DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de uma Plataforma de Software Embarcado para Aerogeradores com Aerofólios Cabeados

Monografia submetida à Universidade Federal de Santa Catarina
como requisito para a aprovação na disciplina
DAS 5511: Projeto de Fim de Curso

Matheus Krüger Winter

Florianópolis, Agosto de 2017

Desenvolvimento de uma Plataforma de Software Embarcado para Aerogeradores com Aerofólios Cabeados

Matheus Krüger Winter

Esta monografia foi julgada no contexto da disciplina
DAS5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação

Prof. Dr. Rômulo Silva de Oliveira

_____

Assinatura do orientador

Banca examinadora:

Eng. Eduardo F. Schmidt
Orientador na empresa

Dr. Rômulo Silva de Oliveira
Orientador no curso

Me. Fernando S. Gonçalves
Avaliador

Eng. Gustavo Kerezi
Debatedor

Eng. Leonardo Rubin Quaini
Debatedor

AGRADECIMENTOS

A Deus, que me conduziu até aqui de forma singular.

À minha família, em especial aos meus pais, Milton Winter e Edelgard Eliane Krüger Winter, que me ensinaram desde cedo o caminho em que deveria andar, por seu esforço, apoio e amor incondicional durante todos esses anos. E às minhas irmãs, Caroline Krüger Winter e Lauren Krüger Winter, pela companhia, conselhos e companheirismo ao longo desta jornada.

Ao laboratório *UFSCkite*, pela estrutura e pela oportunidade de fazer parte da equipe. Um agradecimento especial ao coordenador do projeto Prof. Alexandre Trofino Neto, e aos engenheiros responsáveis Marcelo De Lellis e Ramiro Saraiva, pela orientação profissional, por todos os ensinamentos e pela confiança em mim depositada.

Ao Prof. Rômulo Silva de Oliveira, pela orientação, pelos conhecimentos transmitidos e por todas as contribuições fundamentais à realização e aperfeiçoamento deste trabalho e documento.

Aos meus amigos e colegas de trabalho Eduardo Fensterseifer Schmidt e Lucas Napoleão Coelho, por todo o seu esforço e competência na tomada de decisões durante o desenvolvimento deste trabalho, que foram cruciais ao resultado obtido. Assim como pelos momentos de descontração e conversas que certamente moldaram o seu desenvolvimento.

À Universidade Federal de Santa Catarina, ao Departamento de Automação e Sistemas (DAS), e a todos os professores do curso de graduação em Engenharia de Controle e Automação, que me capacitaram e fomentaram meu interesse pela engenharia ao longo da minha formação acadêmica.

Resumo

O grande crescimento da demanda energética global previsto para as próximas décadas tem fomentado cada vez mais o desenvolvimento e aprimoramento de fontes de energias renováveis. Estudos realizados sobre o potencial de extração de energia eólica em altas altitudes apresentaram resultados extremamente promissores quanto ao potencial energético e custos para a produção de energia elétrica, alavancando o desenvolvimento de tecnologias para tal, como a de aerogeradores com aerofólios cabeados (*Airborne Wind Energy (AWE)*, em inglês). Nesse o contexto, o laboratório *UFSCkite* estuda e desenvolve protótipos visando a extração de energia através desta tecnologia, utilizando sistemas eletro-mecânicos e avançadas técnicas de controle, estimação e instrumentação. Protótipos de sistemas aerogeradores com aerofólios cabeados são dispositivos mecatrônicos complexos, envolvendo aspectos multidisciplinares, e para os quais ainda não existem diretrizes de projeto estabelecidas. Portanto, grupos de pesquisa e empresas novatas no campo são obrigados a comprar soluções genéricas de preço elevado de *software* e *hardware*, ou a construir suas próprias soluções desde o princípio. Como uma tentativa de mitigar esse problema, este trabalho propõe uma plataforma de *software* embarcado sobre a qual aplicações de *AWE* possam ser desenvolvidas. Implementada principalmente na linguagem C de programação e visando inicialmente a sua utilização em computadores de placa única de baixo custo operando com sistemas operacionais *Linux*, a plataforma proposta foi projetada de forma a permitir a decomposição da aplicação de *AWE* em módulos funcionais altamente desacoplados. Estes são executados de maneira distribuída na forma de processos indepedentes, possivelmente sobre múltiplas unidades computacionais, e são capazes de trocar informações através de uma infraestrutura de comunicação padronizada de alta-performance, baseada no padrão *publisher-subscriber*. Além de fornecer um conjunto cuidadosamente selecionado de dependências, as quais são coletadas e instaladas durante uma etapa de construção automática, a plataforma também oferece uma série de mecanismos ao desenvolvedor, incluindo ferramentas de acionamento remoto, monitoramento em tempo real, registro de dados, depuração e instrumentação de código. Através da sua aplicação no protótipo de *AWE* do grupo *UFSCkite*, e da realização de experimentos em laboratório e em campo, onde seus aspectos funcionais e requisitos impostos foram devidamente testados, concluiu-se que a plataforma proposta é capaz de auxiliar o processo de desenvolvimento e operação de protótipos de *AWE*.

Palavras-chave: Sistemas de energia renovável. Protótipos de Aergoreradores com Aerofólios cabeados. Sistemas embarcados. Plataformas de *Software*

Abstract

The increasing global energetic demand forecasted for the next decades has been increasingly promoting the further development and improvement of renewable energy sources. Studies focused on high-altitude wind energy extraction potential report promising results regarding energy production and its costs, leveraging the development of technologies to enable such extractions, such as airborne wind energy (AWE). In this context, the UFSCkite laboratory studies and develops such technologies, by utilizing eletromechanical systems and advanced control, estimation and instrumentation techniques. Airborne wind energy prototypes are complex mechatronic devices involving many multidisciplinary aspects and for which there are currently no established design guidelines. Therefore, newcomers to the field are required to either purchase expensive software and hardware not specifically designed for AWE applications or build their own solutions from scratch. As an attempt to partially overcome this challenge, this work propoes an embedded software platform on top of which AWE systems can be developed. Written mainly in C, and initially targeting low cost single-board computers running Linux, the proposed platform is designed in such a way that it allows for the AWE system to be split into highly decoupled functional modules. These run in a distributed fashion as independent processes, possibly across several computational units, and are capable of exchanging information through a standard, high-performance communication infrastructure based on the publisher-subscriber pattern. Besides providing a carefully chosen set of dependencies, which are fetched and installed during an automatic build phase, the platform also provides a series of facilities to the developer, including remote deployment, real-time monitoring, logging, code instrumentation and debug tools. The designed platform is suitable for application in AWE prototypes and could indeed benefit its growing community, especially in what comes to module reuse and sharing. Through its application in an *AWE* protype developed by the *UFSCkite* group, and the execution of exeperimental procedures both in a lab environment and in real-world conditions, where all its requirements and functional aspects were properly tested, it was concluded that the proposed software plataform is capabled of aiding the development process and the operation of *AWE* prototypes.

Keywords: Renewable energy systems. Airborne Wind Energy prototypes. Embedded systems. Software platforms

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A expansão econômica e populacional de países emergentes, avanços tecnológicos e questões ambientais são alguns dos fatores que influenciam as presentes mudanças no cenário energético global, principalmente em relação ao aumento da demanda por produção de energia elétrica. A agência internacional de energia (*International energy agency (EIA)*, em inglês) prevê em sua publicação mais recente [1], considerando seu cenário principal de projeções, que cerca de 60% do crescimento da demanda energética até 2040 será suprido por fontes renováveis de energia, especificamente, as fontes hídrica, solar e eólica.

Nesse contexto de expansão destaca-se a tecnologia de energia eólica com aerofólios cabeados (*Airborne Wind Energy (AWE)*, em inglês), que visa a extração de energia de ventos em elevadas altitudes a uma fração do custo das turbinas eólicas convencionais. Apesar de promissora, a tecnologia de *AWE* encontra-se em um estágio intermediário de desenvolvimento, e diversos desafios devem ser superados para que chegue ao mercado de forma competitiva. Esta monografia apresenta uma contribuição para um desses desafios, que é o desenvolvimento de *software* para protótipos *AWE* de forma eficiente e organizada.

Este capítulo introduz um breve panorama do cenário energético global, incluindo seu estado atual e as perspectivas de transformação para as próximas décadas, seguido por uma introdução ao conceito de sistemas de *AWE* e seus desafios, assim como por uma apresentação do laboratório onde este trabalho foi conduzido. Em seguida, o objetivo do trabalho é formalmente descrito ao leitor, em conjunto com a metodologia empregada no seu desenvolvimento. Finalmente, a última seção provê uma visão geral da estrutura do documento e seus capítulos.

1.1 Panorama do cenário energético global

O conjunto de matrizes primárias utilizado para a geração de energia elétrica sofreu grandes transformações ao longo das últimas décadas. Apesar das mudanças em direção à utilização de fontes renováveis em decorrência das crescentes preocupações ambientais sobre os efeitos da emissão de gases do efeito estufa, os combustíveis

fósseis continuam sendo a principal fonte da energia primária utilizada no planeta [2]. Segundo um estudo recente [1] publicado pela agência internacional de energia, a demanda energética global deverá crescer cerca de 30% até 2040. Enquanto que a demanda por eletricidade deverá crescer até 65% nesse período, e cerca de 85% desse acréscimo será decorrente da expansão econômica e populacional de países emergentes [3]. Contudo, suprir tal demanda irá requerer grandes esforços no que tange questões ambientais e econômicas relacionadas à produção energética.

O Tratado de Paris, o qual entrou em vigor em novembro de 2016, foi um grande passo em direção a um futuro renovável. Através do estabelecimento da meta de manter o aumento da temperatura do planeta abaixo de 2^oC em relação ao período pré-industrial, as entidades governamentais envolvidas devem gradualmente alterar suas políticas de subsídios, a fim de fomentar a utilização e desenvolvimento de fontes com baixa emissão de carbono. Segundo projeções da própria agência internacional de energia, tal cenário irá alavancar a expansão da indústria de energias renováveis, que será responsável por atender cerca de 60% do crescimento da demanda energética até 2040.

Nesse contexto, destacam-se as energias solar e eólica. A Figura 1.1 evidencia o fato de que apesar de utilizar-se apenas uma fração de sua capacidade, a energia eólica possui um enorme potencial energético ainda não explorado, sendo capaz inclusive de atender a demanda energética global [4]. Contudo, a tecnologia atual de turbinas eólicas possui diversas limitações, tornando-a menos competitiva em relação a usinas termoelétricas, por exemplo. Devido a suas estruturas de grande porte, a sua construção geralmente requer um investimento financeiro considerável, o que eleva diretamente o custo de produção energético de parques eólicos. Além disso, a densidade energética por kilômetro quadrado obtida atualmente é cerca de 200–300 vezes menor do que a obtida em grandes usinas termoelétricas de mesma capacidade de geração, resultando em um alto índice de ocupação de terras e impactos ambientais por parte destes parques [4].

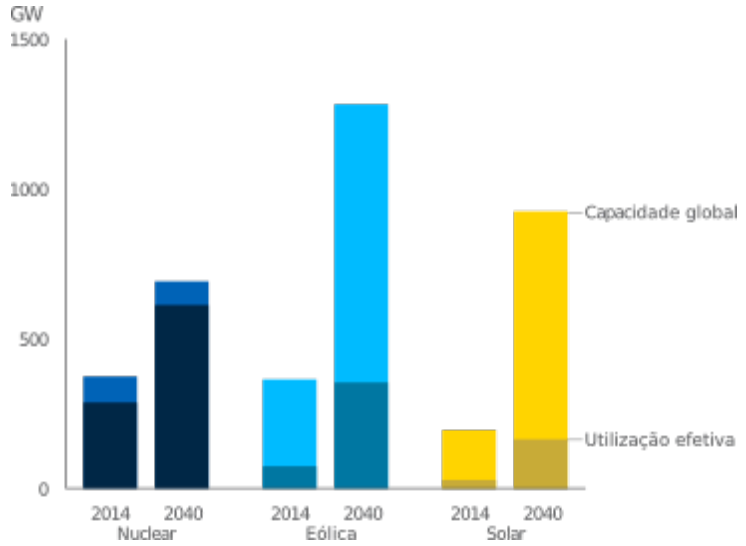


Figura 1.1: *Capacidade de geração energética global e utilização efetiva das fontes nuclear, eólica e solar em GW nos anos de 2014 e sua previsão para 2040.*

Fonte: Adaptado de [3]

1.2 Aerogeradores com aerofólios cabeados

No contexto de fontes renováveis para geração de energia elétrica, o termo *Airbone Wind Energy System (AWES)* surgiu para descrever uma classe emergente de aerogeradores. Tais sistemas usualmente empregam aeronaves ou aerofólios cabeados a fim de alcançar ventos em camadas atmosféricas inacessíveis às turbinas eólicas, e possuem dois componentes principais, uma estação de solo e um dispositivo aéreo, os quais são mecanicamente ou até mesmo eletricamente conectados através de um ou mais cabos [5]. Dentre as diferentes configurações existentes, podemos distinguir sistemas de geração em solo, onde a conversão da energia mecânica em energia elétrica ocorre no solo, e sistemas de geração aérea, onde a conversão ocorre na própria aeronave e a eletricidade é transmitida ao solo através de um cabo. A partir desses dois grupos existem muitas outras possíveis ramificações, as quais variam de acordo com o tipo e número de aeronaves, os conceitos de controle, e a presença ou ausência de movimento na estação de solo. A Figura 1.2 apresenta alguns dispositivos aerogeradores com aerofólios cabeados já desenvolvidos.

Além de alcançar altitudes mais elevadas em comparação à tecnologia eólica tradicional, onde o vento é notoriamente mais constante e forte, conforme demonstrado na Figura 1.3, a tecnologia de *AWE* permite a redução de custos na construção e instalação de usinas de energia. Tais características acabam por simplificar a busca por novos locais de instalação, o que a torna ainda mais atraente. A Figura 1.4 apresenta

uma comparação do custo nivelado de energia[1] com base na velocidade do vento entre uma turbina eólica convencional de 100 kW e um sistema de *AWE* de potencia nominal equivalente, desenvolvido pela empresa alemã EnerKite, evidenciando o seu potencial.

(a) **(b)**

(c)

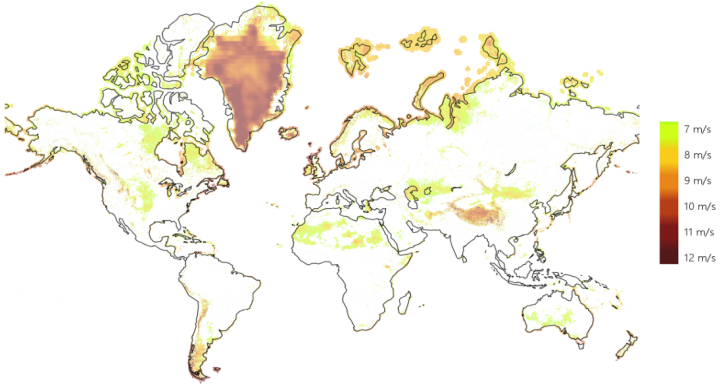
Figura 1.2: *Exemplos de sistemas aerogeradores com aerofólios cabeados. (a) Empresa Enerkite; (b) Empresa Kite Power Systems (KPS); (c) Empresa Makani Power.*

Fonte: Adaptado de [6–8]

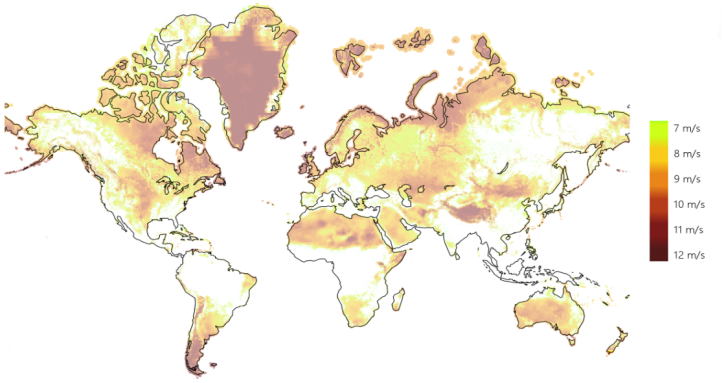
1.3 Laboratório *UFSCkite*

O laboratório *UFSCkite*, localizado na Universidade Federal de Santa Catarina (UFSC), é voltado ao estudo de tecnologias de *AWE* como alternativa para geração de energia elétrica. Desde o início de suas atividades, em meados de 2012, o grupo estuda aspectos teóricos e práticos relacionados ao projeto e ao desenvolvimento da

[1] O custo nivelado de energia (*Levelized cost of energy (LCOE), em inglês*) é comumente utilizado como uma medida de competitividade entre diferentes tecnologias de geração de energia elétrica. O *LCOE* representa o custo por KWh da construção e operação de uma usina de energia considerando sua vida útil, incluindo fatores como custos de capital, de combustíveis, de manutenção, de operação, de financiamento, de incentivos, de seguros, assim como impostos e inflação.



(a) 50 m



(b) 200 m

Figura 1.3: *Disponibilidade e velocidade média dos ventos em (a) 50 m; e (b) 200 m de altitude. Cores mais escuras representam regiões com maior velocidade média.*

Fonte: [9]

tecnologia. A longo prazo, o *UFSCkite* tem como objetivo torná-la viável a ponto de ser comercializada, contribuindo para a diversificação da matriz energética nacional [10].

O grupo possui hoje mais de sete publicações internacionais de destaque, e sua equipe conta com mais de dez integrantes, entre alunos de graduação e pós-graduação. A infraestrutura do grupo inclui um laboratório equipado com estações de trabalho individuais, instrumentos eletrônicos de bancada, e dois protótipos utilizados para teste das técnicas de estimação e controle desenvolvidas, visando, finalmente, a geração de energia elétrica de forma autônoma.

1.4 Motivação

Conforme supracitado, ventos em elevadas altitudes representam um recurso promissor na perspectiva da produção de energia elétrica sustentável. Tal característica estabelece a tecnologia de *AWE* entre as principais emergentes no ramo de ener-

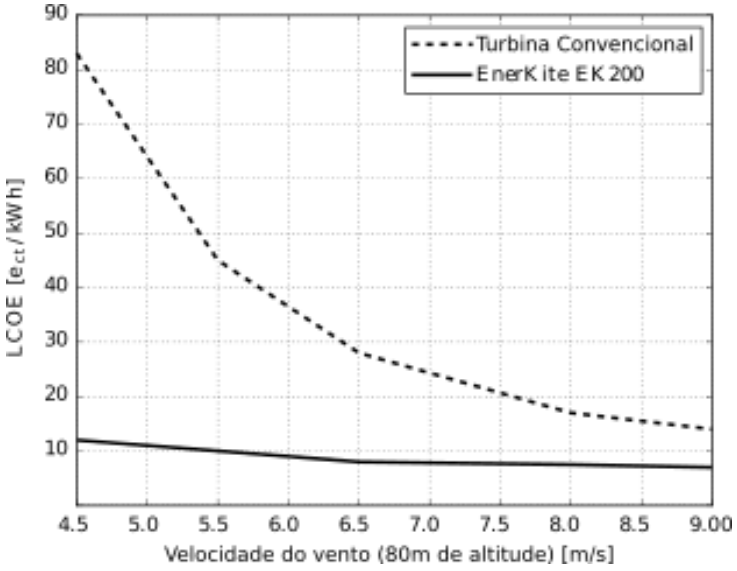


Figura 1.4: *Custo nivelado de energia, em $€_{ct}/kWh$, de uma turbina eólica de 100 kW e do sistema AWE EK200 de potência nominal equivalente, conforme anunciado pela empresa alemã EnerKite.*

Fonte: Adaptado de [6]

gias renováveis, o que é evidenciado pela rápida aceleração do desenvolvimento da mesma. Na última década, o ingresso de diversas empresas ao setor garantiu a criação de patentes, assim como o desenvolvimento de soluções técnicas para a sua implementação, em sua maioria na forma de protótipos. Atualmente, empresas e grupos de pesquisa ao redor do mundo estão trabalhando de forma a aperfeiçoar a tecnologia, abordando aspectos como controle, eletrônica e *design* [5].

Contudo, apesar de promissora em termos de viabilidade econômica, conforme apontado por [11], a tecnologia de aerogeradores com aerofólios cabeados ainda se encontra em estágio de desenvolvimento, com muitos aspectos a serem abordados a fim de torná-la viável comercialmente. Além dos desafios envolvendo o material dos cabos, modelagem aerodinâmica e de aerofólios, instrumentação, atuação e técnicas de controle, uma das grandes dificuldades enfrentadas no seu desenvolvimento é a própria realização de *setups* experimentais [12].

Os protótipos de sistemas de *AWE* são dispositivos mecatrônicos complexos envolvendo noções multidisciplinares [13], para os quais ainda não existem diretrizes de desenvolvimento estabelecidas, obrigando empresas ou grupos de pesquisa que desejam ingressar nesse setor a comprar soluções *off-the-shelf* genéricas de preço elevado de *software* e *hardware* [14], ou a desenvolver as suas próprias soluções desde o princípio. Dada a importância da prototipagem no ambiente de *AWE*, esse cenário impõe uma barreira de entrada não apenas técnica mas também econômica, retardando

resultados experimentais e limitando a cooperação entre diferentes organizações.

Ademais, o processo de desenvolvimento de *software* destas aplicações é complexo. Em adição às inúmeras configurações existentes e aos desafios de controle impostos pela natureza dos sistemas, mudanças em requisitos, em características físicas e em condições de operação de protótipos em desenvolvimento são comuns. Assim, exige-se um grande esforço a fim de manter o processo de prototipagem organizado e flexível sem afetar seu tempo ou custo de execução. De maneira análoga ao sistema operacional de robôs (*Robot Operating System (ROS)*, em inglês) [15], o qual surgiu de forma a mitigar problemas de complexidade e grande variedade de configurações no desenvolvimento de aplicações robóticas complexas, a criação de uma plataforma de *software* padronizada se torna atraente para simplificar e tornar mais eficaz o processo de prototipagem de sistemas *AWE*, sem torná-lo oneroso economicamente.

1.5 Objetivos

O objetivo deste trabalho é, portanto, desenvolver uma plataforma de *software* para sistemas embarcados capaz de simplificar e agilizar o desenvolvimento de protótipos para sistemas aerogeradores com aerofólios cabeados do grupo *UFSCkite*, fornecendo todo o suporte necessário para tal. Além disso, também compõe o escopo deste trabalho a aplicação da plataforma desenvolvida em um dos protótipos do grupo, de forma a avaliar seu desempenho não apenas em tempo de execução, mas também durante o processo de desenvolvimento. De forma mais específica, os propósitos deste trabalho são:

- Estudar a arquitetura de *software* atual empregada pelo grupo, assim como arquiteturas empregadas em aplicações similares;
- Identificar as características desejadas em uma plataforma de *software* para *AWE*, levantando requisitos necessários à tecnologia com base nos protótipos do grupo *UFSCkite*;
- Projetar uma plataforma de *software* capaz de atender aos requisitos previamente estabelecidos;
- Implementar a plataforma de *software* e aplicá-la no desenvolvimento de *software* embarcado para um dos protótipos do grupo *UFSCkite*;
- Avaliar a performance da plataforma e do *software* desenvolvido para o protótipo em ambientes simulados e em campo, a fim de estabelecer as próximas etapas

no desenvolvimento da mesma, aferindo sua viabilidade de aplicação em futuros protótipos do grupo;

1.6 Metodologia

Todas as etapas do trabalho foram desenvolvidas com total suporte do UFSCkite. O início das atividades envolveu uma intensa interação com os membros mais antigos do grupo, Marcelo de Lellis Oliveira e Ramiro Saraiva, que puderam fornecer detalhes em relação às características da aplicação e do software já existente. Essas informações foram fundamentais para o processo de análise do problema, permitindo definir com uma maior segurança os objetivos do trabalho e os requisitos do sistema. Enquanto isso, o projeto e o desenvolvimento propriamente ditos foram orientados pelo membro Eduardo Schmidt, e contaram com colaboração dos membros Lucas Coelho e Jean Damke, responsáveis pelo desenvolvimento do ambiente de simulação e pela interface de usuário, respectivamente. A forte interação entre a equipe, facilitada por meio da utilização de ferramentas colaborativas como Slack, Trello, Gitlab, e outras, foi determinante para o resultado do trabalho.

Durante a sua realização, utilizou-se uma metodologia *top-down* com ênfase no desenvolvimento de sistemas embarcados, conforme descrito em [16]. Essa metodologia sugere uma abordagem iterativa, em que o processo de desenvolvimento inicia com a análise de requisitos e restrições do sistema, passa por uma fase de projeto de alto-nível, na qual são construídos modelos abstratos do *software*, seguida de uma fase de projeto de engenharia, que envolve o detalhamento dos modelos e garante que estão prontos para serem implementados. Na fase de testes, a última etapa do processo de desenvolvimento, avalia-se se o sistema implementado está pronto para produção, ou se ajustes precisam ser realizados, situação em que o fluxo de projeto retorna à etapa inicial.

1.7 Estrutura do documento

Esta monografia é organizada da seguinte maneira. No capítulo 2, a fundamentação teórica de uma série de tópicos fundamentais à compreensão e desenvolvimento do trabalho são apresentados, incluindo conceitos de sistemas embarcados e arquiteturas de *software*. Em seguida, no capítulo 3, é apresentada ao leitor uma introdução a sistemas de *AWE*, onde são descritas características construtivas dos mesmos, assim como os seus princípios de funcionamento mais usuais. O capítulo 4 expõe a

análise do problema cuja solução é o foco da contribuição deste trabalho, abordando aspectos da arquitetura de *software* e do sistema de *AWE* em questão, assim como os requisitos a serem cumpridos pela plataforma de *software*. No capítulo 5, a plataforma de *software* é proposta, fornecendo detalhes sobre a etapa de projeto da mesma, na qual a arquitetura básica é apresenta em conjunto com suas características funcionais e ferramentas adicionais. O Capítulo 6 descreve o processo de implementação da plataforma, especificando as decisões e ferramentas utilizadas para o cumprimento dos requisitos previamente estabelecidos. No Capítulo 7, a aplicação da plataforma desenvolvida em um protótipo do grupo *UFSCkite* é apresentada, explicitando esse processo e apresentando o resultado de testes em ambiente de laboratório e em campo. Finalmente, o Capítulo 8 relata, de forma sucinta, as conclusões obtidas ao longo do desenvolvimento deste trabalho, descrevendo as tarefas não documentadas que já estão em andamento, e sugerindo possibilidades de trabalhos futuros

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta uma breve introdução aos conceitos essenciais à compreensão deste trabalho. São abordados os tópicos estudados durante a sua elaboração, incluindo aspectos de sistemas cíber-físicos, sistemas de tempo real, arquiteturas e plataformas de software e o sistema operacional *Linux*, assim como comunicação entre processos, sistemas e dispositivos periféricos. Aspectos mais específicos serão apresentados com maior detalhe conforme necessário nos capítulos subsequentes.

2.1 Sistemas ciber-físicos

Sistemas ciber-físicos (*Cyber-physical systems* (*CPSs*), em inglês) são definidos como a integração de sistemas computacionais com processos físicos, cujo comportamento é definido tanto pelas partes cibernéticas quanto físicas do sistema. *CPSs* tratam da intersecção do físico com o cibernético, assim não é suficiente compreender o funcionamento de cada um desses sistemas de forma individual, pois a sua interação também deve ser estudada [17].

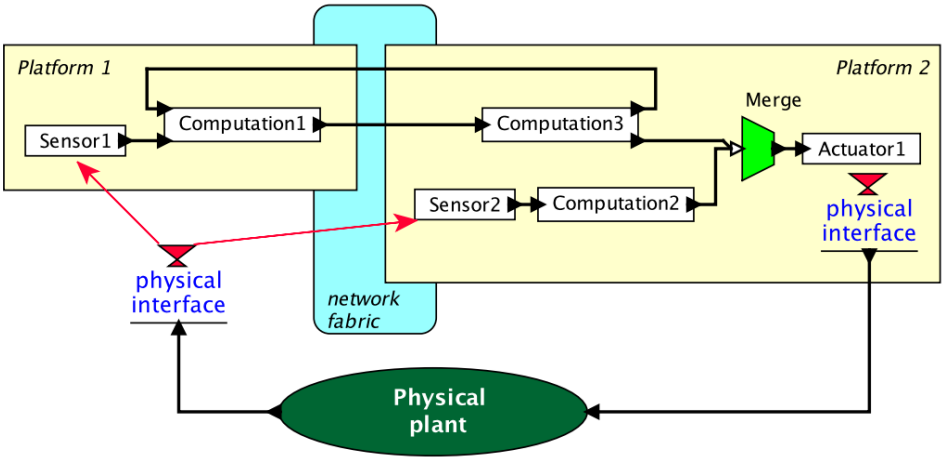


Figura 2.1: *Representação em fluxograma de um sistema ciber-físico, contendo seus elementos e as relações de troca de informação entre eles.*

Fonte: [17]

Os *CPSs* envolvem abordagens multidisciplinares devido à combinação de com-

putação, comunicação e dinâmicas físicas, e são geralmente projetados na forma de uma rede de elementos com saídas e entradas bem definidas, conforme apresentado na Figura 2.1. Um dos maiores desafios presentes no desenvolvimento desses sistemas reside justamente na sua heterogeneidade, pois as diferentes áreas de engenharia envolvidas possuem práticas de projeto e preocupações consideravelmente diferentes. Assim, o projeto de um *CPS* deve considerar os requisitos impostos pelos diferentes subsistemas que o compõe, e ser capaz de explorar as possibilidades de projeto destes de maneira colaborativa, delegando responsabilidades ao elementos físicos e de *software* analisando suas relações.

No contexto de *AWE*, a natureza do processo físico impõe algumas restrições aos seus componentes cibernéticos devido às suas condições de operação, especificamente às unidades de computação de controle de voo. Requerimentos em suas dimensões, peso, consumo de energia, capacidade de processamento e de interação com sistemas periféricos[1] são comuns e acabam por tornar usual a utilização de sistemas embarcados em aplicações de *AWE* [18].

2.1.1 *Sistemas embarcados*

Sistemas embarcados são definidos como dispositivos baseados na utilização de microprocessadores ou microcontroladores que combinam *hardware* e *software* visando a realização de tarefas específicas, em contraste ao propósito generalista presente em computadores pessoais [19]. Utilizados para controlar, monitorar ou auxiliar um sistema, seja esse uma máquina, equipamento ou instalação, os mesmos formam uma classe heterogênea com relação suas características construtivas, as quais são fortemente correlacionadas à sua aplicação.

Usualmente, devido a preocupações com a potência consumida, dimensões físicas, robustez e custo unitário, os mesmos possuem capacidade de processamento e armazenamento limitadas, o que torna o seu desenvolvimento significativamente mais complexo [20]. Contudo, em casos onde o tamanho ou a eficiência energética não são as preocupações principais, seus componentes podem inclusive ser compatíveis aos utilizados em computadores de propósito geral, como é o caso dos chamados computadores de placa única (*Single board computers (SBC), em inglês*)[2]. A vantagem dessa

[1] Nesse caso, o termo periféricos se refere a todos os componentes do sistema à exceção da unidade de computação, como por exemplo sensores, atuadores ou outras unidades de computação, que embora sejam responsáveis por tarefas distintas também integram o sistema ciber-físico.

[2] Usualmente *SBCs* são utilizados de forma embarcada em aplicações de propósito específico, caracterizando-os portanto como sistemas embarcados, apesar de suas características semelhantes a de computadores de propósito geral. São exemplos de *SBCs* os sistemas *Raspberry Pi*, *Beaglebone* ou *Arduino* [21–23]

abordagem reside justamente na possibilidade da utilização de componentes de *hardware* de baixo custo em conjunto com as mesmas ferramentas de desenvolvimento de *software* tradicional, característica a qual é valiosa em processos de prototipagem, por exemplo.

2.1.2 Sistemas de tempo real

Em adição às características acima mencionadas, sistemas embarcados também costumam envolver restrições temporais em suas aplicações, o que os caracteriza como sistemas de tempo real [20]. Sistemas de tempo real são definidos como sistemas onde a exatidão ou correção de uma computação não está relacionada apenas ao seu método de execução, mas também aos tempos que a envolvem. A sua classificação , assim como de seus *deadlines* temporais, se dá através das consequências do não cumprimento de um *deadline*. Em sistemas críticos, um único *deadline* não cumprido significa falha total no sistema, como é o caso de diversas aplicações aviônicas ou militares. Em sistemas não-críticos, a utilidade do resultado é degradada com a perda de um *deadline*, afetando a qualidade do serviço provido, sem causar falhas catastróficas ao sistemas, como é o caso de sistemas telefônicos por exemplo. Uma terceira categoria classifica os sistemas intermediários, onde falhas em *deadlines* não significam necessariamente uma falha total no sistema, mas sim a inutilidade da operação realizada [24].

2.2 Sistemas operacionais

Computadores modernos, inclusive os que compõem sistemas embarcados, são compostos tanto por elementos de *hardware* quanto de *software*. Os elementos de *hardware* são definidos como sendo os componentes físicos do sistema, como por exemplo processadores, memória de armazenamento, dispositivos de I/O (entrada e saída), interfaces de rede e dispositivos periféricos. Já os elementos de *software* são definidos como sendo os componentes lógicos do sistema, os quais controlam o funcionamento do mesmo e, consequentemente, dos elementos de *hardware*. Contudo, tais sistemas podem ser extremamente complexos, o que torna árdua a tarefa de gerenciar o acesso, garantir a correta utilização, e otimizar os recursos de *hardware* disponíveis. Com o intuito de simplificar tal tarefa, surgiram os chamados sistemas operacionais [25].

Os sistemas operacionais (SO) são a primeira camada de *software* que opera sobre o *hardware*. A Figura 2.2 exemplifica tal definição, representando o computador

em camadas de *hardware* e *software*, subdividindo essa em dois modos diferentes. O modo *Kernel* agrupa os elementos de *software* que possuem acesso total a todo o *hardware* do sistema, podendo executar qualquer instrução que a máquina é capaz. O modo de usuário (*user mode*, em inglês) envolve os demais componentes de *software*, como por exemplo programas de aplicação desenvolvidos por usuários, onde apenas um subconjunto de instruções está presente. Portanto, o SO é responsável por desempenhar duas tarefas distintas: fornecer aos programas de aplicação um conjunto de recursos abstrato, limpo e consistente, ao invés do acesso direto aos recursos de *hardware*; e realizar o gerenciamento adequado e eficiente dos mesmos, fornecendo de maneira ordenada e controlada a alocação de processadores, de memória e/ou dispositivos *I/O* aos programas de aplicação em execução [26].

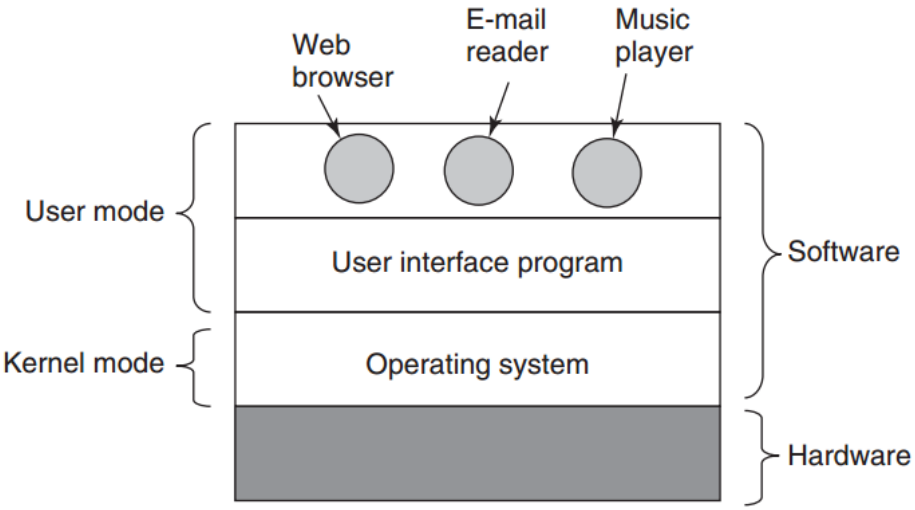


Figura 2.2: *Representação da divisão de um computador em camadas de hardware e software.*

Fonte: [26]

Portanto, o sistema operacional opera justamente como a fundação para o desenvolvimento e execução de aplicações de *software*, proporcionando ao desenvolver liberdade e todo o suporte necessário para tal através de estruturas definidas por chamadas de sistema.

2.2.1 Linux

A filosofia *Unix*[3] surgiu com o propósito de definir um conjunto de regras minimalistas a fim de projetar um sistema operacional compacto e potente com uma interface de serviços simples [26]. Assim, tal filosofia enfatiza a construção de código simples, curto, claro, modular e extensível, o qual pode ser mantido e adaptado por desenvolvedores que o utilizam, e não apenas seus criadores. O *Linux* é um sistema operacional gratuito e de código aberto baseado em *Unix* que foi desenvolvido com o intuito de proporcionar aos seus usuários um ambiente com maior controle, simples, elegante e consistente, de maneira a facilitar o desenvolvimento de aplicações sofisticadas de *software* [28].

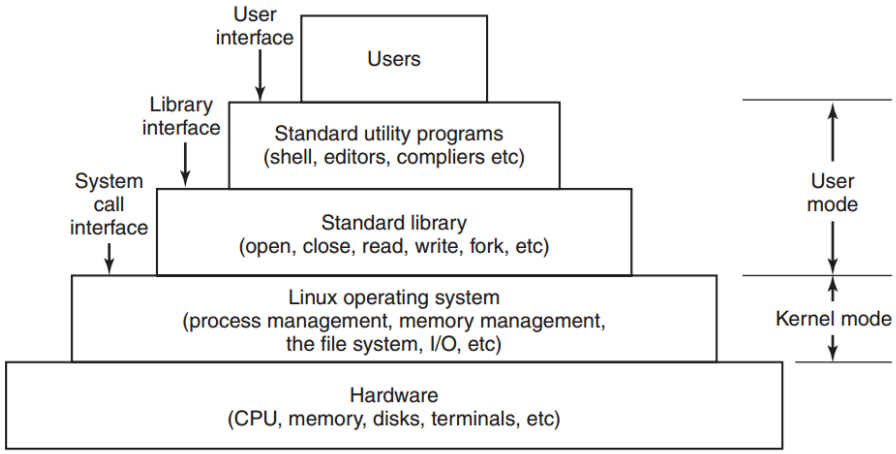


Figura 2.3: *Representação da divisão em camadas de um sistema computacional utilizando o sistema operacional Linux.*

Fonte: [26]

2.2.1.1 Estrutura geral

A Figura 2.3 apresenta a estrutura geral de um sistema *Linux*. A camada de *hardware* contém os elementos físicos do sistema, como processadores e outros dispositivos. Interagindo diretamente com tais elementos se encontra a camada do sistema operacional, o núcleo (*kernel*, em inglês), cuja função é controlar estes elementos e fornecer às camadas superiores uma interface de comunicação que permite aos demais programas a criação e gerenciamento de processos, arquivos e outros recursos de baixo nível [25].

[3] O *Unix* é uma família de sistemas operacionais *multitasking* e multiuso para computadores derivada do sistema *Unix AT&T*, desenvolvido em meados dos anos 1970 [27]

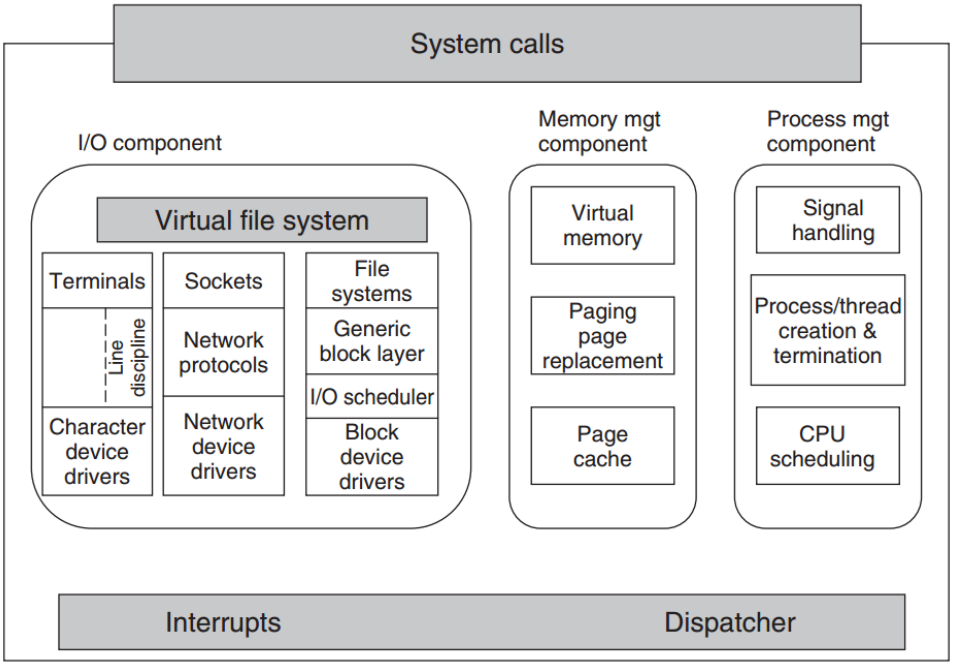


Figura 2.4: *Representação simplificada do kernel Linux.*

Fonte: [26]

O sistema *Linux* faz uso de um *kernel* monolítico, o qual está representado na Figura 2.4 e é descrito detalhadamente em [26]. De maneira simplificada, o mesmo pode ser divido em três módulos principais, os quais são divididos em submódulos: Entrada/Saída(*Input/Output (I/O)*, em inglês), gerenciamento de memória e gerenciamento de processos.

O componente de *I/O* contém os módulos responsáveis pela interação com outros dispositivos e realização de tarefas de rede e armazenamento. Uma camada de abstração denominada sistema de arquivos virtual é utilizada a fim de padronizar o acesso ao diversos tipos de arquivos presentes. O componente de gerenciamento de memória é responsável por manter o mapeamento entre as memórias física e virtual, armazenando e gerenciando de forma eficiente o cache de páginas, neste caso, blocos atômicos de memória virtual. O componente de gerenciamento de processos é responsável basicamente pela criação e término de processos, além do escalonamento do uso do processador e do tratamento de sinais, os quais são provenientes de programas das camadas superior do sistema.

O *kernel Linux* faz uso do tratamento de interrupção para se comunicar diretamente com os dispositivos de *hardware*, ao passo que a comunicação com as camadas de *software* de alto nível é realizada através de chamadas de sistema. Outra característica interessante do *kernel* é o conceito de módulos capazes de serem carregados

durante a execução do sistema, cujo conteúdo varia desde *drivers* de dispositivo até sistemas de arquivos ou protocolos de rede.

2.2.1.2 Sistema de arquivos

Fundamental para o sistema *Linux*, o sistema de arquivos é responsável por implementar um recurso em *software* que não existe no *hardware*. O *hardware* fornece simplesmente espaço em disco, na forma de setores que podem ser acessados individualmente, em uma ordem aleatória. O conceito de arquivo, muito mais útil que um o simples espaço em disco, é uma abstração, um recurso lógico criado pelo sistema operacional a partir dos recursos físicos existentes no sistema computacional [25]. De forma mais específica, sua função é controlar como os dados utilizados pelo sistema são armazenados e acessados.

Arquivos são definidos na literatura como unidades lógicas de informação criadas por processos, ou seja, são mecanismos de abstração que permitem o armazenamento e acesso organizado às informações presentes na memória [25, 26]. Além dos tipos de arquivos convencionais, como os de texto, imagens ou programa compilados, os diretórios, partições e *drivers* de dispositivos de *hardware* também são arquivos no sistema *Linux*. Os diretórios e arquivos são organizados de maneira hierárquica, formando uma espécie de árvore e para escrever ou ler de arquivos, descritores de arquivos são atribuídos a estes. Dispositivos de *I/O* são acessados através de arquivos especiais, contendo as informações necessárias para a identificação dos mesmos.

2.2.1.3 Distribuições

Referenciado como um sistema operacional, o nome *Linux* tecnicamente faz referência apenas ao seu *kernel*, mas, popularmente, este nome designa o sistema operacional como um todo, ou seja, o *kernel*, os programas de sistema e aplicações. O conjunto completo de *software* mais o *kernel Linux* constitui o que se denomina de distribuição *Linux*, como por exemplo as distribuições *RedHat*, *Debian*, *Ubuntu* e *Suse* [25].

Todas essa distribuições fornecem uma versão do *kernel Linux*, um conjunto de *software* e uma interface de instalação. A diferença entre elas reside justamente nesses dois últimos pontos. O conjunto de *software* varia de distribuição para distribuição, assim como seus métodos de instalação e manutenção. A escolha da distribuição mais adequada para o desenvolvimento de uma aplicação depende fortemente da experiência da equipe de desenvolvedores assim como dos requisitos do sistema.

2.2.2 Processos e *threads*

Peças fundamentais para o funcionamento da computação moderna como a conhecemos hoje, os processos são geralmente definidos como instâncias de um programa em execução. Esses podem representar tanto a execução de tarefas do próprio sistema operacional (*daemons*, por exemplo) como tarefas de aplicações, e é através deles que o sistema operacional organiza suas tarefas de gerenciamento [25].

Cada processo é associado ao seu próprio espaço de endereçamento, que nada mais é do que uma lista de endereços na memória disponíveis para serem manipulados, e a um conjunto de recursos e informações necessárias para a execução do programa. Enquanto que o programa em si é composto por uma série de instruções a serem realizadas, o processo pode ser visto como a efetiva execução destas. Tal abstração é fornecida pelo sistema operacional e permite a realização de operações de maneira pseudoconcorrente, por meio do processo de virtualização da unidade central de processamento ou *CPU* (*Central Processing Unit*, em inglês). Através do escalonamento da execução dos diferentes processos ativos, basicamente executando sequencialmente cada um destes por um pequeno período de tempo, o sistema operacional cria a ilusão de que existem CPUs virtuais enquanto que fisicamente existe apenas uma, conforme representado na Figura 2.5. Sistemas que possuem mais de uma CPU são capazes de executar processos de forma verdadeiramente paralela, pois cada CPU processa os dados de forma independente.

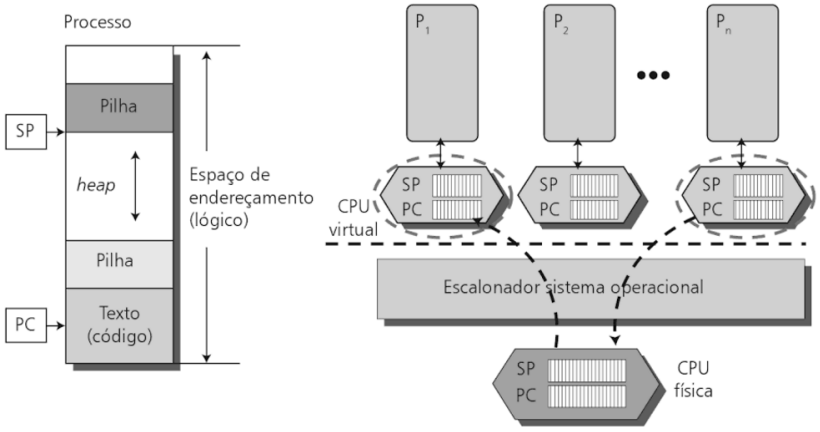


Figura 2.5: *A abstração de processo de um sistema operacional.*

Fonte: [25]

Assim como os processos permitem a abstração da CPU, permitindo múltiplas aplicações operarem de maneira quase paralela, as chamadas *threads* fornecem a mesma funcionalidade ao processo, permitindo com que cada um destes realize múltiplas tarefas sem abrir mão de seu pseudoparalelismo. Essencialmente, *threads* são

subdivisões pertencentes a um processo, as quais compartilham o seu espaço de endereçamento e recursos disponíveis, sendo também escalonadas pelo sistema operacional. Por serem mais leves que os processos, as *threads* são mais fáceis de criar e destruir, e em alguns casos o tempo necessário para criar uma *thread* é na ordem de 10 à 100 vezes menor do que o necessário para a criação de um processo [26].

2.3 Mecanismos de comunicação

Conforme supramencionado, os sistemas ciber-físicos são usualmente compostos por uma ou mais unidades computacionais. Nesse âmbito, o termo periférico se refere a qualquer dispositivo que interage de alguma forma com a unidade computacional, como por exemplo, outras unidades computacionais que compõem o sistema, ou dispositivos de *hardware* na forma de sensores ou atuadores. Assim, de forma a desempenhar suas funções específicas, uma unidade computacional, através de mecanismos e interfaces de comunicação, deve ser capaz de garantir a troca de informações de forma organizada e controlada.

De forma análoga ao *CPSs*, a unidade computacional é composta por diversos processos distintos, os quais possibilitam a realização de tarefas de forma pseudoparalela utilizando de fato apenas uma única *CPU* física. Considerando que a grande maioria das aplicações computacionais não triviais envolve a presença e interação de múltiplos processos, a definição e análise de mecanismos de comunicação entre esses também é de suma importância.

2.3.1 Comunicação entre processos

O termo comunicação entre processos (*Interprocess Communication* (*IPC*), em inglês) tradicionalmente se refere aos diferentes mecanismos para troca de mensagens entre os diferentes processos em execução em algum sistema operacional. A Figura 2.6 apresenta três princípios básicos fornecidos pelo sistema *Linux* para tal: compartilhamento de informação por meio de um arquivo que reside no sistema, o qual é acessado através do *kernel*; compartilhamento de informação que reside no *kernel* do sistema; ou uma região de memória compartilhada entre os processos [29].

Os mecanismos de IPC são classificados baseando-se em requerimentos do *software*, como performance ou modularidade, e em circunstâncias do sistema, como largura de banda e latência. Entre si, eles diferem-se com base em restrições de comunicação, de manipulação dados e de número de processos suportados pelo mecanismo.

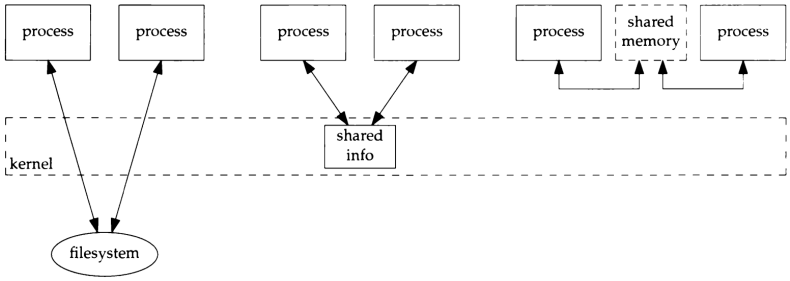


Figura 2.6: *Três maneiras básicas de realizar a comunicação entre processos no sistema Linux.*

Fonte: [29]

2.3.1.1 Arquivos

Processos podem realizar a troca de informações através do compartilhamento de informações contidas em algum arquivo do sistema de arquivos. Para isso, os diferentes processos que desejam acessar e manipular os arquivos utilizam uma interface de comunicação provida pelo *kernel* do sistema. Contudo, se faz necessária a sincronização do acesso ao arquivo compartilhado, de maneira a proteger a integridade dos dados em situações de múltiplos processos realizando ações de escrita, ou de um processo realizando ação de leitura enquanto outro realiza ação de escrita.

2.3.1.2 Soquetes de domínio *Unix*

Em se tratando do sistema operacional *Linux*, tais *soquetes* (*sockets*, em inglês) são pontos finais de comunicação para a troca de dados localmente entre processos executando no sistema operacional. Além de suportar a transmissão ordenada e sequencial de um fluxo de dados, sem a garantia de delimitação entre as mensagens, os *sockets* suportam também a transmissão de datagramas, sem a garantia de sequência ou entrega dos mesmos, porém garantindo a delimitação entre as mensagens. A identificação do *socket* se dá através de um caminho de diretório, e todas as comunicações ocorrem por intermédio do *kernel* do sistema operacional. Para compartilhar os soquetes e estabelecer a comunicação, os processos se referem aos mesmos através de *inodes*, que são estruturas de dados que descrevem um objeto do sistema de arquivos.

2.3.1.3 Fila de mensagens

A fila de mensagens pode ser vista como uma lista encadeada de mensagens, a qual permite a comunicação assíncrona entre múltiplos processos através de ações

de escrita e leitura. Esta oferece espaço dentro do sistema para o armazenamento das mensagens até que o receptor ou receptores estejam prontos para lê-las, sem limitar o tamanho das mensagens a serem transmitidas. Por ser assíncrono, não é necessário aos processos interagir ao mesmo tempo com a fila de mensagens, e a existência da fila permite com que os mesmos não dependam de uma conexão direta para se comunicar.

2.3.1.4 Pipes

Em sistemas *Linux*, *pipes* são canais de transmissão de dados unidirecionais os quais utilizam o sistema de entrada e saída de arquivos do sistema operacional. Os *pipes* são capazes de conectar processos através de suas saídas padrão, pré-conectadas no início da execução dos mesmos. Assim, a conexão é possível apenas entre processos diretamente relacionados através de um processo ancestral em comum. Os *pipes* nomeados, através da utilização caminhos de diretório para sua identificação, possibilitam a comunicação de processos não relacionados. Apesar de serem fundamentalmente unidirecionais, a criação de dois *pipes* distintos possibilita a comunicação bidirecional entre processos.

2.3.1.5 Memória compartilhada

A comunicação através de memória compartilhada é a forma mais rápida de comunicação entre processos disponível. Uma vez que a memória é mapeada para a região de endereçamento dos processos os quais irão compartilhá-la, o *kernel* não tem mais papel ativo na troca de informação entre os processos. Contudo, tal mecanismo usualmente exige a presença de alguma forma de sincronização e coordenação entre os processos que estão armazenando e buscando informação na região de memória compartilhada.

2.3.2 Interfaces de Entrada e Saída

As interfaces de entrada e saída de um sistema são responsáveis pela comunicação deste com dipositivos periféricos, sejam estes dispositivos de instrumentação ou outras unidades computacionais. Tais interfaces de comunicação geralmente utilizam as camadas de rede ou as camadas físicas do sistema [26]. A seguir são descritos alguns exemplos de protocolos.

2.3.2.1 UDP

O *User Datagram Protocol* (UDP) é um protocolo simples da camada de transporte o qual permite que a aplicação escreva um datagrama encapsulado num pacote, e então enviado ao destino. Um datagrama é uma entidade de dados completa e independente que contém informações suficientes para ser roteada da origem ao destino sem precisar confiar em trocas anteriores entre essa fonte, a máquina de destino e a rede de transporte. Mas não há qualquer tipo de garantia que o pacote irá chegar ou não.

Caso garantias sejam necessárias, é preciso implementar uma série de estruturas de controle, tais como timeouts, retransmissões, *acknowledgements* ou controle de fluxo, etc. Cada datagrama UDP tem um tamanho e pode ser considerado como um registro indivisível. Sendo um serviço sem conexão, não há necessidade de manter um relacionamento longo entre cliente e o servidor. Assim, um cliente UDP pode criar um *socket*, enviar um datagrama para um servidor e imediatamente enviar outro datagrama com o mesmo socket para um servidor diferente. Da mesma forma, um servidor poderia ler datagramas vindos de diversos clientes, usando um único socket.

2.3.2.2 TCP

O *Transmission Control Protocol* (TCP) é um protocolo orientado a conexão que provê um fluxo de dados confiável e ordenando entre dois elementos conectados à mesma rede. Para realizar a transmissão entre dois elementos da rede, o protocolo *TCP* define três etapas durante o processo: estabelecimento de conexão, transferência de dados e término de conexão. Cada etapa é projetada de maneira a garantir a robustez e a confiabilidade do protocolo, seja através de confirmações e autenticações da conexão na etapa de estabelecimento ou através de mecanismos como códigos detectores de erros na etapa de transmissão. Assim, o protocolo *TCP* garante que todos os dados recebidos são idênticos aos enviados, inclusive com relação ao seu ordenamento. Devido à suas características, o protocolo TCP é indicado para aplicações sensíveis a falhas na comunicação, e em situações em que características temporais não são críticas ao seu funcionamento.

2.3.2.3 *WebSocket*

WebSocket é uma tecnologia que permite a comunicação bidirecional por canais *full-duplex* sobre um único *socket Transmission Control Protocol*. Ele é projetado para ser executado em browsers e servidores web que suportem o *HTML5*, uma linguagem

para estruturação e apresentação de conteúdo para internet, mas pode ser usado por qualquer cliente ou servidor de aplicativos.

2.3.2.4 *UART*

Um *Universal Asynchronous Receiver/Transmitter (UART)* é um dispostivo de *hardware* capaz de se realizar comunicação serial assíncrona onde o formato dos dados e as velocidades de transmissão são configuráveis. Em conjuntos com os *UARTs* geralmente são utilizados padrões para definir características elétricas de *drivers* e receptores usados na comunicação serial, dado que os *UARTs* utilizam interfaces separadas para converter seus sinais lógicos de e para os níveis de sinais externos.

Usualmente, *UARTs* são um, ou parte de um, circuito integrado, e funcionam através da coleta e transmissão de *bytes* de dados como *bits* individuais de forma sequencial. Ao chegar no destino, um segundo *UART* reconstrói os bits recebidos formando os *bytes* de dados originais.

2.3.2.5 *SPI*

Serial Peripheral Interface (SPI) é uma especificação de interface de comunicação serial síncrona usada para comunicação de curta distância onde o formato e tamanho das mensagens são configuráveis. Os dispositivos SPI comunicam entre si através de canais *full-duplex* usando uma arquitetura mestre-escravo com um único mestre. O dispositivo mestre origina o quadro para a leitura e a escrita. Múltiplos dispositivos escravos são suportados através de selecção com linhas de selecção de escravos individuais, denominadas *SS*.

2.3.2.6 *I2C*

O *Inter-integrated Circuit (I2C)* é um barramento serial multimestre para a conexão de periféricos com baixa velocidade comunicação. O *I2C* faz uso de canais bidirecionais, uma linha serial de dados e uma linha serial de *clock*. No *I2C* são definidos dois papéis para os nós: mestre ou escravo. Os mestres são responsáveis por gerar o sinal de *clock* e iniciar a comunicação com os escravos, que por sua vez recebem o sinal de *clock* e respondem quando endereçados por um mestre.

O *I2C* também define alguns tipos básicos de mensagens, as quais sempre possuem um *bit* de começo e de parada específicos: uma mensagem única onde o mestre envia dados para o escravo, uma mensagem única onde o mestre lê dados do escravo e mensagens combinadas onde o mestre envia pelo menos dois comandos de leitura ou escrita para um ou mais escravos.

2.4 Arquiteturas de *software*

Conforme apresentando anteriormente, sistemas computacionais tendem a possuir estruturas de *software* complexas, e a concepção e representação dessas não é trivial. A arquitetura de *software* de um sistema computacional se refere justamente às estruturas fundamentais relacionadas ao *software* deste, à sua documentação, e ao seu processo de criação. Tais estruturas, essenciais à análise lógica e racional do sistema, consistem dos elementos do *software*, suas relações, e as propriedades dos elementos e dessas relações, em conjunto com a fundamentação das decisões que envolvem a introdução e a configuração de cada elemento [30]. De forma geral, a arquitetura de *software* de um programa ou sistema computacional pode ser vista como uma representação ou abstração do mesmo, no sentido de auxiliar e fundamentar a compreensão de seu comportamento, excetuando detalhes de sua implementação [31].

Através da definição e estruturação de uma solução que atende os requisitos técnicos e operacionais do sistema em desenvolvimento, a arquitetura tem papel fundamental na otimização de características envolvendo decisões críticas, como questões de segurança, performance, capacidade de gerenciamento, confiabilidade e flexibilidade. Tais características tendem a ser tão importantes quanto a garantia de que o *software* é capaz de gerar o resultado correto, e as decisões que as envolvem possuem grande impacto na qualidade, manutenção, performance, e por conseguinte, no sucesso da aplicação [32].

Mesmo não tendo relação direta com a funcionalidade da aplicação, essas questões influenciam diretamente a dinâmica da criação e definição da arquitetura do *software* do sistema, pois projetá-la de forma eficiente envolve a devida compreensão do problema a ser resolvido em conjunto com suas possíveis soluções [30]. Por exemplo, caso seja necessário executar em alta performance, o sistema deve ser capaz de explorar a decomposição das tarefas através de processos cooperativos executando de forma paralela, gerenciando mecanismos de comunicação entre estes ou da rede, bem como acessos à memória. Caso seja necessário operar com alta precisão, deve-se considerar a maneira como os dados são tratados e fluem pelo sistema. Caso a segurança seja crítica, o sistema deve possuir relações que permitam a legislação de restrições entre as partes do sistema, identificando seções críticas e o poder de acesso atribuído aos elementos. Caso seja necessário garantir alta flexibilidade e portabilidade ao sistema, o sistema deve ser projetado considerando as relações entre as partes e como mudanças individuas podem afetar o desempenho e funcionalidades do sistema. Caso o sistema deva ser implementado de forma incremental, através da publicação de sucessivos subconjuntos, o mesmo deve evitar a presença de interde-

pendências diretas entre unidades.

Portanto, o objetivo da arquitetura de *software* é identificar os requisitos que afetam a estrutura da aplicação, de forma a reduzir os riscos envolvidos na construção de uma solução técnica. A eficiência do projeto, ou *design*, está intimamente associada à sua flexibilidade frente às mudanças naturais que ocorrem em tecnologias de *hardware* e *software* com o tempo, assim como em seus requisitos [32].

2.4.1 Estilos e padrões

Um estilo de arquitetura, nesse âmbito, pode ser definido como a especificação dos tipos de elementos que compõe a arquitetura e de suas relações, em conjunto com a descrição das propriedades e das restrições que se aplicam aos mesmos [33]. Muitas vezes denominados padrões de arquitetura, alguns estilos podem ser aplicados a todo sistema de *software*, como é o caso da decomposição em módulos, pois todo sistema acaba sendo decomposto em módulos de forma a distribuir as tarefas a serem realizadas. Sob essa perspectiva, apesar da capacidade de alguns estilos de arquitetura de *software* em atender aos requisitos de diferentes sistemas computacionais, a complexidade e pluralidade destes acaba por tornar usual a combinação de estilos especializados, cada qual é capaz de satisfazer as características e necessidades específicas impostas pelos sistemas.

Contudo, muitos fatores influenciam a escolha dos estilos de arquitetura que melhor se adaptam a uma aplicação. Esses fatores geralmente incluem a capacidade de projeto e implementação por parte da equipe de projeto, as competências e experiência dos desenvolvedores, assim como as limitações de recursos de infraestrutura disponíveis para o processo de criação e manutenção da aplicação. As subseções a seguir apresentam alguns desses padrões relevantes no contexto deste trabalho e suas características, conforme os conceitos apresentados em [32].

2.4.1.1 Cliente e servidor

O estilo de arquitetura baseado em cliente/servidor é utilizado para caracterizar sistemas distribuídos que envolvem a utilização de sistemas distintos do tipo cliente e servidor, conectados através de uma rede. A forma mais simples desse padrão é composta por uma aplicação servidor a qual é acessada diretamente por múltiplos clientes. Outras variações incluem aplicações *Client-Queue-Client*, onde o servidor serve apenas como canal de comunicação de dados entre múltiplos clientes distintos, armazenando as informações e gerenciando seu acesso, assim como aplicações *peer-to-peer*, desenvolvida com base no estilo *Client-Queue-Client*, permitindo com que

o cliente e o servidor alternem seus papéis a fim de distribuir as informações entre múltiplos clientes.

Suas principais vantagens estão relacionadas a centralização das informações no servidor, que apesar de reduzir a flexibilidade do sistema em comparação a outros estilos, oferece maior segurança, facilidade de gerenciamento da informação, assim como manutenção.

2.4.1.2 Orientado a objetos

O estilo de arquitetura orientado a objetos é um paradigma baseado na divisão de responsabilidades de uma aplicação ou sistema em objetos individuáveis reutilizáveis e auto-suficientes, cada qual contendo os dados e o comportamento relevantes a si. Um projeto baseado em orientação a objetos percebe o sistema como uma série de objetos cooperativos, ao invés de um conjunto de rotinas ou instruções. Além disso, objetos são discretos, independentes e fracamente acoplados; se comunicam através de interfaces , invocando métodos ou acessando propriedades em outros objetos, ou enviando e recebendo mensagens. Dentre as principais características desse estilo, destacam-se:

- **Abstração**: isso permite a redução da complexidade de uma operação através de uma generalização que retém suas características básicas, por exemplo através da criação de métodos para o compartilhamento de informações específicas entre objetos;
- **Herança**: objetos podem herdar de outros objetos, e utilizar tais funcionalidades como suas próprias ou sobrescrevê-las a fim de implementar um novo comportamento;
- **Polimorfismo**: isso permite a substituição de comportamentos de um tipo básico que suporta operações na aplicação através da implementação de novos tipos que são intercambiáveis com o objeto existente;

As principais chamarizes desse padrão de arquitetura são a sua compreensibilidade, devido ao mapeamento próximo da aplicação à objetos reais; a sua capacidade de reutilização, através da sua abstração e polimorfismo; e sua alta coesão.

2.4.1.3 Baseado em componentes

O estilo de arquitetura baseado em componentes está relacionado com o processo de projeto e desenvolvimento do sistema, no sentido de que seu foco é a decomposição do projeto em componentes lógicos ou funcionais individuais, os quais expõe

interfaces de comunicação bem definidas contendo métodos, eventos e propriedades. Assim forncendo um nível maior de abstração em relação aos princípios de projeto orientados a objetos, abstraindo seu foco de questões como protocolos de comunicação ou estado compartilhado. Os princípios fundamentais desse estilo residem na utilização de componentes que possuem as seguintes características:

- **Reutilização**: os componentes são usualmente desenvolvidos de forma a permitir sua reutilização em diferentes cenários e diferentes aplicações, sendo possível também o projeto de componentes específicos para algumas tarefas;
- **Substituição**: os componentes devem ser prontamente capazes de serem substituídos;
- **Condições de operação**: componentes devem ser projetados a fim de operarem em diferentes ambientes e contextos. Informações específicas, como informações de estado do sistema, devem ser passados ao componente ao invés de serem acessadas por ou incluídas neste;
- **Extensibilidade**: um componente pode ser estendido a partir de componentes existentes a fim de fornecer novas funcionalidades;
- **Independência**: componentes são projetados de forma a possuir o menor nível de dependência possível de outros componentes. Assim, eles podem ser utilizados ou alterados sem afetar outros componentes ou sistemas;

Arquiteturas baseadas em componentes comumente gerenciam os mesmos ,e suas interfaces através da troca de mensagens ou comandos entre eles, e em alguns casos mantendo seu estado. As principais vantagens de tal abordagem estão na sua facilidade de implementação, devido a possibilidade de substituição de componentes sem afetar o sistema como um todo; no seu custo reduzido, através da utilização de componentes de terceiros já desenvolvidos; no seu desenvolvimento, pois suas interfaces bem definidas auxiliam o processo de desenvolvimento de outras partes do sistema ao fornecer funcionalidades definidas; na sua reutilização, pois múltiplos sistemas ou aplicações podem reaproveitar uma única implementação; assim como a atenuação da complexidade técnica na detecção de eventuais falhas ou problemas e sua soluções.

2.4.1.4 Dividido em camadas

Arquiteturas dividas em camadas tem seu foco em agrupar funcionalidades relacionadas dentro de uma aplicação em camadas distintas que são sobrepostas verticalmente. As funcionalidades dentro de cada camada se relacionam através de uma

função ou responsabilidade em comum. Usualmente cada camada agrega à si as responsabilidades e abstrações das camadas abaixo de si. Em uma configuração estrita, os componentes de uma camada específica só conseguem interagir diretamente entre si ou com a camada diretamente inferior, em contraste com uma configuração mais relaxada, onde estes são capazes de interagir com qualquer camada abaixo de si.

As camadas de uma aplicação podem residir tanto na mesma unidade computacional quanto distribuídas através de diversas unidades computacionais, caso no qual os componentes de diferentes camadas se comunicam através de interfaces bem definidas. Dentre os princípios fundamentais do estilo divido em camadas estão:

- **Abstração**: a divisão em camadas abstrai a visão do sistema como um todo enquanto provê uma descrição detalhada o suficiente para a compreensão dos papéis e responsabilidade de cada camada individual e suas relações;
- **Camadas funcionais bem definidas**: a sepração das funcionalidades entre diferentes camadas é nítida. Camadas superiores enviam comandos a camadas inferiores e podem reagir de acordo com seus eventos, permitindo o fluxo bidirecional de dados entre as camadas;
- **Reutilização**: camadas inferiores não possuem dependência das superiores, permitindo assim a sua reutilização em cenários compatíveis.

Arquiteturas divididas em camadas, ao dividir o sistema de acordo com suas funcionalidades, fornecem grande capacidade de gerenciamento e um ambiente coeso para testar seu comportamento, em função da sua divisão clara e de suas interfaces bem definidas. Além disso, outras características são relevantes a essa, como sua flexibilidade em relação aos seus níveis de abstração, ao permitir diferentes níveis de acordo com a camada do sistema; e sua performance, ao possibilitar a distribuição das camadas através de múltiplas unidades computacionais diferentes.

2.4.1.5 Baseado em barramento de mensagens

Arquiteturas baseadas em barramento de mensagens descrevem o princípio da utilização de um sistema de *software* capaz de receber e enviar mensagens, usualmente de forma assíncrona, utilizando um ou múltiplos canais de comunicação, a fim de permitir a interação entre aplicações independentes entre si com relação aos seus detalhes específicos. As aplicações mais comuns desse estilo utilizam um sistema de roteamento de mensagens ou um padrão de publicação/assinatura (*publish/subscribe*, em inglês), e são comumente implementados através de um mecanismo de troca de

mensagens, como uma fila de mensagens. São características de um barramento de mensagens:

- **Comunicação orientada a mensagens**: toda comunicação entre as aplicações é realizada através da troca de mensagens utilizando padrões bem definidos;
- **Lógicas de processamento complexas**: através da combinação de operações ou funcionalidades mais simples é possível criar lógicas complexas baseadas em múltiplas etapas;
- **Modificações nas lógicas de processamento**: devido a definição do padrão das mensagens, a substituição da lógica que rege a operação de uma aplicação não afeta o sistema;
- **Integração com ambientes diferentes**: também devido a definição do padrão das mensagens, é possível interagir com aplicações desenvolvidas em ambientes diferentes;

Os principais benefícios do estilo baseado em um barramento de mensagens são a sua extensibilidade, pois aplicações podem ser adicionadas ou removidas do barramento sem necessariamente afetar seu funcionamento; a redução de complexidade da aplicação, pois cada aplicação se preocupa com suas funcionalidades específicas e com a comunicação com o barramento; sua redução no nível de acoplamento do sistema, através da possibilidade de substituição de aplicações com mensagens compatíveis entre si; e sua escalabilidade, visto que múltiplas aplicações podem se conectar ao barramento a fim de tratar múltiplas requisições simultaneamente.

2.4.1.6 Orientado a serviços

A arquitetura orientada a serviços possibilita o fornecimento das funcionalidades da aplicação na forma de um conjunto de serviços, e a criação de aplicações as quais os consomem. Os serviços tem como foco proporcionar a interação planejada através de mensagens utilizando interfaces definidas a nível de aplicação, e não de componentes ou objetos. Exemplos comuns da utilização do padrão orientado a serviços incluem o compartilhamento de informação, o tratamento de processos com múltiplas etapas, a exposição de serviços ou informações específicos de um sistema a uma rede externa ou a criação de *mashups* que combinam dados de fontes diversas. São características usuais de estilos orientados a serviço:

- **Serviços automáticos**: cada serviço é mantido, desenvolvido, aplicado e versionado de forma independente;

- **Serviços distribuíveis**: serviços podem estar localizados em qualquer lugar em uma rede, local ou remotamente, desde que haja suporte da rede para tal;
- **Serviços frouxamente acoplados**: cada serviço é independente dos demais, e pode ser substituído ou atualizado sem afetar aplicações que o utilizam, desde que sua interface permaneça compatível;

Suas principais vantagens são a sua interoperabilidade, pois os protocolos e formatações dos dados são baseados em padrões da indústria; sua racionalização, pois os serviços podem ser granulares a fim de prover funcionalidades específicas, ao invés de duplicar funcionalidades de forma desnecessária em múltiplas aplicações; assim como sua capacidade de abstração, devido a autonomia de seus serviços.

2.5 Plataformas de *software*

Inicialmente, o termo plataforma se referia apenas ao *hardware* computacional o qual executava o *software*, mais tarde sendo generalizado a fim de incluir plataformas baseadas em *software*, como sistemas operacionais. Atualmente, o termo implica quais conjuntos de mecanismos *hardware* ou *software* que possibilitam a execução de aplicações de *software* [34].

Assim, uma plataforma de *software* provê uma coleção de recursos ou competências adicionais ao sistema ou a elementos desse. Tais recursos incluem os serviços tradicionais fornecidos por sistemas operacionais, como escalonamento de tarefas, comunicação entre processos, sistemas de arquivos, gerenciamento de memória, entrada e saída ou desemepnho de multitarefas, dentre outros. Outras competências podem ser fornecidas, como um sistema gerenciador de configurações, interfaces de usuário[4], sistemas de registro de dados ou eventos, coletas estatísticas. Tais serviços são geralmente disponibilizados às aplicações do sistema através de interfaces de Programação de Aplicação (*Application Program Interfaces (APIs)*, em inglês).

Através de *APIs*, a plataforma de *software* especifica ao desenvolvedor como acessar suas funções ou módulos de código integrados. Ao utilizar tais funcionalidades no desenvolvimento de novas aplicações, o desenvolvedor pode então reutilizar funções já existentes sem a necessidade de reescrever código. Essencialmente, as aplicações desenvolvidas sobre a plataforma utilizam suas funcionalidades integradas, o que reduz drasticamente a necessidade de desenvolvimento de código a nível dessas

[4] Interfaces de usuário nesse contexto envolvem não apenas interfaces gráficas de usuários (*Graphical User Interfaces (GUIs)*, em inglês), como também interfaces de linha de comando (*Command-line Interfaces (CLIs), em inglês*)

aplicações para a realização de tarefas comuns ou rotineiras no sistema, mantendo o mesmo padrão de estrutura da aplicação.

O principal proveito de uma plataforma de *software* reside justamente em sua capacidade de permitir o desenvolvimento de componentes uma única vez, e de compartilhar a sua implementação através de múltiplas aplicações compatíveis, o que resulta diretamente na redução de custos de desenvolvimento e manutenção.

2.5.1 *Frameworks*

De forma mais específica, plataformas de *software* podem ser desenvolvidas na forma de *frameworks*. *Frameworks* são definidos como uma abstração através da qual o *software* fornecido, que contém funcionalidades genéricas, é utilizado como base para a aplicação em desenvolvimento. As funcionalidades previamente oferecidas usualmente são complementadas por seções adicionais de código implementadas pelo desenvolvedor, assim atribuindo à aplicação funcionalidades específicas. Por exemplo, em um ambiente orientado a objetos, um *framework* consiste de classes abstratas e concretas, e as suas instâncias consistem da composição e herança das classes existentes [35].

Dessa forma, *frameworks* usualmente possuem duas partes distintas, *frozen spots* e *hot spots*. *Frozen spots* definem a arquitetura geral desse sistema de *software*, fazendo referência a seus componentes básica e suas relações, os quais permanecem "congelados" em qualquer instância do *framework*. Já os *hot spots* representam as partes que os desenvolvedores, através da utilização do *framework*, adicionam suas próprias seções de código a fim de integrar funcionalidades características de sua aplicação, as quais estão intimamente relacionadas às necessidades específicas e aos requisitos do sistema [36].

Capítulo 3

Sistemas aerogeradores com aerofólios cabeados

Sistemas aerogeradores com aerofólios cabeados são definidos como sistemas capazes de operar em grandes altitudes e, por meio de dispositivos aéreos e eletromecânicos, realizar a conversão da energia cinética do vento em energia utilizável.

O conceito de extração de energia eólica em elevadas altitudes data de meados da década de 80, contudo, durante a década seguinte os estudos na área foram praticamente abandonados, sendo retomados apenas nos anos 2000 [5]. Desde então o setor tem crescido e se expandido de forma acelerada. Diversas empresas ingressaram nesse mercado, registrando centenas de patentes e desenvolvendo inúmeras demonstrações e protótipos. A Figura 3.1 apresenta os grupos de pesquisa e empresas ao redor do mundo que estudam e desenvolvem atualmente as diversas possibilidades e aspectos da tecnologia. Este capítulo apresenta uma visão geral destes sistemas, incluindo aspectos construtivos e operacionais.

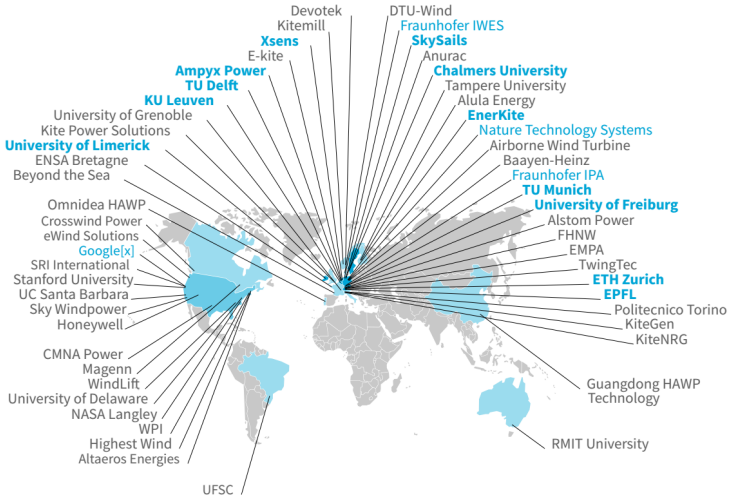


Figura 3.1: *Relação de grupos de pesquisa e empresas envolvidas no desenvolvimento da tecnologia de AWE.*

Fonte: [37].

3.1 Classificação

A maioria dos sistemas de *AWE* possui em comum a presença de uma unidade de solo, de pelo menos um aerofólio ou aeronave, e de cabos realizando a conexão física entre ambos. Apesar disso, são inúmeras as configurações encontradas em protótipos desenvolvidos por diferentes grupos de pesquisa e empresas do setor. Esta seção apresenta uma tentativa de classificação dos sistemas de AWE de acordo com critérios estabelecidos em [5].

3.1.1 Quanto ao princípio de funcionamento

Sistemas de AWE podem ser divididos de acordo com seu princípio básico de funcionamento em dois grandes grupos: sistemas que utilizam as forças aeroestáticas para sustentação e sistemas cujo funcionamento depende de forças aerodinâmicas. Os sistemas baseados em forças de sustentação aerostáticas consistem essencialmente em dispositivos aéreos capazes de flutuar (geralmente uma espécie de balão de gás hélio), conectados ao solo através de cabos. A Figura 3.2 apresenta um protótipo que exemplifica esse tipo de sistema. Em relação a operação, os dispositivos de geração aerostáticos são muito parecidos com os aerogeradores convencionais, gerando energia por meio da movimentação de uma turbina e transmitindo a mesma ao solo através de cabos. A diferença reside na substituição da torre de concreto por cabos, o que permite a sua operação em altitudes elevadas com relação às torres eólicas.

Figura 3.2: *Sistema aerogerador estático desenvolvido pela empresa Altaeros Energies.*

Fonte: [38]

Além dos sistemas que utilizam as forças aerostáticas, existem outros cujo princípio de operação é baseado em forças aerodinâmicas de sustentação e/ou de arrasto,

conforme inicialmente proposto em [39]. Os sistemas baseados em forças aerodinâmicas de sustentação consistem de pelo menos um dispositivo aéreo, geralmente um aerofólio, o qual é conectado a um sistema eletro-mecânico no solo através de um cabo de comprimento variável durante a operação. O voo do dispositivo aéreo é controlado de maneira a tracionar o cabo que o conecta ao sistema em solo, acionando mecanicamente uma máquina responsável pela conversão da energia mecânica em elétrica. Já em sistemas baseados em forças aerodinâmicas de arrasto, o dispositivo aéreo, geralmente uma aeronave, é conectado ao solo através de um cabo de comprimento estático durante a operação. Neste caso, o dispositivo aéreo é equipado com turbinas, as quais são responsáveis pela geração da energia elétrica, enquanto a transmissão desta ocorre por meio do cabo que conecta o dispositivo ao solo. A Figura 3.3 apresenta exemplos de sistemas que utilizam forças aerodinâmicas de arrasto e de sustentação, respectivamente.

(a) **(b)**

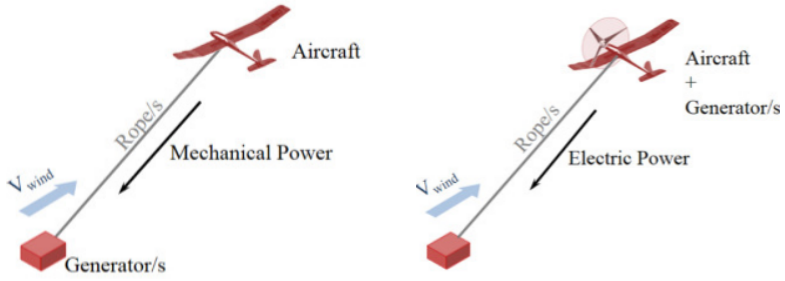
Figura 3.3: *Sistemas AWE baseados em (b) forças de arrasto e (a) forças de sustentação.*

Fonte: Adaptado de [6, 7]

3.1.2 Quanto à localização da unidade geradora

Outro critério comumente utilizado para classificação de sistemas de *AWE* é a localização da unidade geradora. De acordo com esse critério, tais sistemas podem ser classificados em duas categorias distintas, conforme apresentado na Figura 3.4. Nos geradores em voo, a energia mecânica é convertida em elétrica no próprio dispositivo aéreo, já nos geradores em solo, a conversão ocorre na unidade de solo. Nos sistemas com geração em solo, um ou mais cabos são responsáveis pela transmissão da força de tração captada pelo dispositivo aéreo até a unidade de solo, onde a mesma é utilizada para o acionamento mecânico de um gerador elétrico. Já nos sistemas com geração em voo, geralmente pequenos aerogeradores convencionais são acoplados à aeronave transmitindo-se a energia elétrica gerada até a unidade de solo através de

cabos elétricos.



(a) **(b)**

Figura 3.4: *Sistema de geração eólica com (a) gerador(es) em solo e sistema de geração eólica (b) com gerador(es) em voo.*

Fonte: Adaptado de [5].

3.1.3 Quanto à estrutura da asa

Os sistemas de *AWE* fazem uso de dispositivos aéreos para a geração de energia elétrica, mesmo que estes possuam diferentes papéis de acordo com as configurações do sistema. A Figura 3.5 apresenta alguns tipos de dispositivos usualmente empregados, os quais variam de acordo com as configurações e necessidades do sistema, como aerofólios flexíveis, semi-rígidos, aeronaves, dentre outros. Os aerofólios do tipo *Leading Edge Infatable* (LEI) são úteis para manobras de pouso e decolagem. Os aerofólios do tipo *foil* possuem maior eficiência energética se comparados aos do tipo LEI. Os planadores são ainda mais eficientes com relação a sua aerodinâmica, contudo, por serem rígidos, devem ser capazes de resistir às forças aplicadas pelo vento, o que acaba por aumentar seu custo e peso. Por fim, os aerofólios semi-rígidos apresentam uma relação de compromisso entre os do tipo *foil* e os planadores, sendo mais leves que uma estrutura rígida e com melhor aerodinâmica do que uma estrutura flexível.

3.2 Configuração *pumping kite*

Segundo [40], considerando as diversas possíveis combinações presentes na literatura para a composição de sistemas *AWE*, a configuração conhecida como *Pumping-kite* destaca-se devido à sua simplicidade e baixo custo em relação às demais. A operação dos sistemas de *AWE* construídos nessa configuração é realizada através da alternância cíclica entre duas fases distintas de operação, usualmente denominadas "fase de geração"e "fase de recolhimento", conforme apresentado na Figura 3.6. Na

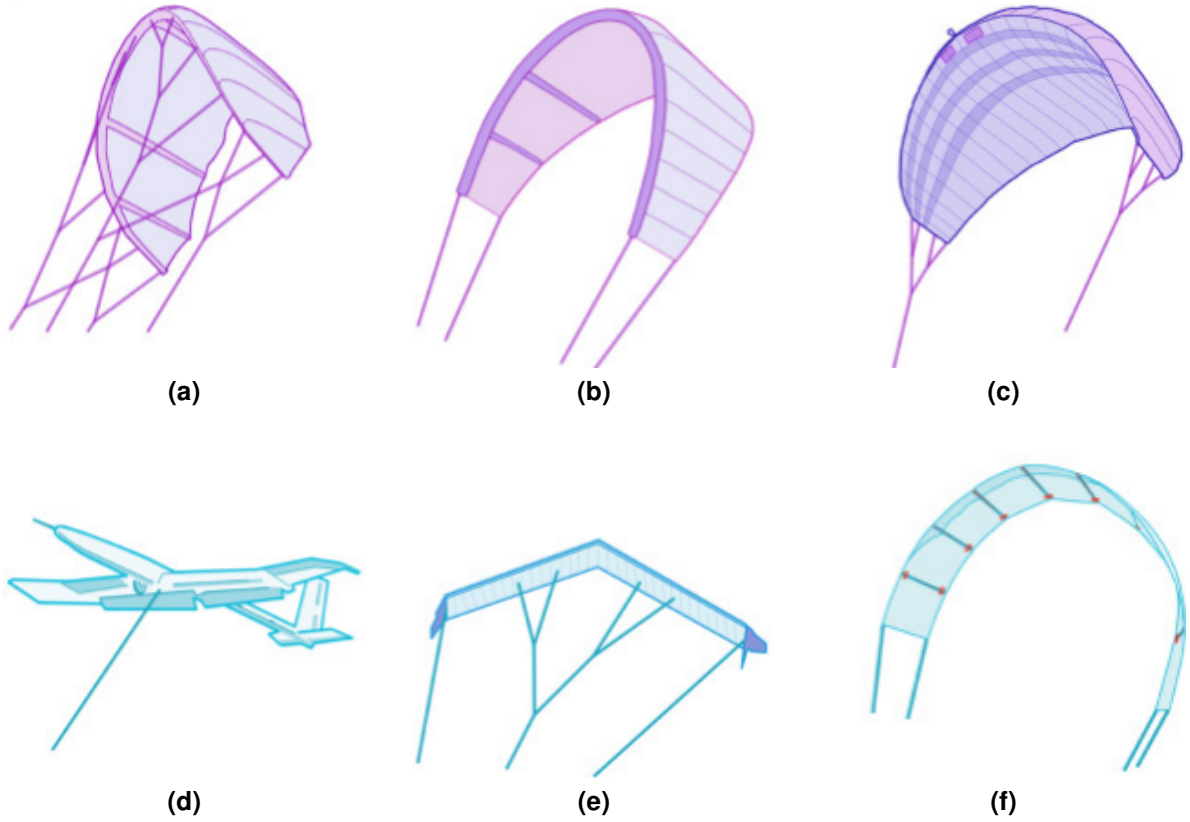


Figura 3.5: *Diferentes tipos de dispositvos aéreos em sistemas de geração em solo. (a) LEI-SLE Kite; (b) LEI-C C kite; (c) Foil Kite; (d) planador; (e) Swept rigid wing; (f) Aerofólio semi-rígido.*

Fonte: Adaptado de [5].

fase de geração, o aerofólio é controlado de modo a percorrer trajetórias na forma de uma lemniscata praticamente perpendicular ao vento, com o objetivo de tracionar o cabo que conecta as unidades de controle de voo e de solo, desenrolando o carretel acoplado à máquina elétrica e consequentemente movimentando a mesma produzindo energia. Na fase de recolhimento, o aerofólio é controlado de maneira a permitir que a máquina elétrica, operando agora como um motor, recolha o cabo que fora previamente desenrolado, consumindo assim energia elétrica e permitindo que o sistema prossiga novamente para a etapa de geração, resultando em um saldo energético positivo ao final de cada ciclo.

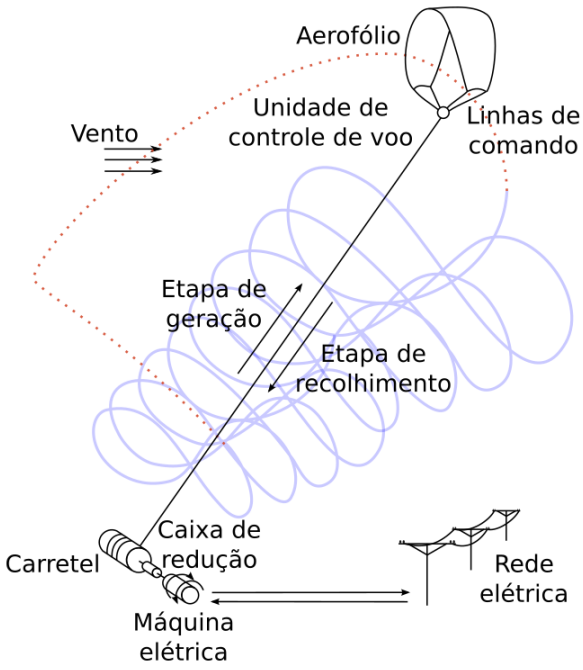


Figura 3.6: *Representação de sistema de AWE operando na configuração pumping kite. Na etapa de geração – linha contínua – o aerofólio se afasta da unidade de solo descrevendo uma trajetória complexa na forma de lemniscata, ao passo que na etapa de recolhimento – linha pontilhada – o mesmo se aproxima da unidade de solo descrevendo uma trajetória simples.*

3.3 Instrumentação

3.3.1 Sistemas de medição

Para o correto monitoramento das condições dos sistemas de AWE, e para o controle dos mesmos, é necessário realizar a medição de grandezas físicas intrínsecas ao processo. Para isso, os seguintes instrumentos de medição são comumente utilizados, conforme exemplificado na Figura 3.7:

- **Encoders:** transdutores de posição angular, são dispositivos capazes de medir a rotação de um eixo em torno de si mesmo. São utilizados para realizar a medição indireta da posição do dispositivo aéreo no espaço, através de operações matemáticas e da combinação com outras informações obtidas do processo.
- **IMU:** unidade de medição inercial (*Inertial Measurement Unit (IMU), em inglês*) é um dispositivo eletrônico capaz de medir as forças e velocidades angulares atuantes sobre um corpo, como por exemplo o aerofólio, através da combinação de acelerômetros e giroscópios.
- **Células de carga:** transdutores de força, são dispositivos capazes de medir de forma indireta a força aplicada, garantindo grande precisão e versatilidade.

São utilizadas para a medição da tração exercida sobre o cabo que conecta o aerofólio ao solo.

(a) (b) (c)

Figura 3.7: *Exemplos de instrumentos utilizados em sistemas de geração eólica com aerofólios cabeados: (a) encoder; (b) IMU; (c) célula de carga.*

3.3.2 Sistemas de atuação

Para fins de controle do aerofólio ou da aeronave, sistemas de *AWE* normalmente utilizam motores elétricos, cujas especificações dependem das características do sistema em questão. Sistemas com aeronaves geralmente possuem motores elétricos acoplados a mesma, ao passo que sistemas que utilizam aerofólios geralmente possuem uma unidade controle de voo, a qual voa adjacente ao aerofólio, e é responsável por controlar a diferença de comprimento entre os cabos que conectam ambas estruturas, direcionando assim o seu voo. Outra possibilidade, é a atuação sobre o aerofólio diretamente da unidade de solo, também fazendo uso de cabos. A Figura 3.8 ilustra as possibilidades de atuação mais comuns em sistemas de *AWE* na configuração *pumping-kite* com aerofólios flexíveis.

3.4 Controle

Devido às características instáveis e às não-linearidades presentes em sistemas de *AWE*, estratégias de controle relativamente complexas são necessárias para garantir a segurança na operação e a máxima eficiência na captação da energia do vento. Conforme [40], a maioria das soluções de controle para sistemas *pumping kite* utilizaram,

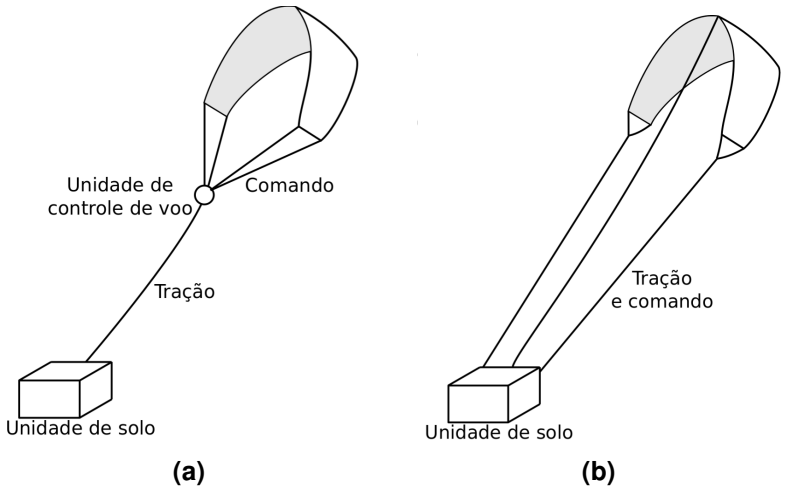


Figura 3.8: *Sistemas de geração eólica com atuação (a) através da unidade de controle de voo; e (b) através da unidade solo.*

ao longo da última década, estratégias preditivas com modelos não-lineares (*non-linear model predictive control* (NMPC), em inglês). Nessa abordagem, os valores das entradas de controle de voo do aerofólio (*steering*, ou a diferença do comprimento de cabo) e do carretel (velocidade angular) no sistema *pumping kite*, são determinados por meio da solução de um problema de otimização cuja função custo considera potência média gerada em cada ciclo de operação. Uma vez que todos os valores são determinadas em conjunto, tem-se uma solução "acoplada"para as entradas de controle do sistema. Soluções de controle do tipo NMPC utilizadas em sistemas de AWE caracterizam-se por não definirem uma trajetória de voo *a priori*, ou seja, a trajetória é um resultado da solução do problema de otimização, e pode variar de acordo com diversos fatores. Devido às características dinâmicas do sistema, a frequência de amostragem necessária para o controle pode fazer com que estratégias baseadas em NMPC se tornem bastante custosas, do ponto de vista computacional, principalmente quando são utilizados modelos mais complexos do sistema.

Uma tendência recente na área de *AWE*, principalmente em sistemas na configuração *pumping-kite*, é a utilização de uma topologia descentralizada de controle, que considera de forma independente os subsistemas em solo e em voo. Dessa forma, é possível calcular os sinais de controle de forma separada, utilizando apenas variáveis disponíveis localmente, e minimizando problemas decorrentes do atraso de transporte devido à comunicação entre as unidades. Além de um aumento na robustez do sistema, essa abordagem permite uma diminuição dos requisitos computacionais associados à execução das malhas de controle do sistema. A Figura 3.9 apresenta uma aerofólio voando em trajetória de lemniscata usando esta topologia de controle descentralizada.

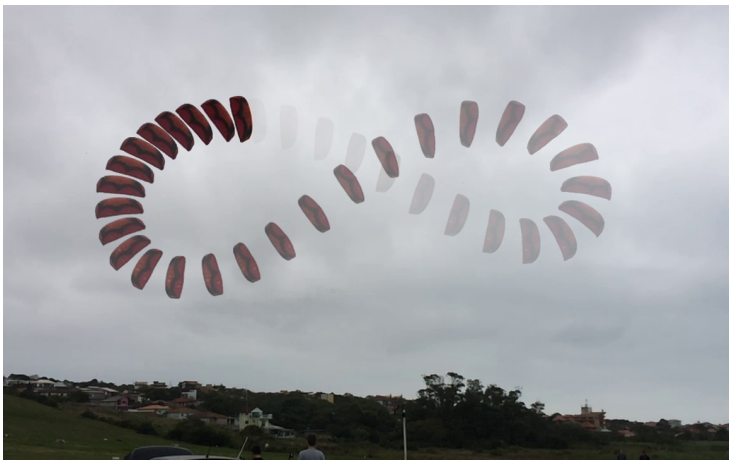


Figura 3.9: *Aerofólio do grupo UFSCkite descrevendo trajetória em formato de lemnsicata.*

Fonte: Adaptado de [10].

Capítulo 4

Análise e identificação de requisitos

Antes da etapa de projeto da plataforma propriamente dita, a análise do contexto no qual a mesma será aplicada foi conduzida, a fim de estabelecer as suas especificações. Essa etapa do trabalho foi marcada por discussões e decisões sobre a estrutura de *software* presente e o ambiente de prototipagem, ponderando suas características para definir de forma concreta os requisitos da plataforma a ser desenvolvida. Neste capítulo será apresentado o resultado de tais análises e discussões, incluindo uma visão geral de um protótipo do grupo *UFSCkite* e do ambiente onde seu desenvolvimento foi conduzido, assim como os requisitos definidos, a análise de sua viabilidade e, finalmente, a decomposição do sistema de *AWE* em componentes funcionais, com o objetivo de guiar o desenvolvimento do trabalho.

4.1 Visão geral do sistema aerogerador

O laboratório *UFSCkite* estuda e desenvolve atualmente um sistema aerogerador composto por uma unidade de solo, uma unidade de controle de voo e um aerofólio flexível. Além da utilização de sistemas puramente computacionais na forma de simuladores para validar o funcionamento desse sistema de aerogeração, o grupo também desenvolve protótipos a fim de colocar em prática os resultados obtidos. Contudo, não existe hoje uma orientação clara para guiar este processo, o que requer esforços no sentido de desenvolver sistemas flexíveis a mudanças em seus requisitos, sejam estas relativas à topologia de controle empregada ou aos seus aspectos construtivos. Assim, os mesmos ainda estão sujeitos a mudanças em suas características físicas, como sensores, atuadores ou componentes de *hardware*, e operacionais, como sua configuração de operação ou a topologia de controle empregada.

4.1.1 Protótipo da unidade de controle de voo

Nesse âmbito, a fim de testar, aprimorar e validar suas estratégias de controle de voo em campo sem a necessidade de possuir o sistema aerogerador completo, o grupo desenvolveu um protótipo, apresentado na Figura 4.1. Construído para operar

fixo ao solo, o sistema atua sobre um aerofólio flexível a fim de controlar a trajetória deste no espaço. O mesmo é constituído por uma estrutura mecânica embutida de uma série de componentes de *hardware*, incluindo um *single board computer*, assim como sensores e atuadores, responsáveis pela sua instrumentação. Apesar de ser um sistema de controle de voo, a operação em solo permite a realização de testes graduais em um ambiente mais controlado com relação ao cenário final, onde a unidade estará voando junto com o aerofólio.

Figura 4.1: *Protótipo desenvolvido pelo grupo UFSCkite a fim de testar, aprimorar e validar as estratégias de controle desenvolvidas.*

4.1.1.1 Componentes físicos e de *Hardware*

O sistema possui dois componentes principais, a unidade de controle de voo e um aerofólio flexível com cerca de 2 metros de envergadura, conectados através de dois cabos de controle e um cabo de tração, de aproximadamente 43 metros de comprimento.

A unidade computacional embarcada no protótipo é uma *BeagleBone*, um *single board computer* desenvolvido pela *Texas Instruments*. O mesmo possui memória *RAM*

de 512 Mb, *clock* de processamento de até 1 GHz[1], conector *Ethernet* para conexões de rede cabeadas e diversos pinos *I/O* para a comunicação com dispositivos de *hardware* periféricos [22].

A fim de interagir com o meio físico e controlar a trajetória de voo do aerofólio, o protótipo faz uso de alguns sensores e atuadores, cada qual com a sua interface de comunicação específica. Os dois *encoders*, responsáveis pela medição dos ângulos do cabo de tração principal, fazem uso de uma interface de comunicação serial RS-485. A medição da força de tração no cabo principal é realizada através do sinal analógico fornecido por uma célula de carga, o qual é convertido por um conversor analógico digital. A atuação do sistema sobre o aerofólio flexível é realizada por meio de dois servo-motores *brushless DC*, e a comunicação com seus *drives*[2] é realizada por meio de duas interfaces seriais RS-232.

4.1.1.2 *Software* embarcado

A primeira versão do *software* de controle do protótipo foi projetada e desenvolvida a fim de ser executada em um computador, em conjunto com seu sistema supervisório. Sem a tarefa de computação dos sinais de controle, o foco do sistema embarcado se conteve à instrumentação do processo, coletando e retransmitindo informações dos sensores, e à atuação sobre os motores, de acordo com os sinais de controle recebidos remotamente do computador. Executados no computador, os controladores e o sistema supervisório foram estruturados de forma monolítica, sendo um processo responsável pela leitura dos dados na rede, cálculo das leis de controle e envio dos sinais ao sistema embarcado. De forma equivalente, o *software* embarcado foi estruturado em um único processo, responsável pelo gerenciamento de diversas *threads* dos sensores e atuadores e pelo seu envio ao computador através da rede local. Apesar de permitir a realização de testes do sistema nos modos automático e manual, essa configuração apresentou algumas limitações na perspectiva do controle e do *software*. A distribuição da malha de controle através da rede acaba por introduzir incertezas temporais e detalhes de comunicação indesejados ao processo, assim como a estrutura monolítica de *software* adotada em ambas as unidades computacionais dificulta sua manutenção, extensibilidade e compreensão.

A fim de amenizar tais problemas, uma nova arquitetura de *software* embarcado foi projetada em conjunto com um novo sistema sistema supervisório, com o objetivo

[1] Para fins de obter a melhor performance possível, as placas *Beaglebone* utilizadas pelo grupo foram devidamente configuradas para operar em caráter permanente utilizando sua capacidade máxima de processamento de 1 GHz

[2] O *drive* dos motores é um programa sequencial que utiliza instruções e registradores fornecidos pelo fabricante para prover funcionalidades básicas aos mesmos

de embutir as responsabilidades das malhas de controle na unidade computacional do protótipo, utilizando o computador apenas para fins de supervisão e simulação. Essa nova arquitetura baseou-se na divisão do sistema em um conjunto de módulos independentes, implementados utilizando a linguagem C de programação na forma de processos no sistema operacional *Linux*. Para fins de comunicação entre os diferentes módulos, foi utilizada a infraestrutura de troca de mensagens fornecida pela biblioteca open-source *nanomsg* [41], que disponibiliza uma série de modelos (e.g. *publish-subscribe*, *request-reply*, e *survey-response*), bem como transportes (e.g. TCP, IPC, WS). Sobre esta infraestrutura foi construída uma camada de abstração, de modo a possibilitar, caso necessário, a substituição da biblioteca *nanomsg* por outra similar. Todas as mensagens trocadas entre módulos do sistema são codificadas no formato de serialização binária *Protocol buffers*, desenvolvido pelo Google [42]. Contudo, mesmo apresentando diversas vantagens sobre a configuração distribuída na rede com aplicações monolíticas anterior, a nova estrutura também apresentou limitações em relação a sua estrutura e, principalmente, à performance dos mecanismos de comunicação durante a execução das malhas de controle, agora de responsabilidade total da unidade computacional embarcada no protótipo.

4.1.2 Sistema supervisório

Atualmente implementado utilizando tecnologias *web* na forma de uma interface gráfica de usuário, o sistema supervisório foi desenvolvido com o objetivo de permitir aos operadores do sistema monitorar e interagir com os protótipos desenvolvidos. Através dessa, as diversas informações provenientes do sistema, como medições de sensores e sinais de controle, são apresentadas na forma de gráficos ou indicadores, como exemplificado na Figura 4.2. Além disso, tal interface gráfica permite também a alteração de parâmetros do sistema, sejam esses construtivos ou diretamente relacionados com a execução do sistema, como por exemplo o modo de operação automático ou manual e as fontes dos dados utilizados para estimação de estado e controle de voo.

4.1.3 Simulador

O simulador do sistema aerogeador foi desenvolvido pelo grupo de maneira a permitir testes em laboratório através da simulação completa das dinâmicas do processo. O mesmo pode ser utilizado para a realização de testes do tipo *hardware-in-the-loop (HIL)*, configuração apresentada na Figura 4.3, onde uma parte do sistema é simulada, enquanto que as demais operam normalmente, permitindo assim a analise de

Figura 4.2: *Imagem obtida da tela principal do sistema supervisório.*

Fonte: [10]

comportamento do sistema sem a necessidade de testes utilizando componentes reais [43]. Assim, é possível planejar e estudar as tecnologias envolvidas, bem como realizar testes preliminares do sistema de forma mais rápida, segura e controlada.

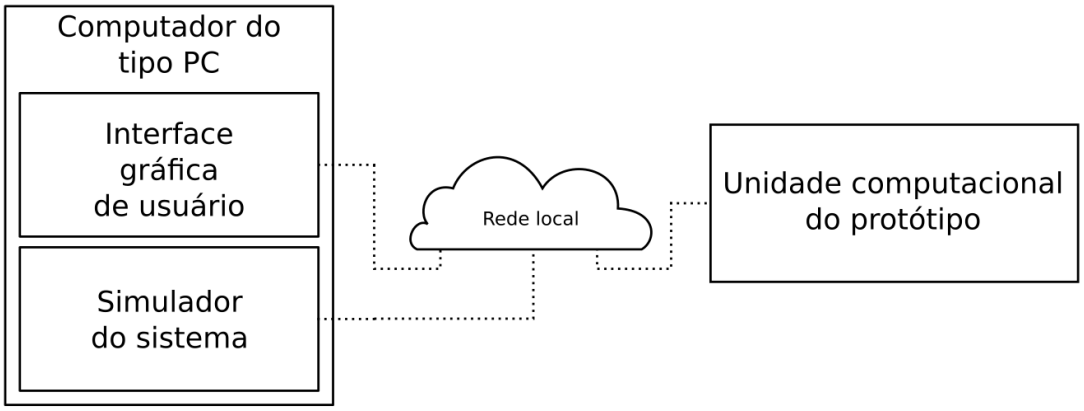


Figura 4.3: *Representação da utilização do sistema em uma configuração de simulação hardware-in-the-loop.*

4.2 Requisitos da plataforma

Nesse contexto, fica evidente a relação do ambiente no qual são conduzidos os estudos e desenvolvidos os sistemas do grupo com o processo de concepção, desenvolvimento e manutenção de protótipos. A fim de estabelecer o conjunto de especifi-

cações necessários à plataforma para torná-la uma ferramenta útil neste cenário, as demandas do laboratório para o desenvolvimento de *software* embarcado foram então compiladas na forma de requisitos bem definidos, livres de ambiguidades. Esse conjunto de requerimentos especifica que a plataforma de *software* embarcado proposta deverá ser capaz de fornecer as seguintes funcionalidades:

R1: Ser agnóstica com relação aos componentes de *hardware* utilizados:

- (a) Alterações em algum componente de *hardware* não devem afetar partes do sistema, além das que possuem interação direta com este;

R2: Oferecer suporte a alterações em elementos de *software*:

- (a) Alterações de funcionalidade ou de lógica em algum elemento de *software* não devem afetar funcionalidades de outros elementos;

R3: Oferecer suporte à análise das aplicações em tempo de execução e *offline*:

- (a) Prover métodos ou ferramentas de análise da performance e das funcionalidades da aplicação em tempo real;
- (b) Prover métodos ou ferramentas de armazenamento de dados em tempo de execução e análise *offline* na forma de um *logger* do sistema;

R4: Oferecer suporte a configuração de parâmetros:

- (a) Fornecer um sistema de configuração para características construtivas ou específicas a determinados componentes do sistema;
- (b) Fornecer suporte a alteração em tempo de execução de parâmetros e configurações do sistema;

R5: Oferecer suporte a diferentes modos de operação:

- (a) Permitir a criação de um sistema transparente em relação ao seu modo de operação, podendo ser configurado a fim de operar conforme a situação desejada;
- (b) Permitir a reutilização de estruturas e componentes do sistema comuns entre os diferentes modos de operação;
- (c) Garantir a independência das funcionalidades relativas a cada modo de operação.

R6: Oferecer suporte a sistemas de instrumentação:

(a) Fornecer suporte a comunicação com elementos de *hardware* através de interfaces bem definidas;

R7: Oferecer suporte a malhas de controle:

(a) Permitir a criação de um sistema capaz de cumprir os requisitos impostos por malhas de controle da aplicação, e.g. requisitos temporais;

(b) Fornecer suporte a instrumentação do sistema para análise de seus requisitos de controle;

R8: Comunicar-se com sistemas externos através da rede local:

(a) Conectar-se simultaneamente a múltiplos sistemas externos;

(b) Utilizar um padrão de comunicação bem definido;

(c) Permitir a alteração de parâmetros através de sistemas externos;

(d) Realizar a sincronização de parâmetros entre os componentes do sistema;

(e) Permitir a troca de informações entre a aplicação e sistemas externos;

4.3 Análise de viabilidade

A viabilidade dos requisitos estabelecidos foi analisada a partir de estudos e comparações com funcionalidades já implementadas na estrutura de *software* atual, assim como com funcionalidades de aplicações similares. Os principais critérios de avaliação de cada requisito foram seu impacto na complexidade, escalabilidade, portabilidade, flexibilidade, performance e manutenibilidade da plataforma de *software* e de aplicações de *AWE* desenvolvidas sobre esta, assim como a avaliação do valor agregado ao garantir o cumprimento do requisito em termos de funcionalidades do sistema. Com base nesses critérios, todos os requistos listados anteriormente foram devidamente aprovados e considerados viáveis.

4.4 Decomposição em módulos

Com base na versão atual de *software*, nos requisitos levantados e no cenário de desenvolvimento de protótipos para sistemas aerogeradores com aerofólios cabeados, foi estabelecida uma estrutura de categorias funcionais para representar os diferentes componentes das aplicações de *AWE*, de acordo com o diagrama da Figura 4.4. A divisão funcional em módulos teve como objetivo servir de diretriz para as decisões

de projeto e implementação da plataforma, auxiliando a escolha de métodos ou ferramentas para atender os requisitos estabelecidos.

- **Sistemas externos**: categoria que classifica os módulos responsáveis pela infraestrutura de comunicação através da rede com sistemas externos ao sistema, como as interfaces gráficas de usuário e o simulador.
- **Sensores**: categoria que classifica os módulos de instrumentação responsáveis pela infraestrutura de comunicação através de interfaces de *hardware* com os sensores conectados ao sistema, como as células de carga ou *encoders*.
- **Atuadores**: categoria que classifica os módulos de instrumentação responsáveis pela infraestrutura de comunicação através de interfaces de *hardware* com os atuadores conectados ao sistema, como motores elétricos.
- **Controladores**: categoria que classifica os módulos responsáveis pela realização de operações relacionadas ao controle do sistema, como controladores de voo ou de estado do sistema.
- **Estimadores e filtros**: categoria que classifica os módulos responsáveis pela realização de operações relacionadas à estimação ou cálculo de parâmetros do sistema e variáveis do processo, como algoritmos de filtragem ou fusão de dados de sensores.

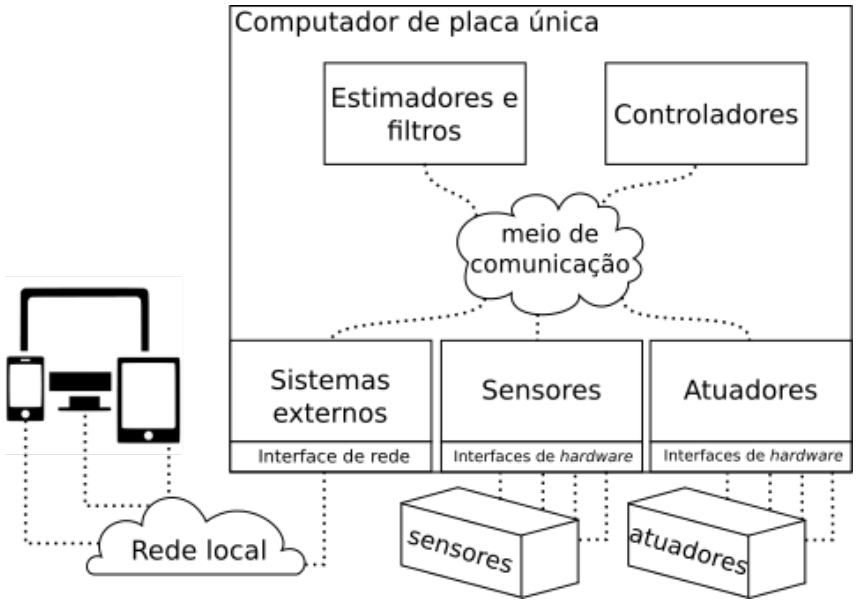


Figura 4.4: *Representação modular de um sistema AWE através da divisão em categorias funcionais.*

Capítulo 5

Projeto da plataforma de *software*

Uma vez finalizada a etapa de análise do problema e identificação de requisitos, no final do Capítulo 4 foi estabelecida a decomposição do *software* para protótipos de sistemas de *AWE* em categorias de módulos funcionais, a fim de guiar a etapa de projeto da plataforma de *softawre* embarcado. O propósito principal da plataforma projetada é fornecer um *framework* de *software* sobre o qual aplicações de *software* para sistemas de aerogeradores com aerofólios cabeados possam ser desenvolvidos. Este capítulo detalha o resultado dessa etapa, na qual foram especificadas as características necessárias ao cumprimento dos requisitos estabelecidos. A arquitetura básica proposta é apresentada, especificando cada parte do sistema com relação a sua infraestrutura, bem como suas relações. Em seguida, é apresentado também o seu funcionamento básico, considerando a estrutura fornecida para a concepção de módulos de aplicações de *AWE*. Por fim, são apresentadas funcionalidades e ferramentas adicionais projetadas a fim de complementar a plataforma, tanto na perspectiva do desenvolvimento de *software* quanto da operação dos protótipos.

5.1 Arquitetura de *software*

Sistemas de *AWE* são usualmente compostos por dispositivos em solo e em voo, cada qual com seus próprios subcomponentes. Isso resulta em sistema inerentemente distribuído, com múltiplos elementos heterogêneos físicamente separados e acoplados por meio de conexões elétricas ou mecânicas. Resultados apresentados em [44], e obtidos a partir do desenvolvimento de uma arquitetura distribuída para sistemas eólicos baseados em aerofólios cabeados, indicam que uma abordagem distribuída facilita a criação de links de comunicação redundantes e componentes de controle, e poderia ser usada mesmo em aplicações comerciais. Assim, a arquitetura básica proposta para a plataforma consiste da divisão do sistema em módulos funcionais e na definição das suas conexões, realizadas através da troca de mensagens. Além da divisão em módulos com relação a suas funcionalidades na perspectiva do sistema aerogeador, os módulos foram subdivididos em componentes de acordo com suas funções internas, denominados submódulos. Por meio desta subdivisão, as fun-

cionalidades que requerem a interação entre os módulos ou sistemas externos são completamente desacopladas das demais, não interferindo caso alterações em sua lógica ou estrutura sejam necessárias, conforme exemplificado na Figura 5.1.

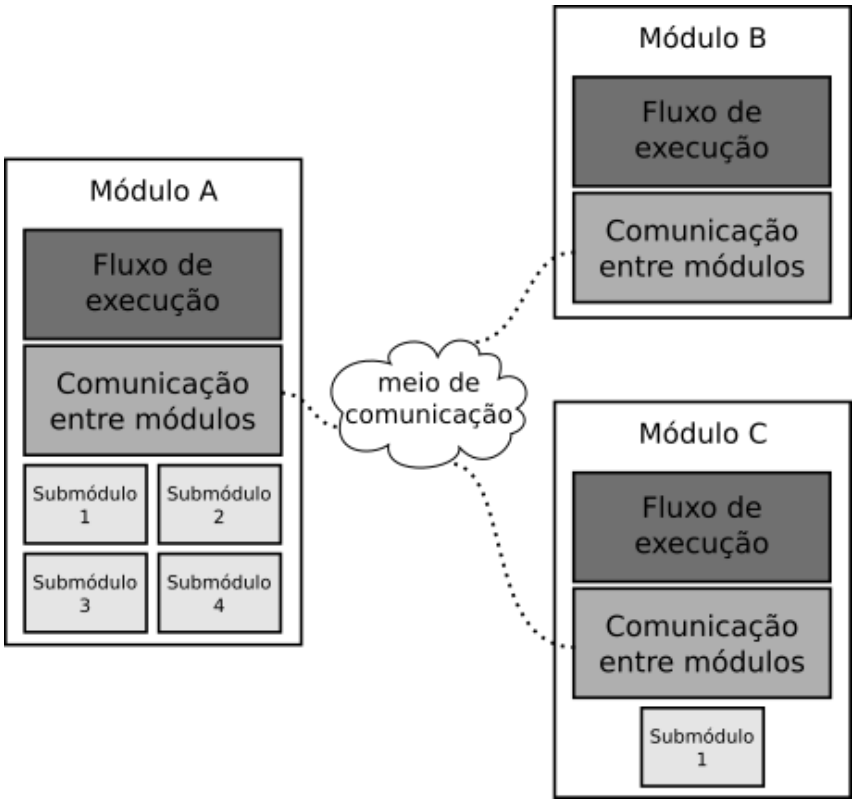


Figura 5.1: *Representação da composição e interação entre módulos e seus submódulos na perspectiva da plataforma de software.*

5.2 Módulos e submódulos

Através dessa estrutura, cada módulo é responsável por uma ou mais funcionalidades específicas dentro do espectro de aplicações do sistema aerogerador, e a união de tais funcionalidades através das conexões entre os módulos e sistemas periféricos é o que possibilita a operação do sistema. Além de permitir que suas funcionalidades sejam desacopladas, a subdivisão do sistema possibilita sua independência em relação aos diferentes cenários de utilização do mesmo: a operação em laboratório em conjunto com o simulador e a operação completa do sistema em campo.

Com o objetivo de padronizar e simplificar o desenvolvimento do *software* embarcado para os protótipos de sistemas de *AWE*, a plataforma fornece ao desenvolvedor uma estrutura de módulo base. Essa estrutura pode ser considerada o esqueleto do módulo funcional da aplicação de *AWE* sobre o qual submódulos serão acrescentados

ou configurados. A estrutura genérica de cada módulo é composta por submódulos de forma a abstrair os detalhes do mecanismo de comunicação entre módulos, bem como fornecer uma estrutura básica para a operação do módulo, fazendo a distinção entre as etapas de configuração e execução do mesmo. Basicamente, o módulo possui um método de inicialização, que é utilizado para a criação do módulo da aplicação de *AWE*, o qual possui dois métodos a serem programados pelo desenvolvedor: um método de configuração e um método de processamento. O método de configuração é responsável pela configuração do módulo e de todos os seus submódulos. Já o método de processamento é responsável pela execução periódica da funcionalidade específica atribuída ao módulo em questão, de forma a garantir que sua execução ocorra de forma independente.

Para definir o padrão de comunicação e sua estrutura, foi atribuído ao sistema um conjunto de mecanismos e estruturas de mensagens. Cada mensagem possui um tópico, responsável por sua identificação , e um conjunto de dados, o qual é consumido pelo destinatário. De forma a permitir a comunicação entre diferentes módulos, os diferentes tipos de mensagens são padronizadas. Assim, os módulos são capazes de enviar diferentes mensagens e de filtrar as mensagens recebidas, sem a necessidade de interagir ou estabelecer uma conexão direta com os demais módulos. A Figura 5.2 exemplifica o sistema de comunicação especificado e suas estruturas de mensagens.

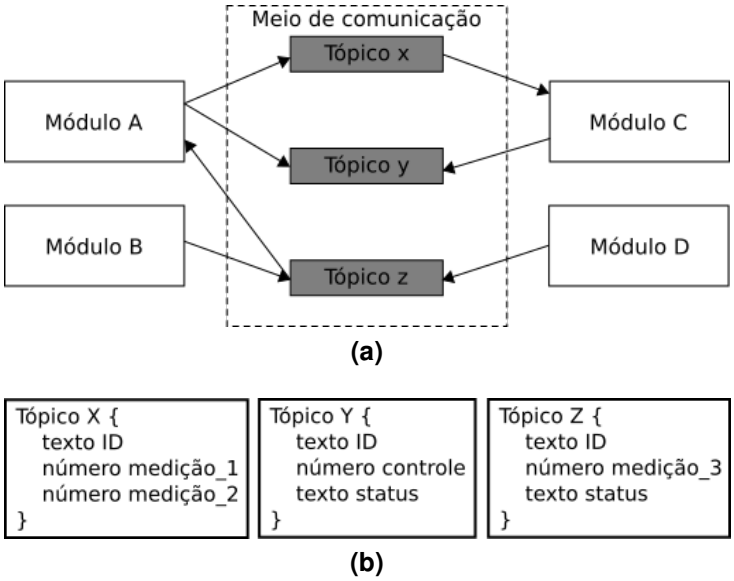


Figura 5.2: *Representação do sistema de comunicação especificado para a plataforma de software, (a) estrutura para a troca de mensagens no sistema; e (b) estrutura de cada mensagem.*

5.3 Funcionamento

A Figura 5.3 apresenta de maneira simplificada o funcionamento básico da estrutura genérica de módulo fornecida pela plataforma. Inicialmente, o módulo deve ser configurado a fim de permitir o estabelecimento da sua comunicação com os demais componentes do sistema, sejam esses outros módulos ou sistemas periféricos. Nessa etapa de configuração são estabelecidas as mensagens de seu interesse, assim como especificados os seus métodos de tratamento. Também diz respeito a essa etapa o processo de especificação das conexões com sistemas externos, como as interfaces gráficas de usuário, o simulador ou dispositivos de *hardware*, caso seja necessário. Finalmente, faz parte da etapa de configuração do módulo a configuração de parâmetros necessários a sua operação, assim como a criação de submódulos completos[1], adicionais aos fornecidos pela plataforma.

Uma vez concluída a etapa de configuração, o módulo passa à etapa de processamento, que é repetida periodicamente. Nessa etapa, o sistema executa os métodos necessários para cumprir com a sua funcionalidade sob a perspectiva global do sistema. São exemplos de tais métodos operações matemáticas, o processamento de dados coletados do sistema, bem como o envio dos resultados para os demais módulos ou sistemas externos. É importante ressaltar que o tratamento das mensagens trocadas entre os módulos ocorre de forma independente da etapa de processamento, o que permite aos módulos realizarem suas funcionalidades de processamento de maneira desacoplada, mesmo quando existe uma relação de dependência de informação entre os mesmos.

5.4 Funcionalidades e ferramentas adicionais

Além de fornecer a estrutura modular básica para o desenvolvimento das aplicações de *AWE*, a plataforma de *software* também oferece algumas funcionalidades e ferramentas adicionais, relacionadas a sua manutenção, análise, desenvolvimento e operação.

5.4.1 Configuração dos módulos

Protótipos são por definição sistemas passivos de mudança em diferentes aspectos, não somente estruturais, mas também de configurações construtivas ou operaci-

[1] Apesar do prefixo "sub", os submódulos tem à sua disposição a mesma infraestrutura de um módulo convencional, sendo estruturado portanto da mesma forma que este.

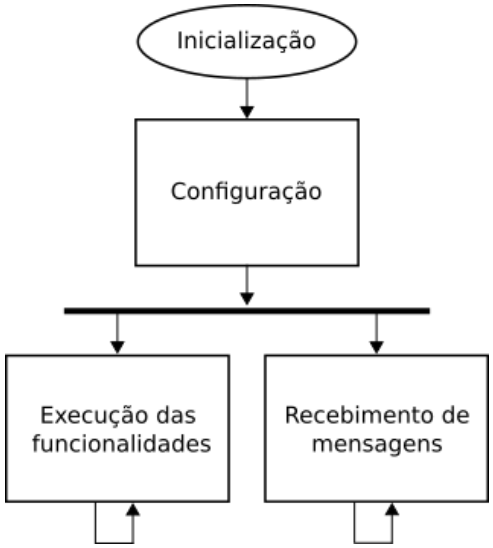


Figura 5.3: *Representação do funcionamento básico de um módulo através de um diagrama de atividades.*

onais. Assim, de modo a facilitar a operação e manutenção dos seus componentes de *software*, é importante atribuir aos mesmos um sistema que permita a edição de configurações de cada módulo de maneira simples e ágil, sem a necessidade de alteração no código fonte. O que também possibilita a reutilização da mesma estrutura de código em módulos com as mesmas funcionalidades, diferindo apenas em suas configurações.

5.4.2 Parâmetros do sistema

Por se tratarem de sistemas complexos, sistemas de *AWE* possuem muitos parâmetros envolvidos em suas malhas de estimação e controle. Além disso, a possibilidade de alternância entre diferentes modos de operação em tempo de execução acrescenta uma série de parâmetros a estes. A fim de permitir a alteração de parâmetros do sistema em tempo de execução, e.g através das *GUIs*, é importante atribuir ao mesmo um sistema de gerenciamento e armazenamento de parâmetros relevantes a múltiplos módulos, garantindo sua sincronia em todos os níveis da aplicação de *software*.

5.4.3 Gerenciamento da execução dos módulos

Dada a estrutura modular e a existência de diferentes cenários de operação do sistema, é importante garantir que seus operadores gerenciem a execução da aplicação e seus módulos de forma fácil, intuitiva e eficaz, sem a necessidade de alterações di-

retas a nível de código em qualquer componente do sistema. Além disso, a execução isolada de módulos ou conjuntos específicos auxilia também a realização de testes de funcionalidades específicas.

5.4.4 Depuração em tempo de execução

Com o elevado número de variáveis de processo e controle envolvidas na operação do sistema, e a presença de malhas de controle operando na ordem de dezenas de milissegundos, não é viável depurar as informações do sistema diretamente em forma de texto, pois além de afetar a performance do sistema, seria muito difícil acompanhar e avaliar seus valores com precisão. Assim, é necessário o desenvolvimento de um sistema minimamente invasivo, capaz de coletar informações dos módulos do sistema em tempo de execução e de apresentar essas informações de forma intuitiva ao desenvolvedor/operador do protótipo.

5.4.5 Registro de dados

É necessária a criação de um macanismo capaz de registrar as variáveis do processo para sua posterior análise *offline*, e.g. para identificar possíveis causas de eventuais falhas durante a execução de testes em campo ou avaliar a performance obtida pelo sistema. Assim, é possível a realização da avaliação formal e estatística do desempenho do sistema utilizando dados quantitativos e não dependendo apenas das observações realizadas durante a operação do sistema.

Capítulo 6

Implementação da plataforma de *software*

Este capítulo apresenta detalhadamente o processo de implementação da plataforma de *software* embarcado. São descritas as motivações suportando as decisões envolvidas na sua implementação, através da correlação das mesmas com as decisões de projeto da plataforma e dos requisitos estabelecidos, e as ferramentas aplicadas em seu desenvolvimento.

6.1 Considerações gerais

Para garantir o cumprimento de todos os requisitos especificados no Capítulo 4, uma série de decisões foram tomadas com relação a aspectos da implementação da plataforma de *software* projetada. Como a mesma deve ser independente do *hardware* utilizado nos protótipos, é fundamental a utilização de um sistema operacional para relacionar a mesma com os recursos de *hardware* disponíveis na unidade computacional. Utilizado também em versões anteriores de *software* embarcado pelo grupo, o *Debian Linux* foi escolhido a desempenhar tal função. De código aberto, o mesmo fornece uma vasta gama de ferramentas e funcionalidades compatíveis com as necessidades de um sistema embarcado, como alta performance, flexibilidade e acessibilidade aos elementos periféricos de *hardware* pelo usuário desenvolvedor. Assim, com base no sistema operacional *Debian Linux*, e na estrutura de *software* projetada, deu-se início à etapa de implementação da plataforma, através da seleção, adaptação e desenvolvimento de ferramentas e mecanismos para desempenhar as funcionalidades desejadas. As seções subsequentes apresentam as decisões desse processo.

6.2 Módulos

Conforme discutido ao longo deste documento, a plataforma de *software* proposta baseia-se na divisão do sistema em um conjunto de módulos independentes e com funcionalidade bem definida. Os módulos executam sob a forma de processos no

sistema operacional Linux e foram implementados na linguagem C, seguindo uma estrutura padronizada e de certa forma similar a utilizada pelos módulos de *kernel* do Linux. A linguagem C foi escolhida devido a sua popularidade, a sua compatibilidade com sistemas operacionais baseados em *Unix* e ao excelente compromisso oferecido entre desempenho, eficiência, complexidade e flexibilidade de código [45].

De acordo com o que foi apresentando no Capítulo 5, a premissa básica da plataforma é a herança de propriedades contidas no módulo base por parte do módulo funcional, cuja implementação/configuração é de responsabilidade do desenvolvedor da aplicação de *AWE* específica. Assim, o módulo funcional da aplicação é uma extensão do módulo base fornecido.

Dessa maneira, é possível permitir a criação de módulos customizáveis que compartilham uma mesma estrutura básica organizada, abstraindo do desenvolvedor detalhes da estrutura de comunicação, configuração e operação do sistema, simplificando o processo de desenvolvimento e manutenção do *software*. Apesar de compartilharem da mesma estrutura de um módulo, os submódulos executam na forma de *threads*, e não de processos, justamente por sua relação de dependência direta.

Basicamente, a estrutura de módulo fornecida é formada por duas camadas funcionais integradas, uma responsável pela comunicação entre módulos e outra responsável por seu fluxo de execução, e por dois sistemas de configuração distintos, um estático utilizado na inicialização do módulo e um dinâmico utilizado durante a sua execução. As subseções a seguir apresentam detalhadamente as funcionalidades e a implementação de cada uma dessas estruturas, assim como algumas de suas características adicionais.

6.2.1 Fluxo de execução

De modo a fornecer um padrão estabelecido para a execução dos diferentes módulos que compõem uma aplicação de *AWE*, simplificando a organização do código na perspectiva do desenvolvedor que a constrói utilizando a plataforma, esta proporciona um fluxo de execução bem definido para o módulo. Esse fluxo consiste de duas etapas distintas e sequenciais: configuração e processamento. Implementadas na forma de funções da linguagem C, ambas possuem como argumento a estrutura do módulo e têm suas ações são definidas pelo desenvolvedor no processo de criação do módulo, de acordo com as suas necessidades funcionais.

A função de configuração é executada apenas um única vez, no início do fluxo de execução. Nesta etapa são realizados todos os procedimentos necessários à garantia da correta operação do módulo, como a configuração de seu parâmetros internos, a configuração dos mecanismos de comunicação entre módulos, a criação de submó-

dulos ou a execução de qualquer rotina necessária ao seu funcionamento. Uma vez concluída a etapa de configuração, o módulo passa a executar sua função de processamento.

Baseada no período de amostragem determinado pelo desenvolvedor na etapa de configuração, a função de processamento é executada de forma periódica até o termino da execução do módulo. Nesta etapa o módulo desempenha a funcionalidade principal para o qual foi concebido, como o cálculo de leis de controle, a leitura da medição de um ou mais sensores, o envio de um sinal para um ou mais atuadores ou qualquer outra funcionalidade específica.

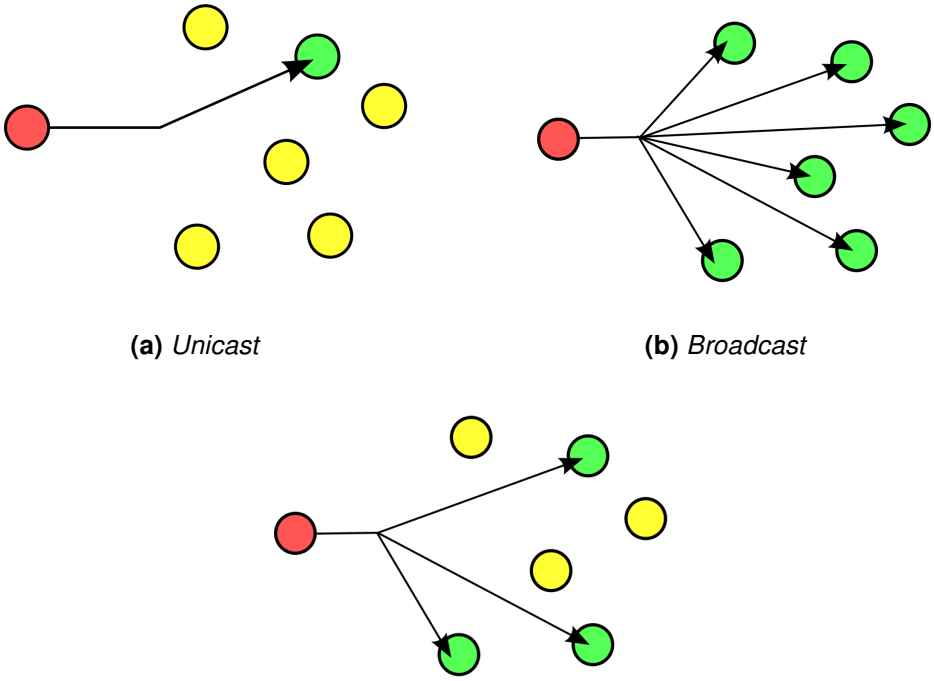
6.2.2 Comunicação entre módulos

Para fins de comunicação entre os módulos que compõe as aplicações, foi utilizada a infraestrutura de troca de mensagens e *marshalling*[1] fornecida pela biblioteca de código aberto *LCM* (*Lightweight Communications and Marshalling*) [46], que disponibiliza um modelo do tipo *publish/subscribe* associado a um mecanismo automático de *marshalling/unmarshalling* e geração de código. A escolha dessa biblioteca se deu por suas características de baixa latência e boa performance em cenários com restrições temporais, assim como pela sua utilização em aplicações comerciais como o *Volvo Car Group*, *BAE systems*, *Ford Motor Company* e *Google* [46].

A biblioteca utiliza *Multicast UDP* para realizar o transporte de mensagens com baixa latência, porém de forma não confiável, evitando assim a necessidade de um barramento centralizado ou de estabelecimento de conexão direta entre os módulos. *Multicast* é o processo de transmissão de dados para um número determinado de usuários através de múltiplos canais distribuídos. Geralmente, tal mecanismo envolve algum tipo de identificador ou canal, que permite aos receptores a sincronização com a transmissão de dados na rede. Assim, em contraste com uma transmissão *Unicast*, onde múltiplas cópias dos dados são transmitidas individualmente, e com uma transmissão *Broadcast*, onde todos os componentes da rede recebem as mensagens enviadas, a transmissão via *Multicast* permite o use eficiente dos recursos da rede [47]. A Figura 6.1 apresenta esta comparação de forma gráfica.

O formato e as informações de cada mensagem são padronizados através da sua especificação em uma linguagem de tipo formal em um arquivo de texto, de forma semelhante ao mecanismo utilizado pela biblioteca *Protocol buffers* da *Google*. A compilação desse arquivo de texto gera o código necessário para a transmissão e recebimento da mensagem, incluindo seu processo de *marshalling/unmarshalling*. Como o

[1] *Marshalling* é o processo de transformação da representação de memória de um objeto em um formato de dados compatível para armazenamento ou transmissão, similiar à serialização.



(a) *Unicast* **(b)** *Broadcast*

(c) *Multicast*

Figura 6.1: *Representação gráfica da transmissão de dados entre elementos em uma rede utilizando (a) Unicast; (b) Broadcast; e (c) Multicast.*

mecanismo *Multicast UDP* não exige o estabelecimento de conexão, a identificação do canal de cada mensagem é realizada por de uma palavra definida pelo desenvolvedor no momento de envio da mensagem.

A fim de simplificar ainda mais o processo de comunicação, a estrutura do base módulo realiza a configuração básica necessária ao funcionamento da biblioteca e disponibiliza ao usuário dois macros[2] para o envio e recebimento de mensagens. Para o envio de mensagens são passados como argumento da macro o módulo, o identificador do canal, o tipo da mensagem e a própria mensagem a ser enviada, ao passo que para registrar o recebimento de uma mensagem são especificados o tipo da mensagem a ser recebida, o identificador do canal pelo qual ela será enviada, e uma função de tratamento da mensagem, procedimento que deve ser realizado na etapa de configuração no fluxo de execução do módulo para cada mensagem a ser recebida durante sua operação. Essa função será executada a cada vez que a mensagem associada a mesma é recebida, e seus argumentos devem seguir um padrão estabelecido pela plataforma, o qual permite o acesso ao módulo e o armazenamento de informações

[2]Macros são uma ferramenta do pré-processador da linguagem C e, basicamente, são fragmentos de código aos quais são atribuídos uma espécie de abreviação, que quando usada durante o código será substituída pelo conteúdo do macro.

neste sem a necessidade da utilização de variáveis globais. É importante ressaltar que o recebimento de mensagens é realizado em uma *thread* dedicada a esta função, o que possibilita o troca de informações entre módulos/submódulos sem interferência no seu fluxo de execução.

Portanto, a plataforma, através da utilização da biblioteca *lcm*, fornece aos módulos um mecanismo de comunicação flexível e eficaz, que permite o envio e recebimento de mensagens de forma independente entre si, sem a necessidade de estabelecimento de uma conexão direta entre qualquer módulo do sistema.

6.2.3 Gerenciamento de parâmetros

Por ser uma tecnologia em desenvolvimento, ainda não existe uma configuração estabelecida como ideal para a operação de sistemas *AWE*, inclusive no que tange questões de controle do sistema. A fim de suportar as alterações de parâmetros da aplicação durante sua operação e a persistência desses durante vários ciclos de operação dos protótipos, um sistema de banco de dados foi desenvolvido utilizando uma *API* implementada sobre a ferramenta *SQlite* [48]. Carregado na inicialização do módulo, é de responsabilidade do desenvolvedor estabelecer a correta conexão entre os elementos do banco de dados e os parâmetros do módulo funcional em desenvolvimento na etapa de configuração do fluxo de execução da aplicação.

6.2.4 Configuração estática

Como os módulos do sistema foram implementados utilizando a linguagem de programação *C*, o código-fonte deve ser ser previamente compilado para a criação dos arquivos executáveis. A presença de diversos parâmetros individuais a cada módulo, como período de amostragem, parâmetros construtivos do sistema, parâmetros de configuração de operação e configurações para comunicação com dispositivos de *hardware* ou sistemas externos, torna o sistema mais propício a necessidade de alterações. Ademais, como alterações mínimas em qualquer elemento do código de um módulo implicam na necessidade de recompilar seu código para a atualização de seu arquivo executável, um sistema de configuração *offline* foi estabelecido através da utilização de um arquivo de texto e da implementação de um sistema capaz de interpretá-lo. Carregado na inicialização do módulo, é de responsabilidade do desenvolvedor estabelecer a correta conexão entre os elementos do arquivo de configuração e os parâmetros do módulo funcional em desenvolvimento na etapa de configuração do fluxo de execução da aplicação.

6.3 Características e dependências de *software* adicionais

Além das estruturas e funcionalidades supracitadas, a plataforma possui algumas características adicionais, implementadas a fim de torná-la mais flexível e completa. Além da definição das rotinas de configuração e processamento no momento da inicialização do módulo, o programador também tem a possibilidade de habilitar ou desabilitar a funcionalidade de receber mensagens de outros módulos, assim como de configurar o módulo como um submódulo, o que implica que sua execução ocorrerá em uma *thread* invocada pelo processo do módulo que o contém. Além disso, ao utilizar uma estrutura de comunicação baseada em *Multicast UDP*, a biblioteca *lcm* permite também a comunicação transparente entre módulos em unidades computacionais diferentes conectadas à mesma rede local, fornecendo ainda mais flexibilidade às aplicações.

A fim de complementar as funcionalidades básicas oferecidas pela plataforma, as seguintes bibliotecas foram minuciosamente selecionadas para fornecer todo o suporte necessário ao desenvolvedor da aplicação *AWE* durante esse processo:

- **C-periphery**: a biblioteca *C-periphery* fornece uma série de *APIs* na linguagem C para a interface de acesso aos *I/Os* periféricos em sistemas *Linux*, como *GPIO, SPI, I2C, MMIO e UART* [49]. Através da sua inclusão na plataforma, o desenvolvedor possui recursos disponíveis caso seja necessário implementar a comunicação com dispositivos de *hardware* periféricos, situação comum em sistemas de instrumentação aplicados à *AWE*.

- **LibWebsockets**: a biblioteca *LibWebsockets* fornece uma série de mecanismos na linguagem C para o estabelecimento de conexão com outros sistemas via *websockets* da rede [50]. Através da sua inclusão na plataforma, o desenvolvedor possui recursos disponíveis caso seja necessário implementar a comunicação com sistemas externos através da rede que sigam algum protocolo específico, situação comum em sistemas que utilizam configurações de *Hardware-in-the-loop* ou sistemas supervisórios.

- **JSON-ccan**: a biblioteca *JSON-ccan* fornece uma *API* na linguagem C para a codificação/decodificação de *JSON*[3]. Através da sua inclusão na plataforma, o desenvolvedor possui recursos disponíveis caso seja necessário implementar a

[3] *JSON* é uma sigla para *JavaScript Object Notation* e, basicamente, é um formato de arquivo padronizado que utiliza texto ”legível por humanos“ para a transmissão de objetos de dados consistindo de pares atributo-valor.

padronização de mensagens trocadas através de um mecanismo de comunicação que não forneça um padrão de forma intrínseca.

- **GSL**: a biblioteca(*GNU Scientific Library* (*GSL*) fornece uma ampla gama de rotinas matemáticas implementadas na linaguagem C, como por exemplo geradores de números aleatórios, funções especiais e minimos-quadrados para ajuste de curvas [51]. Através da sua inclusão na plataforma, o desenvolvedor possui recursos disponíveis caso seja necessário implementar operações matemáticas sofisticadas, situação comum em sistemas de controle complexos.

- **BLAS**: a biblioteca *Basic Linear Algebra Subprograms* (*BLAS*) fornece uma série de estruturas e rotinas que compõem blocos padronizados para a computação de operações com vetores e matrizes na linguagem C [52]. Através da sua inclusão na plataforma, o desenvolvedor possui recursos disponíveis caso seja necessário implementar operações com vetores ou matrizes, situação comum em sistemas multivariáveis, como estimadores de estado ou filtros.

6.4 Ferramentas

Para auxiliar a implementação da plataforma foram utilizadas e até mesmo desenvolvidas algumas ferramentas de suporte, relacionadas à organização do projeto e operação do sistema, tanto em nível estrutural quando em nível de planejamento e acompanhamento dos processos de desenvolvimento e implementação.

6.4.1 Controle de versão

O sistema de controle de versão distribuído *Git* foi utilizado para a realização de controle de versão durante o desenvolvimento deste trabalho, através da plataforma *GitLab* [53, 54]. O *Git* geralmente é utilizado a fim de monitorar e manter um registro das alterações realizadas em arquivos, o que auxilia a coordenação do desenvolvimento de *software* por múltiplos desenvolvedores. Assim, é possível ter controle sobre o progresso do desenvolvimento do sistema e de suas funcionalidades ao longo deste processo.

6.4.2 Integração continua

Integração contínua é uma prática de desenvolvimento que exige dos desenvolvedores a integração de código múltiplas vezes por dia em um repositório compartilhado.

Cada integração é verificada por um sistema automático, o que permite aos desenvolvedores identificar e localizar problemas/incompatibilidades rapidamente. Visto que a plataforma foi desenvolvida utilizando controle de versão, um sistema de integração continua foi implantado para garantir a integridade do código carregado no repositório remoto. Através de um sistema de *container*[4], chamada *Docker* [55], uma distribuição *Debian Linux* foi preparada com os pré-requisitos necessários para compilar a plataforma. Utilizando um serviço integrado ao próprio *GitLab*, um script de teste de compilação foi configurado a fim de permitir a integração do código apenas mediante sua aprovação neste.

6.4.3 Compilação automática

Como a plataforma é uma aplicação distribuída, na forma de vários módulos separados, a compilação manual de seus componentes é trabalhosa e indesejada. Além disso, dada a grande variedade de funcionalidades embutidas no sistema, é necessária a inclusão de algumas dependências de *software* para o funcionamento correto do sistema.

Assim, optou-se pela utilização de uma ferramenta denominada *CMake* a fim de automatizar esse processo [56]. Através de *scripts* em sua própria linguagem de programação é possível configurar as etapas de instalação, configuração e compilação do projeto, definindo caminhos de diretórios, bem como das dependências externas as quais devem ser instaladas antes da compilação propriamente dita. Para auxiliar a utilização da ferramenta, o projeto foi organizado em diretórios separados para as diferentes partes do sistema, permitindo assim a fácil adaptação da plataforma à aplicação de *AWE* a ser desenvolvida.

6.4.4 Gerenciamento de execução dos módulos

Como a plataforma foi projetada e implementada para suportar a criação de aplicações modulares, e não uma aplicação monolítica única, a inicialização do sistema deve ser adaptada. Para tal, a ferramenta *Supervisord* foi adotada, devido a sua simplicidade de configuração, manutenção e operação. O *Supervisord* é um sistema cliente/servidor que permite ao seus usuário monitorar e controlar processos em sistemas operacionais baseados em *Unix* [57].

[4]Nesse contexto, um *container* se refere a um recipiente no qual um sistema de *software* é configurado e isolado, a fim de servir como base para testes ou até mesmo a execução de uma aplicação de *software* [55]

O gerenciador dos processos é configurado através de um simples arquivo *INI*, que permite o estabelecimento de múltiplas opções por processo, como gerenciamento do *log* do processo ou sua reinicialização em caso de falhas. Além disso, o sistema fornece uma forma centralizada para controlar a execução de seus processos, tanto de forma individual ou em grupos, conforme estabelecido no arquivo de configuração. Finalmente, a ferramenta permite o monitoramento não somente local, mas também remoto, através de uma interface *web*, o que é interessante na perspectiva de sistemas embarcados. A Figura 6.2 apresenta um exemplo de interface *web* para o gerenciamento remoto de processos com o *supervisord* no contexto deste trabalho.

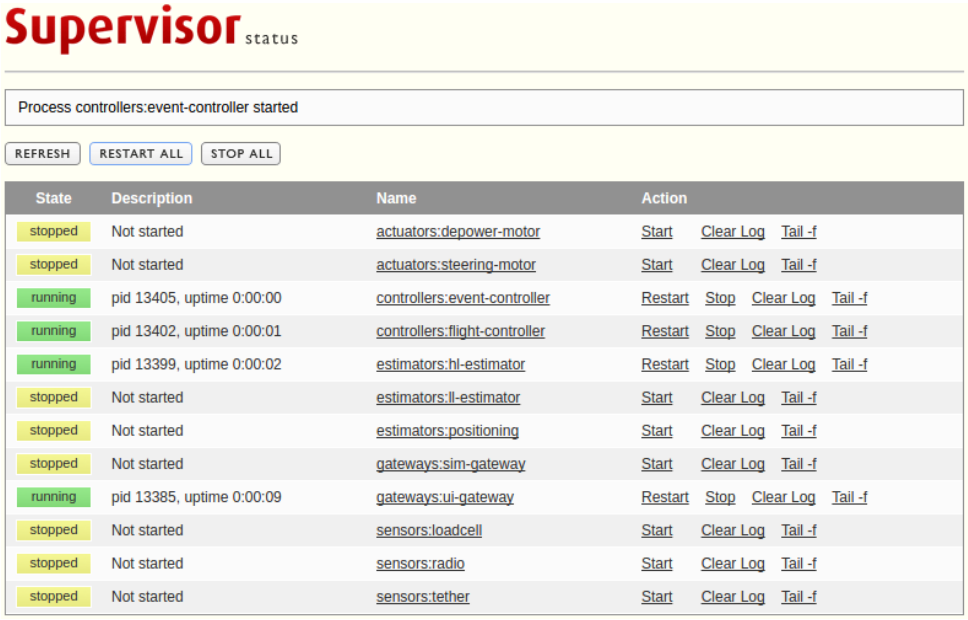


Figura 6.2: *Interface web para o gerenciamento remoto de processos com o supervisord.*

6.4.5 Depuração em tempo de execução

A fim de monitorar variáveis do sistema durante a execução da aplicação, sem a necessidade de depender do sistema supervisório, uma ferramenta gráfica foi desenvolvida utilizando a linguagem *Python*. A ferramenta é, basicamente, um *sniffer*[5] a ser executado em um computador conectado à mesma rede onde a plataforma está operando. Utilizando a biblioteca *lcm*, que permite o envio de mensagens através da rede local de forma transparente, a ferramenta captura mensagens que possuem um canal identificador específico, denominadas sondas. Essas mensagens foram estruturadas de forma a permitir um número flexível de elementos, possibilitando a utilização

[5]Um programa *sniffer*, também chamado de analisador de pacotes, é capaz de interceptar e registrar o tráfego de dados em uma rede de computadores, decodificando e analisando o seu conteúdo de acordo com algum protocolo definido [58].

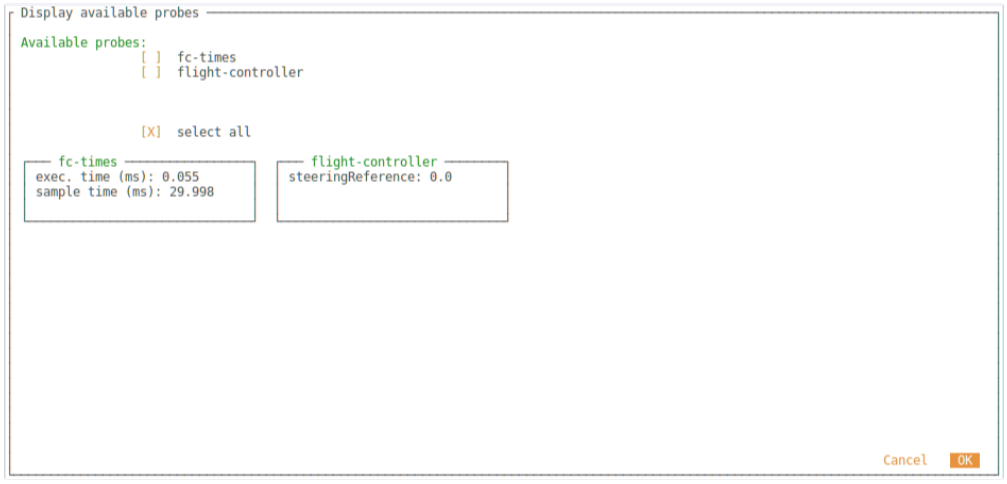
de um padrão de mensagem unificado entre a ferramenta e os módulos da aplicação. Ao utilizar essa estrutura padronizada para as mensagem, a ferramenta elimina a necessidade de alteração de seu código para exibir mensagens em diferentes formatos, sendo necessário apenas que o desenvolvedor acrescente a sonda desejada no código do módulo através da adição de uma chamada de função fornecida pela plataforma.

A Figura 6.3 apresenta as duas telas existentes na ferramenta: a tela inicial, onde o sistema registra novas sondas, e a tela de monitoramento, onde as mensagens de cada sonda são exibidas ao usuário de acordo com a sua escolha.

6.4.6 Registro de dados

Conforme mencionado neste capítulo, a biblioteca *lcm* permite a comunicação transparente entre diferentes unidades computacionais através da rede local. Além disso, a mesma possui uma ferramenta própria de captura de mensagens, a qual é capaz de registrar as mensagens trocadas entre os módulos da aplicação. Como as mensagens envolvem as variáveis de processo do sistema, este mecanismo pode ser utilizado como forma de registrar os dados da aplicação através de um computador conectado a rede local da aplicação de *AWE* durante sua operação para posterior análise. Por registrar a estrutura completa das mensagens, tal mecanismo permite também a sua reprodução, o que é útil para a análise do comportamento do sistema após a realização de experimentos.

(a) *Tela inicial*



(b) *Tela de monitoramento*

Figura 6.3: *Ferramenta de monitoramento de sondas de depuração dos módulos da aplicação AWE. (a) Tela inicial da ferramenta, e (b) Tela de monitoramento.*

Capítulo 7

Aplicação e Resultados

De maneira a validar a plataforma de *software* embarcado implementada, a mesma foi aplicada no desenvolvimento da nova versão do *software* do protótipo da unidade de controle de voo do grupo *UFSCkite*. Este capítulo sumariza esse processo, apresentando a metodologia empregada para testes e seus resultados. Para avaliar o comportamento do sistema do ponto de vista de suas principais características foram conduzidos testes de operação em laboratório e em campo. Além da apresentação dos resultados, os mesmos também são comparados com os requisitos de projeto da plataforma, a fim de avaliar seu desempenho.

7.1 Aplicação da plataforma em protótipo de sistema aerogeador

Conforme apresentado no Capítulo 4, o protótipo da unidade de controle de voo tem como principal objetivo a realização de testes de controle de voo de um aerofólio flexível no espaço, possuindo um papel fundamental no processo de desenvolvimento da tecnologia estudada pelo grupo. Além de permitir a verificação da performance da plataforma em operação, o processo de aplicação da plataforma de *software* possibilitou também a sua análise durante o processo de desenvolvido de *software*.

7.1.1 Módulos

Uma vez que a versão de *software* anterior dividiu o sistema em componentes de acordo com suas características funcionais, o processo de implementação da nova versão de *software* através da plataforma desenvolvida foi simplificado. Com base nessa versão, e com o apoio da equipe de engenheiros responsáveis pelo projeto do protótipo, o *software* embarcado foi decomposto em um conjunto de módulos funcionais, representados na Figura 7.1. Dada a estrutura modular da aplicação, as funcionalidades especificadas para a plataforma de *software* embarcado no Capítulo 5, e descritas em detalhe no Capítulo 6, como a configuração estática e dinâmica de parâmetros ou os mecanismos de troca de mensagens entre módulos, foram testadas

e aprimoradas de maneira incremental durante o processo de desenvolvimento dos módulos.

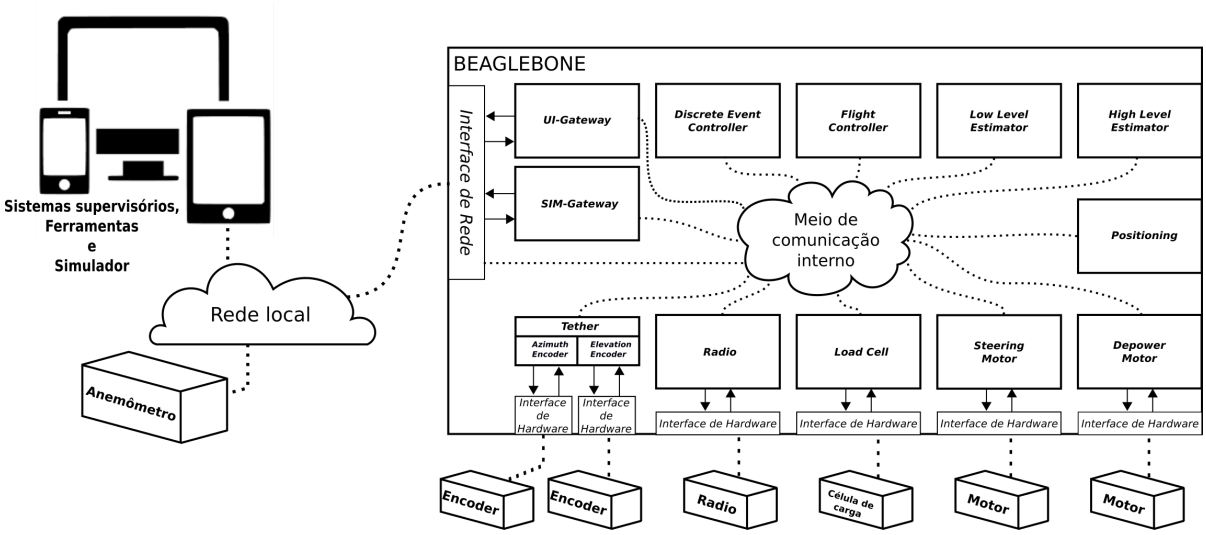


Figura 7.1: *Representação do protótipo da unidade de controle de voo sob a perspectiva dos módulos de software que o compõem e elementos externos.*

Por ser um sistema complexo, que envolve dispositivos de instrumentação operando em conjunto com múltiplas malhas de controle, o ajuste de diversos parâmetros é fundamental à sua correta operação. São exemplos desses parâmetros as especificações de comunicação com dispositivos de *hardware*, parâmetros de controladores, e características construtivas do sistema, como as dimensões e o peso do aerofólio ou a espessura e comprimento do cabo que o conecta ao protótipo.

A cada módulo foi atribuído um arquivo de configuração individual onde suas configurações estáticas foram concentradas, como seu período de amostragem ou aspectos de comunicação com dispositivos de *hardware*. Já os parâmetros dinâmicos, e que dizem respeito a múltiplos módulos, como ganhos de controladores ou características construtivas do sistema foram armazenados em um banco de dados ao qual todos os módulos possuem acesso através da *API* implementada.

Uma vez que tais sistemas de configuração foram testados e são funcionais, apesar da importância dos parâmetros do sistema no processo de operação, individualmente tais características não apresentam grande relevância na perspectiva da avaliação da performance da plataforma de *software*. Assim, considerando a natureza de tempo real do sistema e para fins de complementar a avaliação do desempenho da plataforma, foram estabelecidos requisitos temporais à cada um dos módulos do protótipo, em adição aos requisitos funcionais mencionados no capítulo 4. É importante ressaltar que apesar de possuir restrições de tempo real, o seu não cumprimento pontual por parte de um módulo não necessariamente afeta a operação do sistema de

forma global ou negativa.

O Quadro 7.1 apresenta os módulos de sensores implementados, os quais utilizam protocolos de comunicação de baixo nível para a realização de comunicação com os sensores propriamente ditos. O Quadro 7.2 apresenta os módulos de estimação e filtragem responsáveis por auxiliar o sistema de medição do protótipo, a fim de viabilizar sua operação e torná-la mais robusta frente a incertezas de medição. O Quadro 7.3 apresenta os módulos de atuação implementados, responsáveis pela atuação do protótipo sobre o aerofólio com base nos resultados dos cálculos das leis de controle. O Quadro 7.4 apresenta os módulos responsáveis pelo controle do sistema, seja com relação ao controle da dinâmica do processo ou dos eventos que o regulam. Com relação a topologia de controle implementada, a mesma consiste de um controlador de dois estágios similar ao descrito por [59], com ganhos ajustados de forma conservativa a fim de garantir uma resposta super-amortecida em malha fechada, resultando em uma trajetória de voo em lemniscata. Finalmente, o Quadro 7.5 apresenta os módulos de comunicação com sistemas externos, responsáveis por sua integração com os sistemas supervisórios e o simulador.

De forma geral, cada módulo foi implementado através da extensão da estrutura de módulo base fornecida pela plataforma, assim como seus sistemas de configuração de parâmetros. As funções de configuração e processamento de cada módulo/submódulo foram programadas de acordo com suas demandas funcionais, e auxiliaram diretamente na sua organização e na detecção de problemas durante a sua implementação. Além disso, as bibliotecas fornecidas pela plataforma foram capazes de suprir as necessidades funcionais de todos os módulos do sistema.

Quadro 7.1: *Descrição e restrição temporal estabelecida à cada módulo do conjunto de sensores que integram o sistema embarcado.*

Módulo	***Categoria***	***Descrição***	***Período de amostragem nominal (ms)***
Tether	módulo	Utilizado para a medição de posição e velocidade angular do cabo de tração do aerofólio através da composição e transformação das medições dos submódulos Azimuth e Elevation encoder ao eixo de referência do protótipo	30
Azimuth encoder	submódulo	Utilizado para a comunicação com o encoder responsável pela medição do ângulo azimute descrito pelo cabo de tração do sistema em seu eixo de referência	30
Elevation encoder	submódulo	Utilizado para a comunicação com o encoder responsável pela medição do ângulo de elevação descrito pelo cabo de tração do sistema em seu eixo de referência	30
Load cell	módulo	Utilizado para a comunicação com a célula de carga responsável pela medição da força de tração no cabo principal que conecta o aerofólio ao proótipo	30
Anemometer	módulo	Utilizado para a comunicação com o anemômetro responsável pela medição da velocidade e direção do vento	100
Radio	módulo	Utilizado para a comunicação com o sistema de rádios responsável pela medição de suas posições relativas ao aerofólio a fim de fornecer as informações necessárias ao algoritmo de multilateração do módulo "positioning"	30

Quadro 7.2: *Descrição e restrição temporal estabelecida à cada módulo do conjunto de estimadores e algoritmos de filtragem que integram o sistema embarcado.*

Módulo	***Categoria***	***Descrição***	***Período de amostragem nominal (ms)***
Positioning	módulo	Utilizado para o cálculo da posição do aerofólio no espaço através da multilateração de diferentes distâncias mensuradas entre o aerofólio e pontos definidos no espaço	30
High level estimator	módulo	Utilizado para o cálculo de estimações de variáveis de processo e parâmetros de alto nível do sistema, como as curvas aerodinâmicas do aerofólio ou a velocidade do vento.	30
Low level estimator	módulo	Utilizado para o cálculo de estimações de variáveis de processo e parâmetros de baixo nível do sistema, como o atraso de transporte ou compensações de controle.	30

Quadro 7.3: *Descrição e restrição temporal estabelecida à cada módulo do conjunto de atuadores que integram o sistema embarcado.*

Módulo	***Categoria***	***Descrição***	***Período de amostragem nominal (ms)***
Steering motor	módulo	Utilizado para a comunicação com o motor responsável pelo ângulo de rolagem do aerofólio, através do qual a sua trajetória de voo é controlada	30
Depower motor	módulo	Utilizado para a comunicação com o motor responsável pelo ângulo de ataque do aerofólio, através do qual a sua resistência ao vento incidente é controlada	60

Quadro 7.4: *Descrição e restrição temporal estabelecida à cada módulo do conjunto de controladores que integram o sistema embarcado.*

Módulo	***Categoria***	***Descrição***	***Período de amostragem nominal (ms)***
Flight controller	módulo	Utilizado para o cálculo das leis de controle que regem a trajetória de voo do aerofólio	30
Discrete event controller	módulo	Utilizado para a determinação do estado atual do sistema, controlando assim seu comportamento	30

Quadro 7.5: *Descrição e restrição temporal estabelecida à cada módulo do conjunto de interfaces com sistemas externos que integram o sistema embarcado.*

Módulo	***Categoria***	***Descrição***	***Período de amostragem nominal (ms)***
UI-Gateway	módulo	Utilizado para a comunicação com os sistemas supervisórios do sistema (GUIs), enviando informações de todos os módulos a estes, assim como recebendo e encaminhando informações das interfaces para os módulos embarcados	50
SIM-Gateway	módulo	Utilizado para a comunicação com o simulador do sistema, enviando e recebendo variáveis de processo e controle, assim como repassando informações recebidas das GUIs através do módulo UI-Gateway	30

7.2 Testes funcionais

Contudo, este é um sistema complexo, e testar o seu funcionamento de forma eficiente e completa é uma tarefa de similar complexidade. A seguir são descritos os procedimentos adotados para a realização de testes funcionais em laboratório e em campo, bem como são apresentados e analisados os resultados obtidos, tanto na perspectiva do protótipo quanto da plataforma de *software*.

7.2.1 Testes em laboratório

A estrutura do sistema e as condições requeridas à sua operação inviabilizam a realização de testes completos em laboratório. Para amenizar tal dificuldade, o grupo

UFSCkite desenvolveu uma ferramenta de simulação do ambiente de voo e das dinâmicas do protótipo. A utilização desta em conjunto com o *software* embarcado possibilitou a realização de testes em condições controladas, permitindo a avaliação do sistema como um todo e de seus componentes individualmente, antes de levá-lo a campo.

Durante todos os testes realizados, a simulação do sistema considerou as dimensões do protótipo da unidade de controle de voo e do aerofólio descritas no Capítulo 4, de maneira a operar da forma mais similar possível às condições reais encontradas em campo. Visto que o simulador é responsável por reproduzir as dinâmicas do sistema e as condições do ambiente, os módulos utilizados no teste se limitaram aos módulos de controle, estimação e interface com sistemas externos, não utilizando portanto os módulos de instrumentação do sistema. O diagrama da Figura 7.2 apresenta a configuração do sistema utilizada para a realização dos testes em laboratório.

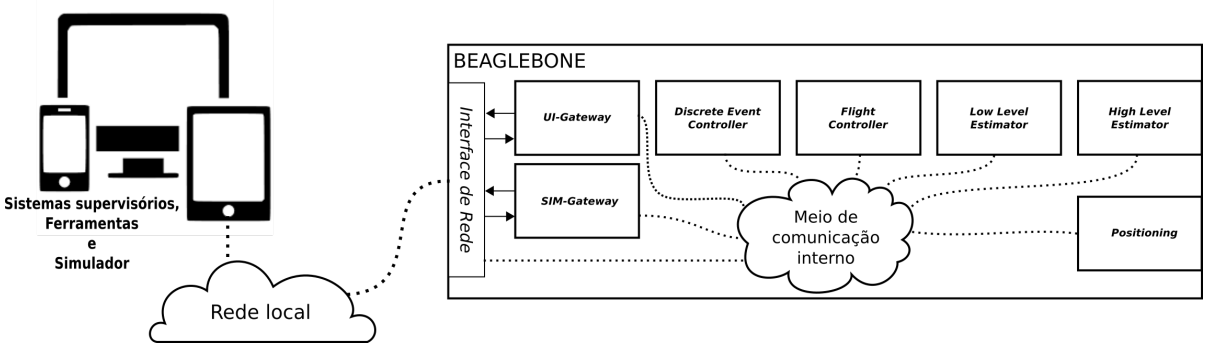


Figura 7.2: *Diagrama representando a configuração do sistema utilizada para a realização de testes em laboratório com o protótipo da unidade de controle de voo.*

7.2.1.1 Resultados

Os principais objetivos deste teste foram a verificação das funcionalidades básicas oferecidas pela plataforma, especificadas através dos requisitos citados no capítulo 4, a verificação do desempenho de cada módulo durante a operação, e a avaliação da performance do sistema com relação ao controle da trajetória de voo do aerofólio.

Durante a operação foram testadas e aprovadas as funcionalidades fornecidas pela plataforma, como a utilização de arquivos de configuração, a alteração e sincronização de parâmetros do sistema através das interfaces gráficas de usuário de forma dinâmica e o mecanismo de troca de mensagens entre os módulos. A Figura 7.3 apresenta uma das telas de operação do sistema supervisório durante a realização do teste.

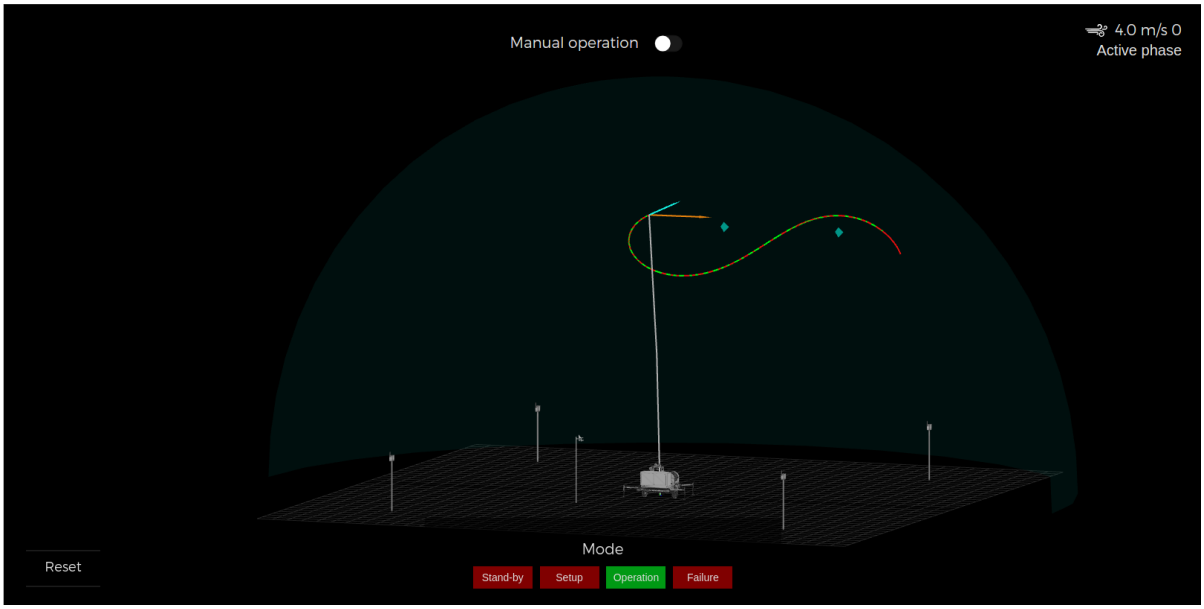
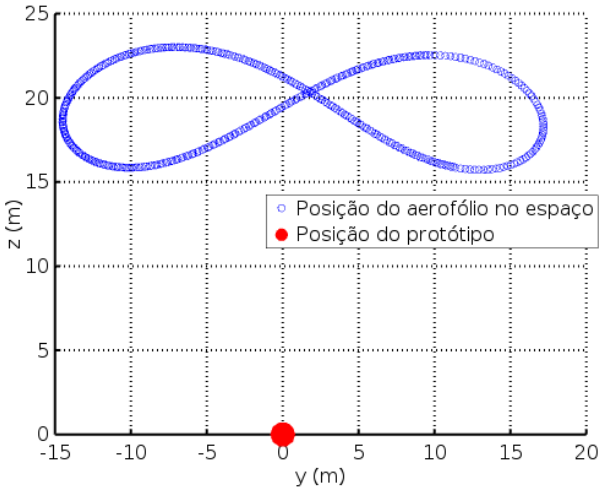
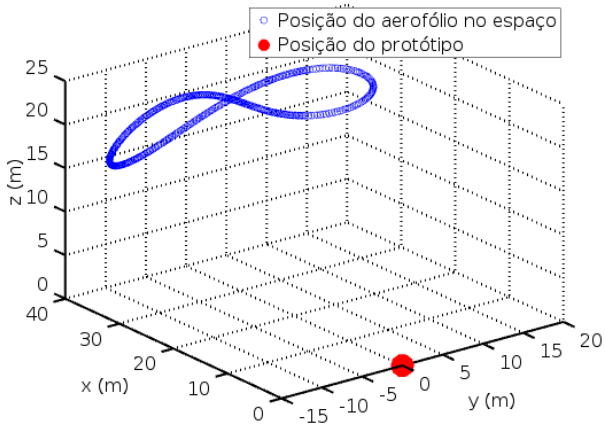


Figura 7.3: *Tela de operação do sistema supervisório do protótipo durante sua operação em laboratório.*

Também foram avaliados aspectos relativos à capacidade funcional do *software* desenvolvido para o controle do protótipo. A Figura 7.4 apresenta um trecho da trajetória descrita pelo aerofólio no espaço durante a operação do sistema embarcado em conjunto com o simulador. Os dados foram obtidos durante a operação através da ferramenta de registro de dados, e analisados posteriormente com o auxilio de ferramentas gráficas. Na perspectiva da plataforma de *software* o desempenho obtido foi excelente, visto que nessa configuração foram realizados múltiplos testes, durante as quais o sistema se manteve estável e não apresentou falhas.



(a)



(b)

Figura 7.4: *Trecho da trajetória descrita pelo aerofólio no espaço durante experimento em laboratório, obtida através da ferramenta de log fornecida pela plataforma. (a) Vista frontal; (b) Vista em perspectiva.*

Foram avaliados e aprovados também aspectos operacionais da plataforma, especificamente as ferramentas fornecidas para o controle dos processos e para a depuração de sondas em tempo de execução. A Figura 7.5 apresenta a tela do sistema de supervisão de processos utilizado para gerenciar a execução dos módulos do *software* embarcado.

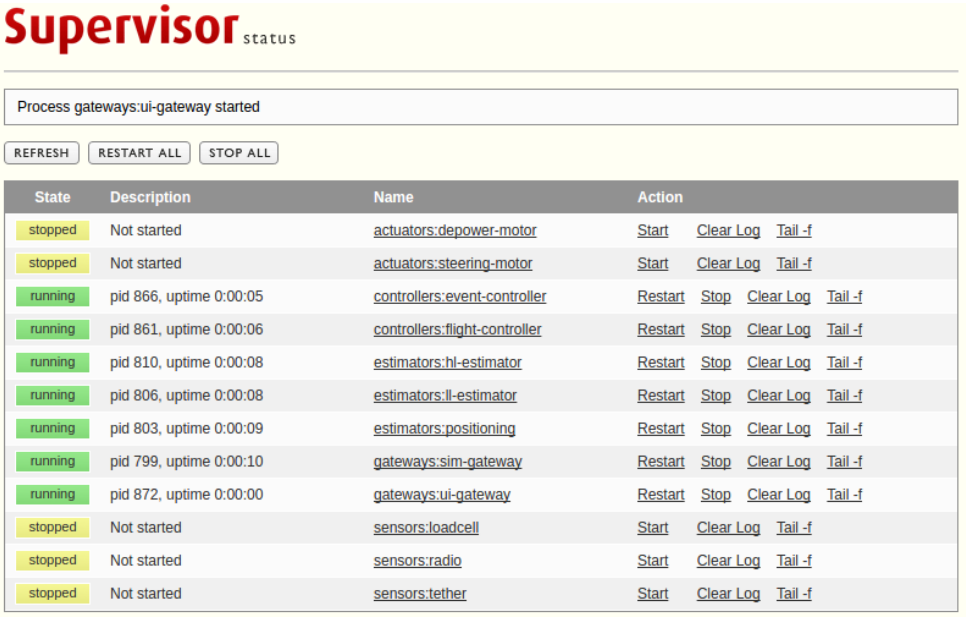


Figura 7.5: *Ferramenta de supervisão de processos supervisord durante experimento em laboratório.*

A Figura 7.6 apresenta a tela do sistema de depuração de sondas utilizado para depurar variáveis de interesse dos desenvolvedores, como por exemplo os tempos consumidos por cada módulo para a execução de sua função de processamento e o intervalo de tempo entre duas execuções consecutivas.

Figura 7.6: *Ferramenta de depuração de sondas em tempo de execução durante experimento em laboratório.*

Os módulos e submódulos foram instrumentados de forma a mensurar o seu período de amostragem a cada ciclo de execução da sua função de processamento. Assim, foi possível verificar a capacidade da estrutura fornecida pela plataforma de cumprir com os requisitos temporais estabelecidos. A Tabela 7.6 apresenta o período de amostragem nominal de cada módulo, o período de amostragem médio observado e seu desvio padrão durante a operação do sistema, assim como o número de amostras em que o valor observado excedeu em mais do que 5% o nominal neste período. Apesar de não garantir a execução das funções de processamento de forma crítica, visto que a plataforma não possui até o momento nenhum mecanismo de prioridade de execução implementado, o comportamento do sistema foi excelente, cumprindo de maneira satisfatória com os requisitos temporais estabelecidos.

Tabela 7.6: *Módulos utilizados durante o experimento em laboratório, seus períodos nominais de ciclo, as médias e desvios padrão observados e o número de amostras que excederam em 5 % ou mais o valor nominal estabelecido.*

Módulo	Tempo de ciclo (ms)			
	Nominal	**Média observada**	**Desvio padrão observado**	**>5%**
UI-Gateway	50	49.984	0.043	2
SIM-Gateway	30	29.995	0.035	1
Discrete Event Controller	30	29.982	0.032	1
Flight Controller	30	29.821	0.023	3
Low Level Estimator	30	29.962	0.033	2
High Level Estimator	30	29.893	0.048	1
Positioning	30	30.04	0.024	1

7.2.2 Teste preliminar em campo

Uma vez que o *software* embarcado foi capaz de controlar o aerofólio de maneira satisfatória operando em conjunto com o simulador em laboratório, a próxima etapa do processo de desenvolvimento foi testá-lo em campo. A realização de um teste preliminar em campo possibilitou a avaliação da performance do sistema e das funcionalidades propostas no contexto real de sua aplicação.

Nesse cenário, o sistema necessita de seus módulos de instrumentação, além daqueles utilizados em testes em laboratório, à exceção do módulo de interface com o simulador. O diagrama da Figura 7.7 apresenta a configuração do sistema utilizada para a realização do teste em campo. Já a Figura 7.8 apresenta uma das telas de operação do sistema supervisório durante a realização do teste.

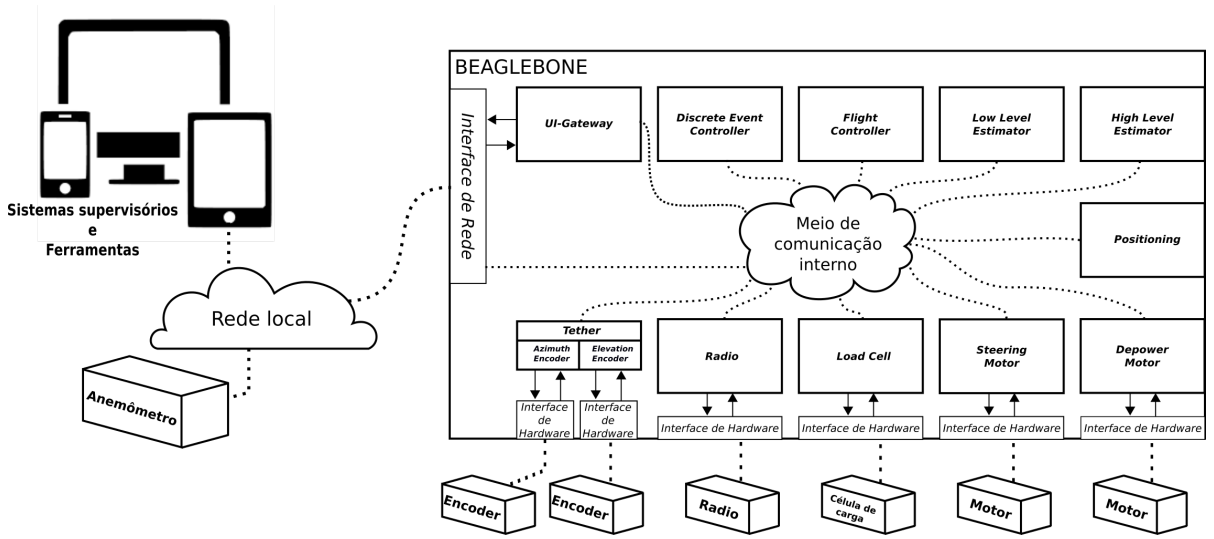


Figura 7.7: *Diagrama representando a configuração do sistema utilizada para a realização do teste em campo com o protótipo da unidade de controle de voo.*

7.2.2.1 Resultados

O principal objetivo deste teste foi a utilização do sistema de instrumentação completo em conjunto com as malhas de controle em condições de operação reais. Apesar de que as funcionalidades da plataforma foram testadas e aprovadas em laboratório, a verificação do desempenho de cada módulo durante a operação e a avaliação da performance do sistema com relação ao controle da trajetória de voo do aerofólio também foram objetivos do teste preliminar em campo.

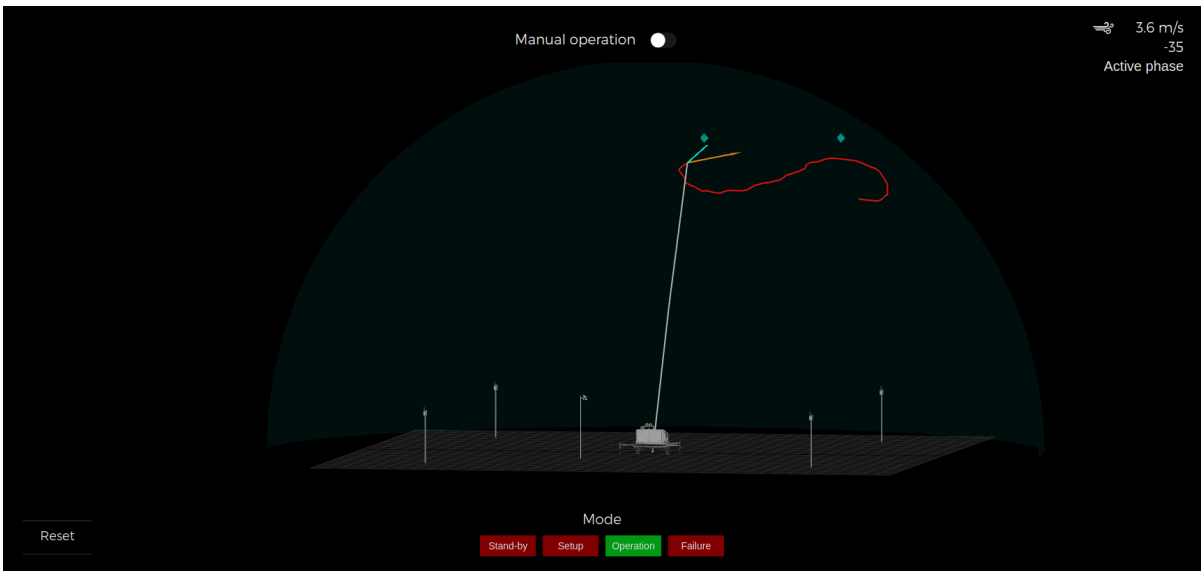
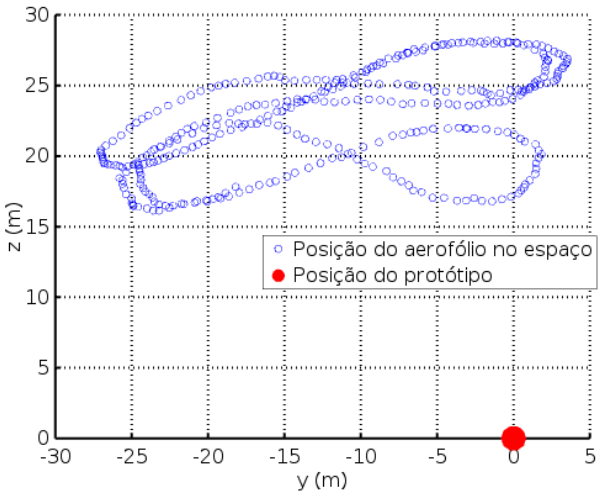
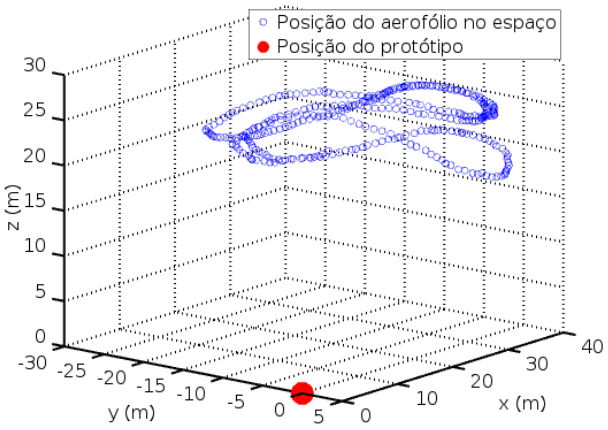


Figura 7.8: *Tela de operação do sistema supervisório protótipo durante sua operação em campo.*

Também foram avaliados aspectos relativos à capacidade funcional do *software* desenvolvido para o controle do protótipo. A Figura 7.9 apresenta um trecho da trajetória descrita pelo aerofólio no espaço durante sua operação. A diferença entre a trajetória observada com relação ao cenário de laboratório se dá principalmente por conta da dificuldade de reproduzir o ambiente e as dinâmicas do sistema em simulação. Na perspectiva da plataforma de *software* o desempenho obtido foi excelente, visto que o protótipo realizou o voo mais estável desde o inicio do seu desenvolvimento, e não apresentou falhas relacionadas ao *software* durante toda a sua operação. A Figura 7.10 apresenta o protótipo controlando o aerofólio durante a realização do experimento.



(a)



(b)

Figura 7.9: *Trecho da trajetória descrita pelo aerofólio no espaço durante experimento em campo, obtida através da ferramenta de log fornecida pela plataforma. (a) Vista frontal; (b) Vista em perspectiva.*

(a)

(b)

Figura 7.10: *Imagens do experimento realizado em campo com o protótipo.*

As ferramentas fornecidas pela plataforma para o controle dos processos e para a depuração de sondas em tempo de execução foram utilizadas a fim de auxiliar e simplificar a realização do teste. A Figura 7.11 apresenta a tela do sistema de supervisão de processos utilizado para gerenciar a execução dos módulos do *software* embarcado, e a Figura 7.12 apresenta a tela do sistema de depuração de sondas utilizado para depurar variáveis de interesse dos desenvolvedores.

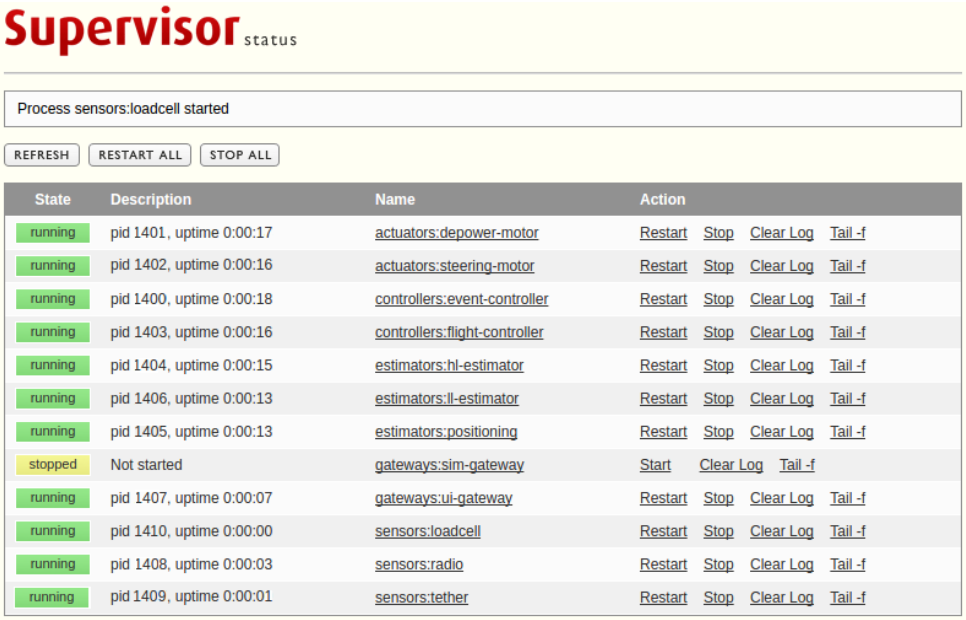


Figura 7.11: *Ferramenta de supervisão de processos supervisord durante experimento em campo.*

Da mesma forma que durante os testes em laboratório, os módulos e submódulos foram instrumentados de forma a mensurar o seu período de amostragem a cada ciclo de execução da sua função de processamento. Assim, foi possível verificar a capacidade da estrutura fornecida pela plataforma de cumprir com os requisitos temporais estabelecidos. A Tabela 7.7 apresenta o período de amostragem nominal de cada módulo, o período de amostragem médio observado e seu desvio padrão durante a operação do sistema, assim como o número de amostras em que o valor observado excedeu em mais do que 5% o nominal neste período. Mesmo com o número elevado de módulos e submódulos ativos com relação à configuração de laboratório, 13 no total, o comportamento do sistema foi excelente, cumprindo com os requisitos temporais estabelecidos de maneira satisfatória.

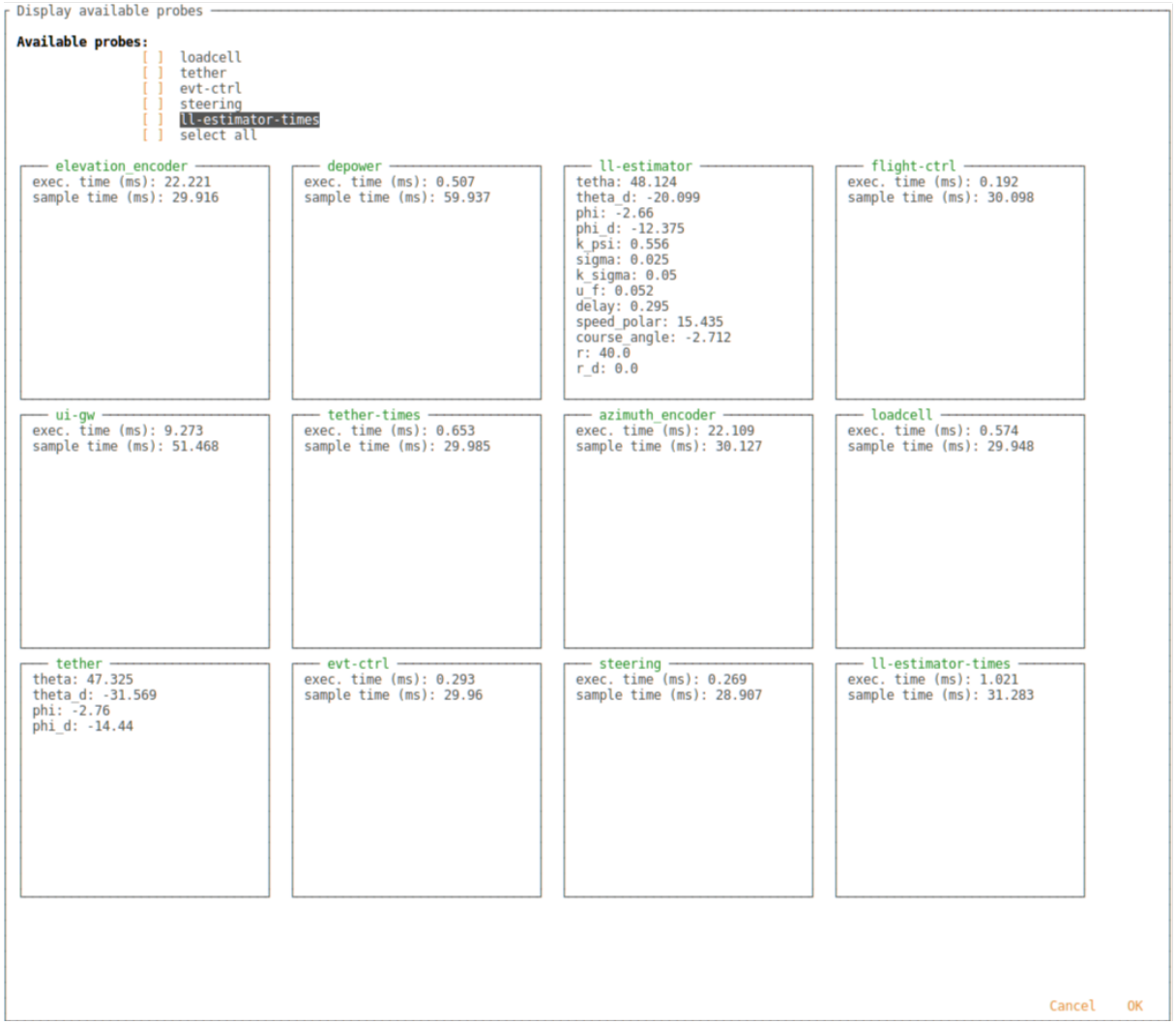


Figura 7.12: *Ferramenta de depuração de sondas em tempo de execução durante experimento em campo.*

Tabela 7.7: *Módulos utilizados durante o experimento em campo, seus períodos nominais de ciclo, as médias e desvios padrão observados e o número de amostras que excederam em 5 % ou mais o valor nominal estabelecido.*

Módulo/Submódulo	**Tempo de ciclo (ms)**			
	Nominal	**Média observada**	**Desvio padrão observado**	**>5%**
UI-Gateway	50	49.964	0.045	3
Discrete Event Controller	30	29.942	0.036	1
Flight Controller	30	29.895	0.033	2
Low Level Estimator	30	29.914	0.036	1
High Level Estimator	30	29.937	0.043	3
Positioning	30	29.959	0.021	2
Radio	30	29.998	0.056	3
Tether	30	30.023	0.029	2
Azimuth Encoder	30	29.898	0.031	1
Elevation Encoder	30	29.974	0.048	1
Load Cell	30	29.856	0.026	1
Steering Motor	30	29.996	0.038	1
Depower Motor	60	60.017	0.032	1

7.3 Observações

Conforme mencionado ao longo deste Capítulo, a aplicação da plataforma de *software* embarcado ao protótipo da unidade de controle de voo foi de grande utilidade para sua avaliação. A seguir são discutidos alguns dos pontos observados ao longo do desenvolvimento e operação da aplicação através da plataforma proposta neste trabalho, na perspectiva de suas características e ferramentas:

- **Estrutura modular:** apesar de criar a necessidade do estabelecimento de padrões e mecanismos de comunicação para garantir o funcionamento de malhas de controle de forma satisfatória, a opção por uma estrutura de módulos funcionais na forma de processos independentes atribuiu grande flexibilidade ao sistema, que foi essencial ao cumprimento de diversos dos requisitos estabelecidos. Isso foi evidenciado não apenas no processo de desenvolvimento da própria plataforma, como também nas etapas de aplicação desta ao protótipo e sua posterior realização de experimentos em diferentes modos de operação.

- **Fluxo de execução:** o estabelecimento de um fluxo de execução padrão para cada módulo, composto por uma etapa de configuração e uma etapa de processamento periódica, foi essencial para simplificar a organização do protótipo na perspectiva do desenvolvimento do *software*. Tal estrutura também facilitou a manutenção e depuração ao longo do processo de aplicação da plataforma,

visto que cada etapa programada foi bem definida de acordo com as suas funcionalidades.

- **Mecanismo de comunicação:** a utilização de um mecanismo de comunicação entre módulos baseado em *Multicast UDP* utilizando a biblioteca *lcm* apresentou excelentes resultados. Além de suas características de baixa latência e boa performance, a mesma excluiu a necessidade do estabelecimento de uma conexão direta entre os módulos, e permitiu também a comunicação transparente entre diferentes unidades computacionais através da rede local. Através da sua execução na forma de uma *thread* associada a cada módulo, foi possível desacoplar a execução do mecanismo de comunicação da etapa de processamento do módulo.

- **Mecanismo de configuração:** eliminando a necessidade de alterar o código fonte de cada módulo para realizar alterações pontuais de parâmetros, o mecanismo de configuração destes através de um arquivo de texto permitiu a fácil manutenção do sistema frente a mudanças de características específicas aos módulos, como seus períodos de amostragem ou configurações de interfaces com dispositivos de instrumentação.

- **Gerenciamento de parâmetros:** a utilização do banco de dados, e sua implementação através da estrutura do *SQlite*, mostrou-se vantajosa na perspectiva do gerenciamento de parâmetros do protótipo, e também na sua operação. Dada a complexidade do sistema e o elevado número de parâmetros configuráveis compartilhados por componentes diversos, a centralização destes tornou ágil e simples a sua correta configuração durante a realização dos experimentos em laboratório e em campo.

- **Ferramentas de suporte ao desenvolvimento:** a utilização de um sistema do controle de versão *Git*, associado a um mecanismo de integração continua do *GitLab*, foi fundamental para garantir a integridade da plataforma desenvolvida, principalmente devido ao grande número de funcionalidades e componentes integrados a esta. Posteriormente, durante a sua aplicação, estes mecanismos apresentaram grande valor no controle das versões funcionais do sistema. Além disso, a utilização do mecanismo de compilação automática *CMake* para auxiliar o processo de integração de diferentes dependências de *software* e os módulos desenvolvidos também simplificou e tornou mais eficaz o processo de desenvolvimento. Finalmente, o conjunto de dependências selecionado para dar suporte ao desenvolvimento foi utilizado em toda sua extensão, e se mostrou capaz de suprir as necessidades desse processo com sucesso.

- **Ferramentas de suporte a operação:** na perspectiva da operação do sistema, a ferramenta de gerência de processos *Supervisord* permitiu simplificar e organizar o processo de inicialização e execução dos módulos. Apesar de ainda não estar integrada com o sistema supervisório utilizado pelo grupo *UFSCkite*, a ferramenta oferece suporte para isso através da sua interface gráfica de gerenciamento remoto. A ferramenta de depuração em tempo de execução, apesar de simples, mostrou-se útil na análise do comportamento de componentes complexos da aplicação, como os estimadores de estado. A mesma também foi útil para observar o desempenho de cada módulo com relação aos seus requisitos temporais, através da instrumentação do sistema com suas sondas de depuração, processo que foi facilitado pela organização modular do sistema. De forma similar ao gerenciador de processos, a mesma oferece suporta a sua integração com o sistema supervisório. O mecanismo de registro de mensagens provido pela biblioteca *lcm* apresentou-se como uma boa solução para a realização do registro das variáveis de processo durante os experimentos. Contudo, este apresenta dificuldades com relação ao registro das configurações do sistema e alterações nestas, visto que outros mecanismos além da troca de mensagens são utilizados para realizar esta tarefa. Assim, a fim de torná-lo completo, é necessário adaptá-lo a esta situação.

- **Performance da plataforma:** apesar da plataforma não ter como foco a otimização da performance computacional das aplicações desenvolvidas, visto que é de responsabilidade do desenvolvedor da aplicação programar as rotinas funcionais de cada módulo, a estrutura implementada apresentou um excelente desempenho na execução do sistema de controle do protótipo. Ademais, a opção pela implementação das estruturas de módulo base da plataforma na linguagem C mostrou-se sólida durante todo o processo de elaboração deste trabalho. Além da sua capacidade de produzir aplicações computacionalmente eficientes e de ser uma linguagem bem estruturada, a sua compatibilidade inerente com sistemas *Linux* ofereceu uma vasta gama de ferramentas e funcionalidades compatíveis com as necessidades de um sistema embarcado, como alta performance, flexibilidade e acessibilidade aos elementos periféricos de *hardware*, essenciais à elaboração da plataforma de *software* e sua aplicação.

Capítulo 8

Considerações finais

Este documento apresentou um fluxo de projeto completo referente ao desenvolvimento de uma plataforma de *software* embarcado para protótipos de sistemas aerogeradores com aerofólios cabeados, bem como a sua aplicação e a realização de testes com o protótipo da unidade de controle de voo do grupo *UFSCkite*. Os resultados obtidos foram excelentes, e apesar de possuir pontos passivos de melhoria, a plataforma foi capaz de fornecer flexibilidade ao processo de desenvolvimento e operação do protótipo, mantendo uma estrutura de simples e de fácil manutenção.

Inicialmente, a fim de contextualizar o cenário energético global e suas tendências, foram apresentados dados concretos sobre a utilização e previsões para demandas energéticas. Em seguida, foi introduzida a tecnologia de aerogeradores com aerofólios cabeados e a situação atual do grupo *UFSCkite* no cenário de desenvolvimento de protótipos. Nesse contexto, foram apresentadas as motivações para o desenvolvimento de uma plataforma de *software* voltada a criação e operação de sistemas aerogeradores com aerofólios cabeados. Além de fatores como a existência de múltiplos protótipos que necessitam de um padrão bem estabelecido para a sua elaboração, a plataforma tem como propósito simplificar e fornecer maior flexibilidade ao sistemas e seu processo de desenvolvimento, assim como torná-los menos vulneráveis a mudanças em diferentes partes do sistemas, as quais são relativamente comuns durante o desenvolvimento de protótipos.

O desenvolvimento do trabalho teve início com uma revisão bibliográfica contendo os principais pontos relevantes à sua compreensão, envolvendo os aspectos teóricos necessários para a sua realização. Foram apresentados conceitos relevantes ao desenvolvimento de uma plataforma de *software* para sistemas embarcados, como sistemas ciber-físicos e seus componentes, arquiteturas e plataformas de *software*, sistemas operacionais e seu funcionamento, protocolos de comunicação e suas especificações. Em seguida, foram abordados tópicos relacionados a sistemas de *AWE* em geral, introduzindo algumas de suas principais características, funcionalidades e especificações. O auxílio do laboratório *UFSCkite* e seus integrantes possibilitou a construção rápida de uma fundamentação teórica completa e pautada pelo problema que se desejava resolver. Esse estudo foi muito útil durante a elaboração do sistema como um todo, e permitiu a identificação prévia de eventuais problemas logo no início

deste processo, guiando muitas das decisões tomadas. Além disso, a pesquisa realizada proporcionou a ampliação dos conhecimentos da grupo *UFSCkite* quanto ao projeto de *software* para sistemas embarcados e, atualmente, a compilação construída durante este trabalho é utilizada como base para o desenvolvimento de *software* pelo grupo.

Uma vez que os conhecimentos necessários foram apresentados, o documento expõe a análise detalhada do problema a ser resolvido, definindo os requisitos necessários à plataforma de *software*, com base nas versões anteriores do *software* utilizado pelo grupo e nas necessidades de sistemas de *AWE*. Essa etapa envolveu a contribuição de outros membros do grupo no sentido de auxiliar na identificação de quais seriam as funcionalidades desejadas, bem como para avaliar se os requisitos estabelecidos estavam de acordo com o planejamento do projeto como um todo.

As etapas seguintes foram as fases de projeto e implementação da plataforma de *software* propriamente ditas, que tomaram como diretriz os critérios estabelecidos anteriormente e descrevem detalhadamente as decisões envolvidas na escolha da estrutura baseada em módulos funcionais, os elementos que compõe a plataforma, as suas principais características, assim como as ferramentas e técnicas empregadas na sua concepção. Implementada principalmente na linguagem C de programação e visando inicialmente a sua utilização em *single board computers* de baixo custo operando com sistemas operacionais *Linux*, a plataforma proposta foi projetada de forma a permitir a decomposição da aplicação de *AWE* em módulos funcionais altamente desacoplados. A utilização da estrutura modular atribui grande flexibilidade a esta, facilitando os processos de desenvolvimento, manutenção e documentação do *software*. Os módulos são executados de maneira distribuída na forma de processos indepedentes, possivelmente sobre múltiplas unidades computacionais, e são capazes de trocar informações através de uma infraestrutura de comunicação padronizada de alta-performance, baseada no padrão *publisher-subscriber*. O uso de *threads* e da biblioteca *lcm* permitiu a separação da camada de comunicação entre módulos e da camada de processamento de suas funcionalidades, assim como a troca de mensagens sem a necessidade do estabelecimento de uma conexão direta. Além de fornecer um conjunto cuidadosamente selecionado de dependências, as quais são coletadas e instaladas durante uma etapa de construção automática, a plataforma também oferece uma série de mecanismos ao desenvolvedor, incluindo ferramentas de acionamento remoto, monitoramento em tempo real, registro de dados, depuração e instrumentação de código.

A seguir, o documento descreve a aplicação da plataforma implementada e suas ferramentas no desenvolvimento do *software* embarcado para o protótipo da unidade de controle de voo do grupo *UFSCkite*. Durante esse processo, a plataforma e suas

ferramentas apresentaram resultados excelentes, atestados pelo cumprimento de todos os requisitos e especificações atribuídas a sua utilização na aplicação a protótipos de *AWE*. Em seguida, são apresentados os resultados e a metodologia proposta para os testes de avaliação de desempenho e performance da plataforma de *software* durante a operação do protótipo, tanto sob a perspectiva das funcionalidades requeridas por este, quanto sob aspectos relevantes ao *software*. Os cenários dos testes em laboratório e do teste preliminar em campo são apresentados, incluindo os resultados obtidos. Em ambos os cenários o resultado observado foi excelente, inclusive com relação aos objetivos de controle da trajetória do aerofólio no espaço e aos requisitos estabelecidos pelo protótipo ao sistema de *software* durante sua operação, como restrições temporais e requisitos funcionais. Finalmente, são apresentadas observações realizadas durante estes experimentos, referentes a aplicação e utilização da plataforma.

Durante sua realização, o trabalho proporcionou ao autor o contato com um problema real de engenharia, o qual permitiu a aplicação de conhecimentos adquiridos ao longo da graduação. O mesmo também possibilitou aprofundar os conhecimentos em diversas áreas de suma relevância para um engenheiro de Controle e Automação. Além dos aspectos técnicos, o mesmo também forneceu a oportunidade de trabalhar em equipe com profissionais qualificados, os quais foram fundamentais para o sucesso deste trabalho. Do ponto de vista das expectativas, espera-se que a plataforma de *software* desenvolvida seja refinada a fim de continuar sendo utilizada como base para os diferentes protótipos em desenvolvimento pelo grupo *UFSCkite*,e que este documento seja proveitoso para o grupo no sentido de fundamentar esse processo, principalmente no que diz respeito a reutilização do fluxo de projeto adotado.

Algumas etapas de trabalhos não documentadas já estão em desenvolvimento. Esse é o caso do desenvolvimento de *software* para os protótipos da unidade de solo do grupo *UFSCkite*, essencial a operação do sistema na configuração *pumping-kite*. Inicialmente, estes protótipos serão utilizados para a realização de testes de geração elétrica em laboratório em uma configuração de *Hardware-in-the-loop*, e uma vez funcionais, serão utilizados para testes de geração em campo, em conjunto com a uma nova versão do protótipo da unidade de controle de voo descrita neste trabalho.

Além do desenvolvimento do *software* embarcado para os protótipos da unidade de solo, outra tarefa em andamento é o aprimoramento das ferramentas fornecidas em conjunto com a plataforma, no sentido de tornar a sua utilização mais simples durante a realização de testes em campo através da integração destas ao sistema supervisório desenvolvido pelo grupo.

Do ponto de vista de melhorias, um trabalho futuro que necessita ser realizado é a

comparação da solução fornecida pela plataforma com outras plataformas de *software* embarcado que se adequam aos requisitos de sistemas aerogeradores com aerofólios cabeados, a fim de identificar pontos de melhoria. Ademais, a fim de otimizar seu desemepenho, existe a necessidade do estabelecimento de critérios temporais e de performance rigorosos, processo que requer a completa instrumentação da plataforma e do desenvolvimento de uma metedologia de testes para esta. Outro ponto passivo de ser trabalhado é a criação de um banco de módulos otimizados para aplicações de *AWE*, para a aplicação em protótipos variados com diferentes configurações.

Finalmente, o desenvolvimento e resultado deste trabalho será apresentado na Conferência de *Airborne Wind Energy* de 2017 [60], com o objetivo de obter *feedback* na perspectiva de outros desenvolvedores da tecnologia. Também é objetivo da participação na conferência a obtenção de informações sobre os sistemas de software utilizados por empresas e grupos de pesquisa ao redor do mundo para o desenvolvimento de suas aplicações de *AWE*, tendo em vista a abertura de canais de comunicação para o realização de trabalhos futuros nesta área.

Bibliografia

[1] IEA, *World Energy Outlook*. IEA, 2016.

[2] EIA, *International Energy Outlook*. EIA, 2016.

[3] ExxonMobil, *Outlook for energy*. ExxonMobil, 2016.

[4] M. Canale, L. Fagiano, and M. Milanese, "High altitude wind energy generation using controlled power kites," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 279–293, 2010.

[5] A. Cherubini, A. Papini, R. Vertechy, and M. Fontana, "Airborne wind energy systems: A review of the technologies," *Renewable and Sustainable Energy Reviews*, vol. 51, pp. 1461–1476, 2015.

[6] "*Enerkite brochure*." `http://enerkite.de`, Acesso em Março, 2017.

[7] "*Makani Power*." `https://x.company/makani/`, Acesso em Março, 2017.

[8] "*Kite Power Systems*." `http://www.kitepowersystems.com/`, Acesso em Março, 2017.

[9] "*Twingtec AWE advantages*." `http://twingtec.ch/advantages/`, Acesso em Julho, 2017.

[10] "Laboratório ufsckite." `https://ufsckite.gitlab.io`, Acesso em Junho, 2017.

[11] M. De Lellis, A. Mendonça, R. Saraiva, A. Trofino, and Á. Lezana, "Electric power generation in wind farms with pumping kites: An economical analysis," *Renewable Energy*, vol. 86, no. C, pp. 163–172, 2016.

[12] L. Fagiano and T. Marks, "Design of a small-scale prototype for research in airborne wind energy," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 1, pp. 166–177, 2015.

[13] L. Fagiano and M. Milanese, "Airborne wind energy: An overview," in *American Control Conference (ACC), 2012*, pp. 3132–3143, IEEE, 2012.

[14] Ampyx, "Ampyx power and lynx partnership." `https://www.ampyxpower.com/2016/04/press-release-ampyx-power-selected-lynx-as-software-platform`, Acesso em Junho, 2017.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[16] J. Valvano, *Introduction to Embedded Systems: Interfacing to the Freescale 9S12*. Cengage Learning, 2009.

[17] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016.

[18] R. van der Vlugt, J. Peschel, and R. Schmehl, "Design and experimental characterization of a pumping kite power system," in *Airborne wind energy*, pp. 403–425, Springer, 2013.

[19] B. Group, "Barr group embedded systems glossary." `https://barrgroup.com/Embedded-Systems/Glossary`, Acesso em Junho, 2017.

[20] S. Heath, *Embedded systems design*. Newnes, 2002.

[21] "The raspberry pi foundation." `https://www.raspberrypi.org/`, Acesso em Junho, 2017.

[22] "The beagleboard.org foundation." `http://beagleboard.org/`, Acesso em Junho, 2017.

[23] "The arduino company." `https://www.arduino.cc/`, Acesso em Junho, 2017.

[24] K. G. Shin and P. Ramanathan, "Real-time computing: A new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, 1994.

[25] R. S. Oliveira, A. S. Carissimi, and S. S. Toscani, *Sistemas Operacionais-Vol. 11: Série Livros Didáticos Informática UFRGS*. Bookman Editora, 2009.

[26] A. S. Tanenbaum, *Modern operating systems*. Pearson Education, 2009.

[27] O. Ritchie and K. Thompson, "The unix time-sharing system," *The Bell System Technical Journal*, vol. 57, no. 6, pp. 1905–1929, 1978.

[28] R. Petersen, *Linux: the complete reference*. McGraw-Hill Professional, 2000.

[29] W. R. STEVENS, "Unix network programming: Interprocess communications with advanced programming in the unix environment," 2001.

[30] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little, *Documenting software architectures: views and beyond*. Pearson Education, 2002.

[31] L. Bass, *Software architecture in practice*. Pearson Education India, 2007.

[32] M. Patterns, *Microsoft® Application Architecture Guide*. Microsoft Press, 2009.

[33] C. Hofmeister, R. Nord, and D. Soni, *Applied software architecture*. Addison-Wesley Professional, 2000.

[34] D. Seth, *A platform based approach for embedded systems software development*. PhD thesis, Massachusetts Institute of Technology, 2006.

[35] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, "Pattern-oriented software architecture volume 1," 1996.

[36] W. Pree, "Meta patterns—a means for capturing the essentials of reusable object-oriented design," *Object-oriented programming*, pp. 150–162, 1994.

[37] "Airborne wind energy conference 2015, book of abstracs." `http://awec2015.com/abstracts.html`, Acesso em Junho, 2017.

[38] "*Altaeros Energies*." `http://www.altaerosenergies.com/`, Acesso em Março, 2017.

[39] M. L. Loyd, "Crosswind kite power," *Journal of Energy*, vol. 4, pp. 106–111, 1980.

[40] M. De Lellis *et al.*, *Airborne wind energy with tethered wings*. PhD thesis, 2016.

[41] "Nanomsg socket library." `http://nanomsg.org/`, Acesso em Junho, 2017.

[42] "Protocol buffers library." `https://developers.google.com/protocol-buffers/`, Acesso em Junho, 2017.

[43] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999.

[44] U. Fechner and R. Schmehl, "Design of a distributed kite power control system," in *Control Applications (CCA), 2012 IEEE International Conference on*, pp. 800–805, IEEE, 2012.

[45] D. M. Ritchie, "The development of the c language," *ACM SIGPLAN Notices*, vol. 28, no. 3, pp. 201–208, 1993.

[46] D. Moore, E. Olson, and A. Huang, “Lightweight communications and marshalling for low-latency interprocess communication,” 2009.

[47] H. Lawrence, “Introduction to data multicasting,” 2008.

[48] “*SQlite documentation*.” `https://www.sqlite.org/docs.html`, Acesso em Junho, 2017.

[49] “*C-periphery source code repository*.” `https://github.com/vsergeev/c-periphery`, Acesso em Junho, 2017.

[50] “*LibWebsockets library*.” `https://libwebsockets.org/`, Acesso em Junho, 2017.

[51] “*GNU Scientific Library for C and C++*.” `https://www.gnu.org/software/gsl/`, Acesso em Junho, 2017.

[52] “*Basic Linear Algebra Subprograms*.” `http://www.netlib.org/blas/`, Acesso em Junho, 2017.

[53] “*GIT reference manual*.” `https://git-scm.com/docs`, Acesso em Junho, 2017.

[54] “*GitLab platform*.” `https://about.gitlab.com/`, Acesso em Junho, 2017.

[55] “*Docker overview*.” `https://www.docker.com/what-docker`, Acesso em Junho, 2017.

[56] “*CMake documentation*.” `https://cmake.org/documentation/`, Acesso em Junho, 2017.

[57] “*Supervisor: A Process Control System*.” `http://supervisord.org/`, Acesso em Junho, 2017.

[58] K. J. Connolly, *Law of Internet security and privacy*. Panel Publishers, 2001.

[59] M. De Lellis, R. Saraiva, and A. Trofino, “Turning angle control of power kites for wind energy.,” in *CDC*, pp. 3493–3498, 2013.

[60] “Airborne wind energy conference 2017.” `http://awec2017.com/`, Acesso em Junho, 2017.