima, e todos os demais eventos controláveis possuem prioridade mais baixa que os eventos não-controláveis. No caso do próximo evento ser não-controlável, então o GM assigna prioridade baixa para todos os eventos controláveis e aguarda que o evento não-controlável seja gerado pela planta. Caso o evento corresponda o próximo passo da missão, então se atualiza o passo seguinte até que não existam mais eventos no plano da missão.

Para o cancelamento de uma missão, duas são as sequências pré-programadas, também denominadas de modo de operação na seção 5.4.4: *com retorno ao ponto de recuperação* – ocorrência de falhas simples (*f*), nível baixo de bateria (*btr_wrn*), falhas nas manobras de descida (*er_md*) ou de navegação (*er_m*); *subir imediatamente à superfície a partir do ponto atual* – falhas graves (*fg*), colisões (*er_col*). Nestes modos de cancelamento, com retorno ou subida à superfície, todos os eventos controláveis que correspondem às ações de ligar um sensor são colocados como a prioridade mais baixa possível, ao passo

que as ações de desligar sensores, recebem prioridade elevada. O mesmo ocorre com as manobras de navegação e descida, onde os eventos controláveis de suspender, finalizar manobra recebem prioridade máxima e o início e retomada de eventos recebem prioridade mínima. Assim, a tendência do GM é desligar todos os sensores e finalizar as manobras de descida ou navegação, caso estejam em curso. A manobra de subida à superfície recebe prioridade de início ou retomada máxima para o caso de cancelamento com emersão imediata. Procedimento similar é realizado para a manobra de retorno.

Para a correção por GPS, quando um evento não-controlável de falha de posição (*fpos*) é gerado, uma sequência de subir à superfície, ligar o GPS, corrigir o erro, desligar o GPS, submergir e retornar à tarefa em curso é inserido no meio do plano da missão, através da modificação das prioridades dos eventos. Neste modo as prioridades são configuradas em duas etapas. Na primeira, recebem maior prioridade os eventos que conduzem o veículo à superfície e à calibração por GPS. Na segunda etapa, os eventos que reposicionam o veículo na tarefa que estava sendo realizada antes da detecção da falha recebem maior prioridade.

## 6.4. ESTRUTURA GERAL DO AMBIENTE DE SIMULAÇÃO

De modo a validar a arquitetura proposta para o SCM, desenvolveu-se um ambiente de simulação distribuído para emular o funcionamento dos componentes remanescentes da arquitetura, como sensores, atuadores e a bateria. O ambiente está compreendido por dois componentes: o simulador da dinâmica contínua do veículo e outro simulador para a dinâmica discreta. Tal ambiente permite a realização de testes do SCM sem a necessidade de dispor do veículo real, sempre que o ambiente de simulação seja capaz de prover os mesmos sinais e comandos físicos gerados pelos equipamentos do AUV. Contudo, a validação do SCM no ambiente de simulação desenvolvido não se constituem da técnica de simulação denominada de *Hardware-in-The-Loop* (HIL). No HIL, o sistema desenvolvido a ser testado tem as suas entradas e saídas reais conectados a um modelo dinâmico simulado da planta, cujos sinais de entrada e saída são os mesmos da planta real (BACIC, 2005; SHIXIANJUN, JIAKUN e HONGXING, 2006). Desse modo, a simulação HIL permite realizar comprovações antes da construção de todo o sistema completo. Neste trabalho, devido a falta de um AUV com as características consideradas, optou-se pelo

desenvolvimento de uma plataforma de simulação que fosse capaz de representar os demais sistemas do AUV, porém, sem necessariamente gerar todos os sinais de baixo nível da arquitetura do veículo.

A figura 6.5 apresenta o diagrama geral do ambiente desenvolvido para o sistema operacional Linux, com os três componentes: o SCM com execução gerenciada pelo ROS; a dinâmica contínua do movimento do AUV e do sistema de controle, simulada pelo software Matlab/Simulink; e a dinâmica dirigida a eventos emulada por uma IHM desenvolvida em Java. Cada componente é executado em um computador separado, conectado aos demais via uma rede de comunicação dedicada, do tipo *Ethernet* (IEEE 802.3) ou *Wi-Fi* (IEEE 802.11). As simulações são realizadas de modo *on-line*, ou seja, com os três componentes funcionando ao mesmo tempo ou em paralelo.

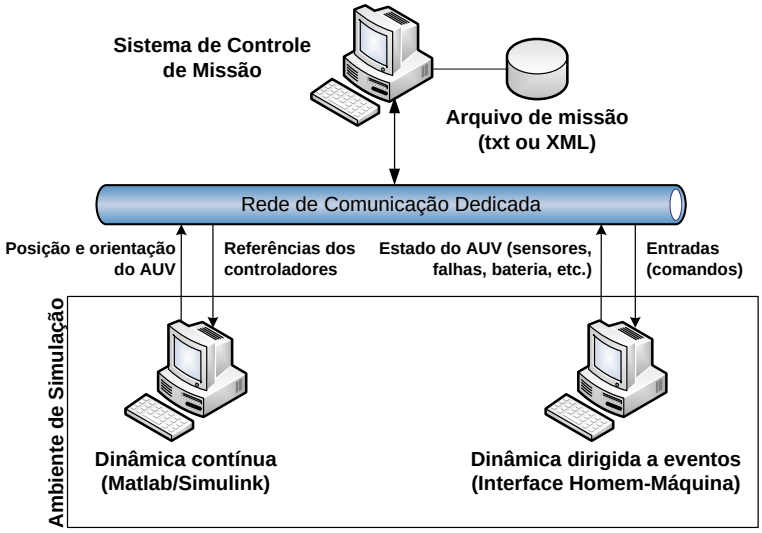


**Figura 6.5:** Organização geral do ambiente de simulação e o SCM

A figura 6.6, por sua vez, apresenta as portas de comunicação TCP/IP dos processos em execução no ROS conectados via *parsers* ao software Matlab/Simulink e à IHM. O *parser* é desenvolvido em Java e transforma os *bytes* que contêm a posição e orientação no formato empregado pelo Matlab/Simulink em um formato que seja entendível pelo processo C/C++ que recebe tal informação, e vice-versa.

d

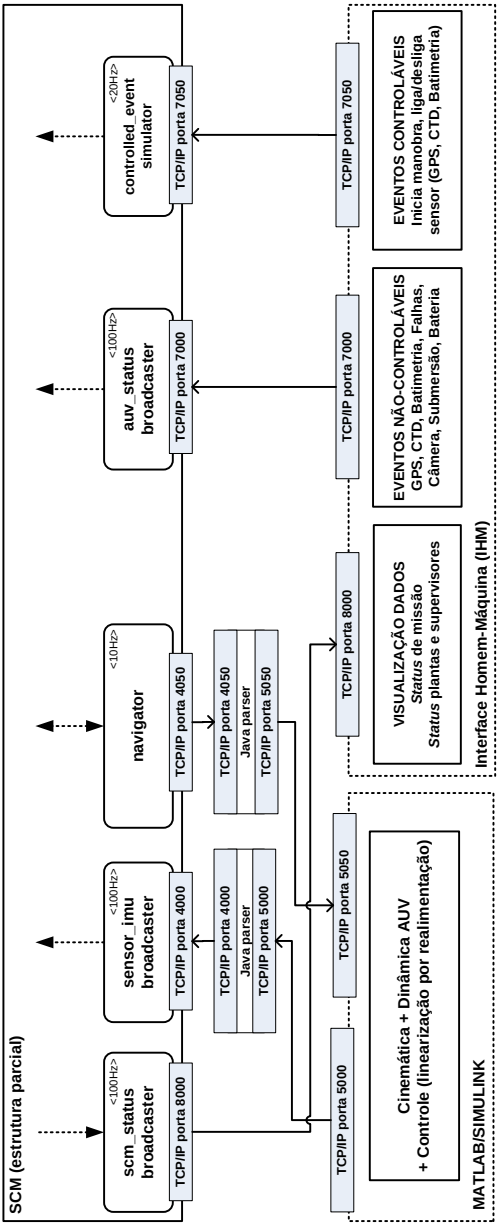


**Figura 6.6:** Conexão Matlab/Simulink, IHM e processos ROS

O SCM em execução no ROS emprega vários processos para capturar os dados proveniente dos demais subsistemas do AUV e para enviar comandos. Tais processos empregam como modelo de comunicação portas baseadas em *sockets*, que fornecem a infraestrutura para comunicação em uma rede com padrão TCP/IP. Como a rede é dedicada, ou seja, assume-se que somente o SCM, o software Matlab/Simulink e a IHM estão conectadas a mesma, a carga ou tráfego de dados não possui congestionamento. Além disso, para garantir a implementação de um canal seguro de comunicação, sem a perda de pacotes na rede, foi adotado o protocolo TCP que garante entrega confiável de pacotes. Diferente do protocolo TCP, o protocolo UDP não apresenta garantia de entrega confiável, porém apresenta uma carga ou *overhead* no envio de informações muito menor, porém, por se considerar que a rede de comunicação é dedicada, optou-se pelo TCP. Algumas implementações de veículos subaquáticos que também empregam infraestruturas de comunicação baseada em Ethernet com TCP/IP podem ser encontradas em Freezor, Sorrell e Blankinship (2001), Sangekar e Chitre (2008) e Huang, Li e Jin (2013). Contudo, a arquitetura proposta para o SCM não depende do padrão de comunicação TCP/IP.

A implementação real do sistema deverá considerar o tipo de rede a ser empregada dependendo, principalmente, das interfaces de comunicação dos diversos sensores. Entretanto, tal alteração somente necessita ser feita nos processos que periféricos do SCM, que trocam informações com a dinâmica contínua e discreta, sendo possível manter todo o restante do SCM conforme proposto neste trabalho. Tal característica também foi explorada no sentido de que a implementação do ambiente de simulação viabiliza a validação do SCM proposto, sem necessariamente dispor de um veículo real, desde que a natureza temporal dos dados (frequência e quantidade) seja mantida.

### 6.4.1. Simulação da Dinâmica Contínua do AUV

A dinâmica do movimento contínuo do AUV, conforme comentado na seção 2.1.1, é descrita pelas equações cinemática, que detalham os aspectos geométricos do movimento, e dinâmica resultante da análise de forças e momentos produzidos no contexto do movimento do AUV (FOSSEN e BALCHEN, 1991; FOSSEN, 1994; SOUZA, 2003; ROBERTS e SUTTON, 2006). Para esse trabalho, considerou-se o veículo NEROV, cujos parâmetros são obtidos a partir de Fejllstad

(1994) e Tavares (2003). A modelagem cinemática e dinâmica do NEROV é apresentada em Fossen e Balchen (1991) e em Tavares (2003). A estratégia de controle adotada é a linearização por realimentação (FOSSEN, 1994; TAVARES, 2003). Optou-se pelo emprego de modelos dinâmicos simplificados para o AUV e o seu sistema de controle pois o objetivo deste trabalho não é a comprovação do modelo dinâmico ou da eficiência do controle em malha-fechada, permitindo reduzir, portanto, a complexidade dos modelos dinâmicos e facilitando a sua simulação.

O modelo dinâmico e a respectiva lei de controle são então implementados no software Matlab/Simulink. O modelo dinâmico funciona como fonte de dados do movimento do veículo, substituindo o sistema de localização e os sensores de posição, como IMU e LBL. Admite-se que a estimação da localização é suficiente precisa para as missões consideradas nesse trabalho, com exceção dos erros de posicionamento, que são gerados pela IHM que simula o comportamento discreto do AUV. Os dados de posição $<x, y, z>$ e orientação $<\phi, \theta, \psi>$ atuais do movimento do veículo são enviados pelo Matlab/Simulink ao SCM mediante a porta de comunicação TCP/IP 5000. O *parser*, por sua vez, repassa os bytes convertidos para o novo padrão mediante a porta TCP/IP 4000, sendo recebido pelo processo *sensor_imu_broadcaster*, que é o responsável de informar aos demais componentes do SCM a posição e orientação atuais do AUV. A frequência em que os dados são enviados corresponde ao passo mínimo de simulação do Matlab/Simulink, configurado variável.

A troca dos valores de referência dos controladores entre o software Matlab/Simulink e o SCM é realizada mediante outro *parser*, também desenvolvido em Java, que recebe do processo de navegação (*navigator*), via porta TCP/IP 4050, o valor de referência para os controladores em malha-fechada sempre que uma nova manobra for iniciada, e os reenvia para o Matlab/Simulink, via porta TCP/IP 5050.

### 6.4.2. Simulação da Dinâmica Dirigida a Eventos do AUV

Os aspectos discretos da dinâmica do AUV são simulados através de uma Interface Homem-Máquina (IHM) desenvolvida em linguagem Java. Basicamente, mediante botões, é possível gerar múltiplos comandos ou sinais que estão associados aos eventos controláveis e não-controláveis que são enviados ao SCM. Em um modo de operação manual, no qual o SCM não gera o plano de missão ou os comandos que

irão guiar o veículo, também é possível empregar a IHM para comandar o movimento do veículo. Contudo, no modo normal de operação esses comandos estão desabilitados e são ignorados pelo SCM, que guia o veículo a partir dos algoritmos de planejamento e replanejamento.

A IHM permite emular os sinais provenientes dos seguintes subsistemas do AUV: bateria (nível baixo e nível crítico de bateria); sensores de carga útil (erro no funcionamento de um sensor); estado da submersão do veículo (emerso ou submerso), obtido através do sensor de profundidade; o modelo de falhas que encapsula em falhas simples, graves e de posição os vários tipos possíveis de erro em vários subsistemas, como a presença de água ou temperatura elevada no compartimento de instrumentação, um sinal de erro estimado por algum mecanismo de detecção de falhas, ou ainda o erro elevado entre a estimação da posição do veículo gerado pelos algoritmos de localização. As falhas nas manobras ocorrem em razão dos *timeouts* programados para realização do movimento e, portanto, não necessitam ser gerados pela IHM.

A IHM também possui interfaces para acompanhamento da missão, onde é possível visualizar a posição e orientação atual do veículo, bem como todas as fases que compõem a missão em realização. Na IHM é possível também seguir a evolução dos modelos ou os estados das plantas do nível SP do CSML. No apêndice D é possível visualizar as telas da IHM.

## 6.5. RESUMO

O SCM, implementado no *middleware* ROS, emprega dois componentes principais, o CSML e o GM, encarregados, respectivamente, de garantir a segurança e integridade da missão ao atender requisitos e especificações, e pelo planejamento e execução das missões. A geração automática de código e a arquitetura de implementação do CSML permitem integrar diretamente no SCM, os modelos formais baseados em autômatos, diminuindo consideravelmente os erros inerentes ao processo de codificação de tais estruturas. O *middleware* ROS fornece diversas ferramentas e bibliotecas, com soluções consolidadas para diversos problemas usualmente encontrados no desenvolvimento de aplicações robóticas, como mecanismos de comunicação, operações matemáticas de transformações de coordenadas e processamento de dados, modelos de

sensores, algoritmos de navegação, entre outros, justificando assim, o seu uso nas etapas de projeto e construção desse tipo de sistemas.

De modo a permitir simular os aspectos contínuos e dirigidos a eventos do comportamento dinâmico do AUV, foi desenvolvido um ambiente de simulação com objetivo de suprir a ausência de um veículo real e, assim mesmo, permitir validar o SCM proposto. Baseado no software Matlab/Simulink, para simulação da parte contínua, e em uma IHM, desenvolvida em Java, para emulação dos aspectos discretos do AUV, o ambiente é integrado ao SCM através de uma rede de comunicação dedicada que suporte o protocolo TCP/IP. A validação da arquitetura completa é realizada por meio de simulações de vários cenários de missão, que serão apresentadas no próximo capítulo.

# 7. TESTES E ANÁLISES

O objetivo desse capítulo é testar, através de simulações, o comportamento discreto ou dirigido a eventos do SCM em missões de AUVs com as características descritas pelo ambiente de simulação, em contextos nas quais as ações de planejamento, replanejamento do GM e garantia de especificação de segurança e operação do CSML sejam observadas e, a partir de tais observações, validadas. Portanto, não é objetivo deste trabalho avaliar os modelos contínuos usados na simulação do veículo ou ainda a performance dos controladores em malha-fechada do movimento do AUV. Para a realização dos testes, são simulados diversos cenários de missão, inspirados nos ambientes não-estruturados de lagos de barragens.

Na segunda parte do capítulo, são apresentadas as análises referentes ao desempenho do SCM observado nas simulações, relacionando também algumas vantagens e desvantagens com respeito as principais arquiteturas baseadas em SEDs, brevemente mostradas no capítulo 3, além de discutir aspectos práticos e limitações relacionadas a questões de tempo-real.

## 7.1. SIMULAÇÕES DE CENÁRIOS DE MISSÕES

Os cenários considerados para esse trabalho objetivam apresentar o funcionamento do SCM em quatro tipos de condições distintas, a saber:

- Caso 1: realização da missão em um cenário ideal.
- Caso 2: realização completa da missão, mas com necessidade de correção por GPS, porém sem a necessidade de replanejamento.
- Caso 3: realização completa da missão com replanejamento, onde a carga insuficiente da bateria exige a eliminação de uma ou mais fases da missão.
- Caso 4: missão com desvio de obstáculo e cancelamento de missão devido a uma falha grave.

A missão considerada será a mesma para todos os cenários e está composta pelas seguintes fases:

- **Fase 1:** aquisição de dados via sensor CTD.
- **Fase 2:** aquisição de imagens em área submersa.
- **Fase 3:** aquisição de dados de topografia do fundo do leito empregando sonar de batimetria.
- **Fase 4:** aquisição de dados via sensor CTD e sonar de batimetria.
- **Fase 5:** aquisição de dados via sensor CTD e sonar de batimetria.

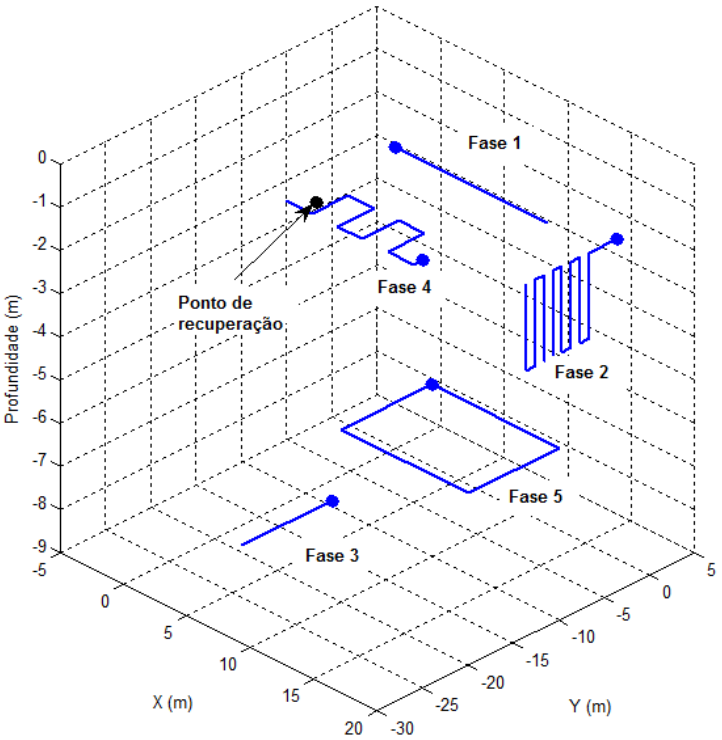


**Figura 7.1:** Missão com as cinco fases

A figura 7.1 apresenta a região de realização de cada uma das fases, onde o círculo representa o início da fase e o ponto de lançamento do veículo é considerado na origem do sistema inercial. Observa-se na figura que os eixos *y* e *z* aparecem em coordenadas negativas pois, ainda que possuam valores positivos no ROS, no software Matlab/Simulink são representados com um giro de 180º em torno ao eixo *z* e, por esse

motivo, surgem os valores negativos de modo a manter o mesmo traçado apresentado pela ferramenta de visualização RVIZ do ROS. A unidade empregada nos eixos coordenados é metros (m).

Devido à implementação de uma estrutura simplificada de controle e sem a geração suave de trajetórias de referências, o desempenho do controle de movimento torna-se insatisfatório para degraus de referência de posição acima de 5 m. Isso ocorre devido à saturação da ação integral na lei de controle e pela ausência de uma trajetória suave de referência. Esse problema pode ser resolvido através da inclusão de uma janela de esquecimento (GOMES e BIER, 1998), trazendo mais estabilidade ao controlador. Neste trabalho, para contor-

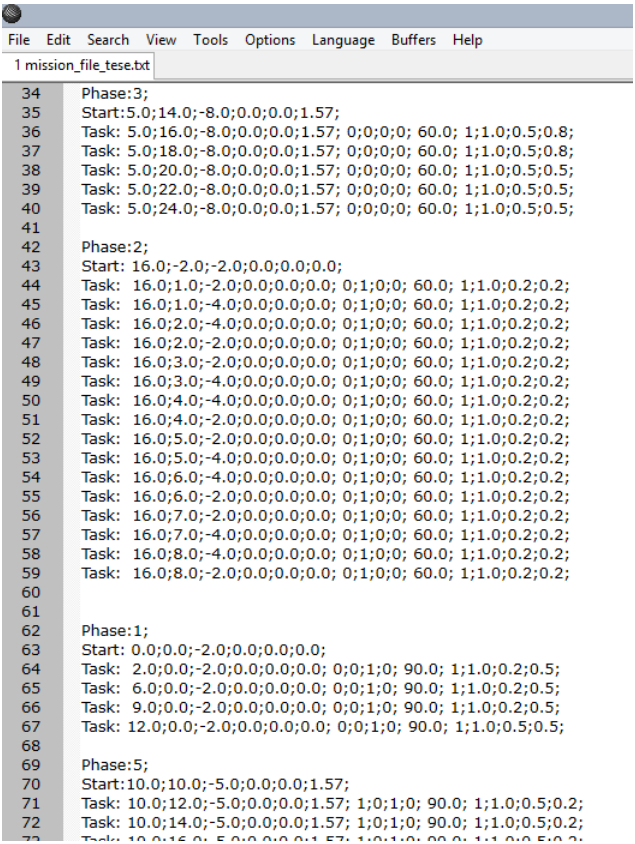


**Figura 7.2:** Arquivo de missão

nar a limitação da performance dos controladores, sempre que os deslocamentos são superiores a valores da ordem de 3 a 5 m, optou-se

por dividir manualmente a trajetória em segmentos. Contudo, tal divisão em segmentos pode ser feita automaticamente pelo próximo sistema de leitura do arquivo de missão. Por esse motivo, o arquivo de missão contém a divisão das trajetórias maiores em diversos segmentos e, consequentemente, o plano de missão irá conter as manobras de navegação correspondentes a cada um dos segmentos representadas como tarefas individuais.

A figura 7.2 apresenta uma vista parcial do arquivo de missão, em formato TXT, com as especificações das fases 1, 2 e 3. É possível observar no arquivo de missão que, para evitar a aplicação de variações nas referências de posição superiores a 5 m, trajetos maiores a 5 m são divididos em segmentos. Tal divisão é feita manualmente, durante a edição do arquivo de missão.

O plano de missão gerado pelo GM, com a sequência de fases que otimiza o consumo e deslocamentos é dada por $F_1 \rightarrow F_2 \rightarrow F_5 \rightarrow F_4 \rightarrow F_3$. Segundo as entradas nos arquivos de *log* do SCM, o tempo para realização do planejamento dessa missão é inferior a 1 s. A sequência de fases com as respectivas tarefas e eventos é apresentada resumidamente na tabela 7.1.

**Tabela 7.1:** Plano de missão com a sequência de fases para o melhor caminho

| **Fase** | **Descrição da fase** | **Sequência de Referência de Eventos** | **Controlável** | **Descrição do evento** | ***Timeout*[12]** |
|---|---|---|---|---|---|
| $F_{ap1}$ | Aproximação 1: deslocamento entre o ponto de lançamento e início da fase 1 | *st_md*<br>*end_md* | sim<br>não | Iniciar manobra<br>Aguardar fim da manobra | 90 s |
| $F_1$ | Realização manobra de navegação com aquisição de dados via sensor CTD | *on_ctd*<br>*st_m*<br>*end_m*<br>...[13] | sim<br>sim<br>não<br>... | Ligar CTD<br>Iniciar manobra<br>Aguardar fim da manobra<br>... | 60 s para cada manobra (segmen-to) |

[12] O *timeout* é especificado no arquivo de missão, porém, para as manobras de aproximação entre as fases, emprega-se o valor *default* de 90s.

| Fase | Descrição da fase | Sequência de Referência de Eventos | Controlável | Descrição do evento | *Timeout*[12] |
|---|---|---|---|---|---|
| | | *st_m* | sim | Iniciar manobra | |
| | | *end_m* | não | Aguardar fim da manobra | |
| | | *off_ctd* | sim | Desligar CTD | |
| $F_{ap2}$ | Aproximação 2: deslocamento entre fases 1 e 2 | *st_m* | sim | Iniciar manobra | 90 s |
| | | *end_m* | não | Aguardar fim da manobra | |
| $F_2$ | Realização de manobra de navegação com câmera ligada | *on_cam* | sim | Ligar câmera | 60 s para cada manobra (segmento) |
| | | *st_m* | sim | Iniciar manobra | |
| | | *end_m* | não | Aguardar fim da manobra | |
| | | ... | ... | ... | |
| | | *st_m* | sim | Iniciar manobra | |
| | | *end_m* | não | Aguardar fim da manobra | |
| | | *off_cam* | sim | Desligar câmera | |
| $F_{ap3}$ | Aproximação 3: deslocamento entre fases 2 e 5 | *st_m* | sim | Iniciar manobra | 90 s |
| | | *end_m* | não | Aguardar fim da manobra | |
| $F_5$ | Realização de manobra como aquisição de dados com sonar de batimetria e sensor CTD | *on_bat* | sim | Ligar sonar | 90 s para cada manobra (segmento) |
| | | *on_ctd* | sim | Ligar CTD | |
| | | *st_m* | sim | Iniciar manobra | |
| | | *end_m* | não | Aguardar fim da manobra | |
| | | ... | ... | ... | |
| | | *st_m* | sim | Iniciar manobra | |
| | | *end_m* | não | Aguardar fim da manobra | |
| | | *off_bat* | sim | Desligar sonar | |
| | | *off_ctd* | sim | Desligar CTD | |
| $F_{ap4}$ | Aproximação 4: deslocamento entre as fases 5 e 4 | *st_m* | sim | Iniciar manobra | 90 s |
| | | *end_m* | não | Aguardar fim da manobra | |
| $F_4$ | Realização de manobra de | *on_bat* | sim | Ligar sonar | 60 s para cada |
| | | *on_ctd* | sim | Ligar CTD | |

[13] Representa a quantidade de segmentos representados no arquivo de missão, cuja realização é feita com a operação de iniciar manobra de navegação e terminar manobra. Tais segmentos são omitidos com objetivo de apresentar somente um resumo do plano de missão.

| Fase | Descrição da fase | Sequência de Referência de Eventos | Controlável | Descrição do evento | *Timeout*[12] |
|---|---|---|---|---|---|
| | navegação com aquisição de dados com sonar de batimetria e sensor CTD | *st_m*<br>*end_m*<br>...<br>*st_m*<br>*end_m*<br>*off_bat*<br>*off_ctd* | sim<br>não<br>...<br>sim<br>não<br>sim<br>sim | Iniciar manobra<br>Aguardar fim da manobra<br>...<br>Iniciar manobra<br>Aguardar fim da manobra<br>Desligar sonar<br>Desligar CTD | manobra (segmento) |
| $F_{ap5}$ | Aproximação 5: deslocamento entre as fases 4 e 3 | *st_m*<br>*end_m* | sim<br>não | Iniciar manobra<br>Aguardar fim da manobra | 90 s |
| $F_3$ | Realização de manobra de navegação com aquisição de dados com sonar de batimetria | *on_bat*<br>*st_m*<br>*end_m*<br>...<br>*st_m*<br>*end_m*<br>*off_bat* | sim<br>sim<br>não<br>...<br>sim<br>não<br>sim | Ligar sonar<br>Iniciar manobra<br>Aguardar fim da manobra<br>...<br>Iniciar manobra<br>Aguardar fim da manobra<br>Desligar sonar | 60 s |
| $F_r$ | Retorno ao ponto de recuperação | *st_mr*<br>*end_mr* | sim<br>não | Iniciar manobra de retorno<br>Aguardar fim da manobra | 90 s |

O SCM foi testado em um PC i5 1.6 GHz com 4 Gb de RAM, a IHM foi executada em um PC Core2Due 2.0 GHz com 2 Gb de RAM e o software Matlab/Simulink, em um PC i5 1.6 GHz com 4 Gb de RAM. Em todas as máquinas o sistema operacional utilizado foi o Linux Ubuntu 12.04. Os experimentos foram realizados empregando redes de comunicação dedicadas do tipo Ethernet (IEEE 802.3) e Wi-fi (IEEE 802.11), não havendo diferença significativa de desempenho devido a tempos de respostas ou atrasos na comunicação.

### 7.1.1. Caso I: Missão em Cenário Ideal

No primeiro caso, considera-se a realização da missão em um cenário ideal, ou seja, sem a ocorrência de erros e com energia suficiente para a realização de toda a missão. A figura 7.3 mostra a simulação da realização completa da missão, onde o veículo submerge em direção à fase $F_1$, com todos os sensores de carga útil desligados. O plano de missão com a sequência de referência dos eventos é obtido durante o processo de planejamento que ocorre enquanto o veículo encontra-se no ponto de lançamento.

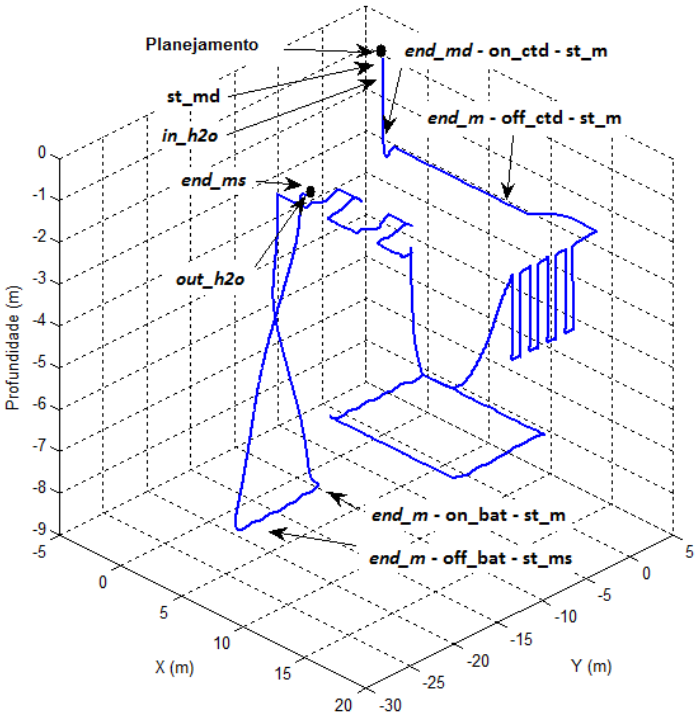


**Figura 7.3:** Cenário com realização completa da missão (Matlab/Simulink)

A figura 7.3 mostra várias sequências de referências parciais para ilustrar a dinâmica discreta do AUV. A primeira trajetória consiste na submersão do veículo onde o SCM seleciona o evento controlável de início de manobra de descida (*st_md*) da sequência de referências. O veículo inicia sua descida e o evento não-controlável de imersão é gerado (*in_h2o*) indicando que o veículo se encontra submerso. Ao

AUV alcançar o final da primeira trajetória de descida o evento não-controlável *end_md* é gerado. Na sequência o SCM inicia a sequência para realização da fase 1, que consiste em ligar o sensor CTD (evento controlável *on_ctd*) e iniciar a manobra de navegação (evento controlável *st_m*). A fase de aquisição continua até o final da trajetória ser alcançada (evento não-controlável *end_m*), ocasião em que o sensor CTD é desligado (evento controlável *off_ctd*) e uma manobra de aproximação à próxima fase é iniciada (evento controlável *st_m*). O mesmo procedimento ocorre para todas as fases remanescentes. A figura 7.3 também ilustra as sequências de referências de eventos para a fase 3 e para a fase de retorno ao ponto de recuperação do veículo.

Como a sequência de comandos (eventos controláveis) e sinais provenientes do AUV (eventos não-controláveis) corresponde exatamente à sequência de referência do plano de missão, não há necessidade de modificá-lo (replanejamento). A figura 7.4 apresenta a mesma informação, porém, gerada pela ferramenta de visualização do ROS denominada de RVIZ.

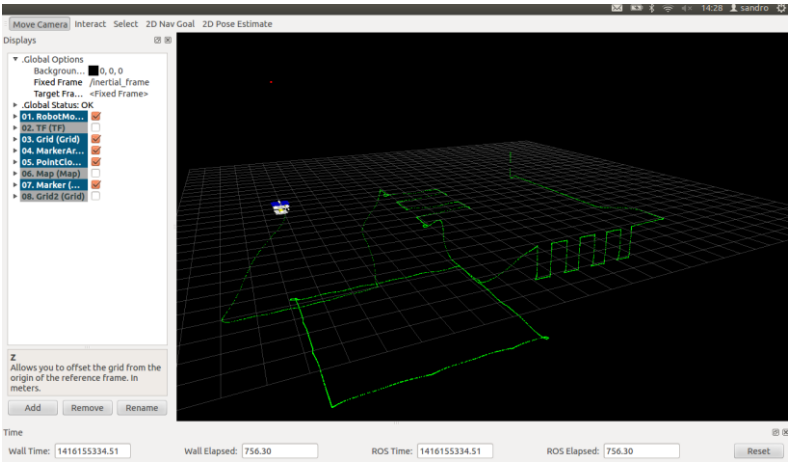


**Figura 7.4:** Cenário com realização completa da missão (RVIZ)

### 7.1.2. Caso II: Correção por GPS e Cancelamento de Missão

O segundo cenário é empregado para ilustrar os modos de operação normal, correção por GPS e cancelamento de missão. Neste cenário, considera-se que há um valor de erro considerável na estimação

da localização do veículo, e, por esse motivo, o SCM decide pela suspensão da manobra em curso para subir à superfície e realizar a recalibração do subsistema de localização mediante o uso do GPS. Tal procedimento é realizado em duas ocasiões. Nesses casos, não há necessidade do SCM ativar o replanejamento da missão. Além disso, o SCM retoma a fase suspensa a partir da última tarefa em curso, sempre que a fase corresponda a uma aquisição de dados. Contudo, caso o erro de localização ocorra durante a realização de uma fase de aproximação, ou seja, durante uma fase intermediária entre duas fases de aquisição de dados, o SCM então opta por guiar o veículo até o início da próxima fase de aquisição de dados.

Basicamente, a ação do SCM consiste em, a partir do recebimento de um evento não-controlável de falha de posição (evento não-controlável *fpos*), atualizar os modelos e supervisores e comprovar que os caminhos habilitados pelo CSML somente permitem sequências de eventos que levem o veículo à superfície. Esse procedimento é realizado através da sequência de referência e da política de prioridades do modo de operação de correção por GPS. Por esse motivo, o SCM suspende e finaliza a manobra em curso, inicia a manobra de subida à superfície. Uma vez na superfície, o GM consulta novamente o CSML para identificar que somente o caminho que permite a ativação do GPS e correção do erro está disponível. Uma vez corrigido o erro de posicionamento (evento controlável *rst_f*), os modelos e supervisores do CSML, atualizados, habilitam a sequência de eventos que possibilita o AUV retomar a manobra em curso e continuar com a realização da missão.

Também se considera nesse cenário, a ocorrência de uma falha simples (evento não-controlável *f*), que não impede a navegação do veículo, porém impede a continuidade da missão. A ocorrência desse evento de falha atualiza os estados dos modelos baseados em autômatos da planta, e mediante a ação dos supervisores, somente os caminhos que conduzem o veículo ao retorno encontram-se habilitados e, por esse motivo, o GM escolhe o caminho mais prioritário, que corresponde à manobra de retorno ao ponto de recuperação. O modo de operação ativo nesse caso é o de cancelamento de missão com retorno ao ponto de recuperação.

A figura 7.5 mostra a trajetória simulada do AUV, obtida a partir do software Matlab/Simulink, para o cenário do caso II, com as duas correções por GPS, indicadas na figura, e com o cancelamento da missão e retorno ao ponto de recuperação.

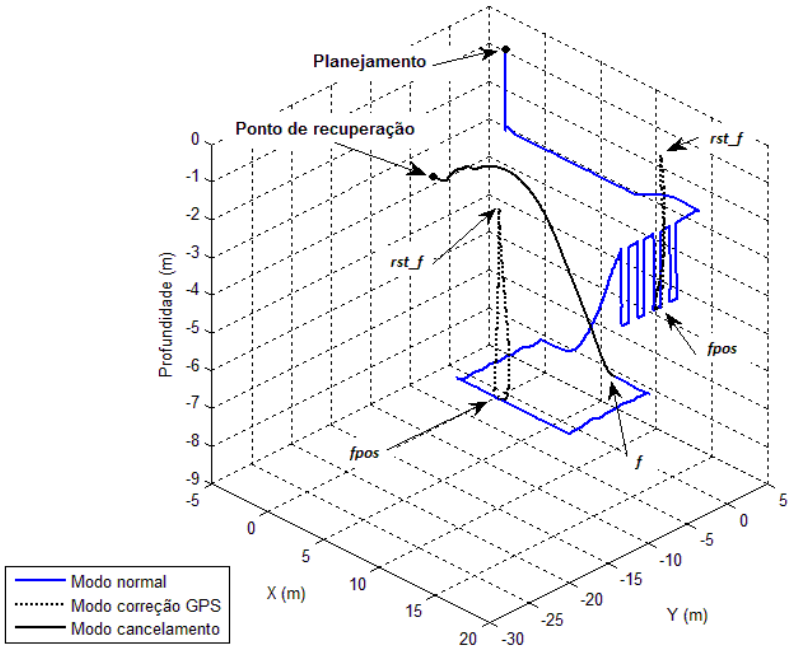


**Figura 7.5:** Trajetória missão do caso II (Matlab/Simulink)

### 7.1.3. Caso III: Replanejamento de Missão

Para o cenário de missão do caso III foi considerado a ação de replanejamento do GM para a obtenção de um novo plano de missão contendo somente as fases possíveis de serem completadas com a carga de bateria disponível. O consumo inicial previsto para a realização da missão pode não ser igual ao consumo real em um cenário onde a presença de correntes, diferenças entre a trajetória estimada e a realizada, falhas em subsistemas, correção por GPS ou desvios de obstáculos acabam por exigir mais energia do sistema de potência. A ação do GM consiste em monitorar o consumo estimado até o final da missão, estimação esta realizada pelo processo *power_estimator*, e caso seja superior ao disponível, então o replanejamento é ativado. Sem a necessidade de suspensão da manobra em curso, ou seja, de modo *on-line*, o replanejamento seleciona qual o maior conjunto de fases possíveis de serem realizadas com a carga disponível, gerando um novo

plano de missão, possivelmente em uma ordem diferente da original, dando prioridade para finalização da fase em curso.

A figura 7.6 apresenta a trajetória completa da missão, considerando o replanejamento de missão, que exclui as fases 2 e 5 da missão devido à baixa quantidade de bateria disponível para completar a missão. A missão é replanejada ao final da fase 1, e tal informação é obtida a partir da consulta do arquivo de registro (*log*) do sistema.

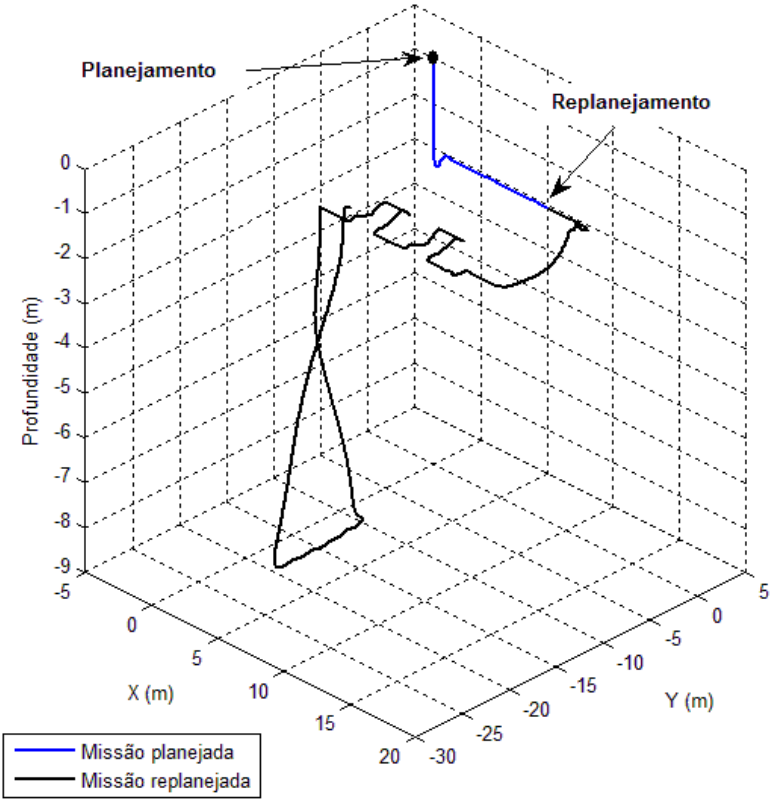


**Figura 7.6:** Trajetória da missão para o caso III (Matlab/Simulink)

### 7.1.4. Caso IV: Desvio de Obstáculo e Cancelamento

O desvio de obstáculos não é implementado pelo SCM, mas sim pelo comportamento reativo incorporado diretamente no processo de navegação do AUV. Contudo, o SCM emprega um modelo para

representar o estado do desvio de obstáculos e assim poder cancelar a missão sempre que uma colisão ocorrer. O presente trabalho, conforme já comentado na seção 6.1, emprega uma estrutura de mapeamento baseada em *octotree*, que constrói um mapa de ocupação com as células livres de obstáculos e as células ocupadas na região do obstáculo detectado. O processo *obstacle_tree*, com base no mapa de ocupação, calcula se na rota atual do AUV, compreendida entre as coordenadas atuais do veículo e a desejada para o final da manobra, há alguma célula ocupada por algum obstáculo. Caso exista algum obstáculo, o procedimento do processo de navegação consiste em contornar o obstáculo e direcionar o veículo novamente para a sequência de manobras original, desconsiderando o trajeto em que o objeto estava presente. Para isso, considera-se a presença de um obstáculo em $x$ = 2.0 m, com 4,0 m de largura (direção $y$) e altura de 2,0 m (direção z), reconhecido pelo *feedfoward looking sonar* com capacidade de detecção de até 5 m adiante do veículo. A figura 7.7 contém o início do trajeto de desvio, com o obstáculo detectado, obtido a partir da ferramenta RVIZ.

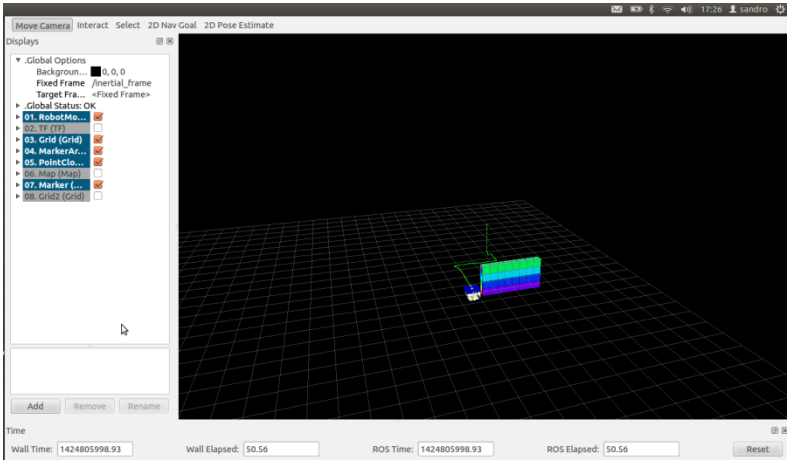


**Figura 7.7:** Detecção de obstáculo e início de trajeto de desvio

Para ilustrar também a ação de cancelamento da missão e emersão imediata à superfície, também foi considerada para esse cenário de missão a ocorrência de uma falha grave (*fg*). A figura 7.8 apresenta a trajetória realizada pelo veículo ao contornar o obstáculo e o cancelamento da missão. Na figura 7.9 é possível observar na ferramenta RVIZ, as células de ocupação e o obstáculo detectado.

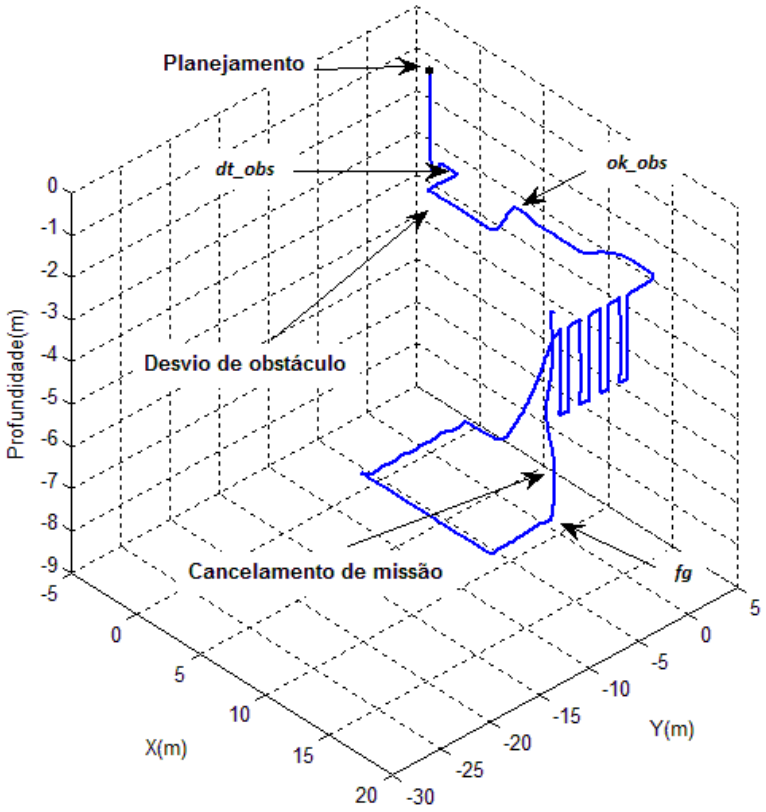


**Figura 7.8:** Cenário de missão com desvio de obstáculos (Matlab/Simulink)

## 7.2. ANÁLISES E CONSIDERAÇÕES

As simulações apresentadas na seção anterior demonstram que o SCM é capaz de tratar os principais problemas relacionados à representação (modelagem, análise) e execução de missões para um AUV atuando em um ambiente não-estruturado. As simulações também mostraram que as características de funcionamento e flexibilidade desejados para o SCM foram atendidas pela implementação do SCM. Assim, o sistema foi capaz de cumprir plenamente com quesitos para funcionamento seguro para as missões e durante toda a sua realização, definidos pelas especificações do CSML. Ao mesmo tempo, ante a ocorrência de eventos não-deterministas e não previstos no plano de missão, representado pelos eventos não-controláveis, e ante variações

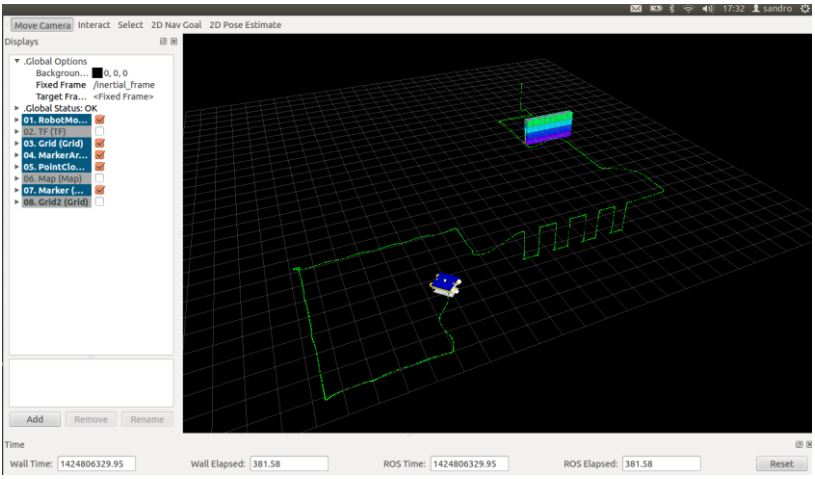


**Figura 7.9:** Missão com obstáculo detectado e mapa com célula de ocupação (RVIZ)

nas condições do veículo, como níveis de bateria inferiores ao considerados no planejamento da missão, foram tratados com suficiente segurança e flexibilidade pelo CSML e pelo GM ao considerar as diversas opções de replanejamento de missão, cancelamento de fases ou o cancelamento em si da missão. Do ponto de vista lógico da missão, ou seja, da representação discreta dos diversos componentes e subsistemas do AUV, a abordagem baseada em CSML se mostra adequada para descrever a evolução dinâmica dirigida a eventos da missão, mas também, com a arquitetura de implementação do CSML, é possível empregar diretamente tal formalismo no sistema embarcado do veículo. Tais características são discutidas na seção 7.2.1.

Ainda que o SCM tenha sido validado em um ambiente de simulação, considera-se que, conceitualmente, o seu emprego em um veículo real seja viável, principalmente devido: aos tempos de execução relativamente baixos, se comparados aos processos de tempo-real mais críticos do AUV; a possibilidade de empregar outros protocolos de comunicação – tempo-real por exemplo; e a relativa independência dos demais processos e algoritmos, como localização, fusão de dados sensoriais ou estratégias de controle. Tais aspectos serão brevemente comentados, respectivamente, nas seções 7.2.2, 7.2.3 e 7.2.4, com objetivo de apresentar algumas relações com uma possível implementação em um veículo real do SCM proposto. Finalmente, uma

análise qualitativa do SCM proposto e uma breve comparação com outras arquiteturas baseada em SEDs é mostrada na seção 7.2.5.

### 7.2.1. Uso de Modelos Formais

Os modelos e especificações desejadas para a operação e segurança do sistema são garantidas no nível de Supervisores Modulares (SM) e Sistema-Produto (SP). As simulações permitem validar que todas as especificações são plenamente atendidas e, pelo próprio processo de modelagem e síntese e pela arquitetura de implementação do CSML, garantidas também na execução do SCM. A vantagem da adoção de uma arquitetura de implementação, no caso, a proposta pelo CSML, permite manter os modelos empregados durante a fase de análise, assim o comportamento do AUV no nível de missão é aquele dado pelos modelos baseados em autômatos. A inclusão dos modelos formais diretamente no AUV conferem ao sistema as propriedades inicialmente definidas para o modelo baseado em SED, como ausência de *deadlocks* e *livelocks* e atendimento de especificações, diferente da maioria das abordagens mencionadas no capítulo 3, como em Xu, Zhang e Feng (2004a, 2004b), Molina *et al*. (2010), Liu e Darabi (2002), Wang *et al*. (1991), Chang *et al*. (2005), Bian *et al*. (2009b), Dias *et al*. (2006) e Tangirala *et al*. (2005). A inclusão dos autômatos no sistema embarcado reduz também os erros decorrentes da adaptação de tais estruturas, conforme apontado por Liu e Darabi (2002). Contudo, devido aos diversos processos em execução no ROS, os componentes dinâmicos contínuos simulados pelo Matlab/Simulink, e os demais subsistemas cujo comportamento discreto é emulado pela IHM, tal garantia, ainda que mostrada pelas simulações, não é comprovada formalmente, via *model checking* ou outro método de verificação formal (CLARKE, GRUMBERG e PELED, 1999). Deste modo, a análise formal do SCM é realizada no nível SM e SP e garantida pela arquitetura de implementação do CSML. Contudo, para o sistema completo ou toda a estrutura de software essa análise é realizada de modo qualitativo através de simulações e não por um método formal.

Ainda que o veículo contenha comportamentos contínuos, no nível de missão a abordagem é sempre dirigida a eventos, pois ao modelar as manobras como uma sequência de ações, descrita por um autômato, foi possível evitar o uso de abordagens baseadas em Sistemas Híbridos (SH), reduzindo assim a complexidade na modelagem e implementação do SCM, e diminuindo também o custo computacional

usualmente elevados em SH. As ações consideradas deliberativas na arquitetura, ou seja, aquelas que necessitam consultar a estrutura de planejamento antes da tomada de decisão, são realizadas pelo GM, enquanto que a ação em si do CSML é reativa. O nível de SO do CSML realiza a correspondência entre as ações do nível de missão, representadas no alfabeto de eventos do CSML, também usado pelo GM e, porventura, associado a parâmetros de missão, com os comandos e eventos dos níveis inferiores da arquitetura. Assim, uma sequência de eventos no nível da missão corresponde a uma sequência de comandos e sinais nos níveis inferiores da arquitetura. O comportamento reativo de desvio de obstáculos, contudo, não necessita da ação de planejamento e tomada de decisão do CSML e do GM, garantindo segurança ante eventos não-deterministas não contemplados no nível de representação de missão. Tal procedimento é realizado devido à necessidade de reação imediata que não necessita da ação de um sistema de planejamento de alto nível. Assim, é possível considerar a arquitetura do SCM híbrida em dois sentidos: na teoria de SEDs, ao abordar a dinâmica contínua e discreta do AUV; e na arquitetura robótica, por apresentar comportamentos deliberativos e reativos em uma mesma arquitetura.

Outros trabalhos aplicam a TCS a ambientes estruturados e totalmente conhecidos, especificando todas as opções de movimento possíveis em seus modelos (MOLINA *et al.*, 2004) e, portanto, tendo aplicação limitada em ambientes não-estruturados e com movimento em 6 graus de liberdade. Contudo, a abordagem empregada neste trabalho permite separar a representação contínua do ambiente, trajetórias e obstáculos, usualmente definidas no espaço de variáveis contínuas, da evolução discreta da missão, reduzindo consideravelmente a complexidade da formulação, modelagem e análise de missões, ao mesmo tempo em que requisitos de segurança e operação são mantidos para qualquer tipo de missão. Tal característica permite o processamento relativamente rápido dos algoritmos de planejamento justamente pelo fato dos mesmos atuarem sobre uma representação discreta, no caso, baseada em tarefas e fases.

Ainda que vários trabalhos que empregam a TCS para o problema de missões de veículos autônomos restrinjam a realização da missão a um único caminho de eventos, como em Xu, Zhang e Feng (2004a, 2004b), nesta tese, o uso da TCS em controle de missão explora o fato do processo de síntese dos supervisores ser minimamente restritivo ou ótimo, ou seja, garante que vários são os caminhos habilitados pelos supervisores e, portanto, habilita várias opções para cumprimento da missão. Além disso, esta tese realizou a aplicação da TCS ao problema

de controle de missões de AUVs de modo sistemático, diferente dos demais trabalhos onde uma abordagem mais experimental, inicial e sem um método definido para aplicação da TCS é encontrada. O mecanismo de escolha dos caminhos habilitados pelo CSML emprega os recursos de replanejamento, modos de operação e política de prioridades permitindo justamente explorar esses vários caminhos, o que evita que a missão possua somente uma única opção viável de sequência de eventos. Tal característica confere flexibilidade no momento de escolher qual a melhor opção para realização da missão, além de possibilitar o emprego de métodos de otimização para escolha da melhor opção.

Outra questão relacionada aos modelos formais, consiste na modificação do comportamento discreto ao introduzir uma política de prioridades implementada por um mecanismo de prioridades dinâmicas na escolha dos eventos controláveis pelo GM. Como a escolha das prioridades é implementada de modo heurístico, ou seja, a partir da definição de quatro níveis de prioridades e pela atribuição das mesmas aos eventos conforme um dos quatro modos de realização das missões (normal, correção por GPS e dois modos de cancelamento da missão), não foi empregado nenhum formalismo para validar o comportamento dos modelos baseados em autômatos mais a ação do GM. Um exemplo de uma possível abordagem para o projeto conjunto da estrutura de controle supervisório e do GM é apresentada em Chung, Lafortune e Lin (1992). Nesse artigo, os autores propõem a aplicação de uma política para seleção de eventos baseada na antecipação (*lookahead*) do comportamento da planta ao invés de encontrar uma estrutura completa de supervisão que contemple todas as possíveis sequências de eventos, ao escolher um evento a partir da projeção de *n* passos a frente do comportamento da planta representada por uma estrutura em árvore. Contudo, por questões de limitação de tempo, tal estudo não foi realizado.

### 7.2.2. Aspectos de Natureza Tempo-Real

Atualmente o *middleware* ROS provê poucas ferramentas ou mecanismos para tratar com questões relacionadas ao processamento tempo-real, mesmo quando em execução em um sistema operacional tempo-real (*real-time operating system* ou RTOS). Assim mesmo, o SCM e os múltiplos processos em execução no ROS mostraram, nas simulações, desempenho razoável por apresentar políticas ou mecanismos de programação que permitem forçar a execução dos laços

principais dos programas dentro de certas margens de tempo, ainda que não garantidos pelo sistema operacional.

Observa-se também pelas simulações, que os tempos de resposta do SCM são compatíveis com os tempos de resposta desejada para o veículo. Enquanto que os períodos de amostragem usados para a publicação dos dados provenientes dos modelos dinâmicos sejam da ordem de 10 ms (100 Hz), o período de amostragem do SCM é da ordem de 100 ms (10 Hz). Usualmente empregam-se valores ainda menores para as malhas de controle (1, 5, 10 ou 20 ms) e que são implementadas em processadores mais rápidos, separados do SCM, devido à necessidade de garantia de tempos de processamento (tempo-real). Os processos tempo-real mais críticos são aqueles realizados pelos controladores em malha-fechada, simulados pelo software Matlab/Simulink, e pelos algoritmos de predição e detecção de faltas (não implementados neste trabalho e abstraídos pelos eventos não-controláveis de falhas gerados pela IHM), enquanto que as tarefas realizadas pelo SCM necessitam de tempos de resposta maiores. Além disso, o veículo atua com velocidades abaixo de 1 m/s, devido às características geométricas tornando-o mais adequado para realização de atividades de inspeção e aquisição de dados à baixa velocidade, porém com alto grau de manobrabilidade, e, portanto, exigindo tempos de respostas relativamente mais lentos que os veículos em formato torpedo, que funcionam a velocidades superiores.

Conceitualmente, a arquitetura do SCM pode ser implementada em tempo-real, pois uma vez traduzida para o sistema embarcado do veículo, o número de operações é fixo, sendo sempre possível, portanto, calcular o *worst case execution time* (WCET), que representa o tempo máximo de execução de uma tarefa para a plataforma de hardware considerada (LIU, 2000). Pela análise das simulações e pelas informações contidas nos arquivos de registros (*log*s) é possível deduzir que os tempos de cálculo da estrutura do CSML é inferior aos 100 ms do SCM. Além disso, devido ao mecanismo de interrupção do código principal via *callbacks*, os eventos não-controláveis são tratados no momento em que chegam à fila de eventos, ou seja, a chegada de dois ou mais eventos não-controláveis pode ser feita em uma taxa inferior ao período de 100 ms do processo principal, no caso, o SCM, conforme mostrado no algoritmo básico de implementação do SCM na seção 6.3.1.

As ações mais lentas do SCM correspondem à fase de planejamento e replanejamento. O planejamento é realizado antes do veículo iniciar a missão (*off-line*) e, portanto, não apresenta restrições

quanto a sua execução. O algoritmo de replanejamento, contudo, para missões da ordem de 5 a 10 fases, com 3 tarefas em média, é executada em paralelo com a fase em curso, ou seja, *on-line*, e seu tempo de execução é da ordem de 1 a 2 segundos. Além disso, como o replanejamento é executado paralelamente à tarefa em realização pelo SCM, tal atividade não interfere diretamente na execução dos demais componentes do sistema.

Desse modo, é possível deduzir que a estrutura do SCM pode ser implementada em um sistema tempo-real, uma vez que os tempos envolvidos no nível de controle de missão são muito menores se comparados aos tempos associados às malhas-fechadas de controle de movimento e ao processamento sensorial, e pelo fato do *wcet* de diversas funções e processos em execução no ROS serem conhecidos e bem definidos (frequências). Além disso, os comportamentos reativos não dependem da ação do CSML e do GM e ocorrem sem a necessidade dos algoritmos de planejamento ou replanejamento. Portanto, a tomada de decisão realizada pelo SCM pode ser implementada em um sistema tempo-real. Contudo, somente através de testes em uma plataforma de simulação tempo-real ao modo do HIL ou na implementação real do veículo será possível validar com segurança se os tempos de respostas do SCM são compatíveis com processos tempo-real mais rápidos.

### 7.2.3. Comunicação

Outro aspecto que limita a garantia de tempos de respostas no nível dos processos em execução no ROS é o tipo de comunicação Ethernet ou Wi-fi, com protocolo TCP/IP, o que gera um *overhead* na troca de mensagens entre os processos, ainda que tais canais de comunicação sejam dedicados ou exclusivos para o SCM e o ambiente de simulação. O fato da rede de comunicação ser empregada para trocar dados entre os processos com período de execução relativamente grande, como o SCM, é outro fator que favorece a adoção do Ethernet ou Wi-fi como tecnologia de acesso ao meio, e o uso de *sockets* TCP/IP, com mensagens do tipo TCP, como protocolo de comunicação. Porém, esse fator é limitante na implementação realizada, mas não na arquitetura para o SCM proposta neste trabalho.

Assim mesmo, para evitar perda de dados devido às diferenças dos tempos de execução dos processos no ROS, e as diferenças de velocidades devido ao tráfego de dados na rede de comunicação, cada canal de comunicação possui uma fila de dados de entrada, limitada a

100 ou 1000 mensagens. Sempre que o canal fica cheio, as mensagens antigas são descartadas. Por sua vez, os processos clientes que consomem os dados dos canais de comunicação, também possuem filas para transferência dos dados. Tais mecanismos são transparentes ao programador, e são disponibilizados pelas bibliotecas de programação do ROS.

Para os processos mais críticos, como as malhas de controle ou o processamento sensorial, protocolos tempo-real são fundamentais para a garantia do desempenho do sistema, ainda que protocolos mais simples, como o CAN (*controller area network*), podem ser usados para diminuir a quantidade de *overhead* gerada para o tráfego de dados em algumas partes do sistema. Para o SCM proposto, a adoção de novos protocolos impactaria, em princípio, somente nos processos do ROS conectados à rede de comunicação Ethernet ou Wi-fi, porém, entre estes processos os mecanismos de comunicação seriam os atualmente empregados.

### 7.2.4. Modelagem Dinâmica e Sistema de Controle

O presente trabalho não tem como objetivo analisar o modelo dinâmico e o desempenho do sistema de controle de malha-fechada, dois dos problemas fundamentais da robótica móvel. Assim, foram desconsideradas no modelo dinâmico questões importantes, como presença de correntes, variações e incertezas paramétricas, que influenciam consideravelmente o desempenho dos controladores de movimento. A limitação na velocidade do veículo, cerca de 1 m/s nos eixos $<x, y, z>$ existe devido à própria limitação nos propulsores do veículo. O sistema de controle utilizado apenas para testar o SCM é simples, não empregando geração de trajetórias de referência suaves e tampouco resolvendo o problema da influência do erro da ação integral (GOMES e BIER, 1998). Tais características exigem um sistema de controle mais sofisticado ao considerado nesse trabalho, o que gerou a necessidade de limitar os deslocamentos do veículo para cerca de 5 m. Assim, trajetórias com deslocamentos superiores a 5 m, foram divididas em segmentos menores para contornar o problema de realização da trajetória.

Além disso, o trabalho empregou como método de guiagem o uso de referências de coordenadas finais desejadas, denominadas de *waypoints*, adequadas para veículos com as características do NEROV sob ação de controladores relativamente simples e para as condições de missão consideradas para esse trabalho. Contudo, em veículos do tipo

torpedo, por exemplo, a estrutura de navegação é diferente, exigindo a geração de trajetórias, parametrizadas ou não no tempo, mas que levem em consideração as restrições não-holonômicas do veículo e as estratégias de controle consideradas. Por exemplo, em Engelhardtsen (2007) apresenta-se uma estratégia de navegação desacoplada para um AUV tipo torpedo, com malhas de controle separados para controle do ângulo em torno ao eixo *z* ou $\psi$ (*heading control*), controle do ângulo em torno ao eixo *y* ou $\theta$ (*roll control*) e controle de velocidade.

Assim, neste trabalho, por simplicidade, foi considerado uma estrutura de controle linearizante por realimentação com erro de seguimento de trajetória definido por controladores do tipo PID e desacoplados para cada um dos graus de liberdade *x*, *y*, *z*, $\phi$, $\theta$ e $\psi$. Além disso, desconsiderou-se, também por simplicidade, uma malha de controle de velocidade, assumindo que o veículo se desloca a velocidade constante, conforme o sistema de propulsão disponível. Tais simplificações implicaram em trajetórias com custo superior ao custo estimado, mas foram plenamente atendidas pelo SCM.

Contudo, a modificação no sistema de controle somente interfere no SCM caso haja necessidade de modificar os parâmetros associados aos eventos de início de manobra, empregando outros valores de referência ou ainda, um sistema de geração de trajetórias parametrizadas no tempo e / ou que levem em consideração as restrições não-holonômicas. Mas mesmo sob essas condições, a estrutura essencial do SCM, com a escolha de eventos, a tradução da missão, o atendimento de especificações, os modelos formais do CSML, o planejamento e o replanejamento podem ser mantidos.

### 7.2.5. Análise Qualitativa do SCM Implementado

Como as demais arquiteturas de SCM baseadas em SEDs apresentadas no capítulo 3, a presente arquitetura também emprega a organização lógica do sistema em níveis, possibilitando separar a dinâmica contínua da dirigida a eventos. Contudo, o trabalho emprega uma arquitetura robótica híbrida, e não hierárquica, como a maioria dos trabalho, onde os comportamentos reativos, como o desvio de obstáculo, não dependem diretamente da capacidade de planejamento e representação da tomada de decisão, aumentando a segurança do AUV ao realizar a missão em ambientes não-estruturados e parcialmente desconhecidos.

A adoção de um *framework* baseado em SEDs para modelar e executar missões torna o SCM relativamente independente quanto aos componentes dos níveis inferiores da arquitetura, podendo, em tese, ser aplicada em diversos tipos de veículos. No nível lógico do SP e SM do CSML, e no GM, a missão é representada pelos modelos abstratos descritos por um alfabeto de eventos que é posteriormente traduzido para sinais do veículo real pelo nível SO. Além disso, diferente da maioria absoluta das arquiteturas analisadas na revisão bibliográfica, com exceção dos trabalhos de Palomeras *et al.* (2006) e Palomeras *et al.* (2009), o SCM proposto emprega diretamente os modelos teóricos para representação da missão como estruturas para controle e supervisão da missão em si, através da geração automática de código. Portanto, o nível SO é o que apresenta o maior nível de dependência dos demais componentes do veículo e, por esse motivo, é implementado manualmente através de programação C/C++.

O vocabulário de ações representado no plano de missão inclui o movimento em si do AUV, mas também as ações de ligar ou desligar sensores, além da ocorrência não-determinista de eventos de erros, possibilitando planejar a correção de missões ou exclusão de parte da missão sem necessariamente implicar no cancelamento completo da missão. Além disso, a opção de replanejamento permite que o veículo se adapte a variações no consumo de bateria, ocasionado pelas variações no movimento devido à ação de correntes por exemplo, ou a presença de eventos não-deterministas, evitando também que missões sejam canceladas completamente ou mesmo que o veículo não tenha energia suficiente para retornar ao ponto de recuperação.

O emprego de um algoritmo de busca em largura permite realizar o planejamento e replanejamento de missões otimizando critérios de consumo e/ou deslocamento entre fases de missão, escolhendo a melhor sequência realizável de fases. Contudo, tal otimização não é empregada para a geração das trajetórias de referência dos controladores de movimento, assumindo o movimento baseado na geração de sequências de pontos (*waypoints*). Devido ao algoritmo ser aplicado para a escolha de fases e tarefas da missão e não para o planejamento de trajetórias, o processamento exigido para o mesmo é relativamente pequeno, o que possibilita a sua execução em poucos segundos, sem comprometer a ação dos demais módulos ou processos do sistema. Além disso, o planejamento é realizado no nível das fases e não na sequência de eventos ou nas trajetórias, reduzindo assim a complexidade do algoritmo de planejamento e replanejamento.

Com respeito ao ambiente de atuação do AUV, este não é considerado fixo, representado por uma estrutura de transição, ou mesmo estruturado, com objetos ou geometria descritos por ângulos retos e bem conhecidos, devido às características inerentes encontradas em lagos de barragens de hidrelétricas. Por esse motivo, considera-se a presença de obstáculos de tamanho qualquer. O SCM também não emprega um mapa para planejamento de trajetórias, pois a não existência de um mapa conhecido ou mesmo que incorpore as variações existentes nesse tipo de ambiente, levaram a adoção de uma estratégia coordenada de navegação baseada em um componente deliberativo, através da geração de pontos finais desejados (*waypoints*) com um componente reativo para o desvio de obstáculos. Contudo, o sistema de desvio de obstáculos mantém um mapa relativo com os obstáculos detectados pelo veículo. O estado do comportamento reativo de desvio de obstáculo é também empregado pelo planejamento no sentido que colisões geram o cancelamento completo da missão, com o objetivo de evitar danos ao AUV.

A representação, no plano de missão, da dinâmica contínua do AUV é realizada mediante o modelo de manobras, apresentado na seção 4.3. Desse modo, é possível empregar o mecanismo de geração de pontos desejados para o movimento do veículo (*waypoints*), sem necessariamente representar a dinâmica contínua explicitamente no plano de missão, diminuindo a complexidade na modelagem e implementação do SCM, e permitindo manter o foco do trabalho na análise do uso do CSML ao problema de missão de AUVs e no desenvolvimento do SCM.

Outro aspecto do SCM desenvolvido consiste no fato do mesmo ter sido implementado em linguagem C/C++ com uso de programação concorrente (*threads*) para evitar bloqueios ou tempos de respostas elevados. O uso do *middleware* ROS para a construção do SCM possibilitou o foco no desenvolvimento em si do SCM, e não nas implementações das estruturas de comunicação, sincronização de processos, garantia relativa de tempos de execução, desenvolvimento de ferramentas de visualização, mapeamento e detecção de obstáculos, já incorporados e disponíveis na ferramenta. Contudo, a implementação do SCM, por estar baseada no ROS, exige que este também esteja presente no sistema embarcado veículo, condição essa possível devido às versões disponíveis existentes para alguns sistemas operacionais para dispositivos embarcados.

Assim, é possível resumir as principais vantagens do SCM desenvolvido como sendo as seguintes:

- Emprego de um formalismo para representação e modelagem de missões, conferindo propriedades desejáveis para o sistema.
- Uso dos modelos formais para execução da missão diretamente no sistema embarcado do veículo, através da geração automática de código.
- A abordagem modular do CSML permite a reutilização dos modelos formais em outras aplicações devido à própria modularidade, ao nível de abstração destes modelos e à relativa independência da modelagem com respeito a implementação do sistema.
- Modelagem independente das características do ambiente não-estruturado onde o AUV atua, considerando presença de obstáculos e ocorrência não-determinista de eventos de erros diretamente no nível de representação e execução da missão.
- Relativa independência do tipo de missão, sendo possível representar e executar missões com vários tipos de fases e tarefas, em regiões diversas.
- Execução segura de missões, com modos de operação de cancelamento de fases ou de missões, e correção automática por GPS.
- Capacidade de replanejamento de missão de modo a permitir ajustar o número de atividades a ser realizada pelo robô com a quantidade disponível de bateria.
- Relativa independência dos componentes do sistema embarcado do AUV ao empregar o nível de SO para traduzir sinais e comandos do AUV em eventos controláveis e não-controláveis.
- Uso de uma arquitetura robótica híbrida deliberativo-reativa, onde os componentes deliberativos de planejamento são responsáveis pela execução segura da missão conforme um arquivo de missão, e comportamentos reativos do CSML, para garantia de especificações, e do desvio de obstáculos ocorrem sem a necessidade de planejamento.
- Arquitetura orientada à implementação, onde o SCM corresponde ao sistema a ser embarcado diretamente no veículo, cuja validação, entretanto, foi realizada mediante simulação dos demais componentes do AUV.

- SCM implementado no ROS, *middleware* de código aberto com amplo uso acadêmico para desenvolvimento de protótipos na área de robótica.

Contudo, a implementação do SCM realizado neste trabalho apresenta diversas limitações e desvantagens, sejam oriundas do formalismo de TCS empregado, do uso do ROS como ferramenta de desenvolvimento e *middleware* de execução do sistema, ou pela própria ausência de um veículo real que permitisse testes relacionados ao desempenho dos componentes de software e hardware. Assim, algumas das principais limitações do SCM proposto são:

- Curva de aprendizagem relativamente lenta para a aprendizagem de métodos formais, o que pode gerar uma certa resistência aos desenvolvedores à adoção de formalismos para a resolução de problemas da área de sistemas embarcados e robótica em geral.
- Representação discreta da missão somente permite otimizar a sequência de fases da missão, contudo, não permite a adoção de critérios para otimização de trajetórias, o que exigiria, a princípio, algum método de representação contínua do movimento no nível da missão, sendo esse tipo de estratégia mais de acordo com abordagens da área de sistemas e controle híbridos.
- Também decorrente da representação discreta da missão, para casos em que o veículo apresente restrições quanto a sua manobrabilidade, restrições não-holonômicas ou mesmo número de propulsores inferior ao número de graus de liberdade que se deseja controlar (sistema subatuado), exigindo também a incorporação de tais características dinâmicas no nível da missão e, portanto, modificação do SCM proposto.
- Limitações inerentes ao ROS, como ausência de tratamento de aspectos tempo-real do sistema, dependência dos mecanismos e protocolos de comunicação disponíveis ou ainda, a imposição do sistema operacional que seja capaz de executar esse *middleware*. Contudo, tal limitação refere-se à implementação desenvolvida para este trabalho e não ao SCM proposto.
- Ausência de um veículo real para implementação e testes do SCM proposto.

## 7.3. RESUMO

Esse capítulo apresentou as simulações de diversos cenários de missões em que o AUV é comando pelo SCM, e com os componentes contínuos e discretos sendo simulados, respectivamente, pelo software Matlab/Simulink e por uma IHM desenvolvida em Java.

A ação do SCM corresponde às necessidades exigidas pelo AUV durante a realização simulada das missões, como tempo de respostas apropriados, capacidade de planejamento e replanejamento para adequação do plano de missão às ocorrências não-deterministas encontradas em ambientes não-estruturados, como obstáculos, correntes, erros em equipamento, entre outros. Contudo, a não implementação em um veículo real e não garantia de execução dos tempos de execução (tempo-real) constituem-se nas maiores desvantagens do SCM proposto, deixados como perspectiva de continuidade do trabalho.

# 8. CONCLUSÃO E PERSPECTIVAS

Esta tese apresentou uma abordagem para o desenvolvimento de uma arquitetura para Sistemas de Controle de Missão (SCM) baseado no Controle Supervisório Modular Local (CSML), permitindo a representação, planejamento, replanejamento e execução de missões de veículos subaquáticos autônomos (AUVs) atuando em ambientes não-estruturados. O SCM proposto nesta tese está constituído de dois componentes principais, a arquitetura de implementação do CSML e o Gerenciador de Missão (GM), cujas ações complementares garantem que funcionalidades de reação e garantia de especificações formais atuem em conjunto com operações de planejamento e replanejamento de missão.

Através da Teoria de Controle Supervisório (TCS), mais precisamente o CSML, foi possível definir uma lógica de controle para representação e execução de missões. O formalismo do CSML permitiu derivar modelos formais para representação, modelagem e análise da missão de AUVs levando em consideração a dinâmica discreta ou dirigida a eventos dos diversos componentes de hardware e software do AUV. Essa abordagem possibilitou que especificações de operação segura do veículo fossem garantidas para qualquer missão e durante toda a sua realização. O resultado final da modelagem discreta dos componentes do AUV e da síntese de supervisores, que garantem especificações de operação e segurança, consiste na especificação de uma lógica de controle para representação e execução de missões, com propriedades formais bem definidas, como ausência de bloqueios e atendimento por projeto de especificações.

A natureza modular do sistema e das especificações favoreceu a adoção do CSML, permitindo a redução da complexidade na análise e implementação de uma lógica de controle não-bloqueante e minimamente restritiva para a realização de missões de AUVs. Especificações foram empregadas para delimitar a evolução discreta dos componentes do AUV (planta) àquelas condições consideradas desejáveis e seguras, garantindo o não-bloqueio do sistema. A modelagem da missão baseada em CSML empregada neste trabalho não depende do ambiente e da missão em si, permitindo a descrição de vários tipos de missões, além de permitir incorporar com relativa

facilidade outros tipos de manobras, comportamentos, modelos ou especificações. A utilização de modelos modulares para o sistema e especificações favorece também a reutilização ou adaptação destes modelos formais para outros tipos de arquiteturas de sistemas embarcados ou mesmo outros tipos de veículos robóticos autônomos.

A ação ótima ou minimamente restritiva do CSML possibilita que somente as sequências de eventos consideradas indesejáveis sejam desabilitadas pelos supervisores modulares, permitindo que vários sejam os caminhos que conduzem o veículo à finalização da missão. Tal característica faculta o emprego de algoritmos de planejamento de tarefas no nível de representação discreta das missões, uma vez que as missões podem ser realizadas através de diversas sequências de eventos e não somente por meio de uma única sequência. Particularmente, este trabalho empregou como critério de otimização de missões, o nível de bateria disponível, permitindo escolher a melhor sequência de fases, constituídas por sequências de eventos que permitiam finalizar uma missão.

Por meio de uma abordagem baseada em Sistemas a Eventos Discretos (SEDs) em missões de AUV também foi possível separar a representação contínua do ambiente e sistema (movimento, trajetórias, obstáculos, localização) da representação discreta da missão, reduzindo consideravelmente a complexidade na análise do sistema, na síntese da lógica de controle e na aplicação dos algoritmos de busca empregados no planejamento e replanejamento. Tal separação também favoreceu a definição de uma semântica e sintaxe para o arquivo de missão mais independente dos aspectos de implementação do AUV e mais próximos dos conhecimento específico dos profissionais que usam os robôs como plataforma de pesquisa ou para aquisição de dados. Ainda que o planejamento e replanejamento da missão sejam realizados no domínio discreto, o que implica em um custo computacional relativamente pequeno, essa separação não considera a dinâmica contínua do veículo ou do ambiente. Porém, esta característica pode representar uma limitação em aplicações em que se deseja empregar critérios de otimização nos algoritmos de planejamento explicitamente baseados nas dinâmicas contínuas, como por exemplo, para a geração de trajetórias de referência para o movimento do AUV.

Por meio da arquitetura de implementação do CSML, a lógica de controle usada para a representação de missões, com todos os modelos formais relacionados (autômatos modulares e supervisores) foi implementada diretamente no SCM proposto. Essa arquitetura de implementação aliada à geração automática do código dos níveis

Sistema-Produto (SP) e Supervisores Modulares (SM) do CSML garante que a lógica de controle desenvolvida para o sistema embarcado seja a mesma empregada na etapa de análise do sistema, diminuindo também o número de erros devido à codificação manual das estruturas do CSML.

Complementando a ação do CSML, o segundo componente do SCM, o GM, implementa as ações de planejamento, replanejamento e escolha de eventos, ações que não são de responsabilidade do CSML. Para evitar possíveis bloqueios na realização de missão, o GM emprega políticas de atribuição de prioridades e seleção de eventos, possibilitando que a missão seja realizada pela escolha de eventos controláveis ou pela espera de eventos não-controláveis. Ao atribuir prioridades aos eventos, o GM impede que a missão seja executada exclusivamente através da escolha ou espera de um único evento, o que eventualmente poderia gerar bloqueios na missão. Com a definição de uma hierarquia de prioridades, na pior situação, todos os eventos possuem a mesma prioridade, no caso, a mais baixa possível, mas mesmo assim sempre existirão eventos controláveis a serem escolhidos ou não-controláveis a espera de serem gerados. Contudo, a influência ou interferência das prioridades sobre as propriedades lógicas do CSML é um aspecto deixado em aberto neste trabalho.

Os algoritmos de planejamento e replanejamento são realizados diretamente sobre o vocabulário semântico empregado no nível mais alto da arquitetura, ou seja, aquele utilizado para descrever as missões com base nos modelos do CSML. Assim, o GM complementa a ação do CSML, ao ser empregado como elemento para escolha da melhor ordem de fases para realização da missão. Os mecanismos de planejamento, replanejamento, bem como os modos de operação, com sequências específicas para tratamento de problemas como cancelamento de missões ou correções de erros, evitam que o usuário tenha que especificar, no arquivo de missão, todas as ações a serem tomadas pelo AUV reduzindo consideravelmente a complexidade na especificação e execução das missões.

Consoante com abordagens e metodologias da área de Sistemas Embarcados, o desenvolvimento do SCM foi realizado por meio da adoção de diversos níveis de abstrações favorecendo o entendimento dos vários tipos de problemas do sistema de controle do veículo, permitindo a integração dos componentes característicos de múltiplos domínios (navegação, controle, planejamento, etc.). A organização da arquitetura do sistema em camadas combinada com a abordagem de projeto (modelagem, definição de especificações, síntese e testes) do CSML, que aproveita a natureza modular dos componentes e especificações do

sistema, permite definir uma especificação para representação lógica de missões com relativa independência dos aspectos de baixo nível da arquitetura, composto por vários elementos de hardware e software, distribuídos em um sistema embarcado tempo-real. Estas características propiciaram a definição de um método para o projeto e implementação do SCM proposto, porém, por ter como objetivo somente o SCM, tal método necessita ser incorporada ao desenvolvimento completo do hardware e software do AUV e, portanto, integrada a metodologias já estabelecidas na área de sistemas embarcados. Contudo, tal integração não foi realizada nesse trabalho, ficando como proposta de continuidade da pesquisa.

A adoção do ROS (*robot operating system*) como ambiente de desenvolvimento do SCM permitiu o uso das diversas ferramentas, bibliotecas de programação, mecanismos de comunicação já criadas para aplicações robóticas, possibilitando o foco na implementação do SCM em si, da estrutura do CSML e do GM, além de servir também de plataforma para execução do sistema proposto. Apesar do SCM apresentar um número fixo de operações, o que permite obter um valor para o seu *wcet* (*worst case execution time*), os tempos da ordem de segundos do planejamento / replanejamento, a limitação do ROS em não tratar aspectos relacionados à natureza tempo-real das aplicações e a falta de suporte para comunicação tempo-real dos padrões Ethernet (IEEE 802.3) e WiFi (IEEE 802.11) não permitem a garantia na execução tempo-real das tarefas (*threads*) que compõem o SCM. Assim mesmo, através de mecanismos de filas de mensagens, do uso de diretivas de execução do ROS, do emprego de chamadas assíncronas (*callbacks*) de funções, do número fixo de operações do SCM foi possível manter os tempos de execução do SCM na ordem de 100 ms, tempos estes superiores aos valores empregados nos algoritmos de controle e outros subsistemas críticos. Portanto é possível deduzir que a estrutura do SCM pode ser implementada em um sistema tempo-real, porém tal comprovação requer a implementação em um veículo real, ou ainda simulação em uma plataforma do tipo *Hardware-in-The-Loop* (HIL), não realizada neste trabalho. Tais limitações dizem respeito às ferramentas e tecnologias usadas para a implementação. Apesar de que a arquitetura proposta para o SCM independe das mesmas, estudos relacionados à aplicação de outras tecnologias e ferramentas para o desenvolvimento do SCM necessitam ser considerados, principalmente para avaliar as limitações tempo-real das tecnologias usadas na implementação realizada nesta tese.

Finalmente a implementação do ambiente de simulação, através da integração dos modelos contínuos do movimento e do controle realizados pelo software Matlab/Simulink, dos componentes discretos emulados pela interface homem-máquina (IHM) como erros em sensores, falhas gerais, entre outros, com o SCM possibilitou avaliar qualitativamente as características, vantagens e limitações da arquitetura proposta, para diversos cenários de missões em ambientes não-estruturados. As simulações demonstraram que os aspectos de operação segura do veículo, definido pelas especificações do CSML, foram atendidos pelo SCM proposto, ao mesmo tempo em que o SCM apresentou flexibilidade na realização de missões ao permitir a escolha de caminhos alternativos àquele especificado no plano de missão original Contudo, pelo fato de não se dispor de um AUV real para implementação, a validação completa do sistema – SCM em funcionamento com os demais subsistemas de um AUV – não foi realizada, constituindo-se em uma tarefa a ser realizada para continuidade da pesquisa iniciada nesta tese.

## 8.1. PRINCIPAIS CONTRIBUIÇÕES

Como resumo, são apresentadas a seguir as principais contribuições obtidas durante o desenvolvimento desse trabalho.

- Proposição de uma arquitetura para SCM baseada na Teoria de Controle Supervisório.
- Desenvolvimento de um Gerenciador de Missão com responsabilidades de planejamento e replanejamento.
- Sistematização da aplicação da TCS, mais precisamente o Controle Supervisório Modular Local ao problema de controle de missões de AUVs, representando a evolução da missão com base nos modelos discretos (autômatos) dos diversos componentes do veículo e na síntese de supervisores para especificação de uma lógica para o funcionamento seguro do veículo (capítulo 4).
- Desenvolvimento, implementação e testes / simulação de um Sistema de Controle de Missão (SCM) baseado no CSML para um AUV (capítulos 5, 6 e 7) visando à validação da arquitetura proposta.
- Proposição de um método de desenvolvimento de SCM para AUVs (capítulo 5).

- Síntese das principais funcionalidades, subsistemas, componentes e problemas encontrados na área de robótica móvel, em particular a subaquática (capítulo 2).
- Levantamento sistemático das principais aplicações de métodos e técnicas da área de Sistemas a Eventos Discretos (SEDs) em robótica móvel em geral (capítulo 3).
- Integração de algoritmos de busca e planejamento da área de Inteligência Artificial (IA) com o SCM baseado em CSML (capítulos 5 e 6).
- Construção de um ambiente de simulação integrando ao SCM a dinâmica contínua do movimento e sistema de controle de um AUV realizado em Matlab/Simulink e a dinâmica discreta simulada por uma interface homem-máquina (IHM) implementada na linguagem Java (capítulo 6).

As contribuições anteriormente comentadas foram apresentadas à comunidade científica através dos seguintes trabalhos:

- BATTISTELLA, S.; QUEIROZ, M. H.; SANTOS, C. H. F. *Modelagem e Síntese de Supervisores para Controle de Missão de AUVs atuando em Lagos de Barragens de Hidrelétricas baseado na Teoria de Controle Supervisório.* Anais do XIX Congresso Brasileiro de Automática, CBA. p. 1870-1877. 2012.
- BATTISTELLA, S.; QUEIROZ, M. H. *Arquitetura e Ambiente de Simulação para Sistema de Missão de AUVs baseado em Controle Supervisório.* Anais do XX Congresso Brasileiro de Automática, CBA. p. 4044-4051. 2014.
- BATTISTELLA, S.; QUEIROZ, M. H. *Simulation Environment of an Architecture for Mission Control System of AUVs operating in Lakes of Hydroelectric Dams.* Proceedings of the III International Conference on Applied Robotics for the Power Industry, CARPI. p. 1-6. 2014.
- BATTISTELLA, S.; QUEIROZ, M. H. *Development of Mission Control Systems for Autonomous Underwater Vehicles based on Supervisory Control Theory*. IEEE Transactions on Robotics (em fase de submissão).

## 8.2. PERSPECTIVAS

A maior limitação encontrada no desenvolvimento desse trabalho constituiu-se na ausência de um veículo subaquático que permitisse a implementação do SCM proposto. A implementação do SCM em um veículo real permitiria validar a arquitetura proposta, apontando também possíveis falhas e limitações a serem corrigidas. Portanto, a primeira proposta para continuidade deste trabalho reside justamente na implementação do SCM em um veículo real, de modo a comprovar as vantagens e limitações do sistema proposto.

A integração entre o CSML e o GM, sobretudo os mecanismos heurísticos de atribuição de prioridades e a modificação dos modos de operação propostos na seção 5.4, ainda que tenham se mostrados válidos e corretos nas diversas simulações apresentadas, necessitam de uma análise formal mais criteriosa a respeito da influência de tais mecanismos sobre as propriedades dos modelos e supervisores do CSML. Nesse sentido, o desenvolvimento de um GM que explore formalmente as propriedades do CSML, permitindo também o estabelecimento de um método matemático para o projeto do GM torna-se desejável.

Uma estratégia possível para análise e solução desse tipo de questão, e que permitiria incluir diretamente no nível de representação da missão, os aspectos dinâmicos do movimento do AUV, consiste na adoção de abordagens baseadas em controle híbrido. Porém, o custo computacional das operações associadas à implementação das estratégias híbridas de controle, entre eles o controle híbrido modular (LEAL, 2005), pode introduzir complexidades que inviabilizam a sua utilização em sistemas complexos como os de AUVs. Além disso, ao incorporar a dinâmica contínua da operação de AUVs em ambientes não-estruturados, com espaço de manobras em três dimensões, necessidade de geração de trajetórias e presença de obstáculos, correntes e outros eventos não-deterministas (falhas, limitação de potência disponível), o cômputo de soluções ótimas para o planejamento e replanejamento da missão em sistemas híbridos, geralmente com custo computacional elevado e que necessitaria ser realizado diversas vezes durante a execução da missão, nem sempre é viável em plataformas embarcadas com limitados recursos computacionais e níveis de bateria disponível. Porém, uma análise mais criteriosa a respeito de tais afirmações torna-se necessária, constituindo-se em uma das propostas para continuidade deste trabalho.

Outro exemplo de estratégia a ser considerada consiste no emprego de estruturas de supervisão que não se baseiem no cômputo completo e *off-line* da política de controle supervisório, mas sim em esquemas para cálculo *on-line* da ação de controle dos supervisores, como por exemplo em Chung, Lafortune e Ling (1992) que usa como base para o cálculo dos supervisores a projeção a *n*-passos adiante do comportamento do sistema. Outras abordagens que também podem ser consideradas como arcabouço teórico formal para o projeto do SCM, até como alternativa à composição do GM em conjunto com o CSML, consiste nos trabalhos que empregam critérios de otimização para cômputo dos supervisores, como na medição de desempenho de linguagens (FU, RAY e LAGOA, 2004; WANG, MALLAPRAGADA e RAY, 2005) ou na estipulação de funções baseadas em custos para escolha de eventos (SENGUPTA e LAFORTUNE, 1998). Contudo, a análise de métodos formais para composição do GM não foi realizada nesse trabalho, optando-se pelo emprego de uma abordagem heurística, validada pelas simulações.

Veículos subaquáticos usados em inspeção usualmente possuem maior grau de manobrabilidade (menor número de restrições não-holonômicas, número de atuadores igual ou superior aos graus de liberdade desejados para o movimento do veículo) que os veículos usados para aquisição de dados em grandes áreas, necessitando estes de estratégias de geração de trajetórias e controle de movimento mais sofisticados que os apresentados neste trabalho. Considerar restrições não-holonômicas e a parametrização no tempo para a especificação e geração *off-line* ou *on-line* de trajetórias exige uma especificação mais detalhada dos deslocamentos entre os diversos pontos da missão, além de requerer algoritmos mais precisos e sofisticados que os empregados neste trabalho. O impacto de tais modificações no SCM, seja nos modelos baseados em autômatos do sistema e na definição de novas especificações de segurança e operação do CSML ou nos algoritmos de planejamento e replanejamento do GM, em tese, podem ser realizados com o mesmo método proposto, porém com modificações em alguns módulos do SCM. Por exemplo, ao incluir as trajetórias explicitamente em um plano de missão os tempos de execução dos algoritmos de busca se tornam maiores, o que levaria a adoção de critérios não-ótimos para o planejamento mas que permitam, ao mesmo tempo, encontrar sequências de tarefas em tempos hábeis de execução durante a realização de uma missão pelo AUV. Além do fato de, em ambientes não-estruturados e com presença de ocorrências não-deterministas, o replanejamento de trajetória poder ocorrer com maior frequência. Neste

trabalho, a simplificação na representação e planejamento da missão, ao considerar a escolha da sequência de fases e deslocamentos entre pontos de início e fim de manobras, facilita o processamento do planejamento em si, contudo, perde-se em precisão na realização ótima das trajetórias.

Outro ponto pouco explorado na tese, mas de suma importância para o projeto e construção de um AUV, consiste na metodologia de projeto do sistema embarcado do veículo, ao qual o SCM proposto faz parte. Dessa forma, a adequação e integração do método proposto para o projeto do SCM com metodologias de sistemas embarcados consiste em uma interessante área para continuidade do trabalho. Por exemplo, apesar do uso do paradigma de programação orientada a objetos (POO) para a implementação de todo o SCM e do uso da ferramenta de simulação Matlab/Simmulink para o desenvolvimento do modelo dinâmico e do sistema de controle contínuo, o desenvolvimento completo do software não teve sua estrutura e funcionamento modelada através de uma linguagem de modelagem de software e / ou da área de sistemas embarcados (AADL – *architecture analysis & design language*, SysML – *systems modeling language*, UML – *unified modeling language*). Além disso, ainda que a adoção de modelos tenha sido feita somente no âmbito do CSML, ou seja, na especificação de uma lógica de representação e execução de missões, com geração automática de parte das estruturas conforme a arquitetura de implementação do CSML, o projeto integrado do SCM com o sistema embarcado, incluindo a definição de uma plataforma alvo, as características de missões e ambientes consideradas neste trabalho e a possível adoção do ROS como *middleware* para desenvolvimento da arquitetura robótica, não foram demonstrados ou validados, exigindo trabalhos nesse sentido.

Em conjunto com a implementação em um veículo real, dois pontos importantes, já comentados na seção 7.2, dizem respeito à influência da comunicação entre os componentes e os aspectos relacionados às restrições tempo-real de vários módulos do sistema embarcado do veículo. Modificações no sistema de comunicação, em tese, somente teriam impacto nas funções de comunicação do SCM com os demais componentes do AUV (sistemas de controle, navegação, localização, diagnóstico, etc.), contudo tal comprovação somente pode ser realizada em um veículo real. Do mesmo modo, devido aos tempos de resposta relativamente mais altos do SCM se comparados aos processos mais críticos, como os controladores de movimento, e a existência de ações reativas que não dependem diretamente das ações mais lentas do SCM, como o planejamento e replanejamento, também é

possível afirmar que a integração do referido sistema com uma plataforma de execução tempo-real não implicaria, em princípio, em grandes modificações. Porém, tal afirmação necessita da verificação em um veículo real.

O uso do ROS como ferramenta de desenvolvimento, plataforma de simulação, testes e execução de aplicações robóticas se mostrou efetiva e importante, permitindo o foco nos aspectos diretamente relacionados ao trabalho em si, ao possibilitar o uso de diversas bibliotecas, funcionalidades e programas para robótica móvel já disponíveis. Contudo, a utilização desse *middleware* requer que o SCM seja executado diretamente sobre o ROS, exigindo que o mesmo seja integrado ao sistema embarcado do veículo. A literatura apresenta diversos casos em que o ROS é incluído com êxito em robôs móveis, inclusive AUVs, mas considerações a respeito do tipo de plataforma alvo, potência disponível para execução de um sistema operacional com suporte à ROS exigem, em teoria, um hardware mais potente – a exceção das versões experimentais do ROS para sistemas operacionais tempo-real e direcionados a plataformas específicas, conforme brevemente citado na seção 6.1. Portanto, uma análise mais criteriosa entre o compromisso da adoção do *middleware* ROS e o sistema embarcado do AUV se faz necessária, além da avaliação da utilização de outras tecnologias, como por exemplo FGPA (*field-programmable gate array*), para a implementação do SCM proposto.

Finalmente, a adoção ou adaptação da abordagem, método, arquitetura e SCM propostos para a resolução do problema de missão em classes de veículos diferentes ao considerado neste trabalho (AUVs em ambientes não-estruturados) constitui-se em uma linha de pesquisa promissora, uma vez que vários modelos e especificações empregados pelo CSML e vários dos mecanismos que compõem o GM podem, em princípio, ser usados em veículos terrestres e aéreos. O recurso de separação da dinâmica contínua e discreta, por exemplo, incluindo a modelagem de componentes e especificações de segurança e operação do AUV, propiciada pelo CSML e utilizada para definir uma lógica de representação, funcionamento e execução de missões, podem ser utilizados com poucas modificações em veículos terrestres (UGVs) e possivelmente com maiores adaptações para veículos aéreos (UAVs) , ainda que estes últimos empreguem manobras distintas para cada uma das operações de lançamento (*launch*), deslocamento, sobrepairar (*hover*) e aterrissagem (*landing*), similares às manobras de descida, navegação, subida e retorno empregadas neste trabalho (seção 4.3). Contudo, em veículos aéreos, as diferenças no comportamento em cada

uma destas manobras podem exigir a incorporação explícita no plano de missão dos aspectos dinâmicos, como apresentado em Costa (2008) e Alves e Cunha (2010). É também possível afirmar que o SCM implementado no ROS pode ser reutilizado em outras implementações e outros veículos, por diversos motivos. Por exemplo, o SCM é relativamente independente do hardware do veículo concentrando a maior dependência com os subsistemas e equipamentos do AUV no nível de Sequências Operacionais (SO) do CSML, o que permitiria a reutilização dos modelos para outros tipos de problemas e classes de veículos. Além disso, como ROS é destinado para a resolução de problemas da robótica móvel em geral, por isso, é destinado a diversos tipos de veículos robóticos. O fato dos processos individuais que fazem parte do SCM não dependerem da estrutura de funcionamento do CSML e do GM, mas somente da troca de dados entre si, e que modificações nos protocolos de comunicação entre alguns destes processos e os diversos equipamentos do veículo, simulados neste trabalho pelo software Matlab/Simulink e pela Interface Homem-Máquina (IHM), afetarem individualmente somente os processos envolvidos na comunicação, a estrutura geral do SCM pode ser, em princípio, modificada e adaptada com poucos esforços a outros tipos de veículos. Entretanto, a comprovação da viabilidade de reutilização do SCM proposto em classes de veículos distintos ao considerado nesta tese necessita de pesquisas e desenvolvimentos futuros.

# APÊNDICE A – MODELAGEM DAS ESPECIFICAÇÕES

A seguir são apresentados os modelos das especificações remanescentes não mostrados na seção 4.4.

**Falhas de Posição e Sensores de Carga Útil**

A ocorrência de uma falha de posição indica que há um erro considerável na estimação da localização do veículo, sendo desejável suspender a manobra em curso e guiar o veículo até a superfície para correção via GPS. Durante esse trajeto, como o veículo não está mais realizando uma trajetória de aquisição de dados, deseja-se que os sensores de carga útil não sejam mais ativados, com exceção do GPS, que é necessário para a correção quando o veículo chegar à superfície. Assim, são definidas três especificações $E_{FPOS\_s}$, onde $s \in \{batimetria, câmera, CTD\}$, indicando que somente o GPS pode ser ligado, e impedindo que os demais sensores sejam ativados. O evento controlável *rst_f* indica que se as falhas forem corrigidas, o sistema pode voltar a operar normalmente. A figura A.1 apresenta o exemplo do autômato para a uma especificação $E_{FPOS\_s}$ válida para os três tipos de sensores de carga útil anteriormente comentados.

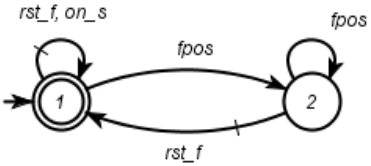


**Figura A.1:** Especificação $E_{FPOS\_s}$ de sensores e falhas de posição

**Falhas Simples e Manobras**

Para a condição de ocorrência de falhas simples define-se duas especificações $E_{F\_mi}$, onde $mi \in \{descida, navegação\}$. Essas especificações expressam o impedimento ao início ou retomada das manobras de navegação e descida, permitindo representar o cancelamento de missões e retorno ao ponto de recuperação. Assim, as manobras de retorno, mas também de subida, estarão habilitadas nesse

tipo de situação. A figura A.2 apresenta a especificação $E_{F\_mi}$ válida para as manobras de descida e navegação.

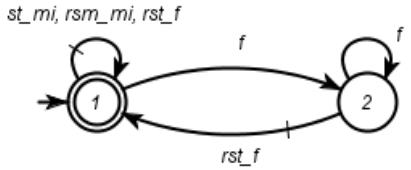


**Figura A.2:** Especificação $E_{F\_mi}$ para falhas simples e manobras de navegação e descida

**Falhas Graves e Manobras**

Para o caso de falhas graves somente a manobra de subida está permitida, indicando que o veículo nesse tipo de situação deverá emergir o mais breve possível, impedindo inclusive o retorno ao ponto de recuperação. O autômato $E_{FG\_mi}$ apresentado na figura A.3 define o modelo geral das três especificações de impedimento do início e retomada das manobras de navegação, descida e retorno, onde *mi* $\in$ *{descida, navegação, retorno}*.

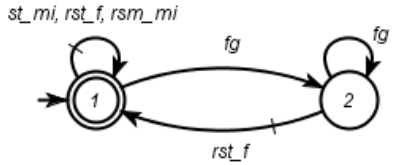


**Figura A.3:** Especificação $E_{FG\_mi}$ para falhas graves e manobras

**Falhas de Posição e Manobras**

Para efetuar a correção por GPS do erro de posicionamento (evento não-controlável *fpos*), o veículo necessita subir à superfície, realizar a calibração e retornar ao ponto em que se encontrava anteriormente e, por esse motivo, somente é permitido que as manobras de subida e descida sejam realizadas. Assim, duas especificações do tipo $E_{FPOS\_mi}$, mostrada na figura A.4, são modeladas definindo a restrição ao início e retomada das manobras de navegação e retorno, com *mi* $\in$ *{navegação, retorno}*.

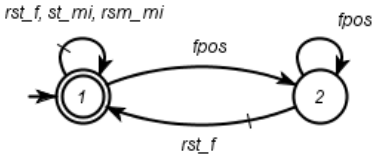


**Figura A.4:** Especificação $E_{FPOS\_mi}$ para falhas de posicionamento e manobras

**Nível Crítico de Bateria e Manobras**

De modo similar a ocorrência de falhas graves, quando o nível de bateria for muito baixo, deseja-se que o veículo rapidamente se direcione à superfície e evite que o mesmo fique sem potência. Nesse sentido, as manobras de navegação, descida e retorno são impedidas de serem iniciadas ou retomadas, condição expressa por três especificações do tipo $E_{btr\_mi}$ apresentada na figura A.5, com $mi \in \{descida, navegação, retorno\}$.

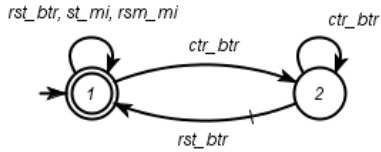


**Figura A.5:** Especificação $E_{btr\_mi}$ para nível crítico de bateria e manobras

**Estado de Submersão e Sensores de Carga Útil**

Os sensores de carga útil câmera, sonar de batimetria e CTD somente são usados durante a realização de manobras abaixo da superfície (evento não-controlável *in_h2o*). Assim, para evitar, por um erro de programação ou implementação do sistema, que os mesmos sejam ativados na superfície, são definidas três especificações do tipo $E_{sub\_s}$ apresentada na figura A.6, onde $s \in \{batimetria, câmera, CTD\}$.

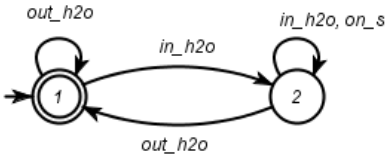


**Figura A.6:** Especificação $E_{sub\_s}$ para submersão e carga útil

Por sua vez, o GPS somente pode ser ativado quando o veículo se encontra na superfície, e o autômato para essa especificação é apresentado na figura A.7. Considera-se que inicialmente o veículo

encontra-se na superfície, e sempre que o mesmo retorne a mesma, o evento não-controlável *out_h2o* é gerado e, por tal motivo, a especificação para essa condição é diferente dos demais sensores de carga útil.

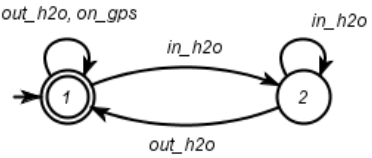


**Figura A.7:** Especificação $E_{sub\_GPS}$ para submersão e GPS

**Estado de Submersão e Manobras**

Enquanto o veículo permanecer na superfície, somente as manobras de descida e retorno estão permitidas e, portanto, não será possível iniciar ou retomar as manobras de navegação e subida. Para essa restrição, define-se a especificação $E_{sub\_mi}$ com $mi \in \{navegação, subida\}$, cujo autômato é apresentado na figura A.8.

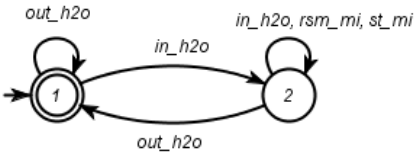


**Figura A.8:** Especificação $E_{sub\_mi}$ para submersão e manobras

De modo análogo, a manobra de descida somente estará ativa quando o veículo se encontrar na superfície, assim, o autômato da figura A.9 apresenta a especificação $E_{sub\_md}$ para essa restrição.

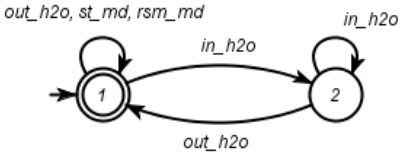


**Figura A.9:** Especificação $E_{sub\_md}$ para submersão e manobra de descida

**Colisões e Sensores de Carga Útil**

Os desvios de obstáculos são manobras realizadas pelo subsistema de navegação, mas monitoradas pelo controle de missão. Assim, sempre que o veículo detectar um obstáculo e não for capaz de desviá-lo, uma colisão se produz e a missão deve ser cancelada de modo

a recuperar o veículo para comprovar o estado do mesmo. Desse modo, ante uma colisão (evento não-controlável *er_col*) todos os quatro sensores de carga útil não poderão ser ligados. Para cada sensor uma especificação similar ao autômato $E_{col\_s}$ apresentado na figura A.10 é empregado, onde $s \in \{batimetria, câmera, CTD, GPS\}$.

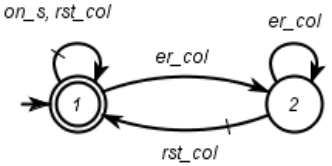


**Figura A.10:** Especificação $E_{col\_s}$ para colisões e sensores de carga útil

**Colisões e Manobras**

Para o caso de realização de manobras, a ocorrência de colisões implica também no cancelamento de missões, sendo necessário levar o veículo ao ponto de recuperação ou à superfície imediatamente acima da posição em que se encontra o AUV. Logo, as manobras de subida e retorno estão permitidas, porém as demais manobras, navegação e descida, são impedidas. Tais condições são expressas por duas especificações similares à $E_{col\_mi}$ apresentada na figura A.11, onde $mi \in \{descida, navegação\}$.

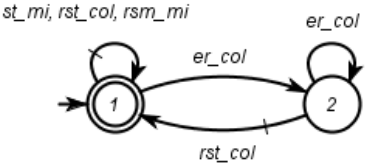


**Figura A.11:** Especificação $E_{col\_mi}$ para colisões e manobras de descida e navegação

# APÊNDICE B – PROCESSOS DO SCM

O SCM está composto por diversos processos desenvolvidos em linguagem C/C++, que são executados pelo sistema operacional Linux e gerenciados pelo *middleware* ROS. Conforme mencionado na seção 6.3, o SCM está composto por 10 processos independentes, ou seja, cada processo se constitui de um programa separado, mas em contato com os demais através de mecanismos de comunicação, mostrados no apêndice E. A comunicação de alguns processos do ROS com os componentes do ambiente de simulação é realizada por sockets TCP/IP, já comentado na seção 6.4. Os processos do SCM são comentados a seguir, e correspondem àqueles mostrados na figura 6.1 e na tabela 6.1.

**Processo I: Sistema de Controle de Missão (SCM)**

O SCM corresponde ao principal processo no nível de missão do sistema embarcado do AUV, e contém a implementação do GM e do CSML. O SCM é responsável pela tradução do arquivo de missão em um plano de missão, pelo replanejamento e também pela execução da estrutura de CSML. Esse processo recebe dados de vários processo: dados de estimação do consumo de bateria proveniente do processo *power_estimator*; o estado do AUV correspondente aos eventos não-controláveis das plantas modulares (eventos de falhas) enviado pelo processo *auv_status_broadcaster*; e do processo *controlled_event_simulator* os eventos controláveis para simulação do sistema sem o modo autônomo. Por sua vez, o SCM envia para o processo *scm_status_broadcaster* os estados dos modelos baseados em autômato e as coordenadas atuais do veículo para visualização na IHM. Empregando o mecanismo do tipo cliente / servidor com possibilidade de preempção de execução de tarefas, o SCM também envia para o processo *navigator* os dados para realização de manobras com a referência de posição e orientação do próximo ponto desejado, além do *timeout* para realização da manobra e da margem de erro.

**Processo II: Aquisição de Posição e Orientação (*sensor_imu_broadcaster*)**

Este processo recebe, pela porta TCP/IP de número 4000, a posição e orientação do movimento do AUV no referencial inercial e as publica em um canal (*pose*), com uma frequência de 100 Hz.

**Processo III: Navegação do Veículo (*navigator*)**

Este processo é responsável por gerar as referências de posição e orientação para os controladores em malha-fechada e por informar ao SCM de que uma manobra chegou ao seu ponto final desejado, dentro da margem de erro programada. As referências de posição e orientação desejadas são enviadas para o controlador de malha-fechada pela porta TCP/IP 4050, e a frequência de execução desse processo é de 10 Hz.

O desvio de obstáculos também é implementada nesse processo. Para isso, sempre que a posição de um objetivo for conhecida (mensagem *obstacle*), o navegador identifica se entre a posição atual e o ponto final da missão existe algum obstáculo. A heurística de desvio implementada nesse trabalho consiste em girar o veículo e identificar se há um trajeto livre ou caso contrário, transladar lateralmente o veículo por 2 m e novamente comprovar se não há obstáculo. Caso exista algum obstáculo, então o veículo é deslocado mais 2 m, e uma nova comprovação é realizada. Assim que um trajeto livre é identificado à frente do veículo, este avança 2 m. Nessa posição, o veículo tenta retornar ao ponto final do trajeto programado sempre que tal ponto não esteja próximo ao obstáculo. Uma vez realizado o desvio, o veículo continua com a missão.

**Processo IV: Estado da Missão e do Sistema (*scm_status_broadcaster*)**

Através desse processo são enviadas à IHM, mediante a porta TCP/IP 8000, o estado dos vários sensores de carga útil do veículo, as coordenadas de posição e orientação atuais do veículo, e os estados lógicos de todos os modelos baseados em autômatos do nível Sistema-Produto (SP) do CSML.

**Processo V: Estado do AUV (*auv_status_broadcaster*)**

Esse processo recebe, pela porta TCP/IP 7000, os comandos gerados pela IHM e que resultam nos eventos não-controláveis. Tais comandos representam a dinâmica discreta do AUV, como erros em sensores, os três níveis de falhas (simples, graves e de posição) e o nível de bateria.

**Processo VI: Eventos controláveis gerados manualmente (*controlled_event_simulator*)**

O sistema possui um modo manual de operação que permite o teste dos componentes do AUV sem a necessidade de seguir o arquivo de missão. Assim, nesse caso, o AUV realiza as ações baseado nos comandos recebidos por esse processo como realizar uma manobra ou ligar / desligar um sensor. A porta TCP/IP de comunicação usada é de número 7050 e recebe os comandos da IHM.

**Processo VII: Estimação de consumo de bateria (*power_estimator*)**

O consumo de bateria é estimado a partir dos deslocamentos do veículo e do estados ligado ou desligado dos sensores de carga útil, onde cada sensor possui uma influência diferente para o consumo. Assume-se que inicialmente o veículo conta com 100% de carga e que gradualmente diminuiu à medida que o AUV realiza a missão. O consumo é estimado como sendo uma ponderação entre a distância percorrida, o tempo em que um sensor fica ligado e o tempo em que o veículo fica parado. O consumo é publicado para os demais processos a uma taxa de 1 Hz.

**Processo VIII: Detecção de obstáculo (*obstacle_sensor_broadcaster*)**

Empregando um *framework* de mapeamento baseado em *octotree* (HORNUNG *et al.*, 2013) integrado ao ROS, esse processo recebe um evento de detecção de obstáculo gerado pela IHM ou também de modo manual pelo próprio código do processo, e cria uma distribuição probabilística simulando a presença de um obstáculo à frente do veículo. O obstáculo é representado por meio de uma estrutura matricial de dados da biblioteca *Point Cloud Library* (PCL, 2014), que também é integrada ao ROS. O obstáculo é então convertido em um mapa de ocupação por um processo já existente no ROS, denominado de *octomap_server* que entre várias funções, realiza a atualização do mapa com células livres de obstáculos e células com obstáculos. Cada célula possui um tamanho mínimo de 0,5 x 0,5 x 0,5 m, modificável através de um arquivo de configuração do ROS. Basicamente o algoritmo divide uma célula ocupada em subcélulas, que podem estar ocupadas ou livres, até o limite do tamanho mínimo. Assim, ao redor do obstáculo é criado uma árvore de nós, onde cada nó contém uma célula que representa a presença ou não do objeto. Por sua vez, o processo *octomap_server* disponibiliza para o ROS várias estruturas de dados e mapas, entre eles

um mapa de ocupação com as células livres e ocupadas, publicadas no canal *octomap*. A frequência de execução do processo de detecção de obstáculo é de 1 Hz.

**Processo XIX: Detecção de colisão (*obstacle_tree*)**

A partir do mapa de ocupação, o processo *obstacle_tree* detecta se entre a posição atual do veículo e a posição desejada (*next_point_mission*) existe alguma célula ocupada por um obstáculo. Caso exista, o processo *obstacle_tree* envia para o processo de navegação as coordenadas desse obstáculo. Como não há necessidade de informar ao SCM para escolha da ação, sendo essa tomada internamente ao navegador, esse tipo de operação caracteriza-se por um comportamento reativo, onde a ação é realizada diretamente com base nas leituras sensoriais, sem a necessidade do planejamento. A frequência de execução desse processo é também de 1 Hz.

**Processo X: Transformação de coordenadas (*tf_imu_broadcaster*)**

Esse processo não pertence diretamente ao SCM, uma vez que é usado para enviar a posição e orientação atual do veículo para a ferramenta de visualização do ROS denominada de RVIZ. A posição e orientação do veículo são transformadas para o referencial do sistema de visualização, que contém o modelo do robô descrito por um arquivo em formato XML com as juntas, elos e matrizes de inércias. Contudo, como não há simulação hidrodinâmica, essa ferramenta somente atualiza o desenho do robô à medida que a sua posição e orientação são modificados. No RVIZ também é incluído o obstáculo detectado para melhor visualização das simulações, conforme pode ser visualizado na simulação do 4º caso apresentado no capítulo 7.

# APÊNDICE C – IMPLEMENTAÇÃO DO CSML

## C.1. GERAÇÃO AUTOMÁTICA DE CÓDIGO

Uma característica da ferramenta Supremica consiste no emprego do formato XML (*eXtensible Markup Language*) para codificar os modelos de planta, especificações e supervisores, e que posteriormente podem ser facilmente traduzidos em código-fonte C/C++. Desse modo, a partir da ferramenta de modelagem e análise, é possível gerar o código a ser compilado que irá compor os níveis de SM (supervisores modulares) e SP (sistema-produto) da arquitetura de CSML, diminuindo erros por ventura decorrentes da digitação, tradução lógica e implementação de código, além de diminuir o tempo de *debug* e testes. A ferramenta para tradução do código XML foi desenvolvida em C#.Net usando o Microsoft Visual Studio 2010.

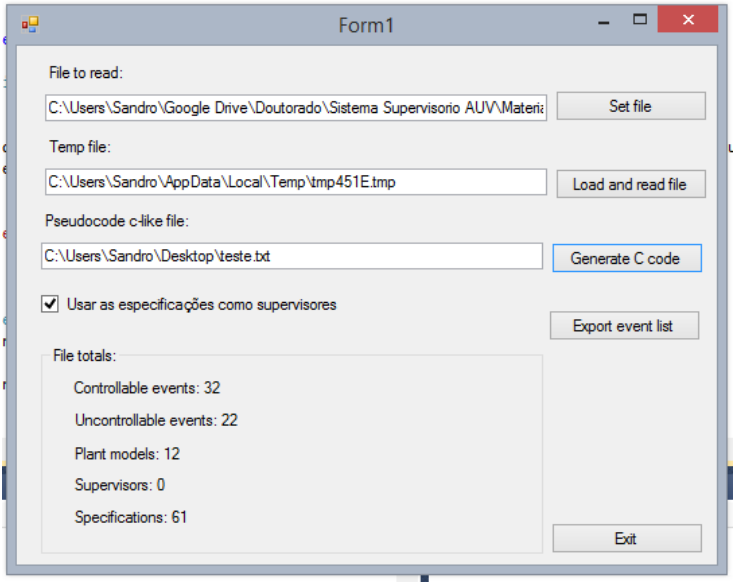


**Figura C.1:** Interface da ferramenta para geração automática de código

A figura C.1 apresenta a ferramenta, onde é possível visualizar o nome do arquivo XML (*file to read*) gerado pela ferramenta Supremica e o arquivo de texto com o código em linguagem C/C++ com as classes dos supervisores modulares (ou especificações) e dos modelos abstratos da planta (*pseudocode c-like file*). Além disso, a ferramenta também permite selecionar como supervisores as próprias especificações e também informa a quantidade total de eventos controláveis, não-controláveis, modelos de plantas, supervisores e especificações codificadas no modelo. Com a tradução automática dos níveis SP e SM do CSML, somente o nível de Sequências Operacionais (SO) precisa ser codificado manualmente, minimizando a ocorrência de erros devido à digitação e tradução dos modelos baseados em autômatos. O paradigma de programação empregado para representar um autômato é baseado em programação orientada a objetos (POO).

Os modelos das plantas locais são representados por uma classe que herda as propriedades básicas de uma classe que representa o Sistema-Produto. Do mesmo modo, um autômato modular é implementado herdando as propriedades básicas de uma classe denominada *Supervisor*. Todos os modelos são obtidos diretamente a partir das informações codificadas no arquivo XML da ferramenta Supremica. A arquitetura de implementação do CSML é construído pela integração dos 3 níveis, codificados manualmente, onde o nível Sistema-Produto (SP) está composto pelas 12 classes com as respectivas plantas locais, o nível Supervisores Modulares (SM) está formado pelas 58 classes que implementam os supervisores modulares e o nível Sequências Operacionais (SO) com as funções de conversão entre os eventos dos modelos formais e os sinais reais para os subsistemas do AUV.

## C.2. TRADUÇÃO DE EVENTOS

Os modelos empregam eventos controláveis e não-controláveis que necessitam ser mapeados para os comandos e sinais reais do AUV. Desse modo, os eventos controláveis, gerados pelo GM, correspondem a comandos que são enviados pelo SCM ao sistema de controle do veículo, e essa correspondência é apresentada na tabela C.1. Os eventos controláveis de início de manobras possuem parâmetros de missão associados, originalmente definidos no arquivo de missão, contendo os dados necessários para guiar o veículo em direção aos pontos objetivos.

Por sua vez, os eventos não-controláveis são mapeados para sinais gerados espontaneamente pelo AUV, e são mostrados na tabela C.2.

**Tabela C.1:** Tradução dos eventos controláveis em comandos para o AUV

| | Evento controlável | Modelo | Parâmetros de missão associados | Comando gerado para o AUV |
|---|---|---|---|---|
| 1 | *rst_btr* | Bateria | - | Reiniciar o dispositivo de informação do nível de carga da bateria |
| 2 | *on_j* | Sensor de carga útil[14] | - | Ligar o respectivo sensor |
| 3 | *off_j* | Sensor de carga útil | - | Desligar o respectivo sensor |
| 4 | *rst_j* | Sensor de carga útil | - | Reiniciar o sensor de carga útil |
| 5 | *st_m_j* | Manobra[15] | $<x_s, y_s, z_s, \phi_s, \theta_s, \psi_s>$: coordenadas do ponto de início<br>$<x_f, y_f, z_f, \phi_f, \theta_f, \psi_f>$: coordenadas do ponto final<br>T: timeout<br>$\delta$: margem de erro para considerar que o veículo está próximo ao ponto<br>Tipo de manobra | Envia nova referência de posição e orientação para os controladores de malha-fechada<br>Inicia temporizador para realização da manobra<br>Atualiza o valor da margem de erro em torno à posição e orientação do veículo<br>Atualiza o tipo de manobra em curso |
| 6 | *sus_m_j* | Manobra | - | Comando de suspensão da manobra em curso |
| 7 | *rsm_m_j* | Manobra | - | Comando de retomada da manobra suspensa |
| 8 | *fin_m_j* | Manobra | - | Comando de finalização da manobra suspensa |

[14] Sensores de carga útil: sonar de batimetria, câmera, CTD e GPS.

[15] Manobras de descida, navegação, retorno e subida. Os eventos de suspensão, retomada e finalização de manobras não estão disponíveis na manobra de subida.

| | Evento controlável | Modelo | Parâmetros de missão associados | Comando gerado para o AUV |
|---|---|---|---|---|
| 9 | *rst_m$_j$* | Manobra | - | Reinício do componente responsável por executar e monitorar a manobra em curso |
| 10 | *rst_col* | Colisões | - | Reinício do componente responsável por monitorar colisões |
| 11 | *rst_f* | Falhas | - | Reinício do componente responsável por monitorar falhas |

**Tabela C.2:** Tradução dos sinais do AUV em eventos não-controláveis

| | Sinal do AUV | Componente[16] | Evento não-controlável | Modelo |
|---|---|---|---|---|
| 1 | Nível baixo de bateria | IHM e estimador de consumo | *wrn_btr* | Bateria |
| 2 | Nível crítico de bateria | IHM e estimador de consumo | *ctr_btr* | Bateria |
| 3 | Erro em sensor de carga útil | IHM | *er_$_j$* | Sensor de carga útil |
| 4 | Submersão do veículo | IHM | *in_h2o* | Submersão |
| 5 | Veículo na superfície | IHM | *out_h2o* | Submersão |
| 6 | Erro na realização de manobra | *Timeout* do processo de navegação que executa e monitora a realização da manobra | *er_m$_i$* | Manobra |
| 7 | Obstáculo | Processo que executa | *dt_col* | Colisões |

[16] A IHM refere-se à interface homem-máquina do ambiente de simulador, a ser apresentada no próximo capítulo. Os demais programas correspondem aos processos que compõem o SCM.

| | Sinal do AUV | Componente[16] | Evento não-controlável | Modelo |
|---|---|---|---|---|
| | detectado | o desvio de obstáculos | | |
| 8 | Obstáculo desviado | Processo que executa o desvio de obstáculos | *ok_col* | Colisões |
| 9 | Colisão com obstáculo | Processo que executa o desvio de obstáculos | *er_col* | Colisões |
| 10 | Falha simples | IHM | *f* | Falhas |
| 11 | Falha grave | IHM | *fg* | Falhas |
| 12 | Falha de posicionamento | IHM | *fpos* | Falhas |

# APÊNDICE D – INTERFACE HOMEM MÁQUINA

A Interface Homem-Máquina (IHM) desenvolvida para esse trabalho, foi criada em linguagem Java, e possui como objetivo simular os sinais relacionados com a dinâmica dirigida a eventos do AUV. Esses sinais correspondem aos eventos não-controláveis que representam erros em sensores de carga útil, falhas no sistema embarcado, estado da submersão e níveis de bateria.

Para permitir testes da plataforma desenvolvida, também foram incluídos os eventos controláveis que, em condições normais do SCM, não estão disponíveis, mas que em modo manual de operação é possível empregá-los para movimentar o veículo. Neste modo de operação, a missão é gerada a partir do arquivo de missão, porém o SCM não gera os comandos para realizar a missão, sendo necessário o usuário usar a IHM para comandar o movimento do veículo. Os comandos implementados no modo manual são: iniciar manobra, para mover o AUV em direção ao próximo ponto especificado no arquivo de missão; suspensão, retomada e finalização de manobras; ações de ligar ou desligar sensor de carga útil.

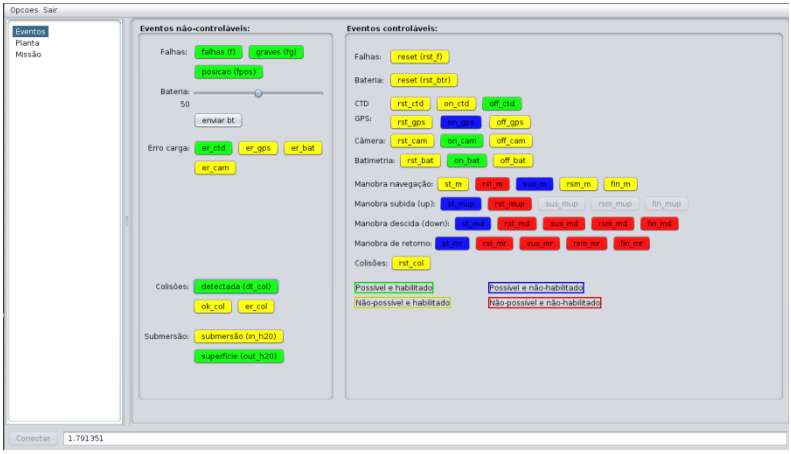


**Figura D.1:** IHM com painel de geração de eventos

A figura D.1 apresenta a tela com o painel de geração de eventos, à esquerda, os não-controláveis e à direita os controláveis, com a manobra de descida em execução e com a câmera ligada e o veículo submerso.

Os estados dos botões contidos no painel com os eventos controláveis e não-controláveis, permite acompanhar, indiretamente, os estados dos modelos locais da planta do AUV. Assim, botões na cor *verde* indicam eventos possíveis de ocorrerem no estado atual do modelo e habilitados pelos supervisores, ou seja, cuja ação não foi impedida por um supervisor; cor *amarela* indica evento não possível e habilitado; cor *azul*, evento possível mas desabilitado; e cor *vermelha*, não possível e não habilitado. Eventos não-controláveis não podem ser desabilitados pelo supervisor e, portanto, não devem aparecer nas cores azul e vermelha.

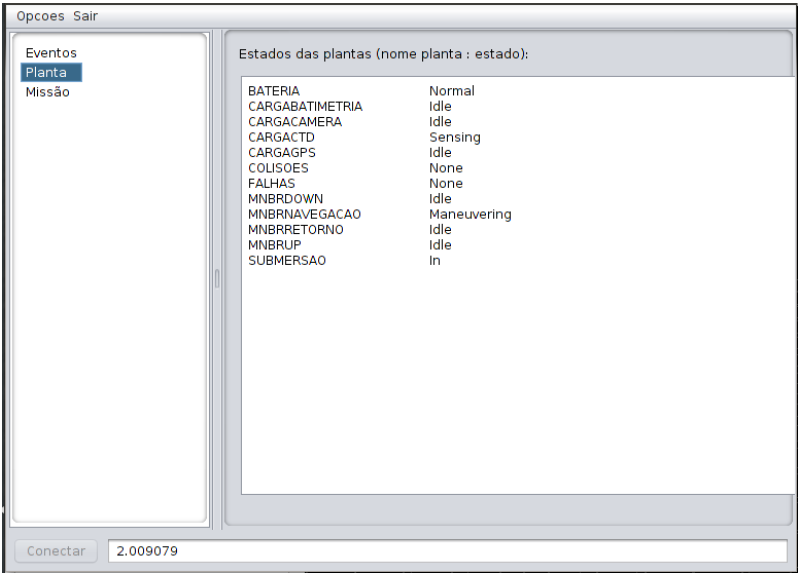


**Figura D.2:** IHM com o estado das plantas locais

O painel *planta* da IHM, apresentado na figura D.2, permite acompanhar o estado dos modelos locais do AUV, apresentados em uma lista, onde na primeira coluna é mostrado o nome do modelo e na seguinte coluna, o seu estado. Os nomes dos modelos e respectivos estados correspondem aos mesmos usados na ferramenta Supremica, devido à geração automática de código. Os estados dos modelos são

enviados pela porta TCP/IP 8000 pelo processo *scm_status_broadcaster* em execução no ROS para a IHM, que atualiza os estados dos botões no painel de eventos e o texto com a indicação dos estados dos modelos no painel da planta.

Finalmente, o último painel, denominado *missão*, contém cada uma das fases a serem realizadas pelo AUV, com as coordenadas de posição e orientação final das mesmas e com a indicação dos sensores de carga útil utilizados em cada uma das etapas. O painel também indica a posição e orientação atual do veículo. A figura D.3 mostra o painel *missão*.

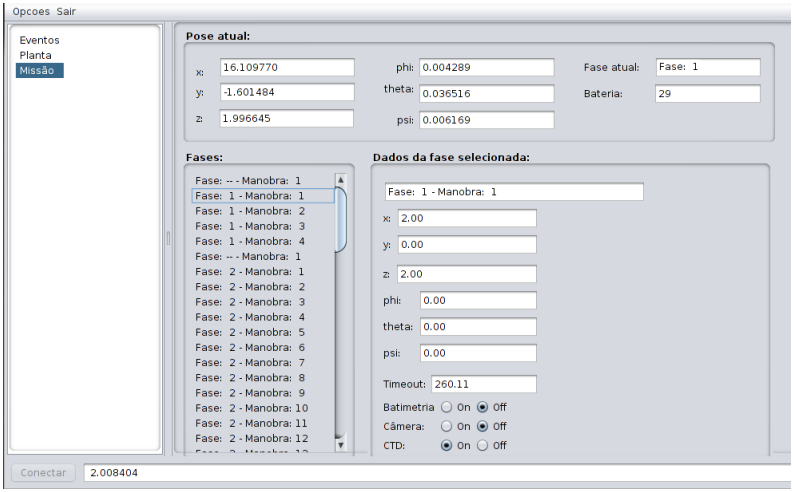


**Figura D.3:** IHM com os dados de uma missão

# APÊNDICE E – COMUNICAÇÃO E MENSAGENS ROS

Para troca de mensagens entre si, os processos do SCM implementados no ROS empregam duas categorias de mensagens: assíncronas e síncronas. As mensagens assíncronas empregam o mecanismo de *publisher* / *subscriber* fornecida pelo ROS, onde um processo publica em um canal a uma dada frequência um determinado tipo de dado, e os processos que necessitam da informação, consultam o respectivo canal. As mensagens síncronas são implementadas através de serviços usando mecanismo do tipo cliente / servidor, com a opção de "preempção"[17]. Nesse tipo de mecanismo, um processo cliente solicita um serviço, usualmente com tempo de execução prolongado, que é executado por uma tarefa no processo servidor. Tal execução é iniciada quando o processo cliente envia um determinado tipo de mensagem implementado pelo ROS. Ao final do processamento o servidor envia uma mensagem com os dados de resposta. O processo cliente também pode cancelar a execução da solicitação, mediante chamadas de funções disponibilizadas pelo ROS.

O mecanismo cliente / servidor com opção de preempção é empregada para a realização de manobras, onde o SCM informa ao processo navegador a necessidade de início de uma nova manobra, enviando as coordenadas finais do próximo ponto desejado. O processo navegador informa ao SCM quando a manobra terminou com êxito, ou com erro caso o tempo máximo (*timeout*) para execução da manobra seja atingido e o veículo não tenha alcançado o ponto final. Caso haja necessidade, o processo navegador também pode solicitar a suspensão, finalização ou retomada da manobra atual.

Com exceção da realização de manobras, todos os demais canais de comunicação entre processos do ROS são realizados mediante mecanismo *publisher* / *subscriber*. Para evitar perda significativa de informação, devido às diferenças nos períodos de execução dos proces-

[17] No ROS a preempção permite que um serviço ou tarefa possa ser finalizado pelo processo que o solicitou mesmo que o processo servidor não tenha terminado o cômputo do resultado final, contudo não permite a suspensão e retomada do serviço ou tarefa, ainda que na documentação o termo usado seja *preemptable tasks*.

```
#include <ros/ros.h>
#include "package/msg_type.h"

int main(int argc, char ** argv){
    ros::init(argc, argv, "thread_name");
    ros::NodeHandle node;

    ros::Publisher publisher_object =
            node.advertise<package::message_type>("channel_name", 100);
    ros::Rate loop_rate(10);

    while(ros::ok()){
        package::message_type msg;
        msd.data1 = "header";
        msg.data2 = 2;
        ...

        publisher_object.publish(msg);

        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

**Figura E.1:** Exemplo de publicação de mensagens por processo do ROS

sos, o ROS permite ao desenvolvedor estipular os tamanhos das filas de mensagens, descartando as mais antigas quando novas mensagens chegam ao um canal com sua capacidade máxima esgotada. A comunicação entre os processos do ROS com os componentes do ambiente de simulação, ou seja, com os componentes que não estão sob execução do ROS, são implementadas via *sockets* TCP/IP.

## E.1.MECANISMO DE COMUNICAÇÃO *PUBLISHER* / *SUBSCRIBER*

O ROS fornece mediante biblioteca de programação um mecanismo para troca de mensagens assíncronas do tipo *publisher* / *subscriber*. As mensagens são definidas em um arquivo de texto, incluída no ambiente de desenvolvimento denominado de *package*, que mediante a compilação gera os arquivos *header* na linguagem de programação escolhida, C/C++ ou Phyton.Na figura E.1 é possível visualizar um programa C/C++ de exemplo de publicação de mensagens. A linha 2 apresenta a inclusão do arquivo *header* de definição da mensagem, gerado automaticamente pelo ROS ao compilar um pacote. O processo ou *thread* em execução pelo sistema operacional é associado a um nó gerenciado pelo ROS, operação essa realizada pela função *ros::init* (linha 5). O nó publica as mensagens do tipo *package::message_type* mediante a chamada à função *advertise*, que recebe como parâmetros o tipo da mensagem, o nome do canal e o número de mensagens de saída que o canal comporta (linha 8). A frequência aproximada para execução do *loop* do processo é definida por um objeto da classe *ros::Rate*, no caso, configurado para 10 Hz (linha 10), contudo tal frequência de execução não é garantida pelo fato do ROS e do sistema operacional utilizados (Ubuntu) não serem tempo-real. A mensagem é então criada e seus campos preenchidos pelos tipos de dados correspondentes (linhas 13, 14 e 15) e publicados no canal pelo comando *publish* (linha 18). A diretiva *spinOnce()* da biblioteca de funções do ROS garante que os demais processos e filas de eventos sejam executados (linha 20) e, finalmente, a diretiva *sleep()* (linha 21) busca garantir que o tempo mínimo de execução do processo atenda à frequência de 10 Hz definida no início do programa.

Para os processos que desejam consultar mensagens publicadas em um canal, necessitam se inscrever ao canal através da função *subscribe*, passando como argumento para a função o nome do canal, o tamanho da fila de mensagens de entrada e o nome da função a ser exe-

```
#include <ros/ros.h>
#include "package/msg_type.h"

void message_Callback(const package::message_type &msg){
    ...
}

int main(int argc, char ** argv){
    ros::init(argc, argv, "thread_other_name");
    ros::NodeHandle node;

    ros::Publisher sub = node.subscribe("channel_name", 100, message_Callback);
    ros::Rate loop_rate(10);

    while(ros::ok()){
        ros::spinOnce();
        loop_rate.sleep();
    }
    return 0;
}
```

**Figura E.2:** Exemplo de consulta de mensagem

cutada (*callback*) sempre que uma nova mensagem é publicada no canal (linha 12 da figura E.2).

A comunicação assíncrona é empregada em quase todas as trocas de dados entre os processos que fazem parte do SCM, conforme pode ser visto na figura 6.1.

## E.2. MECANISMO DE COMUNICAÇÃO CLIENTE / SERVIDOR COM PREEMPÇÃO DE TAREFAS

O ROS disponibiliza uma biblioteca de funções que implementa a execução de serviços em um processo servidor solicitados por um processo cliente. Além disso, esse tipo de mecanismo também permite a troca de mensagens síncronas entre os dois processos durante toda a realização do serviço, com a opção de cancelamento, suspensão e retomada da tarefa que implementa o serviço solicitado pelo processo cliente. A comunicação entre os processos cliente e servidor são feitos de modo transparente ao desenvolvedor, mediante troca de mensagens entre um objeto da classe *action server* e outro da classe *action client*, como pode ser visto na figura E.3.

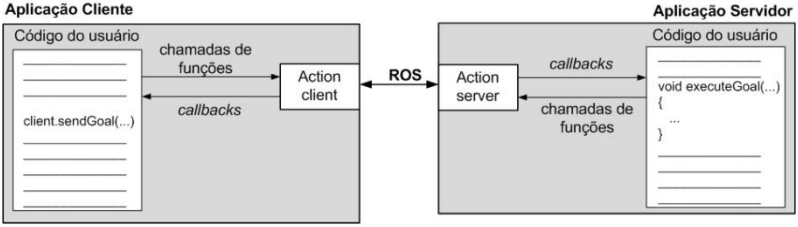


**Figura E.3:** Interação cliente-servidor. Adaptado de ROS (2014).

Basicamente, o mecanismo fornece uma API simples que permite ao processo cliente solicitar ao servidor um conjunto de objetivos (*goal*), que serão realizados pelo processo servidor e enviados ao processo cliente (*result*) uma vez finalizado o processamento, além de permitir ao processo cliente consultar o estado da solicitação (*feedback*) e o estado da execução do processo servidor (*status*). As informações de *goal*, *result* e *feedback* são definidas em um arquivo de texto, que especifica o tipo de mensagem com o objetivo, resultado e estado da solicitação, que são compilados automaticamente e incluídos como *header* pelo ROS.

```cpp
#include <ros/ros.h>
#include "actionlib/client/simple_action_client.h"
#include "package/ExampleAction.h"

int main(int argc, char ** argv){
    ros::init(argc, argv, "client_process");
    ros::NodeHandle node;
    ros::Rate loop_rate(10);

    actionLib::SimpleActionClient<package::ExampleAction> * client;
    client = new actionLib::SimpleActionClient<package::ExampleAction>("server_name", true);

    client->waitForServer();
    package::ExampleGoal goal;
    goal.data1 = "header";
    goal.data2 = 2.3;
    client->sendGoal(goal);
    bool before_timeout = client->waitForResult(ros::Duration(timeout));
    if(before_timeout)
        ROS_INFO("State is %s", client->getState().toString().c_str());
    else
        client->cancelAllGoal();
    int resultado = client->getResult()->result_user_data1;
    return 0;
}
```

**Figura E.4:** Exemplo de processo cliente

```
#include <ros/ros.h>
#include "actionlib/server_action_server.h"
#include "package/ExampleAction.h"
void executeAction(const package::ExampleGoalConstPtr &goal,
                   actionLib::SimpleActionServer::ExampleActionServer * s){
    ... //realizar serviço
    if(s->isActive() && s->isPreemptRequested())
        s->setPreempted();
    else
        s->setSucceeded(ExampleActionResult_data);
}

int main(int argc, char ** argv){
    ros::init(argc, argv, "server_process");
    ros::NodeHandle node;
    ros::Rate loop_rate(10);

    actionLib::SimpleActionServer::ExampleActionServer * server;
    server = new ExampleActionServer(node, "server_name", boost::bind(&executeAction, _1, &server), false);
    server.start();
    return 0;
}
```

**Figura E.5:** Exemplo de processo servidor

Um exemplo de processo cliente pode ser visto no código da figura E.4. Um arquivo definido pelo usuário especifica o tipo de ação (mensagem), com os valores empregados para definir os objetivos (*goal*), recuperar o estado da execução do serviço (*status*) e o valor final do resultado do serviço (*result*). A função *waitForServer* faz com que o cliente aguarde que o servidor informe que será atendida a solicitação do cliente (linha 13). Na sequência o cliente preenche os dados da mensagem (linhas 14, 15 e 16) e as envia para o servidor (linha 17) mediante a função *sendGoal*. O processo cliente então aguarda que o servidor responda a solicitação dentro de um valor máximo de tempo (linha 18), para imprimir o estado da solicitação (linha 20) e usar o valor da resposta (linha 23). Caso o *timeout* expire, então o cliente solicita para o processo servidor cancelar todas as solicitações pendentes (linha 22).

O código de exemplo de um processo servidor simplificado pode ser visualizado na figura E.5. Essencialmente, o processo servidor cria um objeto (linha 18) que executará a tarefa definida por uma função (linhas 4 a 10). Nessa função são processados os dados e atualizados os valores de resultado (*result*), estado da solicitação (*feedback*) e estado da execução da *thread* (*status*). Os valores de *feedback* e *status* podem ser solicitados pelo processo cliente em qualquer momento, mas o valor de resultado final é somente enviado ao cliente quando o processo servidor terminar a sua execução.

## E.3.DEFINIÇÃO DE MENSAGENS ENTRE PROCESSOS

A tabela E.1 contém a relação de todas as mensagens assíncronas trocadas entre os vários processos em execução no ROS e que empregam o mecanismo *publisher / subscriber*.

**Tabela E.1:** Definição das mensagens assíncronas entre processos ROS

| Mensagem | Processo emissor | Processos receptores | Dados |
|---|---|---|---|
| *auv_status* | *Auv_status_ broadcaster* | *SCM Obstacle_ sensor_ broadcaster* | Nível da bateria, o nível de falhas (simples, grave ou de posição), se houve erro em algum sensor de carga útil, e o estado da detecção de obstáculos |
| *event_simulator* | *Controlled_ event_* | *SCM* | Eventos controláveis para realização de missões em |

| Mensagem | Processo emissor | Processos receptores | Dados |
|---|---|---|---|
| | *simulator* | | modo manual |
| *obstacle* | *Obstacle_tree* | *Navigator* | Coordenadas da posição do obstáculo no referencial inercial |
| *octomap* | *Octomap_ server* | *Obstacle_ tree* | Estrutura de dados em formato octomap contendo várias informações, entre elas, as células ocupadas e livres do ambiente |
| *pose* | *Sensor_imu_ broadcaster* | *Navigator* *Obstacle_ sensor_ broadcaster* *Obstacle_ tree* *Power_ estimator* *Scm_status_ broadcaster* *Tf_imu_ broadcaster* | Coordenadas da posição e orientação atuais do AUV no referencial inercial |
| *power_ estimation* | *Power_ estimator* | *SCM* | Dados de consumo até o momento e a quantidade de bateria disponível |
| *next_point_ mission* | *Navigator* | *Obstacle_ tree* | Coordenadas de posição e orientação do próximo ponto objetivo da missão, no referencial inercial |
| *scm_status* | *SCM* | *Scm_status_ broadcaster* | Contém os dados de todos os modelos do nível sistema-produto, incluindo os eventos possíveis, habilitados / desabilitados, bem como as coordenadas atuais do veículo e o estado dos sensores |
| *sonar_ pointcloud* | *Obstacle_ sensor_ broadcaster* | *Octmap_ server* | Estrutura de dados matricial com a distribuição de intensidade de objetos à frente do sonar |

A tabela E.2 apresenta as definições de cada uma das mensagens utilizadas no SCM desenvolvido, com o nome do campo, o tipo de dados e a descrição. Para cada mensagem é criado um arquivo de texto, que posteriormente é compilado pelo ROS em arquivos *header* a serem incluídos no código do sistema.

**Tabela E.2:** Tipos de dados das mensagens entre processos ROS

| Mensagem | Campo | Tipo de dados | Descrição |
|---|---|---|---|
| *auv_status* | *battery_level* | *int64* | Nível da bateria (0 a 100%) |
| | *submersion* | *int64* | Estado da submersão (veículo submerso ou na superfície) |
| | *fail_level* | *int64* | Nível de falha (sem falha, simples, grave ou de posição) |
| | batimetria | *int64* | Erro no sensor de batimetria |
| | gps | *int64* | Erro no GPS |
| | ctd | *int64* | Erro no sensor CTD |
| | colisões | *int64* | Estado da colisão (obstáculo detectado ou não, ocorrência de colisão) |
| | camera | *int64* | Erro na câmera |
| *event_ simulator* | *msg* | *string* | Contém o string com o evento controlável gerado pela IHM, de acordo com os modelos baseados em autômatos |
| *obstacle* | *set* | *int8* | Indica se um ponto foi configurado |
| | *point* | *ROS/geometry_msgs/twist* | Coordenadas <x, y, z, $\phi$, $\theta$, $\psi$> do obstáculo detectado |
| *octomap* | *octomap* | *octomap_msgs* | Tipo de dados empregado pela biblioteca Octomap |

| Mensagem | Campo | Tipo de dados | Descrição |
|---|---|---|---|
| *pose* | *msg* | *geometry_msgs/twist* | Coordenadas <x, y, z, ϕ, θ, ψ> com a posição atual do robô |
| *power_estimaton* | carga | *float64* | Carga disponível da bateria (0 até o valor máximo). Carga + consumo = valor máximo disponível. |
| | consumo | *float64* | Quantidade já consumida de bateria desde o início de operação do veículo. |
| *next_point_mission* | *set* | *int8* | Indica se um ponto foi configurado |
| | *point* | *ROS/geometry_msgs/twist* | Coordenadas <x, y, z, ϕ, θ, ψ> do próximo ponto desejado da missão |
| *scm_status* | *msg* | *string* | String com dados diversos para a IHM |
| *sonar_pointcloud* | *data* | *ROS/sensor_msgs/pointcloud2* | Estrutura de dados com a posição e dimensão do obstáculo detectado (espalhamento do sonar ou câmera) |

## REFERÊNCIAS BIBLIOGRÁFICAS

AGUILAR, L. L. P. **Controle Inteligente para a Navegação de Veículos Submarinos Semi-Autômatos.** Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo. São Paulo. 2007.

AKESSON, K.; FABIAN, M.; FLORDAL, H.; MALIK, R. **Supremica: an Integrated Environment for Verification, Synthesis and Simulation of Discrete Event Systems.** Proceedings of 8th Workshop on Dicrete Event Systems, WODES. p. 384-385. 2006.

ALBIEZ, J.; JOYEUX, S.; HILDEBRANT, M. **Adaptive AUV Mission Management in Under-Informed Situations.** OCEANS. p. 1-10. 2010.

ALVES, O. L. F.; CUNHA, A. E. C. **Modelagem e Controle de uma Aeronave de Asas Fixas para o Planejamento de Missões de Vants.** XVII Congresso Brasileiro de Automática. Bonito, MS, Brasil. p. 1114-1121. 2010.

ALTAMIRANDA, E.; COLINA, E. **Intelligent Supervision and Integrated Fault Detection and Diagnosis for Subsea Control Systems.** Oceans' 2007 – Europe. p. 1-6. 2007.

ANGSTRÖM. **The Angström Distribution.** Disponível em <http://www.angstrom-distribution.org>. Acesso em: outubro, 2014.

ANTONELLI, G. **Open Control Problems in Underwater Robotics.** 4th International Workshop on Robot Motion and Control. June, 17-20, 2004.

ARANDA, J.; ARMADA, M. A.; CRUZ, J. M. **Automation for the Marine Industries.** 273 p. 2005.

BACIC, M. **On Hardware-in-the-loop Simulation.** Proceedings of the 44th Conference on Decision and Control and The European Control Conference. Sevilha, Espanha. p. 3194-3198. 2005.

BAGNITCKII, A. INZARTSEV, A.; SENIN, R. **Facilities of AUV Search Missions Planning.** OCEANS. p. 1-7. 2011.

BAILEY, T. **Mobile Robot Localisation and Mapping in Extensive Outdoor Environments.** Tese de doutorado, Universidade de Sidney, Austrália. Agosto, 2002.

BARROUIL, C.; LEMAIRE, J. **Advanced Real-Time Mission Management for an AUV.** Symposium on Advanced Mission Management and System Integration Techologies for Improved Tactical Operations. Florença, Itália. 1999.

BATTISTELLA, S.; QUEIROZ, M. H.; SANTOS, C. H. F. **Modelagem e Síntese de Supervisores para Controle de Missão de AUVs atuando em Lagos de Barragens de Hidrelétricas baseado na Teoria de Controle Supervisório.** Anais do XIX Congresso Brasileiro de Automática, CBA. p. 1870-1877. 2012.

BATTISTELLA, S.; QUEIROZ, M. H. **Arquitetura e Ambiente de Simulação para Sistema de Missão de AUVs baseado em Controle Supervisório.** Anais do XX Congresso Brasileiro de Automática, CBA. p. 4044-4051. 2014.

BATTISTELLA, S.; QUEIROZ, M. H. **Simulation Environment of an Architecture for Mission Control System of AUVs operating in Lakes of Hydroelectric Dams.** Proceedings of the III International Conference on Applied Robotics for the Power Industry, CARPI. p. 1-6. 2014.

BHATTACHARYYA, S.; KUMAR, R.; TANGIRALA, S.; O'CONNER, M.; HALLOWAY, L. E. **Animation/Simulation of Missions for Autonomous Underwater Vehicles with Hybrid-Model Based.** Proceedings of the American Control Conference. Mineapolis, EUA. p. 4273-4278. 2006.

BECKER, L. B.; FARINES, J. M.; BODEVEIX, J. P.; VERNADAT, F. **Development Process for Critical Embedded Systems.** Anais do I Workshop de Sistemas Embarcados. p. 151-164. 2010.

BECKER, L. L; PEREIRA, C. E. **SIMOO-RT – An Object-Oriented Framework for the Development of Real-Time Industrial**

**Automation Systems.** IEEE Transactions on Robotics and Automation. Vol. 18. N. 4. p. 421-430.

BIAN, X.; ZOU, H.; CHANG, Z.; ZHAO, D.; WANG, Z. **Multi-thread Technology's Application in The Decesion-making System of AUV.** Proceedings of the IEEE International Conference on Mechatronics and Automation. Cataratas do Niagara, Canadá. p. 868-873. 2005.

BIAN, X. Q.; CHEN, T.; ZHOU, J.; YAN, Z. **Research of Autonomous Control Based of Multi-Agent for AUV.** Intelligent Systems and Applications. 2009a.

BIAN, X.; CHEN, T.; YAN, Z. QIN, Z. **Autonomous Management and Intelligent Decision for AUV.** Proceeding of the IEEE International Conference on Mechatronics and Automation. Changchun, China. p. 2101-2106. 2009b.

BLIDBERG, D. R.; TURNER, R. M.; CHAPPELL, S. G. **Autonomous Underwater Vehicles: Current Activities and Research Opportunities.** Robotics and Automation Systems. Holland, p. 139 – 150, 1991.

BRÄUNL, T. **Embedded Robotics: Mobile Robot Design and Application with Embedded Systems.** Ed. Springer-Verlag. Austrália. 542 p. 2008.

BRITO, M. P.; GRIFFITHS, G. **A Markov Chain State Transition Aproach to Establishing Critical Phases for AUV Realibility.** IEEE Journal of Oceanic Engineering. Vol. 36. N. 1. p. 139-149. 2011.

BROOKS, R. A. **A Robust Layered Control System for a Mobile Robot.** IEEE Journal of Robotics and Automation. Vol. RA-2. N. 1. p.14-23. 1986.

BRUTZMAN, D.; MCGHEE, R.; DAVIS, D. **An Implemented Universal Mission Controller with Run Time Ethics Checking for Autonomous Unmanned Vehicles – a UUV Exemple.** IEEE/OES Autonomous Underwater Vehicles. p. 1-8. 2012.

BUSQUETS, J.; BUSQUETS, J. V.; TUDELA, D.; PÉREZ, F.; BUSQUETS-CARBONELL, J.; BARBERÁ, A.; RODRIGUEZ, C.;

GARCÍA, A. J.; GILABERT, J. **Low-cost AUV based on Arduino Open Surce Microcontroller Board.** IEEE/OES Autonomous Underwater Vehicles. p. 1-10. 2012.

CAPKOVIC, F. **Cooperation of Autonomous Agents Based on Supervisory Control of DES.** 15th IEEE Mediterranean Electrotechnical Conference. p. 178-183. 2010.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems.** 2ª ed. Springer. 2008.

CHANG, Z.; FU, M.; TANG, Z.; CAI, H. **Autonomous Mission Management for an Unmanned Underwater Vehicle.** Proceedings of IEEE International Conference on Mechatronics & Automation. Niagara Falls, Canada, p. 1455 – 1459, July 2005.

CHEN, P. C.; HWANG, Y. K. **SANDROS: A Dynamic Graph Search Algorithm for Motion Planning.** IEEE Transactions on Robotics and Automation. V. 14. N. 3. p. 390-403. 1998.

CHEN, P. C. Y.; GUZMAN, J. I.; NG, T. C.; POO, A. N.; CHAN, C. W. **Supervisory Control of an Unmanned Land Vehicle.** Proceedings of the IEEE International Symposium on Intelligente Control. Vancouver, Canadá. p. 580-585. 2002.

CHEN, T.; YAN, Z.; ZHAO, Y. **Scheduling of BSA-AUV Mission Operation: Development, Implementation and Trials.** OCEANS. p. 1-6. 2011.

CHEN, C. W.; KOUH, J. S.; TSAI, J. F. **Modeling and Simulation of an AUV Simulator With Guidance System.** IEEE Journal of Oceanic Engineering. Vol. 38. N. 2. p. 211-225. 2013.

CHITRE, M. **DSAAV – A Distributed Software Architecture for Autonomous Vehicles.** Oceans 2008. Quebec, Canadá. p. 1-10. 2008.

CHRISTENSEN, H. I.; PIRJANIAN, P. **Theorical Methods for Planning and Control in Mobile Robotics.** First International Conference on Knowledge-Based Intelligent Eletronics Systems. Adelaide, Austrália. p. 81-86. 1997.

CHUNG, S. L.; LAFORTUNE, S.; LIN, F. **Limited Lookahead Policies in Supervisory Control of Discrete Event Systems.** IEEE Transactions on Automatic Control. V. 37. N. 12. p. 1921-1935. 1992.

CLARKE, E. M.; GRUMBERG, O.; PELED, D. **Model Checking.** MIT Press. 1999.

COSTA, G. S. **Utilização da Verificação de Modelos para o Planejamento de Missões de Veículos Aéreos Não-Tripulados.** Dissertação de mestrado, Instituto Militar de Engenharia. Rio de Janeiro, 2008.

CROTEAU, A.; DUGUAY, N. **The Maski Underwater Robot: Technology, Field Experience and Benefits.** IEEE International Conference on Applied Robotics for the Power Industry. Canadá. 2010.

CUNHA, A. E. C. **Contribuições ao Controle Hierárquico de Sistemas a Eventos Discretos.** Tese de doutorado, Universidade Federal de Santa Catarina. Florianópolis, 2003.

DIAS, P. S.; GOMES, R. M. F.; PINTO, J.; GONÇALVES, G. M.; SOUSA, J. B.; PEREIRA, F. L. **Mission Planning and Specification in the Neptus Framework.** Proceedings of the IEEE International Conference on Robotics and Automation. Orlando, Florida, EUA. p. 3220-3225. 2006a.

DIAS, P. S.; PINTO, J.; GONÇALVES, G. M.; GONÇALVES, R.; SOUSA, J. B.; PEREIRA, F. L. **Mission Review and Analysis.** International Conference on Information Fusion. Florência, Itália. 2006b.

DIANKOV, R.; KUFFNER, J. **OpenRAVE: A Planning Architecture for Autonomos Robotics.** Robotics Institute, Carnegie Mellon University. Pittsburgh, EUA. 2008.

EBADI, T.; PURVIS, M.; PURVIS, M. **A Distributed and Concurrent Framework for Facilitating Cooperation in Dynamic Environments.** International Conference on Web Intelligence and Intelligent Agent Technology. Toronto, Canadá. p. 287-294. 2010.

ENCARNAÇÃO, P. M. M. **Nonlinear Path Following Control Systems for Ocean Vehicles.** Tese de doutorado, Universidade Técnica de Lisboa – Instituto Superior Técnico. Abril, 2002.

ENGELHARDTSEN, P. M. M. 3D **AUV Collision Avoidance.** Dissertação de mestrado, Universidade de Ciência e Tecnologia da Noruega. 2007.

FERNANDES, D. A. **Sistema de Controle Ótimo para Veículo Submersível Semi-Autônomo.** Dissertação de mestrado, Escola Politécnica da Universidade de São Paulo. São Paulo, 2007.

FEJLLSTAD, O. E. **Control of Unmanned Underwater Vehicles in Six Degrees of Freedom: A Quaternion Feedback Approach.** Tese de doutorado, Instituto Norueguês de Tecnologia. Noruega, 1994.

FENG, X.; CHEN, P. C. Y.; POO, A. N.; GUZMAN, J. I.; CHAN, C. W. **Enhanced Supervisory Control System Design of an Unmanned Ground Vehicle.** IEEE International Conference on Systems, Man and Cybernetics. p. 1864-1869. 2004.

FOSSEN, T. I.; BALCHEN, J. G. **The NEROV Autonomous Underwater Vehicle.** Proceedings of the Ocean Techonologies and Opportunities in the Pacific for the 90's. OCEANS' 91. p. 1414-1420. 1991.

FOSSEN, T. I. **Guidance and Control of Ocean Vehicles.** Ed. John Wiley & Sons. 1994.

FREEZOR, M. D.; SORRELL, F. Y.; BLANKINSHIP, P. R. **An Interface System for Autonomous Undersea Vehicles.** IEEE Journal of Oceanic Engineering. V. 26. N. 4. p. 522-525. 2001.

FU, J.; RAY, A.; LAGOA, C. M. **Unconstrained Optimal Control of Regular Languages.** Automatica. V. 40. N. 4. p. 639-646. 2004.

FUJISAWA, K.; HAYAKAWA, S.; AOKI, T. **Real-Time Decision Making for Autonomous Mobile Robot using Evolution Strategy and Anytime Search.** 20th Annual Conference of the IEEE Industrial Eletronics Society (IECON). Vol. 4. p. 2809-2814. 2000.

GAMAGE, G. W.; MANN, G. K. I.; GOSINE, R. G. **Discret Event Systems based Formation Control Framework to Coordinate Multiple Nonholonomic Mobile Robots.** International Conference on Intelligent Robots and Systems. Saint Louis, EUA. p. 4831-4836. 2009.

GARRIDO, R. O. **Aplicação da Síntese de Supervisores de Sistemas Híbridos ao Planejamento de Missões de Veículos Aéreos Não-Tripulados.** Dissertação de mestrado, Instituto Militar de Engenharia. Rio de Janeiro, 2009.

GE, S. S.; LEWIS, F. L. **Autonomous Mobile Robots: Sensing, Control, Decision Making and Applications.** Ed. Taylor & Francis Group, LLC. 696 p. 2006.

GOMES, S. C. P.; BIER, C. C. **Estudo sobre Trajetórias de Controle para Robôs Manipuladores.** XII Congresso Brasileiro de Automática. 1998.

GORYCA, J.; HILL, R. C. **Formal Synthesis of Supervisory Control Software for Multiple Robot Systems.** American Control Conference. p. 125-131. 2013.

HARRIS, C.; DEARDEN, R. **Contingency Planning for Long-Duration AUV Missions.** IEEE/OES Autonomous Underwater Vehicles. p. 1-6. 2012.

HERNÁNDEZ, C.; SUN, X.; KOENIG, S.; MESEGUER, P. **Tree Adaptive A*.** Proceedings of the 10th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). p. 123-130. 2011.

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. **Introduction to Automata Theory, Languages and Computation.** 2ª ed. Pearson Education. 2001.

HORNUNG, A.; WURM, K. M.; BENNEWITZ, M.; STACHNISS, C.; BURGARD, W. **OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees.** Autonomous Robots. V. 34. N. 3. p. 189-206. 2013.

HUANG, X.; LI, Y.; JIN, S. **A Control System Based on Data Exchange using Ethernet and CANBus for Deep Water AUV.** Asian Control Conference. p. 1-5. 2013.

INSAURRALDE, C. C.; LANE, D. M. **Autonomy-Assessment Criteria for Underwater Vehicles.** IEEE/OES Autonomous Underwater Vehicles. p. 1-8. 2012.

ITAIPU. **Itaipu: Usina Hidrelétrica – Aspectos de Engenharia.** Itaipu Binacional. Foz do Iguaçu, PR. 2009.
ITAIPU. **Site oficial da Itaipu.** <http://www.itaipu.gov.br > Acesso em: junho, 2014.

JENSEN, J. C.; CHANG, D. H.; LEE, E. A. **A Model-Based Design Methodology for Cyber-Physical Systems.** International Wireless Communications and Mobile Computing Conference (IWCMC). p. 1666-1671. 2011.

JI, M.; SARKAR, N. **Supervisory Fault Adaptive Control of a Mobile Robot and Its Application in Sensor-Fault Accommodation.** IEEE Transactions on Robotics. Vol. 23. N. 1. p. 174-178. 2007.

KAPELLOS, K.; JOURDAN, M.; ESPIAU, B.; ABDOU, S. **Specification, Formal Verification and Implementation of Tasks and Missions for an Autonomous Vehicle.** Proceedings of the 4th International Symposium on Experimental Robotics. Vol. IV. p. 412-421. 1995.

KIM, B.; JUN, B. H.; SIM, H. W.; LEE, F. O.; LEE, P. M. **The Development of Tiny Mission Language for the ISiMI100 Autonomous Underwater Vehicle.** OCEANS 2010. Seattle, EUA. p. 1-6. 2010.

KOENIG, S.; LIKHACHEV, M. **Fast Replanning for Navigation in Unknown Terrain.** IEEE Transactions on Robotics. Vol. 21. N. 3. p. 354-363. 2005.

KURFESS, T. R. **Robotics and Automation Handbook.** CRC Press. 2005.

LAVALLE, S. M. **Planning Algorithms.** Cambridge University Press. 2006.

LEAL, A. B. **Controle Supervisório Modular de Sistemas Híbridos.** Tese de Doutorado, Programa de Pós-Graduação em Engenharia de Automação e Sistemas. Universidade Federal de Santa Catarina. 2005.

LEE, E. A.; SESHIA, S. A. **Introduction to Embedded Systems: A Cyber-Physical Systems Approach.** 2011.

LEFEBVRE, D.; LECLERCQ, E. **Stochastic Petri Net Identification for the Fault Detection and Isolation of Discrete Event Systems.** IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans. Vol. 41. N. 2. p. 213-225. 2011.

LIKHACHEV, M.; FERGUSON, F.; GORDON, G.; STENTZ, A.; THRUN, S. **Anytime Dynamic A*: An Anytime, Replanning Algorithm.** Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS). 2005.

LIMSOONTHRAKUL, S.; DAILEY, M. N.; SRISUPUNDIT, M. **A Modular System Architecture for Autonomous Robots Based on Blackboard and Publish-Subscribe Mechanisms.** Proceedings of the IEEE International Conference on Robotics and Biomimetics. Bangkok, Tailândia. p. 633-638. 2009.

LIN, C.; REN, S.; FENG, X.; LI, Y.; XU, J. **Autonomic Element Based Architecture for Unmanned Underwater Vehicles.** Oceans 2010 IEEE. Sidnei. 2010.

LIU, J. W. S. **Real-Time Systems.** Prentice Hall. 2000.

LIU, J.; DARABI, H. **Ramadge-Wonham Supervisory Control of Mobile Robots: Lessons from Pratice.** Proceedings of the IEEE International Conference on Robotics & Automation. Washington, USA. p. 670-675. 2002.

MARCO, K. D.; WEST, M. E.; COLLINGS, T. R. **An Implementation of ROS on the Yellowfin Autonomous Underwater Vehicle (AUV).** OCEAN'S. p. 1-7. 2011.

MARQUES, E. R.; GONÇALVES, G. M.; SOUSA, J. B. **Seaware: A Publish-Subscribe Middleware for Networked Vehicle Systems.** MCMC'06 7th IFAC Conference on Manoeuvring and Control of Marine Craft. Lisboa, Portugal. 2006.

MARTIN, S. C.; WITHCOMB, L. L.; YOERGER, D.; SINGH, H. **A Mission Controller for High Level Control of Autonomous and Semi-Autonomous Underwater Vehicles.** OCEANS' 2006. p. 1-6. 2006.

MATLAB. Disponível em <http://www.mathworks.com/products/matlab>. Acesso em: outubro, 2014.

MCGANN, F. P.; RAJAN, K.; RYAN, J.; HENTHORN, R. **Adaptive Control for Autnomous Underwater Vehicles.** Proceedings of XXIII AAAI Conference on Artificial Inteligence. Chicago, EUA, p. 1319 – 1324, 2008.

MICROSOFT ROBOTICS. Disponível em <http://www.microsoft.com/robotics>. Acesso em: outubro, 2014.

MOLINA, L.; BASILIO, J. C.; FREIRE, E. O.; MOREIRA, M. V. **Desenvolvimento de uma Arquitetura de Navegação Deliberativa para Robôs Móveis utilizando a Teoria de Controle Supervisório.** XVIII Congresso Brasileiro de Automática. Bonito, MS, Brasil. p. 3231-3238. 2010.

MURPHY, R. R. **Introduction to AI Robotics.** MIT Press. Londres, Inglaterra. 466 p. 2000.

OREBÄCK, A.; CHRISTENSEN, H. I. **Evaluation of Architectures for Mobile Robots.** Autonomous Robots. N. 14. p. 33-49. 2003.

OROCOS. **Open Robot Control Software.** Disponível em <http://www.orocos.org>. Acesso em: outubro, 2014.

PALOMERAS, N.; CARRERAS, M.; RIDADO, P.; HERNANDEZ, E. **Mission Control System for DAM Inspection with an AUV.** Proceedings of the IEEE International Conference on Intelligente Robots and Systems. Beijing, China. p. 2551-2556. 2006.

PALOMERAS, N.; RIDAO, P.; CARRERAS, M.; SILVESTRE, C. **Using Petri Nets to Specify and Execute Missions for Autonomous Underwater Vehicles.** The IEEE/RSJ International Conference on Intelligente Robots and Systems. St. Lous, USA. p. 4439-4444. 2009.

PALOMERAS, N.; EL-FAKDI, A.; CARRERAS, M.; RIDAO, P. **COLA2: A Control Architecture for AUVs.** IEEE Journal of Oceanic Engineering. Vol. 37. N.4. p. 695-715. 2012.

PAULL, L.; SAEEDI, S.; SETO, M.; LI, H. **AUV Navigation and Localization: A Review.** IEEE Journal of Oceanic Engineering. Vol. 39. N. 1. p. 131-149. 2014.

PAVEI, J. **Coordenação em Sistemas Multi-Robôs Utilizando Métodos baseados em Autômatos.** Dissertação de mestrado, Universidade Federal de Santa Catarina. Florianópolis, 2011.

PCL. **Point Cloud Library (PCL).** Disponível em <http://www.pointclouds.org>. Acesso em: julho, 2014.

PEBODY, M. **The Contribution of Scripted Command Sequences and Low Level Control Behaviours to Autonomous Underwater Vehicle Control Systems and Their Impact on Reliability and Mission Success.** OCEANS 2007. p. 1-5. 2007.

PERDOMO, E. F.; GÓMEZ, J. C.; BRITO, A. C. D.; SOSA, D. H. **Mission Specification in Underwater Robotics.** Journal of Physical Agents. Vol. 4. N. 1. p. 25-33. Janeiro, 2010.

PERRAULT, D.; NAHON, M. **Fault-Tolerant Control of an Autonomous Underwater Vehicle.** Oceans' 98 Conference Proceedings. Vol. 2. p. 820-824. 1998.

PERRIER, M.; COSTE-MANIÈRE, E. **A Tentative Study of High-Level Reative Mission Programming: Aplication to VORTEX Underwater Vehicle.** Proceedings of Oceans Engineering for Today's Technology and Tomorrow's Preservation. Breste, França. Vol. 1. p. 596-601. 1994.

PERRIER, M.; KALWA, J. **Intelligent Diagnosis for Autonomous Underwater Vehicle using a Neuro-Symbolic System in a Distributed Architecture.** Oceans' 2005 - Europe. p. 350-355. 2005.

PINTO, J.; DIAS, P. S.; GONÇALVES, G. M.; GONÇALVES, R.; MARQUES, E.; SOUSA, J. B.; PEREIRA, F. L. **Neptus – A Framework to Support the Mission Life Cycle.** 7th IFAC Conference on Manoeuvring and Control of Marine Craft. Lisboa, Portugal. 2006.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach.** 5ª ed. McGraw Hill. 2001.

QUEIROZ, M. H.; CURY, J. E. R. **Modular Supervisory Control of Large Scale Discrete Event Systems.** Proceedings of 5th Workshop on Dicrete Event Systems, WODES. p. 103-110. 2000.

QUEIROZ, M. H.; CURY, J. E. R. **Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell.** Proceedings of 6th Workshop on Discrete Event Systems, WODES. p. 377-382. 2002.

RAMADGE, P. J. G.; WONHAM, W. M. **The Control of Discrete Event Systems.** Proceedings of the IEEE. Vol. 77. N. 1. p. 81-98. 1989.

RANGANATHAN, N.; PATEL, M. I.; SATHYAMURTHY, R. **An Intelligent System for Failure Detection and Control in an Autonomous Underwater Vehicle.** IEEE Transactions on Systems, Man and Cybernetics. Vol. 31. N. 6. p. 762-767. 2001.

ROS. **Robot Operating System.** Disponível em <http://www.ros.org/wiki>. Acesso em: outubro, 2014.

ROBERTS, G. N.; SUTTON, R. **Advances in Unmanned Marine Vehicles.** The Institute of Electrical Engineers, IEE. United Kindgom. 442 p. 2006.

ROMAGÓS, D. R. **Underwater SLAM for Structured Environment using an Imaging Sonar.** Tese de doutorado, Universidade de Girona, Espanha. 2008.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Aproach.** 2ª Ed. Prentice Hall. 2003.

SAKAR, N.; PODDER, T. K.; ANTONELLI, G. **Fault-Accomoation Thruster Force Allocation of an AUV Considering Thruster Redundancy and Saturation.** IEEE Transactions on Robotics and Automation. Vol. 18. N. 2. p. 223-233. 2002.

SANGEKAR, M.; CHITRE, M.; KOAY, T. B. **Hardware Architecture for a Modular Autonomous Underwater Vehicle STARFISH.** OCEANS. p. 1 – 8. 2008.

SEIBEL, C. W. **Uma Metodologia Formal para o Planejamento e Controle de Missões de Aeronaves Não-Tripuladas.** Tese de doutorado, Universidade Federal de Santa Catarina. Florianópolis, Novembro, 2000.

SELIC, B. **The Pragmatics of Model-Driven Development.** IEEE Computer Society. p. 19-25. 2003.

SELIG, J. M. **Introductory Robotics.** Prentice Hall. 1992.

SENGUPTA, R.; LAFORTUNE, S. **An Optimal Control Theory for Discrete Event Systems.** SIAM Journal on Control and Optimization. V. 36. N. 2. p. 488-541. 1998.

SHAW, M.; GARLAN, D. **Formulations and Formalisms in Software Architecture. Computer Science Today: Recent Trends and Developments.** Vol. 1000. p. 307-323. 1995.

SHIXIANJUN; JIAKUN, S.; HONGXING, L. **Hardware-in-the-loop Simulation Framework Design for a UAV Embedded Control System.** Proceedings of the 25th Chinese Control Conference. p. 1890-1894. 2006.

SICILIANO, B.; SCIAVICCO, L.; VILLANI, L.; ORIOLO, G. **Robotics: Modelling, Planning and Control.** Springer. 2009.

SIEGWART, R.; NOURBAKHSH, I. R. **Introduction to Autonomous Mobile Robots.** MIT Press. Londres. 2004.

SNAME. **Nomenclature for Treating the Motion of a Submerged Body Through a Fluid.** Society of Naval Architects and Marine Engineers. Technical and Research Bulletin. New York, USA. N. 1-5. 1950.

SOUZA, E. C. **Modelagem e Controle de Veículos Submarinos Não Tripulados.** Dissertação de mestrado, Universidade de São Paulo. São Paulo, 2003.

STANCEL, E.; STOIAN, I.; GYURKA, B.; POSTEUCA, C.; MOLDOVAN, A. **Intelligente System for Underwater Inspection Based on Digital Camera.** IEEE International Conference on Automation, Quality and Testing, Robotics. 2006.

STENTZ, A. **The Focussed D* Algorithm for Real-Time Replanning.** Proceedings of the International Joint Conference on Artificial Intelligence. p. 1652-1659. 1995.

SU, R.; WONHAM, W. M. **Supervisory Reduction for Discrete Event Systems.** Conference on Information Sciences and Systems. p. 786-791. 2001.

SWERE, E.; MULVANCY, D.; SILLITOE, I. **Efficient Incremental Decision Tree Generation for Embedded Applications.** IEEE Conference on Cybernetics and Intelligent Systems. p. 1101-1106. 2004.

TANGIRALA, S.; KUMAR, R.; BHATTACHARYYA, S.; O'CONNOR, M.; HOLLOWAY, L. E. **Hybrid-Model Based Hierarchical Mission Control Architecture for Autonomous Underwater Vehicles.** American Control Conference. Portland, EUA. p. 668-673. 2005.

TAVARES, A. M. **Um Estudo sobre a Modelagem e o Controle de Veículos Subaquáticos Não Tripulados.** Dissertação de mestrado, Rio Grande. 2003.

UBUNTUARM. Disponível em <http://www.ubuntu.com/download/server/arm>. Acesso em: outubro, 2014.

VALAVANIS, K. P.; MATIJASEVIC, M.; KOLLURU, R.; DEMETRIOU, G. A. **Control Architecture for Autonomous Underwater Vehicles.** IEEE Control Systems. p. 48 – 64. Dezembro, 1997.

WANG, F. Y.; KYRIAKOPOULOS, K. J.; TSOLKAS, A.; SARIDIS, G. N. **A Petri-Net Coordination Model for an Intelligent Mobile Robot.** IEEE Transactions on Systems, Man and Cybernetics, vol. 21, n. 4. p. 777 – 789. July/August 1991.

WANG, X.; MALLAPRAGADA, G.; RAY, Asok. **Language-mesaure-based Supervisory Control of a Mobile Robot.** American Control Conference. Portland, USA. p. 4897-4902. 2005.

WEHRMEISTER, M. A. **An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Sytems.** Tese de Doutorado, Universidade Federal do Rio Grande do Sul e Universidade de Paderborn. 2009.

WILLIAMS, S. B.; NEWMAN, P.; DISSANAYAKE, G.; ROSENBLATT, J.; DURRANT-WHYTE, H. **A Descoupled, Distributed AUV Control Architecture.** Proceedings of 31st International Symposium on Robotics. p. 246-251. 2000.

WILLNER, Y; HEYMANN, M. **Supervisory Control of Concurrent Discrete-event Systems.** International Journal of Control. Vol. 54. N. 5. p. 1143-1169. 1991.

XIANG, X.; XU, G.; ZHANG, Q.; XIAO, Z.; HUANG, X. **Coordinated Control for Multi-AUV Systems based on Hybrid Automata.** Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics. Sanya, China. p. 2121-2126. Dezembro, 2007.

XIE, M. **Fundamentals of Robotics: Linking Perception to Action.** World Scientific. Cingapura. 2003.

XU, H.; ZHANG, Y.; FENG, X. **Discrete Hierarquical Supervisory Control for Autonomous Underwater Vehicle.** IEEE. p. 417 – 421. 2004a.

XU, H.; ZHANG, Y.; FENG, X. **Research on the Descentralized Supervisory Control of Autonomous Underwater Vehicles.** Proceedins of the 5th World Congress on Intelligent Control and Automation. Hanghou, China, p. 4909 – 4913, 2004b.

YANG, K. C.; YUH, J. CHOI, S. K. **Experimental Study of Fault-Tolerant System Design for Underwater Robots.** Proceedings of the IEEE International Conference on Robotics & Automation. Bélgica. p. 1051-1056. 1998.

YUH, J. **Underwater Robotics.** Proceedings of the IEEE International Conference on Robotics & Automation. San Francisco, EUA. p. 932 – 937. April, 2000.

ZHANG, Y.; TIAN, J.; SU, D.; WANG, S. **Research on the Hierarquical Supervisory Control of Underwater Glider.** Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. Beijing, China. p. 5509 - 5513. 2006.