Este exemplar corresponde à redação final da Tese/Dissertação devidamente corrigida e defendida por: Luiz Gustavo Bizarro Mirisola

e aprovada pela Banca Examinadora.

Campinas, 28 de agosto de 2001

COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Desenvolvimento da Estação de Terra do Projeto AURORA

Luiz Gustavo Bizarro Mirisola

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Desenvolvimento da Estação de Terra do Projeto AURORA

Luiz Gustavo Bizarro Mirisola

Julho de 2001

Banca Examinadora:

- Dr. Marcel Bergerman (Orientador)
- Prof. Dr. Luiz Marcos Garcia Gonçalves
  Instituto de Computação - UNICAMP
- Prof. Dr. Mario Fernando Montenegro Campos
  Departamento de Ciência da Computação - UFMG
- Prof. Dr. Jacques Wainer (Suplente)
  Instituto de Computação - UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Mirisola, Luiz Gustavo Bizarro

M677d Desenvolvimento da estação de terra do projeto AURORA / Luiz Gustavo Bizarro Mirisola -- Campinas, [S.P. :s.n.], 2001.

Orientadores : Marcel Bergerman; Ricardo Anido

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Telemetria aeroespacial. 2. Aeronave não tripulada. 3. Processamento eletrônico de dados em tempo real. I. Bergerman, Marcel. II. Anido, Ricardo III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Desenvolvimento da Estação de Terra do Projeto AURORA

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Luiz Gustavo Bizarro Mirisola e aprovada pela Banca Examinadora.

Campinas, 26 de julho de 2001.

Dr. Marcel Bergerman (Orientador)

Prof. Dr. Ricardo Anido (Co-orientador)

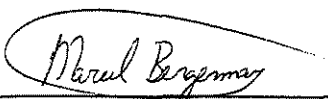
Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 26 de julho de 2001, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Mario Fernando Montenegro Campos
DCC - UFMG

Prof. Dr. Luiz Marcos Garcia Gonçalves
IC - UNICAMP



Prof. Dr. Marcel Bergerman
INTI - CTI

Resumo

Esta dissertação apresenta o desenvolvimento da estação de terra do Projeto AURORA (Autonomous Unmanned Remote Monitoring Robotic Airship). O objetivo do Projeto AURORA é o desenvolvimento de veículos robóticos aéreos para inspeção aérea, evoluindo de veículos puramente tele-operados para veículos telemonitorados. O protótipo da primeira fase, AURORA I, tem como finalidade a demonstração da viabilidade do projeto e a realização de missões de baixa complexidade.

Esta dissertação abrange o projeto e implementação do *hardware* e *software* utilizados pelo operador em terra para monitorar e controlar o vôo do dirigível. São apresentados os diversos componentes da interface com o usuário da estação de terra, o sistema utilizado para a comunicação entre a estação de terra e o sistema embarcado, e o *software* utilizado para se atender aos requisitos de tempo real. Adicionalmente, é apresentada a definição do sistema operacional utilizado na estação de terra e no sistema embarcado do Projeto AURORA.

A estação de terra desenvolvida está sendo utilizada no Projeto AURORA, permitindo a monitoração e o envio de comandos para o dirigível em vôos reais e simulados, e fornecendo suporte para o desenvolvimento dos algoritmos de controle e do sistema embarcado do Projeto AURORA.

Abstract

This dissertation shows the development of the AURORA Project's Ground Station (AURORA stands for Autonomous Unmanned Remote Monitoring Robotic Airship). The AURORA Project aims the developement of aerial robotic vehicles to perform aerial inspection, evolving from purely tele-operated vehicles to tele-monitored ones. The first phase prototype, AURORA I, aims to demostrate the project's viability and to accomplish low complexity missions.

This dissertation presents the design and implementation of the hardware and software used by the ground operator to monitor and control the airship flight. It shows the various components of the user interface in the ground station, the communication system between the ground station and the embedded system, and the software developed to attend the real time requisites. Additionaly, it presents the definition of the operational system to be used in the Project AURORA's ground station and embedded system.

The Project AURORA team is already using the developed ground station in actual and simulated flights to allow flight monitoring and to send commands to the vehicle, providing suport for the development of the control algorithms and the embedded system.

Agradecimentos

Em primeiro lugar agradeço aos meus pais, Oswaldo e Edméia pelo apoio incondicional desde sempre. Se estive em condições de começar (e terminar) este trabalho, devo-o a Deus e a eles.

Agradeço especialmente à orientação do Josué, Marcel e Samuel, e ao Sílvio, cujo código sempre respondeu do outro lado do rádio-modem. E a eles e a toda a equipe do Projeto AURORA, pela agradável convivência e trabalho ao longo dos últimos anos, no ITI, nos vôos, mesmo aqueles às 6 horas da manhã, e nos diversos almoços e jantares. Em ordem alfabética:

Bruno Guedes Faria
Cleyner Soares Pereira
Conrad Tadashi Fujiwara
Ely Carneiro de Paiva
Fábio Augusto G. F. dos Santos
Felipe de Alvarenga Leite
Gabriel Cardoso Martins
Gustavo Sousa Pavani
João Paulo Guimaro Batistela
Josué Jr. Guimarães Ramos
Kiyoshi Asanuma
Luiz Gustavo Corrêia do Nascimento
Marcel Bergerman
Odir Spada Júnior
Omar Esteves Duarte Filho
José Reginaldo Hughes de Carvalho
Ricardo da Rocha Frazzato
Rodrigo Paniago Peixoto
Samuel Siqueira Bueno
Silvio Mano Maeta

Agradeco também à II Cia. de Comunicações Blindada, e aos seus últimos comandantes Maj. Com. Cláudio A. Cunha Dornelles e Maj. Com. Ricardo Henrique Paulino da Cruz, pela cessão de espaço para o hangar do dirigível e para a realização dos vôos.

Finalmente, à FAPESP, que financia parcialmente o Projeto AURORA e concedeu-me a bolsa de mestrado, sob o processo 99/04631-6.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Visão Geral

Esta dissertação apresenta o desenvolvimento da estação de terra do Projeto AURORA (Autonomous Unmanned Remote Monitoring Robotic Airship), abrangendo o projeto e implementação do *hardware* e *software* utilizados pelo operador em terra para monitorar e controlar o vôo do veículo áereo utilizado, um dirigível. Adicionalmente, apresenta a definição do sistema operacional utilizado na estação de terra e no sistema embarcado do Projeto AURORA. Neste trabalho são apresentados os diversos componentes da interface com o usuário da estação de terra, o sistema utilizado para a comunicação entre a estação de terra e o sistema embarcado, e o *software* utilizado para se atender aos requisitos de tempo real.

A estação de terra desenvolvida está sendo utilizada no Projeto AURORA, permitindo a monitoração e o envio de comandos para o dirigível em vôos reais e simulados, e fornecendo suporte para o desenvolvimento dos algoritmos de controle e do sistema embarcado do Projeto AURORA.

1.2 Motivação e Objetivos

Veículos aéreos possuem um enorme potencial não explorado em tarefas de monitoração de tráfego, planejamento urbano, inspeção de grandes estruturas como linhas de transmissão ou oleodutos, retransmissão de sinais de rádio e vídeo, prospecção mineral e arqueológica, policiamento e pesquisa e monitoração ambiental, climatológica e de biodiversidade. No que diz respeito a esta última, subtarefas incluem o sensoriamento e monitoração de florestas e parques nacionais, levantamentos de uso e ocupação do solo, estudos agropecuários, previsão de colheitas, medição da qualidade do ar e de níveis de poluição sobre centros urbanos e industriais e estudos limnológicos. A informação obtida nestes casos

pode ser usada como base de estudos ecológicos e para o estabelecimento de políticas de desenvolvimento sustentado.

Atualmente, muitas das tarefas descritas acima são realizadas a partir de sensoriamento remoto, mais precisamente, de dados obtidos através de sensores e câmeras instalados em balões, satélites ou aviões. Tais veículos, entretanto, possuem inúmeras desvantagens. Balões não são manobráveis e, portanto, não é possível ao seu usuário definir com rigor a área a ser sobrevoada. Imagens de satélites disponíveis para fins civis possuem baixa definição espacial e temporal, além de limitações nas faixas do espectro disponíveis. Levantamentos aéreos, embora permitam ao usuário controlar a área a ser sobrevoada, a resolução dos dados e as faixas do espectro utilizadas, requerem a garantia do conforto e da segurança da equipe de bordo, sendo portanto bastante dispendiosos.

A adoção de veículos robóticos aéreos semi-autônomos não tripulados permite a aquisição de dados de forma que o usuário possa determinar a área a ser monitorada, a resolução espacial e temporal dos dados, e também o sensor ideal para cada tipo de missão, a custos relativamente baixos. Tem-se como consequência uma expansão do uso científico e civil de dados aéreos e a benefícios sociais e econômicos significativos.

O Projeto AURORA foi iniciado pelo Laboratório de Robótica e Visão (LRV) do Instituto Nacional de Tecnologia de Informação (ITI) em 1997 [5], [6], [26]. Nele, o objetivo é o desenvolvimento de veículos robóticos aéreos para inspeção aérea. Como mostrado em [5], para estes tipos de missões, os dirigíveis [25] possuem inúmeras vantagens sobre aviões e helicópteros. No projeto AURORA visa-se o estabelecimento de dirigíveis não-tripulados com significativos graus de autonomia durante todas as fases de suas missões, incluindo a habilidade de planejar e executar sensoriamento e navegação, diagnosticar e recuperar-se de falhas, e adaptativamente replanejar missões baseado na avaliação, em tempo real, de informação sensorial e de restrições ambientais.

O Projeto AURORA consiste em várias fases, envolvendo o desenvolvimento de protótipos sucessivamente com maior capacidade de vôo, capazes de cobrir maiores distâncias, e com graus cada vez maiores de autonomia. Tais protótipos evoluirão de veículos puramente tele-operados para veículos telemonitorados. O protótipo da primeira fase, AURORA I (modelo AS800, fornecido pela empresa inglesa Airspeed Airships, mostrado na Figura 1.1) tem como finalidade a demonstração da viabilidade do projeto e a realização de missões de baixa complexidade.

Os principais componentes do AURORA I são:

* o dirigível;

* o sistema de controle e navegação a bordo com seus sensores internos e atuadores, o qual designaremos por sistema embarcado;

* o sistema de comunicação, que serve de elo de ligação entre o sistema embarcado e a estação móvel de base;

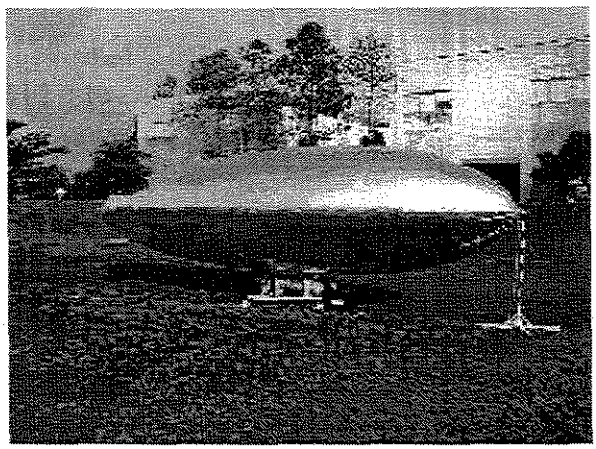


Figura 1.1: O dirigível modelo AS-800.

* os sensores externos, situados a bordo do dirigível, que são necessários para o cumprimento da missão específica um determinado vôo e variam de missão para missão;

* e uma estação móvel de base, que designaremos por estação de terra.

Esta dissertação apresenta a concepção, o desenvolvimento e a validação experimental da estação de terra do protótipo AURORA I. Os requisitos da estação de terra serão definidos na próxima seção, após a revisão bibliográfica pertinente. Adicionalmente, apresenta a definição do sistema operacional utilizado na estação de terra e no sistema embarcado do Projeto AURORA.

1.3 Revisão Bibliográfica

A evolução na área de microeletrônica, incluindo sensores e computadores mais compactos e computacionalmente mais poderosos, abre a possibilidade de execução de vôos autônomos através de UAVs (*Unmanned Aerial Vehicles*) [38], para uma série de aplicações, como aeronaves de reconhecimento militar, vigilância em grandes cidades, mapeamento do solo e análise atmosférica. Nestas aplicações é necessário que o veículo seja autônomo para livrar o ser humano da monitoração e pilotagem de baixo nível, permitindo que ele se dedique ao planejamento estratégico de alto nível.

Dentre os diversos projetos de desenvolvimento de UAVs existentes até o momento, há uma predominância quase absoluta de aviões e helicópteros autônomos em relação a dirigíveis. No entanto, vários destes projetos necessitam também de estações de terra para o monitoramento e controle do vôo e da missão a ser executada, apresentando algumas características e problemas similares. Na sequência citaremos alguns destes projetos cujas referências descrevem aspectos relevantes das respectivas estações de terra:

- Projeto GTAR (*Georgia Tech Aerial Robotics*) [9], desenvolvido no *UAV Research Facility* do *Georgia Institute of Technology*: um helicóptero autônomo que participou da competição internacional de veículos aéreos robóticos, com um *hardware* semelhante ao utilizado no Projeto AURORA;

- Sistema MAGIC [10], desenvolvido pela empresa Herley-Vega Systems: um sistema para operar UAVs militares de reconhecimento da OTAN, incluindo uma estação de terra, sistema de comunicação e interface com o usuário;

- Projeto *Hummingbird* [13], desenvolvido na Stanford University, um helicóptero autônomo capaz de receber da estação de terra uma especificação de uma missão composta de tarefas como, por exemplo, se deslocar até um ponto de passagem, ou permanecer pairando com uma determinada atitude[1] e executar a missão recebida. Também participou das competições internacionais de veículos aéreos robóticos;

- *UAV Research at Sydney University* (Austrália) [37]: um grupo de pesquisa desenvolvendo diversos aviões autônomos, e uma estação de terra com telemetria em tempo real e replanejamento de trajetórias em função de obstáculos conhecidos ou detectados.

Podemos destacar destas referências algumas características importantes de suas estações de terra, como:

- Apresentação da trajetória do veículo em tempo real, em forma de gráfico, bem como algumas variáveis críticas para a segurança do vôo, como altitude, velocidade e modo de operação (vôo automático ou vôo manual, por exemplo), apresentados ao operador da estação de terra na forma de instrumentos similares aos encontrados em veículos tripulados reais (especialmente em [10]) e de gráficos de latitude x longitude (muitas vezes mostrados sobre mapas da área de vôo) e de longitude x altitude;

- Controle da missão: acompanhamento da execução das fases da missão através de uma lista de tarefas a serem cumpridas, e possibilidade de interferir, modificando a lista de tarefas e pontos de passagem da missão, especialmente em [13] e [10];

- Utilização de um visualizador 3D, capaz de ser utilizado tanto em vôos reais como em simulados, mostrando o veículo aéreo voando em um mundo virtual. Em [15] pode-se encontrar um mundo virtual bastante complexo, com boa qualidade de renderização, utilizado no momento para visualização de vôos e futuramente para vôos simulados pilotados;

[1] "Atitude" significa a orientação do veículo em relação ao plano do horizonte e ao norte, ou utilizando os termos da aeronáutica, os ângulos de *roll*, *pitch* e *yaw*.

- Utilização de um rádio-modem conectado às portas seriais para a comunicação entre a estação de terra e o sistema embarcado [15].

Dentre os projetos que tratam especificamente de dirigíveis, citamos:

- Projeto *GPS Formation Flying Blimps* (EUA, *Stanford University*) [11] : Este projeto objetiva demonstrar que um sistema CDGPS (*Carrier Phase Diferrencial GPS*) pode ser usado para controlar o movimento de vários veículos em formação, fornecendo não só a posição dos veículos mas também a atitude. Como o objetivo final é utilizar a tecnologia desenvolvida em satélites, os autores escolheram usar dirigíveis *indoor* pois sua dinâmica é a mais parecida com a de veículos no espaço. Esses dirigíveis *indoor* não têm superfícies aerodinâmicas, sendo controlados apenas por motores elétricos posicionados em volta da gôndola, portanto não é possível operá-los sob vento razoavelmente forte;

- O dirigível SASS-LITE da *Bosch Aerospace* (EUA) [3]: Essa família de dirigíveis foi projetada para executar monitoração e reconhecimento de longa duração em várias aplicações militares, como monitoramento de áreas de fronteira, de rios e como estação repetidora de rádio. Carregam 190 Kg de carga útil, em missões de 12 horas, e são teleoperados;

- O dirigível da *Simon Fraser University* (Canadá) [31]: Vencedor da qualificação da competição de veículos aéreos robóticos em 1998 (apesar de que nenhum competidor foi capaz de cumprir a missão durante a competição), é um dirigível de dez metros de comprimento, com um motor a gás de 5 HP e equipado com GPS Diferencial (DGPS) e bússola, conectados a um *hardware* PC-104 embarcado. Um *link* de vídeo transmite imagens de uma câmera embarcada para a estação de terra, que processa estas imagens procurando detectar objetos de interesse a serem sobrevoados mais de perto e identificados (de acordo com o objetivo da competição, procurar e identificar visualmente tambores de óleo com diferentes rótulos colocados na área de vôo). Uma vez que os objetos são detectados a sua localização é transmitida para o sistema embarcado. Posteriormente o grupo abandonou o desenvolvimento do dirigível e está agora desenvolvendo um avião autônomo;

- Desenvolvimento de um dirigível autônomo no LAAS/CNRS (França) [16]. É o primeiro projeto similar ao projeto AURORA em relação aos objetivos e ao tipo de dirigível, porém ainda está em fase de definição.

Em relação aos requisitos de projeto para uma estação de terra, [35] apresenta de forma relativamente concisa (existem especificações oficiais bastante extensas e detalhadas) os

requisitos dos sistemas de UAVs militares da OTAN. Evidentemente estes sistemas são mais complexos e apresentam requisitos mais rigorosos do que o projeto AURORA, em termos de confiabilidade, *links* de comunicação, tipo, alcance e duração das missões, etc. Dentre os requisitos mostrados no artigo para uma estação de terra, destacaremos alguns considerados mais relevantes no contexto de um projeto de dirigível autônomo:

- Escalabilidade: suportar expansões futuras do sistema sem necessidade de uma completa reestruturação;
- Modularidade: configuração do sistema através do uso de diferentes módulos;
- Suporte a diferentes modelos de UAVs, e a vôos simulados utilizando um modelo matemático do veículo ao invés do veículo real;
- Fácil monitoramento: devem ser continuamente mostrados ao usuário dados que lhe permitam verificar rapidamente o correto funcionamento do sistema bem como a ocorrência de algum problema;
- Ergonomia: controles e mostradores da interface com o usuário devem ser projetados de forma ergonômica;
- Ser facilmente transportável para instalação no local de operação.

O mesmo artigo também apresenta requisitos para as suas estações de terra em relação ao planejamento, controle e monitoramento das missões:

- permitir que o operador gere e processe planos de missões do veículo;
- permitir mudança e cancelamento da missão durante a operação do veículo;
- enviar um plano de vôo para o veículo através do *link* de comunicação veículo-estação de terra;
- verificar a validade da missão e plano de vôo a serem transmitidos ao veículo antes da sua transmissão, verificando restrições de altitude, gasto de combustível, alcance do *link* de transmissão, etc.;
- realizar telemetria de todas as variáveis disponíveis em tempo real, bem como armazenar todos os dados recebidos para posterior avaliação.

Existem também diversos outros requisitos que não se aplicam ao Projeto AURORA, como suporte a múltiplos veículos e múltiplas estações de terra, comunicação protegida contra guerra eletrônica, integração e comunicação do sistema com os sistemas de proteção ao

vôo existentes, controle de *payloads* como câmeras de alta resolução e sistemas de armas, etc.

Em [32] é apresentada uma linguagem de descrição de missão para um helicóptero autônomo. Esta linguagem visa descrever a trajetória a ser seguida pelo veículo, em alto nível - descrevendo *o que* o veículo deve fazer, e não *como* - de forma que descrições da missão possam ser escritas ou lidas por seres humanos. Cada comando desta linguagem descreve uma tarefa a ser executada pelo veículo, como decolar, pousar, e se dirigir até um ponto de passagem. São também utilizados parâmetros como velocidade de movimentação, atitude ao fim da movimentação, etc.

Uma vez que a missão seja mostrada ao operador na forma de pontos marcados sobre um mapa, surge a possibilidade da manipulação direta dos pontos de passagem, permitindo ao usuário mudar as suas coordenadas visualmente sobre o mapa. Na literatura [36] existe uma avaliação de usabilidade de diversos dispositivos utilizados como interface para que o usuário monitore um UAV simulado durante a execução de uma missão: decolagem, replanejamento - mudança de coordenadas dos pontos de passagem - da missão, identificação de um alvo e retorno à base. Os resultados, tanto quantitativos (medição do tempo necessário para completar a tarefa e número de erros) quanto qualitativos (opinião subjetiva do usuário), mostram que o mouse e a *touchscreen* são os dispositivos que permitem executar este tipo de tarefa mais rapidamente e com menos erros, em comparação com o teclado e o *touchpad*.

Apesar das diferenças entre o Projeto AURORA e os projetos apresentados, o exame da literatura permitiu levantar diversos aspectos importantes e ajudou a definir e classificar os requisitos a serem cumpridos pela estação de terra do Projeto AURORA para alcançar os objetivos mostrados na Seção 1.2.

Assim, baseados na revisão bibliográfica e nos requisitos do Projeto AURORA, definimos os principais requisitos da estação de terra do Projeto AURORA (estes requisitos serão detalhados nas Seções 2.1 a 2.6) como:

- Permitir que o operador da estação de terra possa assumir o controle do veículo se ocorrerem problemas inesperados, enviando comandos para pilotar o dirigível manualmente, abortando a missão, ou tomando outras medidas necessárias;

- Permitir o monitoramento do dirigível em vôo através da visualização de dados de telemetria em tempo real, durante o vôo;

- Permitir a definição e envio de uma missão para ser executada pelo dirigível, bem como a monitoração ou a modificação do andamento da missão. Por exemplo, numa missão de inspeção aérea de uma área, seguindo uma trajetória pré-definida, o operador pode desejar acompanhar em tempo real as imagens da inspeção, e, a

partir das observações realizadas, determinar novas trajetórias a serem percorridas durante o curso da missão;

- No caso de existirem dispositivos em terra cujos dados sejam necessários para a operação do sistema embarcado, transmitir as informações relevantes para o sistema embarcado. Por exemplo, dados de uma estação de GPS Diferencial.

- Apresentar facilidades para o desenvolvimento, manutenção e configuração do sistema. Considerar que a mesma estação de terra deve poder ser utilizada como uma interface de operação para vôos simulados, sendo assim uma ferramenta útil para a validação de modelos ou mesmo para treinamento de pilotos;

- Permitir a análise pós-vôo dos dados de telemetria, tanto em relação ao estudo do veículo em si, sua dinâmica e algoritmos de controle, quanto em relação ao cumprimento da missão. Para isto os dados de telemetria, além de serem visualizados em tempo real durante o vôo, também devem ser armazenados;

1.4 Estrutura da Dissertação

Este primeiro Capítulo introduz o leitor no contexto do Projeto AURORA, apresentando as motivações e objetivos do Projeto AURORA e sua estação de terra. A seguir, após a revisão bibliográfica, fornece uma visão geral dos requisitos da estação de terra, que serão expostos mais detalhadamente no Capítulo 2.

O Capítulo 3 apresenta o sistema operacional utilizado, as suas características e limitações, e descreve a sua utilização dentro do contexto do Projeto AURORA. A seguir, o Capítulo 4 descreve o sistema desenvolvido, apresentando cada um dos seus componentes e confrontando o resultado final com os requisitos definidos no Capítulo 2.

O Capítulo 5 descreve experiências de campo realizadas com o sistema descrito no capítulo anterior, ou seja, vôos reais do dirigível realizados pela equipe do Projeto AURORA.

Finalmente, o Capítulo 6 sugere alguns possíveis desenvolvimentos futuros e apresenta as conclusões do trabalho realizado.

Adicionalmente, os apêndices fornecem instruções sobre como instalar todo o *software* necessário em um computador novo, algumas informações julgadas importantes para alguém interessado em desenvolver um projeto semelhante, e finalmente uma lista das publicações do autor em conjunto com o grupo de pesquisadores do Projeto AURORA.

Capítulo 2

Requisitos da Estação de Terra

Esta seção apresenta e classifica os requisitos da estação de terra do Projeto AURORA. Tais requisitos estão classificados em 6 conjuntos, e dentro de cada conjunto são listados em uma ordem decrescente *aproximada* de prioridade em relação às necessidades específicas do Projeto AURORA. Essa ordem pode sofrer alguma alteração de acordo com a aplicação. Se atender um requisito A for condição *sine qua non* para atender outro requisito B, então A será apresentado como mais prioritário do que B nestas listagens (e portanto A é listado antes de B).

2.1 Requisitos relacionados com a segurança do dirigível

A seguir são listados os requisitos necessários para permitir que o operador da estação de terra possa assumir o controle do veículo se ocorrerem problemas inesperados:

- transmitir comandos de um rádio-controle de aeromodelo, que chamaremos de RCU (do inglês *Radio Control Unit*) para o sistema embarcado. Dessa forma um piloto em terra pode comandar manualmente o veículo. Deve também ser possível receber diretamente os sinais do rádio-controle no sistema embarcado através de um receptor de rádio de aeromodelo, sem utilizar a estação de terra, o rádio-modem ou a CPU embarcada. Esta última alternativa deve ser mantida por segurança, como último recurso em caso de pane da CPU embarcada, no entanto tem as desvantagens de ter menor alcance e correr o risco de interferência com aeromodelistas nas proximidades.

- permitir a seleção entre pilotagem manual ou automática: sempre deverá existir um radio-controle na estação de terra que permita a pilotagem manual do dirigível.

O operador dever ser capaz de mudar do controle manual para controle automático (e vice-versa) sempre que desejado, permitindo ou não que a CPU controle o movimento do veículo;

- avisar o operador em caso de falhas de sistemas ou sensores essenciais, antes e durante a missão, usando alarmes visuais ou sonoros: no caso de ser detectada falha de algum sistema ou sensor essencial à execução da missão, o operador deverá ser avisado imediatamente por meio de um sinalizador sonoro ou de cores diferentes na estação de terra. Uma boa abordagem é construir na interface um mostrador do estado de todos os sensores essenciais (GPS, por exemplo) com cores indicando o estado de funcionamento do sensor. Isto é importante tanto durante o vôo, para permitir que a missão seja interrompida imediatamente e um operador em terra assuma o controle do veículo manualmente, quanto antes da decolagem, para que esta seja permitida somente se todos os sistemas estiverem funcionando perfeitamente;

2.2 Requisitos relacionados com a monitoração do vôo do dirigível

Esta seção trata dos requisitos relacionados com monitoração do vôo, que consiste na visualização da trajetória do veículo e das demais leituras dos sensores embarcados. Mesmo que o dirigível não esteja executando nenhuma missão e esteja sendo pilotado manualmente, a sua posição, trajetória e leituras de sensores ainda podem ser monitoradas. Isto também é importante em relação à segurança da operação do veículo. Tais requisitos são:

- representar a trajetória em um mapa da área de vôo: o operador na estação de terra não consegue avaliar visualmente com precisão a localização do dirigível em vôo, mesmo quando o dirigível se encontra visível, e além disso ele pode precisar manter a sua atenção no monitor para acompanhar o desenvolvimento da missão. Dessa forma, é necessário mostrar em tempo real ao operador o movimento do veículo sobre uma imagem do mapa da área de vôo;

- permitir a monitoração das informações essenciais ao acompanhamento do vôo, apresentados de forma contínua em um formato familiar para o usuário. Por exemplo, o Painel de Instrumentos e Comandos (Seção 4.3.1) apresenta variáveis críticas como altitude, velocidade, *roll*, *pitch* e *yaw*, e velocidade vertical na forma de mostradores similares aos instrumentos aeronáuticos encontrados em aviões reais ou em simulações e jogos;

- mostrar os dados de telemetria em tempo real na forma de gráficos: o operador deve poder acompanhar a evolução de determinadas variáveis durante o vôo. Isto pode

ser importante durante o desenvolvimento dos algoritmos de controle, e em relação a variáveis que sejam importantes para a missão específica sendo executada;

- mostrar uma visualização 3D do dirigível em vôo: Pelo mesmo motivo do item anterior. Além disso, é um modo de se visualizar conjuntamente e rapidamente o movimento espacial e a atitude do veículo. Uma representação 3D também será útil para o desenvolvimento e operação de controle por visão para o dirigível, que é um dos temas atuais de pesquisa do LRV.

2.3 Requisitos relacionados à missão a ser executada

Uma das funções mais importantes da estação de terra é permitir a definição e o envio da missão a ser executada pelo dirigível ao sistema embarcado. Uma missão é composta de tarefas, como "ir até um ponto de passagem", e outros tipos de tarefas conforme as capacidades do veículo. Assim, se o veículo permitir decolagem ou aterrisagem automáticas, ou tiver capacidade de pairar *(hovering*, permanecer parado no ar sem se mover), decolar, aterrissar ou pairar podem ser tarefas possíveis em uma missão. Os requisitos relacionados à missão a ser executada são:

- planejar a missão previamente e enviá-la para o sistema embarcado: a missão será composta por uma lista de tarefas a serem executadas. A descrição da missão deverá ser enviada através do rádio-modem para o sistema embarcado;

- monitorar e controlar a execução da missão: o operador deve ser informado sobre que fase da missão o veículo está executando a cada momento - por exemplo, aterrissando ou indo até um determinado ponto de passagem. Além disso o operador deve ser capaz de abortar e modificar a missão durante a sua execução. Isto é necessário para a segurança do vôo em caso de eventos inesperados que impossibilitem a execução da missão ou tornem a execução perigosa para a segurança do veículo ou da área da missão;

- quanto aos pontos de passagem, que são as tarefas mais essenciais da maioria das missões, o operador deverá poder definí-los tanto digitando manualmente as coordenadas quanto clicando sobre uma imagem da área de vôo. A primeira forma permite uma maior precisão se coordenadas precisas puderem ser obtidas na área de vôo com um GPS manual, o que elimina a possibilidade de erros cartográficos. No entanto, caso se disponha de um mapa da área de vôo, a segunda forma permite definir a missão de forma mais simples e visual para o operador;

- os algoritmos de controle embarcados podem necessitar de parâmetros de configuração que variam de acordo com a missão. Por exemplo, pode-se desejar variar os ganhos de cada controlador em cada vôo, testando o sistema com diferentes ganhos durante o desenvolvimento dos algoritmos. A estação de terra deve permitir o envio desses parâmetros para o sistema embarcado durante o vôo.

2.4 Requisito necessário como suporte para o sistema embarcado

Atender o seguinte requisito é necessário para o funcionamento do sistema embarcado:

- transmitir dados de uma estação DGPS para o sistema embarcado: Uma estação DGPS é um dispositivo utilizado para se melhorar a precisão das leituras de um receptor GPS. A estação DGPS permanece em terra, numa posição fixa, e possui um receptor GPS. Como a estação DGPS não se movimenta, ela pode determinar a sua própria posição com maior precisão do que o GPS embarcado. Ela envia informações para o GPS embarcado, que as utiliza para corrigir alguns tipos de erros nas suas leituras, aumentando grandemente a precisão das leituras de posição e velocidade no sistema embarcado. É necessário que a transmissão de dados de DGPS esteja ativa durante todo o tempo de vôo para que se obtenha a melhor precisão do GPS, o que aumenta a segurança da operação do veículo, principalmente considerando que ele deve voar em baixa altitude, e a medida de altitude apresenta menor precisão do que as outras. No entanto, se existirem outros sensores de altitude (barométricos, por exemplo), ou se a altitude de vôo for maior, pode-se optar por voar sem um DGPS, desde que o sinal de SA (descrito no próximo parágrafo) continue desligado. A Figura 2.1 mostra a estação de GPS Diferencial utilizada no Projeto AURORA, colocada em uma caixa plástica com todos os equipamentos necessários (fios, antenas, bateria e rádio-modem) para facilitar o transporte até o local dos vôos.

Até o segundo semestre de 2000, a maior fonte de erro de um receptor GPS era o sinal de SA (*Selective Availability*), um erro que era introduzido propositadamente pelo governo americano, por motivos militares. Apenas usuários autorizados pelo governo americano tinham acesso aos códigos que permitiam obter melhor precisão de um receptor GPS. Com a SA, um receptor GPS sem um GPS Diferencial apresentava erros de posicionamento da ordem de dezenas de metros ou até cerca de 100 metros, o que tornava absolutamente necessário uma estação de GPS diferencial (cujo custo era nessa época pelo menos uma ordem de grandeza maior do que um receptor GPS) para uma aplicação como navegação

de um veículo aéreo. No segundo semestre de 2000, o governo americano decidiu retirar o sinal de SA, permitindo que qualquer receptor GPS alcance precisão de aproximadamente 10 metros. Adicionalmente, mesmo sem o sinal de SA, o GPS Diferencial é capaz de aumentar a precisão do GPS significativamente, já que existem outras fontes de erro menos significativas.

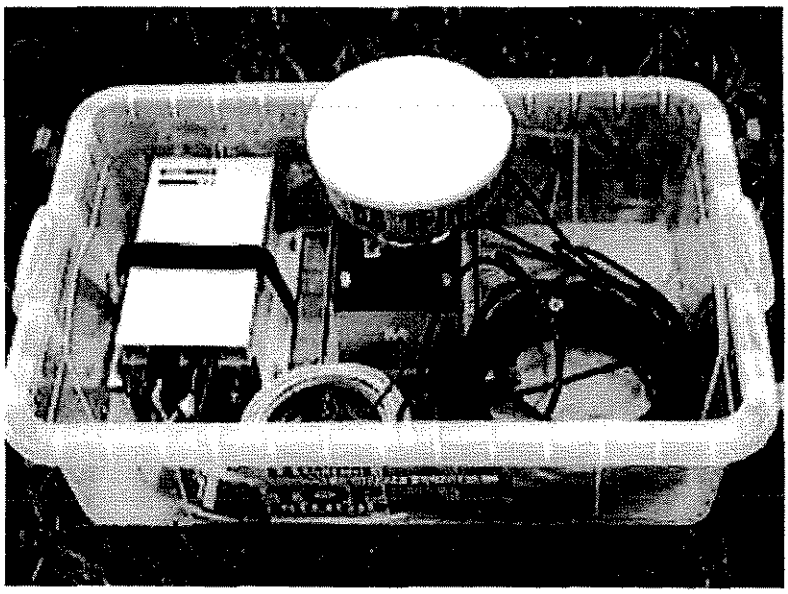


Figura 2.1: Estação de GPS diferencial.

2.5 Requisitos necessários para a manutenção e desenvolvimento da estação de terra

Esta seção apresenta requisitos necessários para facilitar o desenvolvimento e manutenção futuras do sistema:

- permitir fácil reconfiguração em caso de mudança no formato do pacote de dados enviado pelo sistema embarcado: O formato e conteúdo do pacote de dados de telemetria transmitido pelo sistema embarcado para a estação de terra pode mudar, por exemplo, devido a uma mudança no conjunto de sensores embarcados. A estação de terra deve ser facilmente reconfigurável nesta situação;

- utilizar a mesma estação de terra para vôos com o dirigível real e com o modelo simulado. Para a estação de terra, a diferença entre o dirigível real e o sistema

embarcado deve ser apenas o formato do pacote de dados recebido. Quanto a isso, o requisito anterior estabelece que a estação de terra deve ser facilmente reconfigurável;

- manter a modularidade do sistema, fazendo com que cada componente possa ser modificado independentemente dos demais.

2.6 Requisitos necessários para a análise posterior do vôo ou da missão

Esta seção apresenta os requisitos necessários para a análise pós-vôo dos dados armazenados durante os vôos.

- armazenar os dados de telemetria em tempo real: os dados de telemetria recebidos pela estação de terra devem ser integralmente armazenados para permitir análise posterior e reapresentação do vôo realizado. Isto é de capital importância para o desenvolvimento dos algoritmos de controle e para análise da missão executada. Outra alternativa seria armazenar os dados no próprio sistema embarcado para posterior *download* na estação de terra, mas dessa forma não seria possível o monitoramento em tempo real do vôo. Além disso, como o sistema embarcado normalmente não possui grande quantidade de memória RAM, e considerando que podem haver restrições quanto ao armazenamento de dados em memória não-volátil no sistema embarcado - por exemplo, se for utilizado um *flashdisk* (Seção 3.4)- o que poderia inviabilizar o armazenamento dos dados em vôos de duração maior;

- gerar arquivo com todos os dados de telemetria para análise: deve-se gerar um arquivo contendo todos os dados de telemetria em um formato (ASCII, por exemplo) que possa ser aberto em programas de análise estatística ou matemática, para permitir uma análise mais detalhada do vôo realizado. No caso de uma aplicação para o Projeto AURORA que envolva sobrevoar uma área de interesse realizando coleta de dados de sensores, este arquivo seria parte essencial do resultado gerado pelo vôo;

- realizar reapresentação do vôo: deve-se poder utilizar os dados armazenados durante o vôo para reapresentar do vôo realizado com a mesma interface utilizada durante o vôo real, incluindo a telemetria das variáveis, a visualização da trajetória sobre o mapa e a representação 3D do vôo.

2.7 Requisitos do Sistema Operacional

Esta seção apresenta os requisitos do sistema operacional a ser utilizado tanto no sistema embarcado quanto na estação de terra, considerando não apenas características do sistema operacional em si como também a especificação do *hardware* e *software* que ele deve suportar. Portanto esta seção não trata de requisitos da estação de terra do Projeto AURORA, mas é aqui colocada por clareza de apresentação.

2.7.1 Requisitos de suporte ao *hardware* no sistema embarcado

O hardware do sistema embarcado deve ser suportado pelo sistema operacional, o que gera os seguintes requisitos:

- não utilizar disco rígido, devido ao movimento inerente ao vôo do dirigível e à vibração causada dentro da gôndola pelos motores de combustão. Portanto deve-se usar outro meio de armazenamento não vulnerável à vibração, como discos de estado sólido (*flash disks*);
- executar sobre placas no padrão PC-104, com processador Intel x86, o *hardware* escolhido para o sistema embarcado [21] devido ao seu pequeno tamanho, consumo reduzido, e possibilidade de expansão através da ligação de placas adicionais ao barramento;

A Figura 2.2 apresenta uma foto do sistema embarcado, mostrando a CPU, outras placas PC104 e o *flash disk*.

2.7.2 Requisitos necessários para a operação do sistema embarcado

O sistema operacional deve atender aos seguintes requisitos para permitir a operação do sistema embarcado, pricipalmente durante os vôos:

- executar *software* capaz de realizar a leitura dos sensores, controlar o veículo, transmitir os dados de telemetria e receber comandos da estação de terra em tempo real;
- ser robusto, apresentando baixo índice de falhas durante os vôos;
- reiniciar rapidamente, e carregar automaticamente toda a aplicação de controle do dirigível;
- no laboratório, além de funcionar com teclado e monitor, permitir acesso remoto via rede, permitindo assim a operação do sistema e a instalação do *software*.

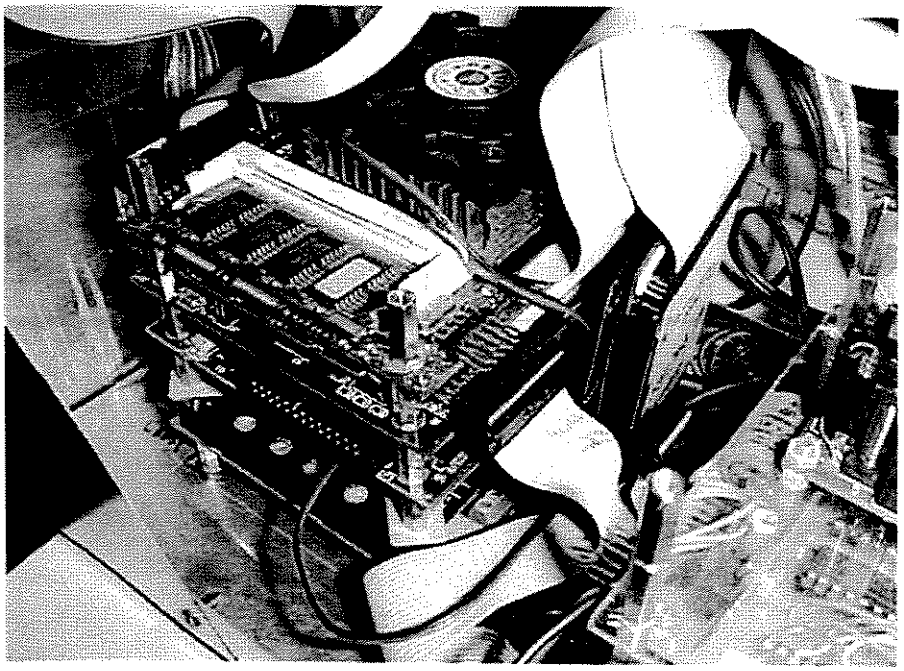


Figura 2.2: A CPU, outras placas PC-104 e *flash disk*, no sistema embarcado.

2.7.3 Requisitos necessários para a operação da estação de terra

O sistema operacional deve atender aos seguintes requisitos para permitir a operação da estação de terra durante os vôos:

- Comunicar-se com o sistema embarcado em tempo real, recebendo dados de telemetria e enviando comandos. O dirigível em vôo apresenta uma dinâmica relativamente rápida (devido ao seu tamanho e à potência da motorização) e um volume potencialmente grande de dados a serem transmitidos a cada passo. Necessita-se portanto garantir que não ocorrerão atrasos na recepção de estados e dados de telemetria e no envio de comandos para o sistema embarcado, mesmo quando a estação de terra estiver executando aplicações computacionalmente custosas, como visualização de gráficos ou ambientes 3D.

Capítulo 3

Sistema Operacional RT-Linux

3.1 Introdução

Nesta seção apresentaremos o sistema operacional de tempo real utilizado na estação de terra e no sistema embarcado do Projeto AURORA, bem como o desenvolvimento realizado utilizando-se esse sistema operacional. Apesar da construção do sistema embarcado estar fora do escopo desta dissertação, a especificação do sistema operacional utilizado foi realizada no âmbito desta dissertação de mestrado.

3.2 Descrição do RT-Linux

3.2.1 Definição de Termos

O *kernel* do Linux, apesar de monolítico, pode ser extendido e modificado dinamicamente com a inserção e remoção de módulos. Estes módulos são arquivos objeto, cujo código é inserido no espaço de endereçamento do *kernel*, portanto tendo acesso às funções da tabela de símbolos do *kernel* (listada em `/proc/ksyms`), e executando com acesso total ao *hardware*. No código de um módulo não há uma função `main()`, mas o módulo deve definir uma função `init_module()` que será chamada quando o módulo for inserido no *kernel*, e outra função `clean_module()`, que será chamada ao se remover o módulo do *kernel*. Durante a inserção do módulo no *kernel*, realizada utilizando-se o comando `insmod`, podem-se passar parâmetros na linha de comando do `insmod`, cujos valores são atribuídos a variáveis globais no código do módulo. Dessa forma um *script* que insira um módulo no kernel pode passar parâmetros para o módulo durante a sua inicialização. Para maior clareza, na sequência os módulos do *kernel* do Linux serão designados por LKM, de *Linux Kernel Module*, evitando-se utilizar simplesmente a palavra módulo.

A expressão "compilar o *kernel*", neste contexto, significa o processo de se executar

os *scripts* do sistema Linux que, a partir do código fonte do *kernel* Linux, geram um arquivo denominado *imagem do kernel*, que deve ser carregado em memória durante a inicialização do sistema por um programa carregador (o programa *lilo* é comumente usado em um sistema Linux). Maiores informações podem ser encontradas nos diversos arquivos de documentação do *kernel* e do *lilo* que não encontrados na maioria das distribuições de Linux.

3.2.2 Breve histórico e descrição geral

O *Real-Time Linux* (RT-Linux) foi originalmente desenvolvido como uma tese de mestrado [1] no *New Mexico Institute of Mining and Technology*. Neste documento será utilizada a denominação RT-Linux, porém também podem ser encontradas na literatura as denominações NMT-RTL (NMT é a sigla da universidade do Novo México) ou simplesmente RTL. O sistema é livremente distribuído sob os termos da *Gnu Public License*, e existem hoje diversos projetos em andamento utilizando o RT-Linux. Atualmente, o esforço de desenvolvimento de novas versões está concentrado principalmente na empresa FSMLabs, fundada pelos desenvolvedores originais, embora contribuições significativas tenham surgido da comunidade de usuários e desenvolvedores, que mantêm uma *mailing list* ativa. O conceito principal do RT-Linux (Figura 3.1) é tratar o *kernel* do Linux como uma tarefa de menor prioridade executando sobre um escalonador de tempo real, juntamente com outras tarefas de tempo real definidas pelo usuário. O *kernel* do Linux (juntamente com todos os programas de usuário) só recebe um *time slot* do escalonador de tempo real quando não existe nenhuma tarefa de tempo real pronta para executar. Dessa forma, o programador tem acesso a um sistema operacional de tempo real, onde as tarefas de tempo real definidas por ele podem ser executadas na velocidade máxima permitida pelo *hardware*, e a todas as funcionalidades de um sistema operacional de propósito geral.

Para isto o RT-Linux insere uma camada de emulação de interrupções sobre o *hardware* de controle de interrupções, de forma que o Linux não pode realmente tratar ou desabilitar as interrupções. Ao invés disto, a camada de emulação, ao receber uma interrupção do *hardware*, verifica primeiro se não existe nenhum *handler* em uma tarefa de tempo real. Se existir, este *handler* será invocado, caso contrário, ou se o *handler* indicar que a interrupção deve ser compartilhada com o Linux, a interrupção será emulada para o Linux posteriormente.

O RT-Linux consiste em um *patch* que deve ser aplicado às fontes do *kernel* do Linux, e em alguns LKM's que devem ser inseridos no *kernel* antes dos LKM's da aplicação. Outros LKM's disponíveis provêem funcionalidades opcionais, por exemplo capacidade de depuração de código em LKM's e acesso às portas seriais. Note que o *kernel* deve ter sido compilado com o *patch* RT-Linux aplicado para permitir a inserção de quaisquer destes LKM's.

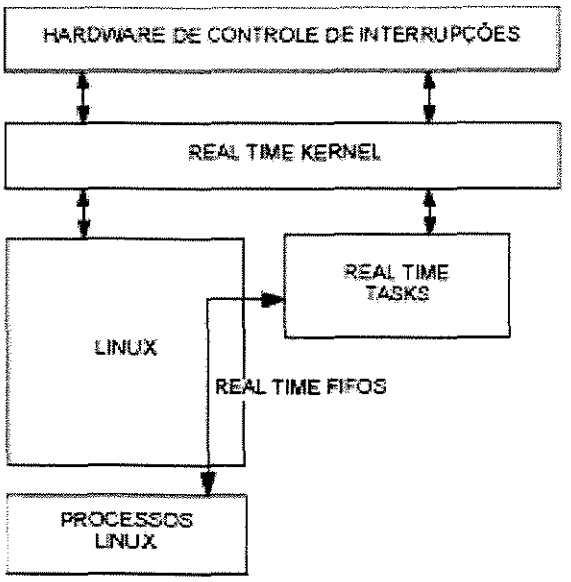


Figura 3.1: Conceito e operação do RT-Linux.

Para se definir uma tarefa de tempo real, deve-se escrever um LKM utilizando a API (*Application Programming Interface*) do RT-Linux, e inseri-lo no *kernel*. Esta API, que segue o padrão POSIX 1001.13 "Minimal RT Application Environment Profile" [12] (com algumas simplificações), permite a definição de tarefas a serem executadas periodicamente ou em um tempo determinado (*one-shot mode*), e a utilização de rt-fifos e memória compartilhada para comunicação com processos do Linux. Rt-fifos são filas, gerenciadas pelo RT-Linux, sendo que um LKM pode acessá-las através de uma API específica, e os programas de usuário podem acessá-las por meio de arquivos de dispositivos no diretório `/dev`, utilizando as funções `read()` e `write()` usuais da biblioteca C padrão.

As tarefas de tempo real, por serem implementadas como LKMs, são executadas no espaço de endereçamento do *kernel*, tendo acesso às funções da tabela de símbolos do *kernel* (listada em `/proc/ksyms`), e não estão submetidas, como um processo de usuário estaria, às proteções de acesso ao disco e à memória providas pelo sistema operacional. Dessa forma, erros de programação em uma tarefa de tempo real podem facilmente levar ao travamento do sistema, implicando em reinicialização. A funcionalidade de depuração introduzida na última versão (3.0) do RT-Linux auxilia o desenvolvimento, inserindo uma camada entre o *hardware* e as tarefas de tempo real do usuário que captura exceções lançadas pelas tarefas de tempo real. Isto permite (na maior parte dos casos) que uma exceção em uma tarefa de tempo real não leve ao travamento de todo o sistema, além de permitir utilizar o depurador `gdb` para depurar o código de um LKM.

As funções das bibliotecas padrão da linguagem C não podem ser utilizadas no *kernel* (e portanto nas tarefas de tempo real), embora, se necessário, algumas funções possam ser ligadas estaticamente em um LKM. Outras funções, por exemplo funções matemáticas como `sin()` e `cos()`, ou funções de tratamento de *strings*, podem ser reimplementadas de forma a serem executadas no *kernel*. Finalmente, há funções que se forem chamadas em um LKM, causam a paralisação do computador, sendo necessário reinicializá-lo. Por exemplo, não podem ser utilizadas funções como `malloc()`, que realizam alocação de memória para processos, e várias outras funções da biblioteca C que internamente chamam `malloc()` ou semelhantes. Em alguns casos pode-se modificar o código das funções existentes nas bibliotecas C padrão, eliminando-se chamadas de funções inseguras (muitas vezes apenas chamadas a funções que imprimem mensagens de erro), em outros é mais rápido reescrever toda a função. Nas páginas sobre RT- Linux na Internet é possível encontrar-se algumas funções da biblioteca C já portadas para serem usadas em LKM's. Bibliotecas da linguagem C++ também não podem ser utilizadas diretamente, e como estas bibliotecas são utilizadas na implementação de várias características da linguagem, existem várias limitações para a utilização da linguagem C++ em tarefas de tempo real, ou em qualquer LKM, que serão descritas na Seção 3.3.

O programador deve implementar como tarefas de tempo real apenas as partes da aplicação que realmente possuírem requisitos de tempo real a serem satisfeitos. Partes não críticas - em termos de requisitos de tempo real - da aplicação, como interface com usuário e serviços de rede, devem ser implementadas como processos de usuário Linux, usando todos os recursos e bibliotecas do sistema operacional sem restrições. Essa separação da aplicação em duas partes é uma das motivações para o desenvolvimento do RT-Linux. Os desenvolvedores acreditam que isto leva a uma melhor modularização, e além disto possibilita que os usuários do RT-Linux possam utilizar qualquer *software* desenvolvido para o Linux na parte da aplicação implementada como processos de usuário. Cabe observar que as tarefas de tempo real têm sempre maior prioridade: caso todo o tempo de processamento da CPU seja ocupado pelas tarefas de tempo real, o Linux e os seus processos de usuário não irão receber *time-slots* para execução, e os processos permanecerão congelados.

O principal desenvolvedor do RT-Linux, Victor Yoidaken, possui os direitos sobre uma patente americana sobre este conceito fundamental, o que permite-lhe impor uma restrição sobre o uso do RT-Linux: permite-se usar o *software* original comercialmente sem restrições, mas não se permite modificar o RT-Linux, ou implementar uma variante utilizando o mesmo conceito, e utilizar comercialmente a versão modificada. Para isso deve-se pagar direitos autorais para a empresa FSMLabs. Para uso acadêmico não há restrições, e além disso, qualquer pessoa pode livremente modificar o código fonte do RT-Linux e utilizá-lo desde que as modificações sejam disponibilizadas publicamente.

3.2.3 O sistema RTAI

Outra alternativa para sistemas Linux de tempo real é o RTAI - *Real-Time Application Interface* - [4],[24] desenvolvido no *Dipartimento de Ingegneria Aerospaziale, Politecnico de Milano* (DIAPM), em Milão, Itália. Os pesquisadores deste departamento desenvolvem aplicações e sistemas de tempo real voltadas para controle em aeronáutica desde o final da década de 80, inicialmente utilizando o sistema MS-DOS e programas residentes na memória. Para implementar esse sistema de tempo real, ponteiros para todos os dados e chamadas de sistema relevantes (para tratamento de interrupções, por exemplo) foram agregadas em uma estrutura, e instalar o sistema de tempo real consiste em fazer o *kernel* chamar os ponteiros da estrutura ao invés das funções originais, e fazer esses ponteiros apontarem para funções substitutas que implementam as funcionalidades de tempo real necessárias. Isso é chamado por eles de camada de abstração de *hardware*, ou RTHAL (*Real Time Hardware Abstraction Layer*), embora alguns pesquisadores questionem se a utilização de uma estrutura contendo ponteiros de funções mereça um nome especial.

Depois de algumas tentativas frustradas de portar e utilizar seu sistema de tempo real sob outros sistemas operacionais, surgiu o RT-Linux, e os pesquisadores de Milão modificaram o *patch* do RT-Linux para implementar o conceito RTHAL e executar as suas aplicações já existentes no seu sistema baseado em DOS quase sem precisar modificá-las. Após recusa por parte dos desenvolvedores do RT-Linux em aceitar estas modificações no *patch*, o novo sistema foi chamado de RTAI, sendo disponibilizado ao público a partir de abril de 1999. Ou seja, surgiu uma variante a partir do RT-Linux original, e as duas linhas de desenvolvimento, agora independentes, têm divergido com o tempo. Ambos os sistemas possuem as mesmas funcionalidades básicas: criar e executar tarefas de tempo real definidas em LKMs, e permitir comunicação com processos de usuário. Ambos implementam uma camada de emulação de *hardware* que trata as interrupções e escalona o *kernel* do Linux ao lado das tarefas de tempo real. Porém, atualmente as APIs são diferentes (o RT-Linux procura seguir o padrão POSIX, e o RTAI não tem este compromisso), bem como a implementação de diversas funcionalidades (as rt-fifos e a depuração de LKMs, por exemplo). O RT-Linux está disponível em diversas arquiteturas (x86, Alfa, PPC), enquanto o RTAI é desenvolvido apenas para a arquitetura x86. Há também algumas funcionalidades desenvolvidas independentemente pelos desenvolvedores do RTAI que não estão presentes no RT-Linux, como chamadas de sistema para IPC no estilo do sistema QNX e algumas facilidades para alocação de memória.

O RTAI é desenvolvido em um departamento de engenharia aeroespacial, para atender as necessidades surgidas na pesquisa em aeronáutica. Conforme as próprias palavras dos desenvolvedores, nos futuros desenvolvimentos do RTAI não há garantias de padronização, de compatibilidade ou mesmo de que o desenvolvimento seja continuado. Já o RT-Linux hoje é o produto principal de uma empresa, a FSMLabs, que se compromete publica-

mente a manter o desenvolvimento em código aberto do RT-Linux (embora submetido às restrições da patente) e além disso suportar várias arquiteturas, manter a modularidade (várias funcionalidades estão implementadas em módulos separados que não precisam ser carregados se não forem utilizados) e a padronização das APIs. Por outro lado a situação pode mudar se outras entidades ou empresas começarem a suportar o desenvolvimento do RTAI. Desde o segundo semestre de 2000, nota-se que o número de desenvolvedores e usuários do RTAI fora do *Politecnico de Milano* vêm aumentando bastante. Além disso o próprio DIAPM desenvolve e utiliza sistemas de tempo real há mais de uma década, e nada parece indicar que num futuro previsível a dedicação do departamento ao desenvolvimento do RTAI irá diminuir.

O RTAI é distribuído sob a licença LGPL, e por ser desenvolvido na Europa, está livre de restrições em relação à patente americana de Victor Yoidaken. Portanto é um sistema completamente *open source*, livre para ser modificado e utilizado sem restrições, pelo menos em relação aos desenvolvedores italianos.

No Projeto AURORA utiliza-se o RT-Linux, que já vinha sendo utilizado antes do aparecimento do RTAI. No estado presente das aplicações do Projeto AURORA no sistema embarcado e na estação de terra, o RT-Linux satisfaz plenamente as necessidades do projeto. Porém, qualquer um dos dois sistemas pode ser utilizado, e caso futuramente deseje-se utilizar o RTAI, basta reescrever o código que utiliza a API do RT-Linux utilizando a API do RTAI. Os LKMs da estação de terra e do sistema embarcado já foram reescritos uma vez, quando foram atualizados da versão 1.0 do RT-Linux, que possui uma API diferente, para a versão 2.0, e esse tipo de conversão pode ser realizado sem dificuldade. O fato de que os LKMs são escritos em C++ facilita ainda mais uma conversão deste tipo, ao concentrar as chamadas à API do RT-Linux em poucos arquivos, notadamente no arquivo principal de cada LKM que cria e dispara a tarefa de tempo real e na classe `rtf_channel`, que encapsula as chamadas para as rt_fifos. (o LKM da estação de terra é descrito na seçaõ 4.2).

3.3 Utilizando C++ em LKM's

Os LKMs da estação de terra e do sistema embarcado foram inicialmente escritos em C. Posteriormente, percebendo-se que o código deveria aumentar (o sistema embarcado deveria tratar de mais sensores, e a estação de terra deveria tratar da transmissão de pacotes relacionados à missão), e que era viável se utilizar C++, foi decidido pela equipe do Projeto AURORA portar o código já existente para C++ a fim de facilitar o desenvolvimento futuro, aproveitando as vantagens da orientação a objeto em termos de modularização e reaproveitamento de código. Primeiramente o LKM foi testado em simulação, e posteriormente passou-se a utilizá-lo com o sistema embarcado real. A Seção 4.2 apresenta

uma descrição mais detalhada do LKM desenvolvido. Estas são algumas das restrições encontradas durante a conversão dos LKMs de C para C++:

- Operadores `new` e `delete`: Não se pode utilizar estes operadores da forma usual, pois eles utilizam internamente funções de alocação de memória da biblioteca C que não estão disponíveis no *kernel*. Há duas opções para se lidar com esta restrição: A primeira é reescrever estes operadores utilizando as funções de alocação de memória do *kernel* (`kmalloc()` e `kfree()`). Implementações podem sem encontradas na Internet. Neste caso deve-se levar em conta que alocação de memória não é completamente previsível, portanto, aplicações com requisitos de tempo real muito rigorosos podem ser inviabilizadas. A segunda opção é não utilizar alocação dinâmica de objetos, alocando cada objeto estaticamente. Na estação de terra e no sistema embarcado escolhemos a segunda opção, por não apresentar grandes inconvenientes no caso do Projeto AURORA, e por ser mais simples e confiável.

- Ausência de RTTI (*Run Time Type Identification*) e polimorfismo: Em um sistema Linux utilizando o compilador `gcc`, um arquivo objeto gerado pelo `gcc` a partir de um código fonte em C++ que utiliza polimorfismo necessita ser ligado com a biblioteca C++ do sistema, para resolver os símbolos referentes a RTTI. Como a biblioteca não está disponível no *kernel*, não é possível se utilizar polimorfismo em um LKM, embora herança possa ser utilizada sem problemas. Deve-se inclusive usar uma opção de compilação (`-no-rtti`) que indica ao `gcc` que não será utilizado RTTI no programa, caso contrário o LKM não poderá ser inserido no *kernel*.

- Ausência de exceções e funções *pure virtual*: pelo mesmo motivo do item anterior, não se pode utilizar exceções ou funções *pure virtual*. Classes abstratas devem implementar métodos abstratos como uma função vazia (e.g. `void foo(){ }`, e não `void foo()=0`);

- Arquivos de cabeçalho (*header*) do *kernel* contém palavras reservadas da linguagem C++. Por exemplo, uma variável chamada `new` não apresenta problema, pois o *kernel* é escrito em C, não em C++. Isso implica que o código C++ deve ser compilado em um arquivo objeto separado e ligado posteriormente ao arquivo principal do LKM, que deve ser escrito em C. O arquivo principal contém os `#include` necessários para se escrever um LKM e realiza chamadas aos métodos dos objetos através de funções C exportadas no arquivo objeto que contem as classes C++. Este problema poderia ser resolvido rapidamente pelos desenvolvedores do *kernel* (bastaria trocar o nome de algumas variáveis), mas alguns deles (conforme mensagens da lista de discussão *kernel traffic*) pretendem realmente impedir o desenvolvimento de partes do *kernel* em C++, devido a todas as dificuldades descritas acima, à possível queda

de performance, além do trabalho adicional de suportar mais uma linguagem. No entanto, tarefas de tempo real não fazem parte do *kernel* (apesar de serem inseridas no *kernel*), no sentido que elas fazem parte da aplicação do usuário, e não do núcleo do sistema operacional.

Outras referências sobre este assunto são [4], e os arquivos das listas de discussão sobre o RT-Linux ou o *kernel* do Linux, acessíveis na Internet.

Apesar destas restrições, é viável utilizar a linguagem C++, como demonstrado na estação de terra e no sistema embarcado [20] do Projeto AURORA (o LKM do sistema embarcado também foi desenvolvido inicialmente em C e depois em C++). Tanto o RT-Linux quando o sistema RTAI incluem em suas distribuições algum suporte para o desenvolvimento de LKM's em C++: a distribuição do RT-Linux contém uma implementação dos operadores `new` e `delete`, enquanto no RTAI estes operadores já fazem parte da própria API do sistema, bem como uma implementação mais segura de alocação dinâmica de memória para as tarefas de tempo real.

Antes de se utilizar o RT-Linux na estação de terra, foram feitos alguns testes utilizando-se o Linux padrão e o *driver* de porta serial do sistema operacional. O *driver* de porta serial do Linux não pode ser utilizado pelo RT-Linux, mas a distribuição do RT-Linux inclui o *driver rt_com*, um LKM específico para permitir que os LKM's da aplicação do usuário acessem as portas seriais. Verificou-se que o sistema não consegue garantir a entrega de todos os pacotes transmitidos em caso de elevada utilização da CPU pois o processo que lê a porta serial não consegue ser sempre escalonado em todos os momentos necessários. Em [23] é apresentado um resultado semelhante utilizando a porta paralela. Embora os requisitos de tempo real do sistema embarcado e da estação de terra não sejam muito rigorosos (os pacotes são transmitidos à taxa de 10 Hz) o RT-Linux nos garante que o processo (a tarefa de tempo real, na verdade) que realiza a comunicação vai ser executado consistentemente uma vez a cada 100ms independentemente da utilização da CPU ou do número de processos executando no momento.

3.4 O Mini-RTL

No Projeto AURORA são necessárias soluções para a configuração e atualização do sistema operacional do sistema embarcado. Inicialmente foi utilizado no sistema embarcado o RT-Linux aplicado manualmente sobre o *Linux Router Project* (LRP) [18], uma distribuição de Linux mínimo ocupando um disquete de 1.44MB otimizada para a utilização como um roteador.

A distribuição do LRP consiste em um disquete contendo:

- Os arquivos necessários para se inicializar o computador e carregar o *kernel* (`LDLINUX.SYS` e `SYSLINUX.CFG`);
- A imagem do *kernel* que será carregada na memória (arquivo `linux`);
- Arquivos "*package*", que serão descomprimidos e copiados no diretório raiz (/), contendo todos os executáveis do sistema, arquivos de configuração, a aplicação do usuário, etc. Por exemplo, um arquivo `root.tgz`, que contém praticamente toda a árvore de diretórios a partir de /, e um arquivo `etc.tgz` que contem os arquivos de configuração do diretório `/etc`.

Na inicialização do sistema (idealizado para um roteador sem disco rígido), o computador é inicializado a partir do disquete, o *kernel* é carregado, um RAMDISK é criado, os arquivos *"package"* são descomprimidos na RAMDISK (o diretório raiz (/) está na RAMDISK), e então é ativado o processo `init` e segue-se a sequência de *scripts* de inicialização comum às plataformas Unix.

Utilizar o LRP dificulta a atualização do sistema RT-Linux , de forma a se utilizar as funcionalidades das versões mais atuais do RT-Linux no sistema embarcado (API padrão POSIX, sistema mais estável, possibilidade de se utilizar semáforos, etc.), pois, além de ser necessário esperar até que a mesma versão do *kernel* seja suportada pelo LRP e pelo RT-Linux (ambos dependem de *kernel patches* que estão disponíveis apenas para versões específicas do *kernel*), a cada nova versão deve-se aplicar manualmente os *patches* sobre o *kernel* original, recompilar o *kernel*, e reconfigurar todo o sistema novamente.

O MiniRTL [23],[22] é baseado no LRP, oferecendo o mesmo tipo de funcionalidade (sistema em um disquete de 1.44MB, contendo um conjunto similar de arquivos, inicializando e executando em RAMDISK, etc.) porém utilizando versões mais atualizadas do *kernel* já com o RT-Linux instalado. Segundo [22], o hardware mínimo necessário para executar o MiniRTL é um computador 386 com 8 MB de memória, sendo 4MB para RAMDISK e 4MB para RAM, além de um *drive* de disco flexível ou um *flashdisk* de 2MB. Com ele não é mais necessário se compilar outro *kernel* e configurar o sistema novamente a cada atualização, basta utilizar as imagens de disquetes fornecidas pela distribuição do MiniRTL, preencher alguns arquivos de configuração do sistema e instalar os LKM's necessários para o RT-Linux e para a aplicação. Portanto utilizamos o MiniRTL ao invés do LRP para facilitar a atualização e a manutenção do sistema embarcado, permitindo que os desenvolvedores se concentrem mais na sua tarefa principal, desenvolver o *software* de controle do dirigível.

No caso do sistema embarcado do Projeto AURORA o conteúdo do disquete foi copiado para um disco de estado sólido (*flash disk*), que é utilizado em vôos ao invés do disco flexível. Isto é necessário devido à vibração existente no interior do dirigível causada pelos motores de combustão, o que inviabiliza a utilização de discos de acionamento mecânico.

Por questão de economia, o *flashdisk* deve ser utilizado somente para leitura, pois este dispositivo apresenta um número máximo de gravações possível. Assim, é conveniente que o sistema seja executado em RAMDISK, sendo apenas carregado a partir do *flashdisk* para a RAMDISK durante a inicialização do sistema. Sugere-se o uso de *flashdisks* com interface IDE, que podem substituir discos rígidos de forma transparente, evitando problemas de compatibilidade ou suporte em relação ao sistema operacional.

O LRP utiliza alguns *shell scripts* para atualizar os arquivos *"package"* armazenados no disquete (que normalmente são utilizados somente para leitura durante a inicialização do sistema) a partir da RAMDISK. O usuário pode modificar os diretórios e arquivos armazenados na RAMDISK, e utilizar os *shell scripts* para gerar novamente os arquivos *"package"* a partir da RAMDISK e salvar esta nova versão no disquete. Assim, utilizando um servidor NFS, pode-se copiar os arquivos da aplicação do usuário da maquina de desenvolvimento para a RAMDISK do sistema embarcado, e utilizar os *shell scripts* do LRP para atualizar o *software* do sistema embarcado.

No entanto o MiniRTL não suporta estes *scripts*, nem clientes NFS (apesar da opção correspondente estar habilitada no *kernel*, os arquivos necessários não foram incluídos na distribuição), o que implica que a cada vez que se desejar atualizar a aplicação durante o desenvolvimento, deve-se gerar manualmente os arquivos *"package"* comprimidos em um outro computador, copiá-los para um disquete, e ainda copiar todos os arquivos manualmente do disquete para o *flashdisk* no sistema embarcado. Portanto, foi necessário instalar um cliente NFS no sistema embarcado, e reescrever os *scripts* necessários para se gerar no próprio sistema embarcado os arquivos *"package"* do sistema e armazená-los no disquete ou no disco de estado sólido (veja Apêndice A.1).

Capítulo 4

A Estação de Terra do Projeto AURORA

4.1 Introdução

Este capítulo apresenta cada um dos componentes da estação de terra do Projeto AURORA. A Figura 4.1 mostra um diagrama em blocos do sistema como um todo, considerando a estação de terra e o sistema embarcado, com o *software* e *hardware* envolvidos.

Vários dos componentes da estação de terra apresentam interfaces gráficas com o usuário. Ao se projetarem estas interfaces, considerou-se como usuário-alvo o próprio grupo de pesquisadores do Projeto AURORA, portanto tratam-se de usuários qualificados e com conhecimento específico sobre a operação do dirigível.

4.2 O Módulo de *Kernel* (LKM) da Estação de Terra

Na estação de terra há um LKM que define uma tarefa RT-Linux periódica, responsável pela comunicação com o sistema embarcado, recebendo as leituras dos sensores enviadas pelo sistema embarcado, e enviando o sinal de correção de DGPS, os comandos recebidos de uma RCU (*Radio Control Unit*, um controle remoto para aeromodelos), e comandos especiais para mudar de vôo manual para vôo automático. Cada funcionalidade do LKM da Estação de Terra é descrita mais detalhadamente nas próximas seções.

4.2.1 Comunicação com o sistema embarcado

A comunicação entre a estação de terra e o sistema embarcado é realizada através de um par de rádio-modems (modelo DGR-115, da empresa *FreeWave Technologies*, mostrado na Figura 4.2), conectados a portas seriais e capazes de transmitir dados à velocidade

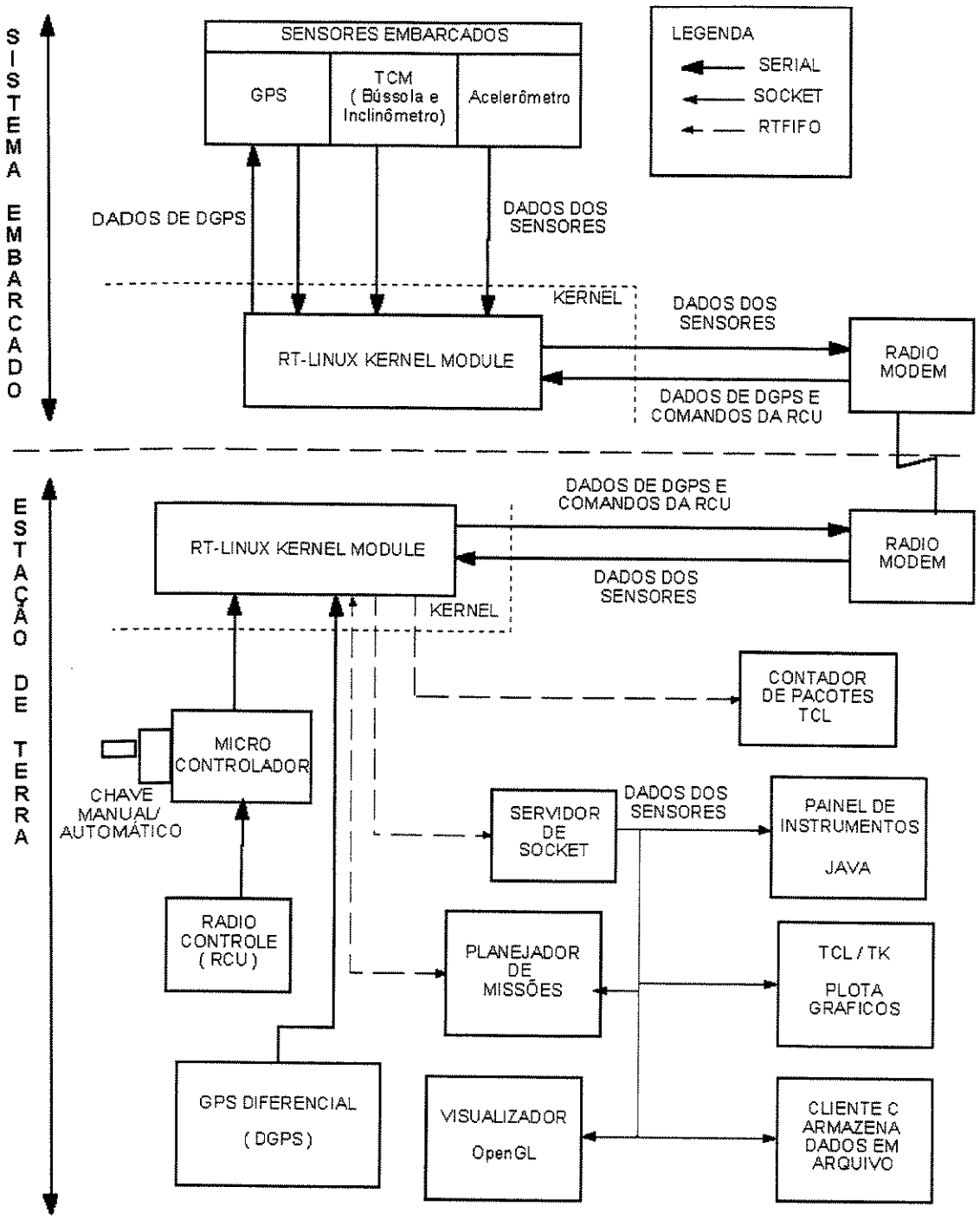


Figura 4.1: Diagrama em blocos do sistema utilizado em vôos reais.

máxima de 115 kbps. Os rádio-modems, depois de configurados, são utilizados como dispositivos RS-232 e podem inclusive ser substituídos transparentemente por um cabo serial sem nenhuma modificação no *software* da estação de terra ou do sistema embarcado.

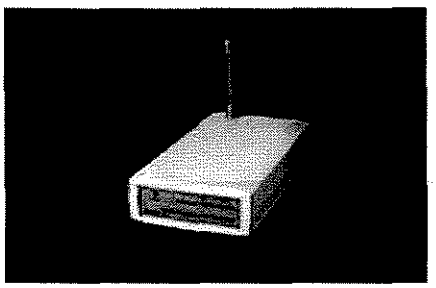


Figura 4.2: Rádio-Modem FreeWawe DGR-115.

Os LKMs da estação de terra e do sistema embarcado, que realizam efetivamente a comunicação, usam o *driver rt_com* (também um LKM) para tratar das interrupções e *buffers* de transmissão das portas seriais. No entanto, o LKM da estação de terra precisa separar os pacotes recebidos e verificar a ocorrência de eventuais erros de transmissão. Para isso, cada pacote transmitido contém um código identificador que permite reconhecer o seu conteúdo (por exemplo, dados de telemetria dos sensores) e um código de CRC (Cyclic Redundancy Code) [34] para deteção de erros de transmissão. Utilizou-se uma rotina de *char-stuffing* [34], que insere os caracteres DLE e STX no início de cada pacote, e os caracteres DLE e ETX no fim de cada pacote. Caso os dados contenham o caracter DLE, o mesmo é duplicado. Assim, o receptor pode detectar o início e término de um pacote procurando uma sequência de tamanho ímpar de caracteres DLE seguidos de um STX ou um ETX, respectivamente.

A tarefa RT-Linux lê os dados de telemetria recebidos pelo rádio-modem e envia-os para uma rt-fifo. Esta rt-fifo pode ser então acessada por processos Linux normais executados em modo usuário, conforme mostrado na Seção 4.3.

A estação de terra deve mostrar ao usuário uma contagem dos pacotes recebidos para que ele verifique se todos os dispositivos estão enviando ou recebendo dados. Isto não pode ser feito pela tarefa RT-Linux, que não executa em um terminal e não pode usar as bibliotecas do sistema como um processo comum. Portanto a tarefa RT-Linux escreve um *byte* em uma rt-fifo para cada pacote processado, associando um código para cada tipo de pacote, e diferenciando pacotes recebidos corretamente de erros de transmissão ou pacotes incompletos. Assim, um *script* Tcl/Tk pode ler a rt-fifo, contar os pacotes processados pelo sistema, e apresentar essa contagem ao usuário numericamente em uma interface gráfica simples. Assim o usuário pode verificar visualmente se a estação de terra está recebendo pacotes do sistema embarcado (simulado ou real), do DGPS, da RCU, e quantos

pacotes de cada tipo são recebidos a cada segundo. Esta é uma interface em um nível mais baixo, utilizada para verificar se todos os dispositivos estão ligados e transmitindo dados adequadamente. Depois de verificado o funcionamento dos dispositivos, o operador da estação de terra pode dirigir a sua atenção para os outros componentes da interface que mostram os estados e a missão do dirigível.

4.2.2 Leitura do DGPS

O receptor DGPS fica localizado em local aberto, onde um maior número de satélites da constelação GPS esteja visível, e transmite dados para a estação de terra, via um par de rádios-modem (ou diretamente através de um cabo serial), transmitindo um pacote de dados por segundo. A tarefa RT-Linux em terra lê os dados do DGPS em uma porta serial, separa os pacotes e retransmite-os para o sistema embarcado utilizando o mesmo rádio-modem utilizado para se enviar os comandos e receber as leituras dos sensores do dirigível.

Alternativamente poder-se-ia embarcar mais um rádio-modem, ligado diretamente ao receptor GPS embarcado, apenas para receber os sinais do DGPS. Porém o peso do rádio-modem (cerca de 400g) é significativo para sistema embarcado, considerando que podem existir sérias limitações de carga útil - dirigíveis com envelope de aproximadamente $30m^3$ (o que implica em cerca de 10m de comprimento), como o dirigível utilizado no Projeto AURORA, suportam cerca de 10 Kg de carga útil ao nível do mar. Além disso a montagem e a manutenção operacional do sistema embarcado será facilitada ao se diminuir o número de dispositivos embarcados.

4.2.3 Leitura dos comandos da RCU

Para fins de aprendizado de pilotagem em ambiente de simulação, e principalmente quando é necessário operar o dirigível manualmente, utiliza-se a RCU. Este controle remoto conecta-se a uma das portas seriais da estação de terra através de um microcontrolador. O conjunto é mostrado na Figura 4.3. O *software* deste microcontrolador bem como o formato dos comandos foi projetado pela equipe do Projeto AURORA [21]. A mesma tarefa RT-Linux lê os comandos da RCU e os envia para a infra-estrutura embarcada pelo rádio-modem.

Opcionalmente, o LKM pode ler os comandos de uma rt-fifo, não utilizando assim a RCU e a porta serial correspondente. Nesse caso é necessário que um processo de usuário gere os comandos no mesmo formato gerado pela RCU e os envie pela rt-fifo. Para isso, utilizamos o Painel de Comandos Java, descrito na Seção 4.3.1.

Verificamos que é muito mais ergonômica a utilização da RCU, pois o piloto do dirigível pode ter um controle mais preciso dos comandos, tem *force feedback* na alavanca de

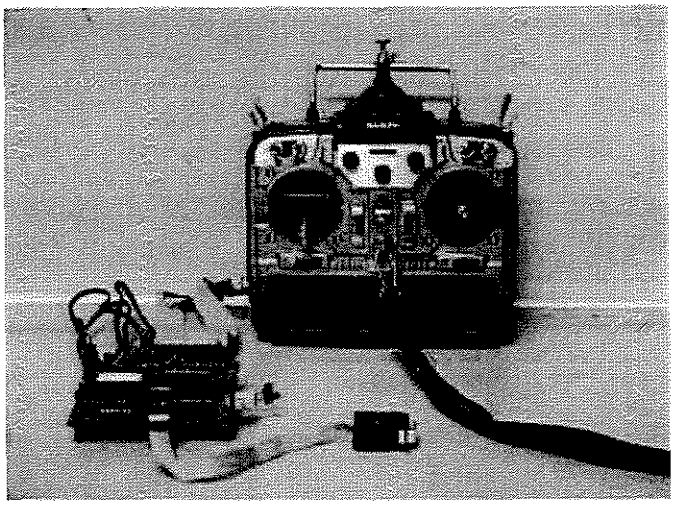


Figura 4.3: RCU (*Radio Control Unit*) e microcontrolador.

comando (uma mola exerce uma resistência ao movimento das alavancas de comando), não precisa mover, arrastar e clicar o mouse (o que causa atraso e erros), não precisa olhar para a tela do computador para mudar os comandos transmitidos, e pode controlar o motor, a direção e a vetorização ao mesmo tempo, segurando a RCU com as duas mãos. Não é recomendada e é considerada perigosa a utilização do Painel de Comandos em vôos reais, por não permitir uma resposta tão precisa e rápida por parte do piloto do dirigível quanto a RCU. No entanto, para vôos simulados (ver Seção 4.6) o Painel de Comandos facilita a operação do sistema, pois não precisamos usar a RCU, o que implicaria manter as suas baterias carregadas (estas baterias demandam horas para serem carregadas) e levar a RCU da área de operação dos vôos de volta para o laboratório. Além disso a RCU pode não estar disponível, como no caso de pesquisadores integrantes do projeto AURORA que usam a estação de terra com um sistema embarcado simulado em outros laboratórios.

4.2.4 Chaveamento entre vôo automático e manual

A estação de terra envia comandos para a pilotagem manual do dirigível simulado ou real. Para a execução de um vôo autônomo, o operador deve ser capaz de enviar um comando para o sistema embarcado de forma que este passe a ignorar os comandos para os atuadores recebidos da estação de terra (que podem deixar de ser transmitidos) e inicie a cxecução dos algoritmos embarcados, que irão gerar os comandos para os atuadores necessários para o vôo autônomo. Isto é especialmente importante enquanto não existirem algoritmos de controle para decolagem ou pousos, portanto o piloto deve conduzir manualmente estas

fases do vôo, e também em caso de problemas durante a execução do vôo autônomo - o operador deve ser capaz de enviar um comando para desligar os algoritmos embarcados e permitir que o piloto reassuma o controle. Isto é válido tanto para vôos simulados quanto para vôos reais.

O microcontrolador que envia os dados da RCU para a estação de terra, e o Painel de Comandos e Instrumentos que pode assumir o papel da RCU, enviam um campo a mais no pacote de comandos que indica se o usuário deseja enviar comandos manualmente para o sistema embarcado ou habilitar os algoritmos de controle embarcados. O valor deste campo corresponde a uma chave no microcrontrolador que pode ser ligada ou desligada pelo operador, ou a uma opção de menu no Painel de Comandos e Instrumentos Java, que é usado apenas em simulação - durante vôos reais o operador não habilita este painel e utiliza-o apenas como Painel de Instrumentos. O LKM então pode enviar ou não comandos para os atuadores do sistema embarcado, e, quando o operador decide mudar o modo de operação, comunicar-se com o sistema embarcado através de pacotes especiais transmitidos pelo rádio-modem: o pacote RCU indica que o operador selecionou o modo de operação manual e o pacote ALG indica que o modo de operação automático foi selecionado. A estação embarcada responde utilizando os pacotes ACK ALG e ACK RCU, indicando que um comando de mudança de estado foi recebido e que a mudança foi efetuada com sucesso. A Figura 4.4 apresenta um diagrama de estados que representa a operação do LKM da estação de terra e a comunicação com o sistema embarcado no que concerne à mudança entre estes modos de operação. Os estados de transição indicam que o operador mudou o modo de operação, mas o sistema embarcado ainda não enviou um pacote de *acknowledge* indicando que a mudança foi efetivada. Enquanto o LKM está nestes estados, ele envia para o sistema embarcado os pacotes indicando a mudança de modo de operação. Para manter a clareza da Figura4.4, eventos que devem ser ignorados não são mostrados (por exemplo, no estado RCU, o recebimento de um ACK RCU deve ser ignorado).

4.2.5 Configuração e linguagem de programação

Através de parâmetros do LKM, o *script* de inicialização de estação de terra pode escolher entre leitura dos comandos da RCU através de uma porta serial ou de uma rt-fifo, utilização ou não do DGPS (o DGPS não é utilizado em simulação) e do rádio-modem. Ao invés de se utilizar o rádio-modem para a comunicação entre a estação de terra e a estação embarcada, pode-se utilizar duas rt-fifos. Isso é utilizado quando, em simulação, deseja-se executar a estação embarcada simulada (descrita na Seção 4.6) na mesma máquina onde é executada a estação de terra. Neste caso os dois LKMs dispensam o rádio-modem, comunicando-se através das duas rt-fifos.

Inicialmente um protótipo deste LKM foi escrito em C. Posteriormente ele foi rees-

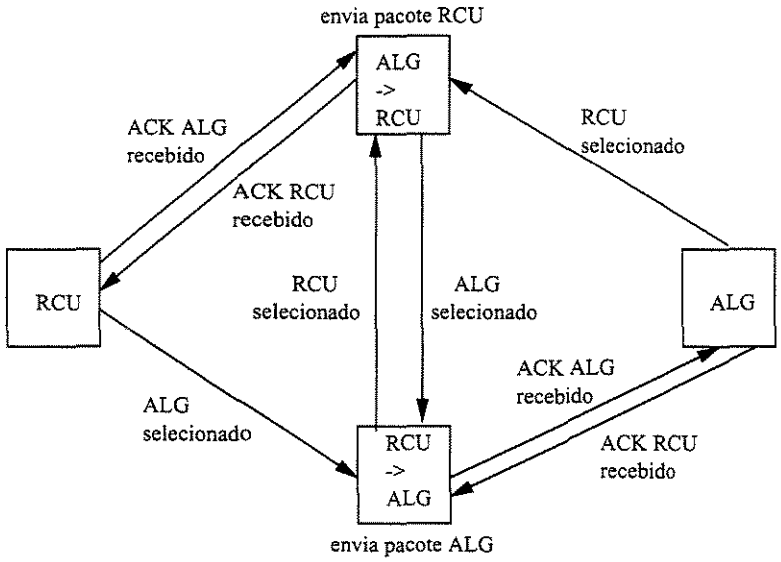


Figura 4.4: Diagrama de Estados do chaveamento Algoritmo / RCU.

crito em C++, permitindo que os próximos passos do desenvolvimento aproveitassem as vantagens da programação orientada a objeto quanto à modularização e reutilização de código. A Seção 3.3 apresenta mais detalhadamente esta conversão.

4.3 Telemetria do dirigível

Como explicado na seção anterior, a tarefa RT-Linux lê os dados dos sensores e os envia para uma rt-fifo. Há vários clientes para estes dados, que podem utilizá-los para diversos fins [30]. Como uma rt-fifo não pode ser lida por mais de um processo simultaneamente, um servidor de *sockets* a lê e envia os dados recebidos para os seus clientes. O servidor, escrito em C, permanece bloqueado até receber uma solicitação de conexão ou desconexão de um cliente, ou até que existam dados a serem lidos na rt-fifo. Assim, para um processo, local ou remoto, receber os dados dos sensores, basta se conectar ao servidor de *sockets* na porta especificada. Além disso a utilização de um servidor permite que cada cliente seja desenvolvido independentemente dos demais, mantendo a modularidade do sistema (Seção 2.5). Este servidor não realiza nenhum tratamento sobre os dados recebidos.

O formato do pacote de dados transmitido está sujeito a mudanças sempre que se modifica o conjunto de sensores embarcados. Além disso, como diversos sensores fornecem dados em diferentes formatos, é necessário que o pacote de dados possa conter dados em diversos formatos diferentes. Um arquivo de configuração na estação de terra indica, para

cada campo do pacote, o seu nome, tipo (por exemplo *string* ASCII, inteiro, *float*, *double*, *string* hexadecimal) e tamanho. Cada cliente deve ler este arquivo de configuração e utilizar as informações deste arquivo para encontrar as informações desejadas dentro do pacote de dados transmitido. Dessa forma, em caso de mudança no conjunto de sensores embarcados ou no conjunto de dados a serem transmitidos, basta atualizar este arquivo de configuração e a estação de terra estará automaticamente configurada.

Um cliente do servidor de *sockets* realiza a gravação dos dados dos sensores na estação de terra. Este é um programa escrito em C que armazena os dados recebidos do *socket* em um arquivo, gravando cada pacote de dados juntamente com a hora em que foi recebido, sendo o tempo contado em centésimos de segundo desde que o processo foi inicializado. Este cliente não realiza nenhum tratamento sobre os dados recebidos, e necessita apenas conhecer o tamanho do pacote de dados.

Na sequência serão apresentados cada um dos outros clientes do servidor de *sockets* que apresentam os dados dos sensores para o usuário da estação de terra.

4.3.1 Painel de Instrumentos e Comandos

O Painel de Instrumentos e Comandos foi desenvolvido a partir do simulador de dirigíveis [29] (Figura 4.5) desenvolvido em 1998 por pesquisadores do Projeto AURORA. O simulador original já contém um painel de instrumentos de navegação com altímetro, indicador de velocidade vertical e à frente, bússola e horizonte artificial, e um outro painel utilizado para se comandar o dirigível simulado, através de um *joystick*, uma alavanca de controle da rotação dos motores e um controle da vetorização dos motores.

Utilizou-se os mesmos painéis de instrumentos e de comandos do simulador, adaptando o *software* para deixar de ser uma *applet* e passar a ser um aplicativo Java; para realizar a leitura do arquivo de configuração da estação de terra e receber os pacotes de dados do servidor de *sockets*; e para enviar comandos no mesmo formato enviado pela RCU para uma rt-fifo, inclusive enviando os comandos de seleção entre vôo automático e manual. O LKM da estação de terra então pode ler estes comandos e enviá-los para a estação embarcada simulada. O envio de comandos pode ser ligado e desligado através de um item do menu, permitindo que se utilize a RCU para enviar os comandos.

O simulador original, que foi desenvolvido em ambiente Windows, contém também uma visualização do vôo do dirigível em um mundo VRML, que não pôde ser aproveitada, pois até o momento não existe um *browser* VRML para Linux capaz de executar a aplicação do simulador (existem *browsers* VRML para Linux, mas eles não implementam ainda a especificação completa da linguagem VRML, e não são capazes de executar a aplicação do simulador de dirigíveis). Dessa forma foi necessário encontrar outra solução para a visualização de vôos em um mundo 3D virtual, conforme mostrado na Seção 4.3.3.

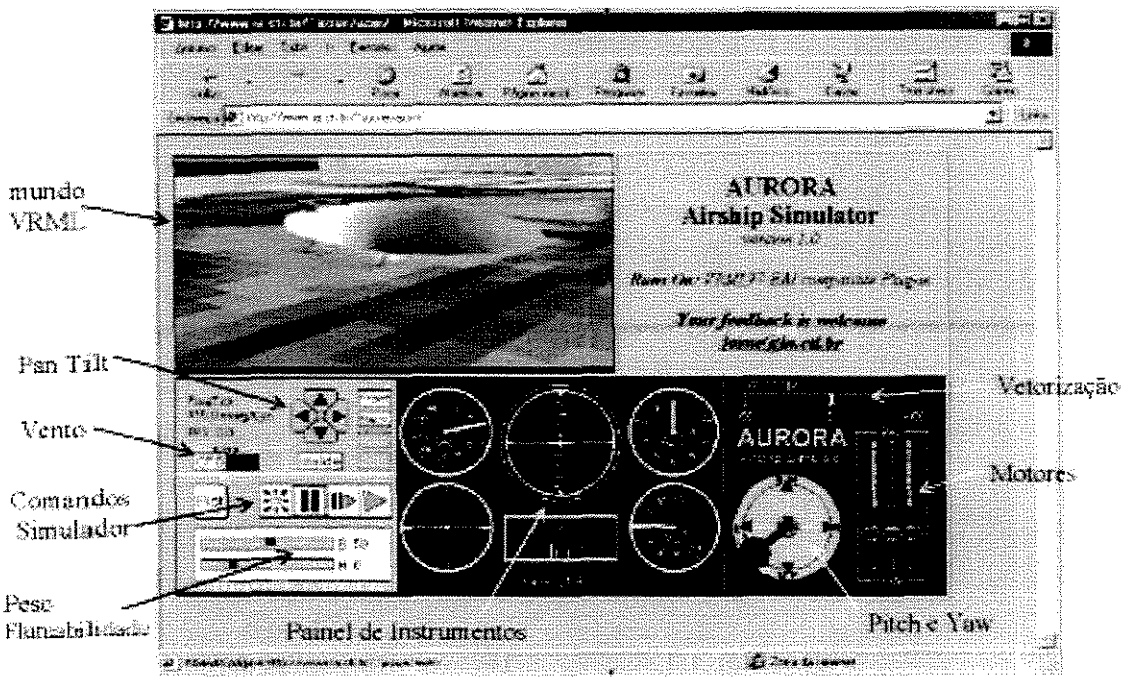


Figura 4.5: Simulador de dirigíveis com o Painel de Instrumentos e Comandos.

4.3.2 Apresentação de gráficos das variáveis de telemetria

Um *script* Tcl/Tk (Figura 4.6) recebe dados do servidor de *sockets* e mostra as leituras recebidas na forma de gráficos, em tempo real. O usuário pode escolher quais variáveis são mostradas (o *script* lista todos os campos disponíveis no pacote de dados, utilizando o arquivo de configuração para identificar o nome de cada campo), além de um gráfico de latitude x longitude, ou seja, um gráfico do movimento do dirigível no mundo. Caso disponha-se de um mapa digitalizado da região de vôo, pode-se colocá-lo no fundo deste gráfico, permitindo a visualização do movimento do veículo sobre o mapa em tempo real. Os gráficos são *widgets* da extensão BLT [2] da linguagem Tcl/Tk, que implementam *auto-scroll, zoom, auto-scale,* e outras funcionalidades.

Além de desenhar os gráficos o *script* mostra continuamente algumas informações críticas: a situação do GPS e do DGPS, que são necessários para o funcionamento dos algoritmos de controle; a altitude, a velocidade do veículo e a distância em relação ao ponto de decolagem; a temperatura interna do veículo aéreo, uma indicação se o veículo está no modo de pilotagem manual ou automático e o tempo decorrido desde o momento em que foi recebido o último pacote de dados do sistema embarcado. Através da utilização de cores para estes mostradores - verde para operação correta, amarelo para situações

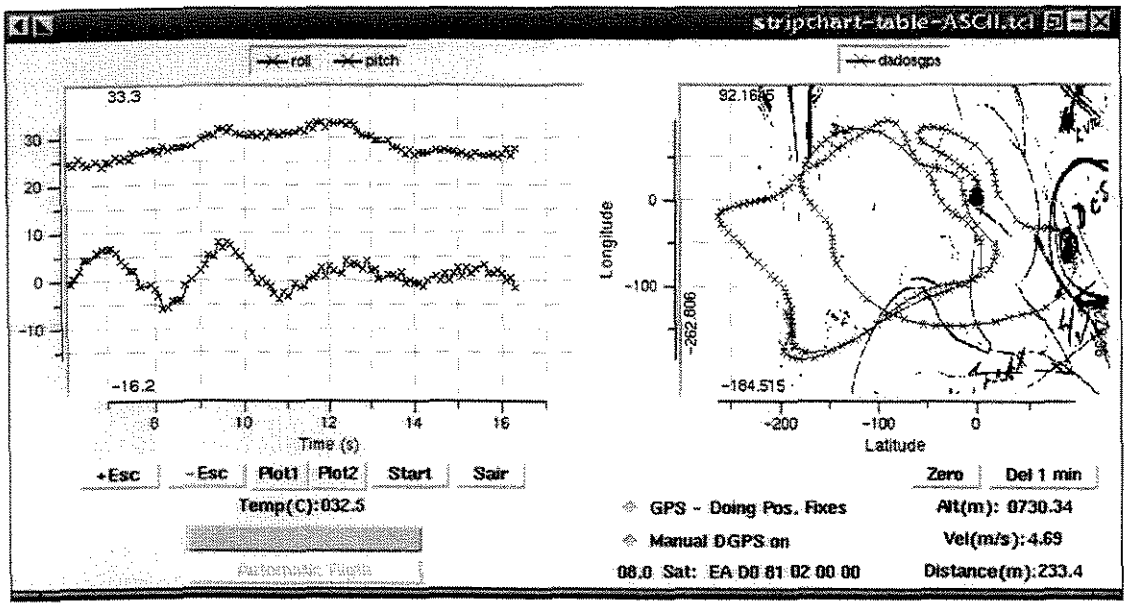


Figura 4.6: Visualização dos dados em tempo real.

perigosas (por exemplo, o GPS encontrou apenas 3 satélites) e vermelho para falhas (por exemplo, GPS é incapaz de fornecer dados de posição) - o operador da estação de terra pode detectar qualquer problema imediatamente. Assim a decolagem deverá ser permitida somente se todos os mostradores estiverem com a cor verde, proporcionando um meio rápido para o operador verificar a segurança do sistema.

4.3.3 Visualizador 3D

Para conseguir uma visualização 3D em ambiente Linux, foi utilizado um visualizador de veículos aéreos (Figura 4.7), desenvolvido no INRIA, França, e no Instituto Superior Técnico, em Portugal, e cedido ao Projeto AURORA através de um acordo de cooperação entre os laboratórios. O visualizador, que utiliza a biblioteca OpenGL, vem sendo utilizado em um projeto de um avião autônomo em Portugal, e foi adaptado para utilizar o arquivo de configuração para receber os estados do dirigível.

4.4 O Planejador de Missões

O operador da estação de terra deve definir uma missão e enviá-la para o sistema embarcado, para ser executada, conforme especificado na Seção 2.3. Deve ser possível definir

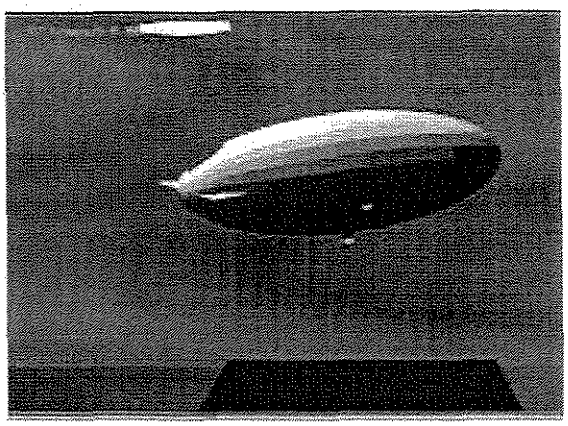


Figura 4.7: Visualizador do dirigível em OpenGL.

a missão de forma *on-line*, sobre o mapa, e armazenar uma descrição da missão em um arquivo, que possa ser utilizado posteriormente e, se necessário, modificado.

O Planejador de Missões foi implementado em C++, utilizando-se a biblioteca Qt, que esta está disponível livremente para uso não comercial em ambiente Linux. O objetivo da biblioteca é principalmente facilitar o desenvolvimento de aplicações utilizando GUI (*Graphical User Interface*), contando também com *class templates* implementando as estruturas de dados mais comuns (pilhas, listas, etc.). Utilizando-se a linguagem C++ consegue-se uma execução mais rápida e utilizando menos memória em relação a Java, além de se contar com uma linguagem orientada a objeto e com uma biblioteca gráfica e de estruturas de dados (graças a biblioteca Qt). Certamente utilizar uma linguagem orientada a objeto permitiu desenvolver um *software* de manutenção mais fácil do que se tivesse sido utilizada a linguagem Tcl/Tk (por exemplo, por permitir a utilização de padrões de projeto).

O primeiro tipo de missão a ser implementado é seguir uma lista sequencial de pontos de passagem definida pelo usuário, e voltar para o primeiro ponto da lista depois de alcançado o último ponto. Isto aplica-se a diversos tipos de monitoração aérea, onde existe uma trajetória pré-definida que o veículo deve seguir, por exemplo para monitorar a área ao redor de uma região ou objeto de interesse. As experiências de campo descritas na Seção 5 se referem a este tipo de missão.

Os componentes da missão (pontos de passagem, por exemplo) podem ser definidos clicando-se sobre o mapa ou através da leitura de um arquivo texto descrevendo a missão.

Conectando-se ao servidor de *sockets*, o Planejador de Missões obtem as informações de posição e atitude do dirigível. Assim pode também desenhar uma imagem do dirigível sobre o mapa, rotacionando-a para representar a orientação do dirigível (*yaw* ou *heading*).

O Planejador de Missões utiliza cores diferentes ao desenhar a representação da mis-

são sobre o mapa para indicar o progresso da missão. Assim, pode-se diferenciar uma missão ainda não enviada ao sistema embarcado, de uma missão já enviada ao sistema embarcado. Adicionalmente, outras cores são utilizadas para o componente da missão sendo executado a cada momento, para componentes já executados, e para componentes que não serão executados (o sistema embarcado pode informar que não irá executar determinados componentes de uma missão, devido a falta de combustível, por exemplo).

Desta forma o usuário pode visualizar o dirigível movimentando-se sobre o mapa, dirigindo-se a cada ponto de passagem, e visualizar inclusive o momento em que o dirigível alcança um ponto de passagem e passa a se dirigir a outro, graças ao código de cores. A Figura 4.8 apresenta uma imagem da tela do programa durante a execução de uma missão simulada.

Cumprindo outro dos requisitos necessários para a estação de terra, o Planejador de Missões também permite que o usuário envie parâmetros de configuração para o sistema embarcado. O usuário define o número, nome e valor padrão dos parâmetros desejados em um arquivo texto de configuração e o programa permite que o usuário edite os valores dos parâmetros e os envie ao sistema embarcado. Isto é utilizado para se variar, por exemplo, os ganhos dos controladores do *software* embarcado, permitindo testar diferentes valores durante um vôo sem necessidade de se aterrissar o veículo para reprogramar o sistema embarcado.

Com a evolução do sistema embarcado, o dirigível deverá se tornar capaz de executar também outros tipos de missões ou tarefas, portanto será necessário criar outros tipos de componentes da missão. Assim, é necessário que o Planejador de Missões seja escrito de forma a ser facilmente modificado para atender aos novos requisitos, principalmente em relação às classes que modelam a missão e seus componentes, bem como a interface com o usuário que permite que este defina e acompanhe o andamento de uma missão. A utilização de padrões de projeto [7] auxilia o desenvolvimento de um *software* melhor modularizado e documentado e de manutenção e atualização mais rápidas.

As classes que representam internamente os componentes de uma missão utilizam o padrão *Composite*. Conforme este padrão, há uma classe base para todos os tipos de componentes (`MissionComponent`), uma classe para tipos de componentes compostos, ou seja, componentes que contém uma lista de componentes (`MissionComposite`, utilizada para se representar a missão como um todo), e classes específicas de componentes descendentes de `MissionComponent` (e.g. `Waypoint`, `Takeoff`, `Hover`, representando respectivamente um ponto de passagem, decolagem e vôo pairado).

A cada classe de componente pode corresponder uma classe de janela de configuração. Tais janelas devem ser criadas por um método do próprio componente, mostrar ao usuário os atributos do componente (e.g. nome e coordenadas) e permitir ao usuário modificá-los. Para isto foi utilizado o padrão *Factory Method*, onde, para cada classe de com-

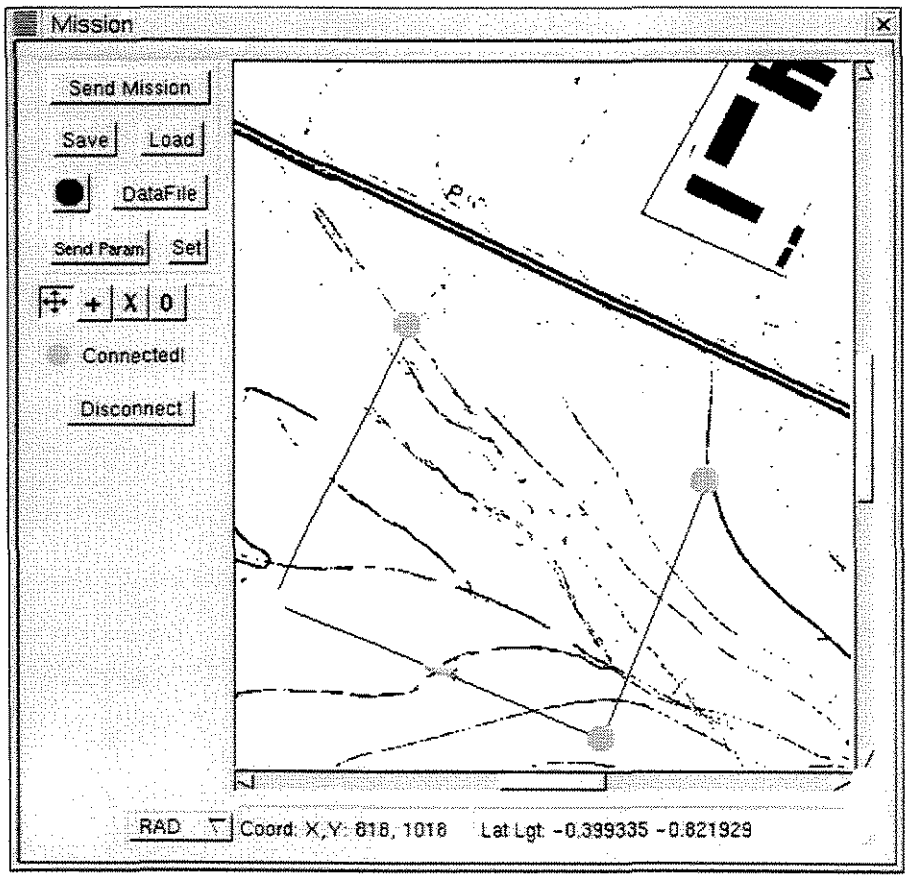


Figura 4.8: O Planejador de Missões, mostrando o dirigível durante uma missão simulada.

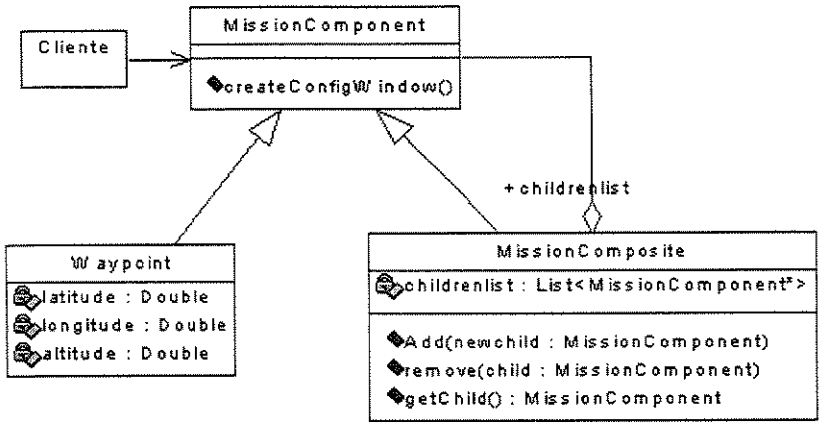


Figura 4.9: Utilização do padrão de projeto *Composite* para representar os componentes de uma missão.

ponente, pode-se criar uma nova classe de janela de configuração (descendente da classe `ConfigurationWindowImpl`) e se reimplementar o método `MissionComponent::createConfigDialog()`, ou não se reimplementar este método e utilizar o mesmo tipo de janela de configuração da classe base.

Deve ser mostrada ao usuário uma representação da missão sobre o mapa, onde cada componente da missão deverá ser representado por uma imagem na coordenada correspondente do mapa. A biblioteca Qt fornece classes para desenho de figuras, que permitem construir um *widget* contendo a imagem da região de vôo ao fundo, e desenhar outras imagens sobre esta imagem. A cada uma destas imagens deve corresponder um objeto da classe `QCanvasSprite` (fornecida com a biblioteca).

A abordagem utilizada para que cada tipo de componente pudesse ter a sua imagem desenhada sobre o mapa, e manter separados a representação interna da missão e o desenho mostrado ao usuário na interface, foi implementar uma classe base descendente de `QCanvasSprite`, chamada `RotatingSprite`, que implementa todas as funcionalidades necessárias para representar um componente sobre o mapa, mas não possui uma imagem, e, para cada tipo de componente, deve ser implementado um descendente de `RotatingSprite` que abre a imagem correspondente e realiza qualquer outra tarefa necessária para desenhar aquele componente específico.

Para ligar um componente com a sua representação gráfica, utilizamos o padrão *Observer*, onde `MissionComponent` atua no papel *subject* e `RotatingSprite` é o *observer*. Assim, cada vez que o usuário, utilizando o mouse, arrastar o desenho do componente na interface gráfica, o objeto MissionComponent é alterado, e, se o objeto `MissionComponent` for alterado de outra forma (por exemplo, por uma mensagem do sistema embarcado),

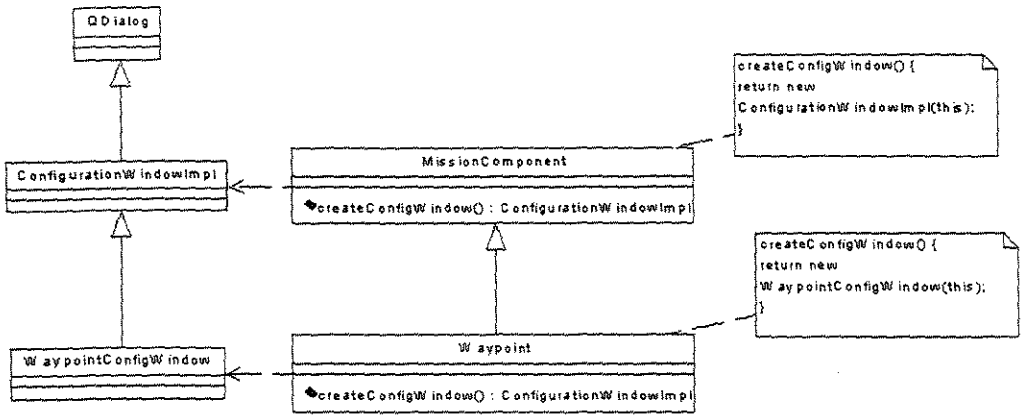


Figura 4.10: Utilização do padrão de projeto *Factory Method* para criar janelas de configuração.

ele utilizará o método `notify()` para notificar os seus *observers* da mudança. Note que pode haver mais de um *observer* por *subject*, o que abre a possibilidade de se implementar outras formas de representar a missão ao usuário graficamente (por exemplo, por meio de uma lista), sem necessidade de se alterar muito o código (este é um dos objetivos do padrão *Observer*).

Portanto, para se adicionar um novo tipo de componente, normalmente é necessário criar uma classe descendente de `Waypoint`, que implementaŕa o novo tipo de tarefa, e uma classe descendente de `RotatingSprite` para representar tal componente sobre o mapa (muitas vezes tudo o que esta classe tem a fazer é carregar a imagem correta). Será necessário alterar também o código de leitura dos arquivos de descrição da missão para reconhecer o novo tipo, e criar uma nova classe de janela de configuração caso nenhuma das existentes seja adequada.

4.5 Reapresentação dos vôos realizados

A partir dos dados armazenados pelo cliente mostrado na Seção 4.3, deve-se mostrar ao usuário uma reapresentação dos vôos realizados, reais ou simulados, utilizando-se a mesma interface utilizada nos vôos reais.

Um *script* Tcl/Tk lê um arquivo de dados de vôo previamente gravado e escreve os pacotes na saída padrão. Um servidor de *sockets* similar ao utilizado nos vôos (que, ao invés de receber os dados por uma rt_fifo, os recebe pela entrada padrão), permite a reapresentação dos vôos armazenados (Figura 4.12). Basta conectar os dois processos a

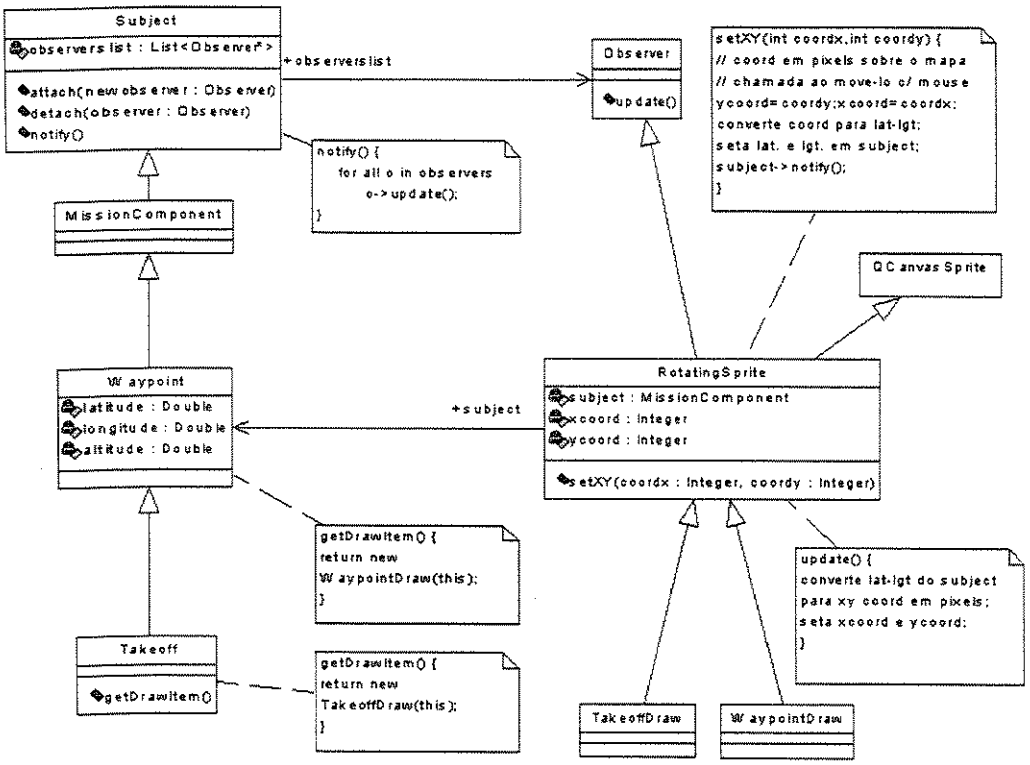


Figura 4.11: Utilização do padrão de projeto *Observer*, separando a representação interna da missão da interface gráfica.

partir da linha de comando utilizando o comando *pipe* ("|"). No *script* Tcl/Tk, botões similares aos de um vídeo cassete (*Play, Fast Forward, Stop*, etc.) permitem o controle da transmissão dos pacotes armazenados no arquivo. O programa tem um contador de tempo, e leva em conta o tempo armazenado juntamente com cada pacote, para transmitir os pacotes na mesma escala de tempo em que foram recebidos, permitindo uma reprodução realista do vôo. Os clientes de *socket* são os mesmos utilizados em vôos reais.

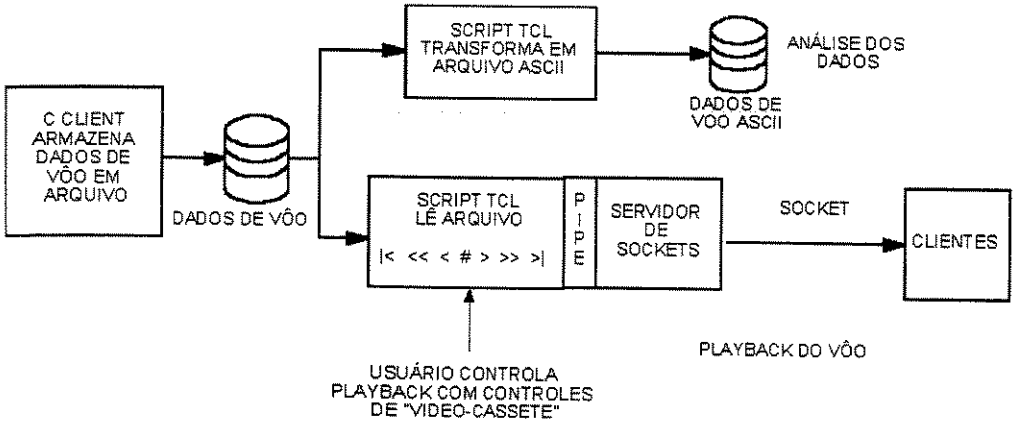


Figura 4.12: Esquema utilizado para reapresentação de vôos.

É importante notar que este servidor de reapresentação não necessitará ser modificado em caso de mudanças no formato do pacote ou se outros sensores forem acrescentados ao sistema, pois supõe-se que o arquivo de dados de vôo contenha pacotes já no formato requerido pelos diversos clientes que recebem os dados. Além disto ele utiliza o arquivo de configuração para identificar o tamanho do pacote de dados a ser lido do arquivo de dados.

Além da ferramenta de reapresentação foi desenvolvido um outro *script* Tcl/Tk que gera, a partir do arquivo de dados de vôo, um arquivo ASCII que contém na primeira linha os nomes das variáveis gravadas (nomes obtidos do arquivo de configuração) e nas linhas subsequentes os valores das variáveis em cada pacote recebido. Um arquivo ASCII desta forma pode ser importado por programas de análise matemática ou estatística, como Matlab, Origin ou MiniTab.

4.6 Validação em Simulação

Para validar o sistema desenvolvido, foram realizados vôos simulados antes dos vôos reais. A estação de terra foi conectada a um sistema embarcado simulado, que por sua vez comunica-se com o modelo dinâmico do dirigível em Matlab. A estação de terra usa

em simulação o mesmo *hardware* e o mesmo *software* utilizados em vôo real. Em uma simulação, a estação de terra envia comandos para o sistema embarcado simulado, que os repassa para o modelo do dirigível. O modelo então realiza um passo da simulação, calculando novos estados para o dirigível simulado, e envia os estados de volta para o sistema embarcado simulado e este por sua vez os envia para a estação de terra.

O sistema embarcado simulado foi desenvolvido em cooperação com a equipe do Projeto AURORA. Nele, um LKM comunica-se com a estação de terra via rádio-modem, simulando a comunicação efetuada pelo sistema embarcado real. Este LKM comunica-se via duas rt-fifos com um servidor de *sockets*, e o modelo do dirigível conecta-se a este servidor para receber os comandos do dirigível e enviar os estados da simulação. Além disso, o servidor de *sockets* também possui uma interface com o usuário onde este pode enviar comandos para o modelo iniciar ou suspender a simulação, bem como visualizar os valores dos comandos e estados transmitidos. O servidor utiliza uma *thread* (utilizando a biblioteca *pthreads*) para realizar a leitura e envio dos comandos do dirigível, uma para os estados do modelo, e uma para os comandos de início e término da simulação.

O LKM do sistema embarcado simulado contém algoritmos de controle, que devem gerar comandos para o veículo a partir do conjunto de estados recebidos do modelo matemático.

A Figura 4.13 mostra o diagrama em blocos do sistema utilizado em simulação. A seguir apresentamos uma decomposição temporal da operação do sistema em um vôo simulado:

1. Os comandos da RCU (ou do Painel de Comandos Java) são lidos pelo LKM da estação de terra e, caso o sistema esteja em modo manual de operação, são enviados para o sistema embarcado pelo rádio-modem. A estação de terra decodifica o campo de seleção entre vôo automático e manual, e se necessário envia pacotes solicitando a mudança no modo de operação (Seção 4.2.4).

2. O LKM do sistema embarcado simulado recebe os comandos enviados pela estação de terra e escreve-os em uma rtfifo (se estiver em modo manual de operação). Em modo automático de operação, os comandos são ignorados (além disso, a estação de terra não os envia neste modo de operação).

3. Em modo automático de operação, o sistema embarcado simulado executa os algoritmos de controle sobre os valores do último conjunto de estados recebido do modelo. O conjunto de comandos gerado é então escrito na mesma rtfifo citada no item anterior.

4. O servidor de *sockets* lê a rtfifo contendo os comandos e os envia para o modelo matemático via rede (*socket*).

5. O modelo matemático, ao receber os comandos, executa um passo de integração do modelo dinâmico do dirigível. As saídas dessa execução são a nova posição e atitude do veículo simulado em relação a um referencial inercial. Estas saídas, os estados do modelo, são enviadas via rede para o servidor de *sockets*.

6. O servidor de sockets recebe os estados do modelo e os escreve em outra rtfifo.

7. O LKM do sistema embarcado simulado lê a rtfifo citada no item 6, repassa os estados do modelo para os seus algoritmos de controle e envia uma cópia dos estados para a estação de terra via rádio-modem.

8. O LKM da estação de terra recebe os estados do modelo e escreve-os em uma rtfifo. Segue-se a operação normal da estação de terra, onde um servidor de *sockets* lê os estados da rtfifo e os disponibiliza para os clientes, que os apresentarão ao usuário (veja as seções anteriores).

O modelo de dirigíveis utilizado foi desenvolvido pelos pesquisadores do Projeto AURORA, utilizando o *software* MATLAB. Este modelo também foi portado para a linguagem Java, para ser utilizado no simulador (mostrado na Seção 4.3.1). Qualquer um destes dois modelos existentes pode ser utilizado. Maiores informações sobre o modelo utilizado podem ser encontradas em [8].

A estação de terra utiliza a mesma interface para mostrar ao usuário os estados da simulação. Para isto, basta trocar, na estação de terra, o arquivo de configuração que representa o pacote de dados do sistema real por um outro que represente o formato dos estados do modelo (normalmente um conjunto de variáveis de ponto flutuante). Deve-se, entretanto, cuidar para que os nomes dos campos relevantes sejam os mesmos, para que as informações possam ser mostradas. Por exemplo, o Painel de Instrumentos procura um campo chamado "altitude" que deve conter a informação a ser mostrada no instrumento altímetro.

4.7 Análise em relação aos requisitos

No Capítulo 2, foram detalhados os requisitos da estação de terra. Para cada uma das seções do Capítulo 2, será especificado como e onde tais requisitos foram atendidos, na enumeração a seguir:

1. **Requisitos relacionados com a segurança do dirigível:**
   O LKM da estação de terra permite o envio de comandos para o sistema embarcado, bem como permite ao usuário selecionar entre vôo automático e vôo manual. Os comandos a serem enviados podem ser obtidos do RCU, ou, em vôos simulados, do

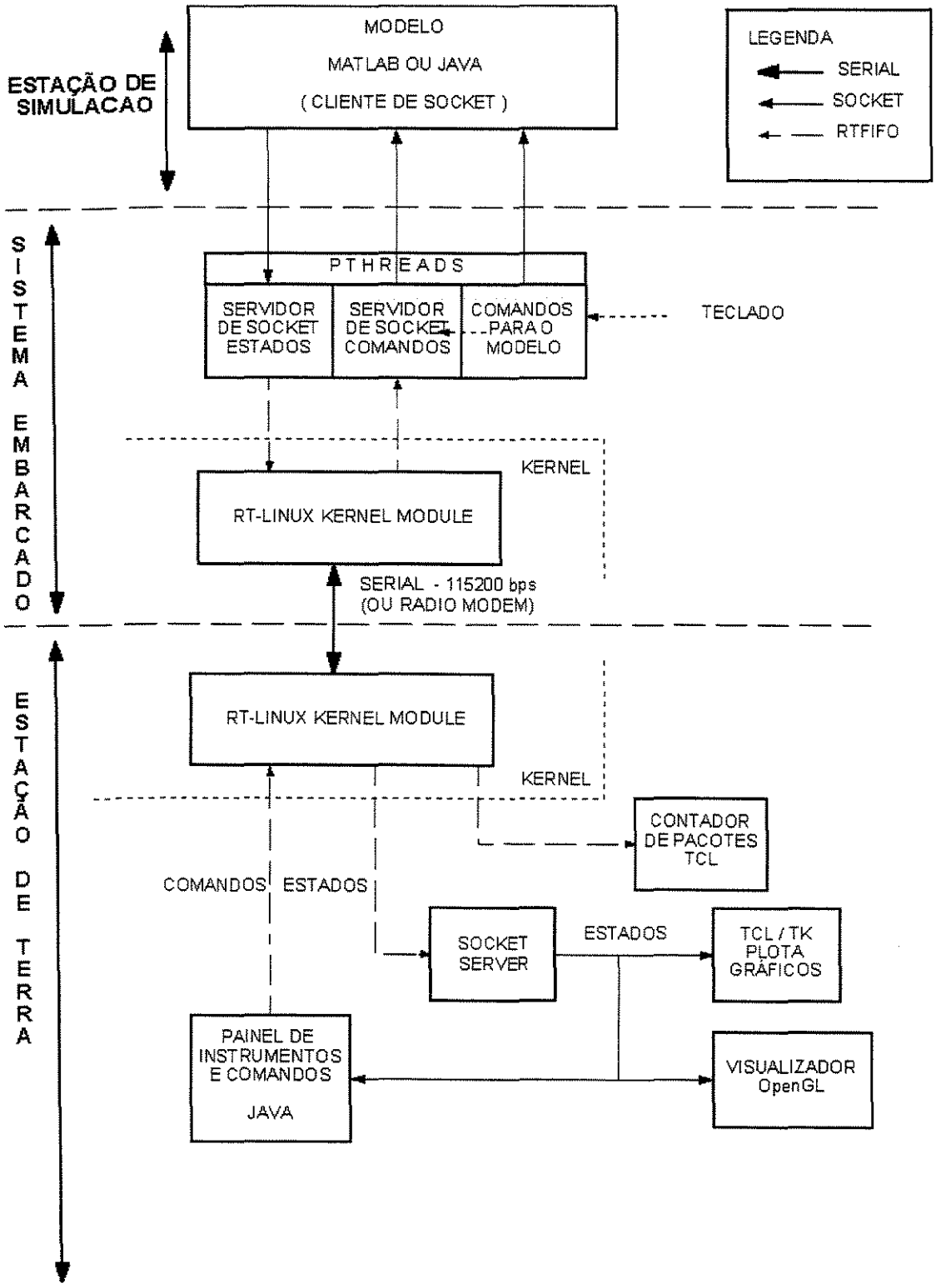


Figura 4.13: Diagrama de blocos do sistema utilizado em simulação.

Painel de Instrumentos Java.
O *script* Tcl/Tk que apresenta gráficos das variáveis de telemetria apresenta mostradores coloridos para algumas variáveis críticas para a segurança do veículo (estado do GPS e DGPS, estado do *link* estação de terra - sistema embarcado, etc.), para chamar a atenção do usuário no caso de problemas.

2. **Requisitos relacionados com a monitoração do vôo do dirigível:**
A trajetória do veículo é mostrada em tempo real sobre um mapa da área de vôo pelo script Tcl que apresenta gráficos e pelo Planejador de Missões. Este último mostra inclusive a orientação do dirigível, desenhando uma imagem do dirigível na orientação dada pela bússola.
O Painel de Instrumentos Java apresenta continuamente ao usuário mostradores com ponteiros similares aos encontrados em aviões e jogos, que permitem um acompanhamento visual e direto de outras variáveis como altitude, atitude, etc.
O mesmo script Tcl permite ao usuário visualizar em tempo real um gráfico de qualquer das variáveis de telemetria transmitidas para a estação de terra pelo sistema embarcado.
O Visualizador OpenGL atende ao requisito de apresentar uma visualização 3D do veículo, mostrando um desenho do dirígivel se movimentando sobre um mundo virtual, sob diversos pontos de vista possíveis.

3. **Requisitos relacionados à missão a ser executada:**
O Planejador de Missões atende a todos os requisitos da Seção 2.3. Permite ao usuário especificar e enviar uma missão ao sistema embarcado, sendo composta de uma lista de tarefas a serem executadas.
O Planejador permite também acompanhar a execução da missão, através da visualização da figura do dirigível se movimentando no mapa sobre os pontos de passagem ou outros componentes da missão, e através do código de cores que indica o *status* da execução de cada componente da missão. Além disso, pode-se modificar uma missão em andamento, sem prejudicar a visualização da missão sendo executada, e enviar a missão modificada ao sistema embarcado quando desejado. Para abortar uma missão basta retornar para o modo manual.
Os componentes da missão podem ser definidos por manipulação direta sobre o mapa ou através da entrada manual das suas coordenadas de latitude, longitude, altitude. Outras informações necessárias, por exemplo, o tempo de espera para um vôo pairado, podem ser definidas através de uma janela de configuração aberta ao se clicar sobre a representação do componente sobre o mapa com o botão direito do mouse. Adicionalmente, a definição da missão pode ser salva ou carregada em arquivos.

Uma janela do Planejador de Missões permite se editar os valores dos parâmetros necessários e enviá-los para o sistema embarcado. O número, nome e valor *default* dos parâmetros são definidos em um arquivo de configuração.

4. **Requisito necessário como suporte para o sistema embarcado:**
   O LKM da estação de terra realiza a leitura de dados de DGPS, conectado via uma porta serial e os envia para o sistema embarcado. Não é necessário se decodificar o conteúdo dos pacotes de dados, já que o receptor GPS embarcado é compatível com o formato de dados enviado.

5. **Requisitos necessários para a manutenção e desenvolvimento da estação de terra:**
   O LKM da estação de terra e o servidor de *sockets* não realizam nenhum tratamento sobre os dados recebidos do sistema embarcado. Os clientes recebem o pacote de dados e utilizam o arquivo de configuração para decodificá-lo. Assim, basta mudar o arquivo de configuração ao se adicionar ou retirar qualquer campo de dados à telemetria.
   Além disso, basta utilizar um outro arquivo de configuração para vôos simulados e se poderá utilizar o sistema embarcado simulado transparentemente no lugar do dirigível real.
   A existência do servidor de *sockets* permite que os clientes sejam desenvolvidos independentemente dos demais. Note também que cada cliente pode ser utilizado ou não, independentemente dos demais, conforme necessário. Além disso, o RT-Linux permitiu separar os *device drivers* e o *software* responsável pela comunicação com o sistema embarcado, que são implementados no LKM da estação de terra, da interface com o usuário, que é implementada como processos de usuário.

6. **Requisitos necessários para a análise posterior do vôo ou da missão:**
   Um cliente do servidor de *sockets* armazena em arquivo todos os pacotes de dados de telemetria recebidos juntamente com o momento da recepção de cada pacote de dados.
   A partir dos arquivos armazenados pode-se gerar um arquivo ASCII, contendo todos os dados de telemetria recebidos e uma linha de cabeçalho contendo o nome de cada campo (que são obtidos a partir do arquivo de configuração da estação de terra), para análise em programas comerciais como Matlab, Origin, etc.
   O servidor de reapresentação faz o papel do servidor de sockets em um vôo real, lendo os dados contidos nos arquivos armazenados e disponibilizando-os para os clientes. Isto permite que os clientes sejam utilizados em uma reapresentação da mesma forma que são utilizados em um vôo real.

Portanto todos os requisitos apresentados foram satisfeitos pelos diversos componentes da estação de terra.

Capítulo 5

Experiências de campo

A estação de terra está sendo utilizada nos vôos do dirigível do Projeto AURORA. Primeiramente foram realizados vôos com pilotagem manual e telemetria dos sensores, para testar o sistema de aquisição de dados sensoriais e obter dados de telemetria, para ajudar os desenvolvedores de algoritmos de controle do Projeto AURORA a modelar a dinâmica do veículo. Posteriormente foram realizados vôos com controle automático das superfícies aerodinâmicas permitindo que o dirigível seguisse autonomamente uma trajetória pré-definida.

A estação de terra foi utilizada nesses vôos para permitir um acompanhamento em tempo real das leituras dos sensores, para armazenar os dados para posterior análise pelos desenvolvedores dos algoritmos de controle, e para enviar comandos para o dirigível, definindo a trajetória a ser percorrida e configurando os algoritmos de controle embarcados. Detalhes sobre os algoritmos utilizados e o experimento realizado podem ser encontrados em [27] e [28].

No decorrer de 2000 e 2001, foram realizados cerca de 30 vôos com o dirigível nas dependências da II Cia. de Comunicações Blindada (II Ciacom), uma unidade do Exército Brasileiro baseada no município de Campinas. Em um destes experimentos, o dirigível foi comandado manualmente por alguns minutos antes de iniciar o controle automático. Foram definidos quatro pontos de passagem, formando um quadrado de lado igual a 150 m. A velocidade do vento estava em torno de 10 m/s. A Figura 5.1, obtida a partir dos dados de telemetria armazenados pela estação de terra, mostra a trajetória percorrida pelo dirigível, enquanto o piloto manualmente controlava a altitude e a rotação dos motores, utilizando a RCU. A linha pontilhada representa a trajetória do dirigível da decolagem até o início da operação por controle automático. A linha contínua representa a trajetória do veículo sob o controle automático. A última parte da trajetória, representada com uma linha tracejada, apresenta a trajetória do dirigível após o controle automático ter sido desligado, sendo o dirigível pilotado manualmente até o pouso. As Figuras 5.2 e 5.3

apresentam fotos da estação de terra instalada nas dependências da II Ciacom, próxima à área de vôo do dirigível.

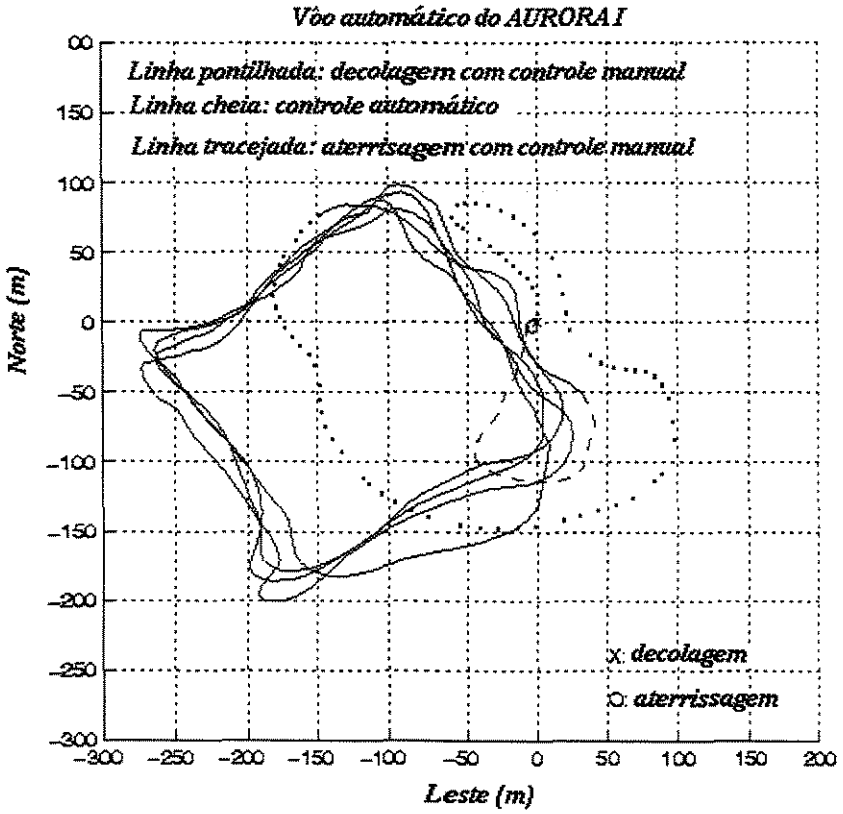


Figura 5.1: Trajetória do dirigível em um experimento real.

Diversos outros vôos foram realizados no mesmo local, apenas com pilotagem manual, com o objetivo de testar novos componentes de *hardware* do dirigível e obter dados de sensores montados no dirigível.

A estação de terra foi utilizada em todos os vôos, permitindo a monitoração do estado do dirigível antes e durante os vôos. Os componentes da estação de terra permitiram detectar falhas nos sensores antes da decolagem, por exemplo em uma ocasião em que o cabo da antena do GPS foi desconectado acidentalmente durante os preparativos para a decolagem de um vôo automático. O operador da estação de terra pôde perceber rapidamente a falha devido ao alarme visual e avisar os responsáveis para suspender a decolagem, evitando a realização de um vôo inútil e perigoso.

A estação de terra também vêm permitindo que os dados de novos sensores - por exemplo, sonda de vento e tacômetros - sejam facilmente integrados ao pacote de dados transmitido durante a telemetria.

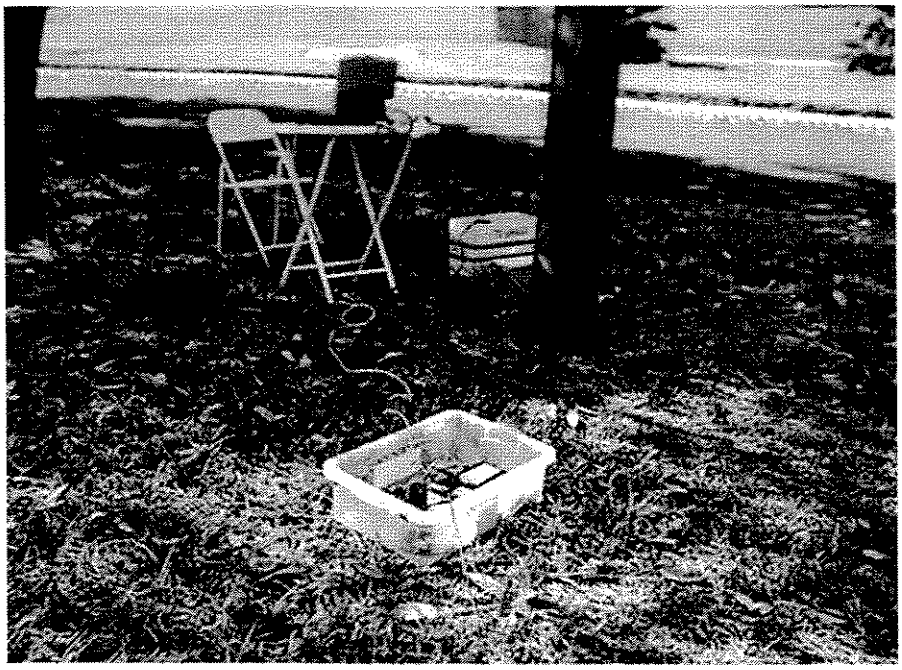


Figura 5.2: A Estação de Terra durante um vôo.

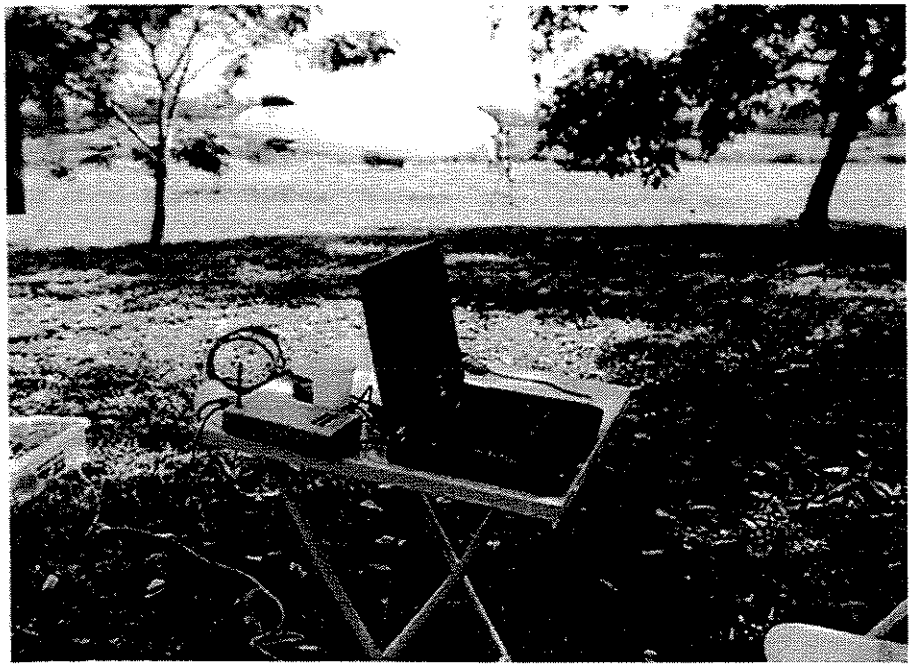


Figura 5.3: A Estação de Terra, em outro ângulo, com o dirigível ao fundo.

Capítulo 6

Conclusões e desenvolvimentos futuros

6.1 Contribuições

Veículos aéreos possuem um enorme potencial não explorado em diversas tarefas de inspeção e monitoração aérea, como monitoração de tráfego, planejamento urbano, inspeção de grandes estruturas como linhas de transmissão ou oleodutos, retransmissão de sinais de rádio e vídeo, prospecção mineral e arqueológica, policiamento, e pesquisa e monitoração ambiental, climatológica e de biodiversidade.

Atualmente, muitas das tarefas descritas acima são realizadas a partir de sensoriamento remoto, mais precisamente, de dados obtidos através de sensores e câmeras instalados em balões, satélites ou aviões. Dirígiveis apresentam diversas vantagens sobre estes veículos para a realização das tarefas descritas acima, conforme apresentado em [5].

A adoção de veículos robóticos aéreos semi-autônomos não tripulados permite a aquisição de dados de forma que o usuário possa determinar a área a ser monitorada, a resolução espacial e temporal dos dados, e também o sensor ideal para cada tipo de missão, a custos relativamente baixos. Tem-se como consequência uma expansão do uso científico e civil de dados aéreos e benefícios sociais e econômicos significativos.

No projeto AURORA visa-se o estabelecimento de dirigíveis não-tripulados para inspeção aérea, contando com significativos graus de autonomia durante todas as fases de suas missões, e envolvendo o desenvolvimento de protótipos sucessivamente com maior capacidade de vôo, capazes de cobrir maiores distâncias, e com graus cada vez maiores de autonomia. Tais protótipos evoluirão de veículos puramente tele-operados para veículos telemonitorados.

Esta dissertação contribuiu para o desenvolvimento do Projeto AURORA definindo, com o auxílio da revisão bibliográfica, os requisitos pertinentes à uma estação de terra apropriada para o tipo específico de veículo utilizado no Projeto AURORA, e implementando a estação de terra visando satisfazer estes requisitos, conforme apresentado na seção

final do Capítulo 4. Assim, a estação de terra pode ser utilizada nos vôos do Projeto AURORA, permitindo a monitoração e o envio de comandos ao veículo, a monitoração da execução da sua missão, e fornecendo suporte, através dos dados coletados, para o desenvolvimento dos algoritmos de controle e do sistema embarcado do Projeto AURORA. Em relação à definição e monitoramento da execução da missão, o Planejador de Missões foi desenvolvido de forma a ser facilmente modificável para refletir novos tipos de tarefas que o veículo seja capaz de realizar futuramente.

Adicionalmente, esta dissertação definiu os requisitos do sistema operacional a ser utilizado tanto na estação de terra quanto no sistema embarcado do Projeto AURORA, e apresenta a utilização do sistema RT-Linux, configurando-o para suportar convenientemente o *hardware* do sistema embarcado e o desenvolvimento do *software* de controle do sistema embarcado e do software da estação de terra, satisfazendo os requisitos de tempo real de ambos os sistemas.

Portanto a estação de terra representa uma contribuição imprescindível para o cumprimento dos objetivos do Projeto AURORA, e permitiu a realização de vôos, reais ou simulados, inicialmente com telemetria e pilotagem manual, e posteriormente com o envio de missões para vôos automáticos.

6.2 Trabalhos futuros

Nesta seção são apresentadas possibilidades para trabalhos futuros a partir da estação de terra desenvolvida, divididos em possíveis novos requisitos, ou seja, adição de novas funcionalidades e melhora da usabilidade, e outras possíveis implementações da estação de terra e seus componentes.

Com relação à possíveis novos requisitos para extender a estação de terra, podemos destacar os seguintes:

A interface com o usuário da estação de terra foi desenvolvida tendo em vista apenas as necessidades atuais do Projeto AURORA e considerou apenas o próprio grupo de pesquisadores como usuários potenciais da estação de terra, conforme definido na Seção 4.1. Pode ser realizado um estudo mais aprofundado na área de interfaces homem-computador (IHC), estendendo-se o grupo de usuários-alvo (por exemplo, considerando os usuários finais de uma das diversas aplicações possíveis para inspeção aérea), e se aplicando os métodos de avaliação e *design* da área de IHC. A estação de terra atual pode ser utilizada nestes testes de usabilidade, como ponto de partida de desenvolvimento ou base de comparação para outras interfaces. Em [19], é conduzido um teste de usabilidade com diferentes usuários avaliando um *software* que permite programar uma missão para um robô autônomo, avaliando o tempo necessário para que os usuários programem uma missão descrita textualmente em um robô simulado.

A estação de terra foi desenvolvida tendo em vista a existência de apenas um dirigível. Extendê-la para possibilitar a monitoração e o envio de missões para mais de um veículo aéreo ou para outros tipos de robôs autônomos como veículos terrestres permitiria desenvolver aplicações envolvendo a cooperação entre diversos robôs, uma área de pesquisa promissora dentro da robótica.

Para isto se necessitaria, supondo que a estação de terra receberia dados de todos os robôs, via rádios-modem ou rádio-*ethernet* (que seria desejável caso seja necessário que os robôs se comuniquem entre si), modificar o LKM da estação de terra para ser capaz de estabelecer comunicação com cada robô (um rádio-modem para cada robô ou um endereço IP no caso de rádio-*ethernet*). Além disso o servidor de *sockets* deveria receber todos estes dados e permitir aos clientes escolher um (ou vários) robô(s) dentre os disponíveis por meio de um protocolo a ser definido, e receber os dados do(s) robô(s) escolhido(s) para visualização. Provavelmente seria necessário estabelecer um identificador único para cada robô existente, para que o servidor de *sockets* e os clientes possam diferenciá-los. Embora a maioria dos clientes possam simplesmente receber dados de apenas um robô de cada vez para visualização, particularmente o Planejador de Missões deveria ser modificado para suportar a definição de missões envolvendo mais de um robô, e monitorar a execução destas missões, necessitando portanto receber dados de diversos robôs.

Considerando que existe uma missão, composta de diversos componentes, a ser enviada ao dirigível e executada pelo mesmo, podem ocorrer eventos e contingências que devem provocar replanejamento da missão (por exemplo, devido à falta de combustível, pode ser impossível executar todos os componentes de uma missão longa) ou a ativação de comportamentos reativos de emergência (por exemplo, realizar um pouso de emergência devido à falha de um dos motores do veículo). Para isso é necessário utilizar uma arquitetura de *software* capaz de monitorar o estado do dirigível e as informações sobre o ambiente e supervisionar a execução da missão, de forma a detectar contingências e tomar as decisões necessárias para garantir a segurança do veículo, ativando comportamentos pré-definidos para cada tipo de contingência.

Uma grande parte do *software* necessário deve ser executado no sistema embarcado, porque uma das contingências possíveis é a perda de comunicação com a estação de terra, e o dirigível poderia ter capacidade para tomar a decisão do que fazer nesse caso (tentar pouso de emergência em coordenadas previamente conhecidas como seguras, por exemplo). No entanto, em condições normais, uma parte, provavelmente a de mais alto nível, do replanejamento poderia ser realizado pela estação de terra sob solicitação do sistema embarcado. Exatamente o que deve ser executado na estação de terra depende de uma decisão de projeto, mas no mínimo a estação de terra deveria ser informada sobre as contingências detectadas e decisões tomadas pelo sistema de supervisão do sistema embarcado, e informar o usuário. E, adicionalmente, realizar parte do replanejamento

da missão. Foi realizado um trabalho preliminar neste sentido, no entanto apenas em simulação, utilizando diversos componentes da estação de terra. Utiliza-se a arquitetura para programação de robôs TDL (*Task Description Language*) [33], que é disponibilizada na forma de uma biblioteca C++.

Para se incorporar este novo conjunto de funcionalidades, o Planejador de Missões deveria ser modificado, adicionando-se os algoritmos necessários (é conveniente que ele tenha sido implementado em C++, a mesma linguagem da arquitetura TDL), e modificando-se a interface com o usuário para refletir as ações de replanejamento e detecção de contingências.

Para se possibilitar uma aplicação real com grande autonomia de vôo a partir do sistema desenvolvido. o sistema supervisório deve evoluir tendo em vista a tolerância a falhas do sistema. Deve-se considerar, no caso de um dirigível maior, e portanto com maior capacidade de carga, quais sistemas poderiam ser duplicados provendo redundância de *hardware*. Além disto, o conjunto de contingências e de soluções ou medidas de emergência a serem tomadas deve incluir todas as condições de vôo passíveis de serem encontradas em um vôo real, incluindo falhas de componentes de *hardware*, mecânico ou eletrônico, *software*, e *links* de comunicação.

Isso possibilitaria o desenvolvimento de um sistema capaz de voar autonomamente por longos períodos com pouca necessidade de intervenção ou monitoração humana, portanto capaz de realizar missões de elevada complexidade e duração.

Em relação à possíveis novas implementações para os componentes da estação de terra, destacamos os seguintes:

O sistema Real-Time Linux atende satisfatoriamente os requisitos do Projeto AURORA no seu estágio atual. No entanto, podem ser considerados também outros sistemas operacionais de tempo real (SOTR) para utilização no sistema embarcado ou na estação de terra, de acordo com a evolução dos requisitos e funcionalidades necessárias para ambos os sistemas.

Dentre os SOTR mais utilizados está o sistema VxWorks, da empresa Wind River Systems, que fornece um ambiente de desenvolvimento e execução de tarefas de tempo real bastante complexo, suportando uma grande variedade de processadores. Neste sistema há três componentes altamente integrados: um sistema operacional de alta performance executado no processador alvo, um conjunto de ferramentas de desenvolvimento executadas em um sistema hospedeiro, e um conjunto de componentes de comunicação entre o processador alvo e o hospedeiro.

Um desenvolvimento recente na área de *software* para UAVs é a utilização da plataforma CORBA (*Common Object Request Broker Architecture*), principalmente o CORBA *Real-Time*, e uma rede *ethernet* conectando o veículo aéreo e a estação de terra. Assim, o veículo pode disponibilizar as informações de estados e telemetria para a estação de terra,

e esta pode comandar o veículo acessando serviços CORBA disponibilizados na rede [15], [14]. Neste caso necessita-se realmente de um rádio-*ethernet* devido à maior velocidade de transmissão necessária para se transmitir as requisições CORBA. Pode-se utilizar os serviços fornecidos pelo ORB (*Object Request Broker*), como serviço de nomes e serviço de eventos, para construir objetos independentes que são conectados para se compor o sistema, e a utilização de interfaces IDL permite o desenvolvimento diferentes implementações do mesmo subsistema que podem ser conectadas e desconectadas ao sistema, inclusive em tempo real. O resultado deve ser uma maior facilidade de reconfiguração e reusabilidade do sistema.

A especificação do CORBA *Real Time*, ainda não está terminada, porém ORBs de tempo real já foram implementados na maioria das principais plataformas existentes. Uma avaliação de diversos sistemas operacionais (Windows NT, Solaris, VxWorks, LynxOS e também Linux) em relação ao suporte à execução de um ORB de tempo real pode ser encontrada em [17]. O RT-Linux poderá vir a melhorar os resultados obtidos pelo Linux nesta avaliação, ao se utilizar os recursos do RT-Linux para atender os requisitos de tempo real do ORB, porém não existe ainda nenhuma implementação neste sentido.

O fato do *software* de tempo real da estação de terra e do sistema embarcado estarem escritos em uma linguagem orientada a objeto certamente facilitaria uma eventual mudança visando a adoção do CORBA como infra-estrutura de comunicação.

Uma característica da interface com o usuário da estação de terra é a diversidade de componentes, muitas vezes escritos em linguagens diferentes, apresentando informações ao usuário em várias janelas ao mesmo tempo. Mesmo considerando que a resolução da tela de um monitor de qualquer forma não permitiria que se visualize ao mesmo tempo todas as informações referentes à missão e a telemetria (mapa da trajetória, gráficos, indicadores visuais e instrumentos, contagem de pacotes e tempo, etc.), isso apresenta o inconveniente de dificultar que o usuário localize a informação desejada.

Outras estações de terra também apresentam este problema, como em [13] e [32]. Uma alternativa para minimizar este problema, utilizada em [32], é utilizar como estação de terra dois *laptops*, ligados em rede *ethernet*. Um dos *laptops* comunica-se com o sistema embarcado via o rádio-modem, e disponibiliza os dados de telemetria em um servidor na rede. Então o usuário pode executar os clientes em ambos os *laptops*, obtendo assim maior espaço para visualizar as informações. Esta solução também pode ser utilizada no Projeto AURORA, pois muitos dos componentes da estação de terra são clientes do servidor de *sockets* e portanto podem ser executados em máquinas diferentes.

Os gerenciadores de janelas encontrados para ambiente Linux apresentam uma vantagem sobre o ambiente Windows: existem vários *desktops* virtuais, portanto o usuário pode distribuir os diversos componentes da estação de terra entre eles e alternar entre eles utilizando o teclado ou mouse. Desta forma consegue-se utilizar satisfatoriamente a

estação de terra sem necessidade de dois computadores.

Em [10], uma única aplicação contém toda a interface da estação de terra, permitindo que o usuário escolha e alterne entre as diversas funções (mapa e acompanhamento da missão, instrumentos de navegação, telemetria). Além disso a disponibilidade de dois monitores permite a visualização de quase todas as informações simultaneamente.

Atualmente utiliza-se um anemômetro manual para se medir a velocidade do vento na área de vôo. É desejável adquirir e acrescentar à estação de terra um sensor capaz de medir e enviar dados de velocidade e direção do vento, para gravação destes dados juntamente com a telemetria do veículo e para automatizar a monitoração das condições metereológicas durante os vôos. Existem anemômetros disponíveis comercialmente, que montados em um mastro a 5 m de altura são capazes de enviar as leituras recebidas já em formato digital e se conectar a um computador por uma interface serial ou porta de joystick (*game port*).

O rádio-modem apresenta as vantagens de ter um alcance elevado (aproximadamente 30 km), ser confiável, facilmente configurável, e ser utilizado transparentemente entre duas portas seriais.

Um rádio-*ethernet*, embora seja uma opção mais cara do que um rádio-modem serial, e apresente um alcance muito menor (poucos quilômetros), possui as vantagens de suportar uma velocidade de transmissão muito maior (1Mb/s, contra 115 Kb/s do rádio-modem), e suportar a pilha de protocolo TCP como padrão. Para algumas placas de rede *ethernet*, existem *drivers* que permitem a utilização da rede *ethernet* diretamente pelas tarefas de tempo real do RT-Linux.

A necessidade ou não de uma maior velocidade de transmissão dependerá da quantidade de sensores colocados no sistema embarcado, e da frequência de envio das suas leituras para a estação de terra. Pode-se subtituir o rádio-modem por um rádio-*ethernet* caso a velocidade de transmissão necessária seja maior do que a oferecida pelo rádio-modem. Considerando que a maioria dos sensores envia dados a uma taxa de 10 Hz, e que a estação embarcada transmite os dados em um formato binário (mais compacto do que a saída ASCII da maioria dos sensores), a velocidade de transmissão do rádio-modem têm sido suficiente até o momento: o pacote de dados transmitido tem cerca de 250 bytes, ou seja, à taxa de dez pacotes por segundo a banda utilizada é aproximadamente 20 Kb/s.

Bibliografia

[1] M. Barabanov. *A Linux-based Real-Time Operating System* . Tese de Mestrado, New Mexico Istitute for Mining and Technology, 1997. http://rtlinux.org/rtlinux/papers/thesis.ps.

[2] *BLT Extension to Tcl/Tk Language.* Web Page, Maio 2001. http://www.tcltk.com/blt.

[3] *Bosch Aerospace Inc. Bosch Aerospace Military Use Airships.* Web Page, 1999. http://www.boschaero.com.

[4] P. Cloutier, P. Montegazza, S. Papacharalambous, I. Soanes, S. Hughes, e K. Yaghmour. "DIAPM-RTAI Position Paper". *I Real Time Operating Systems Workshop and II Real Time Linux Workshop*, Orlando, FL, EUA, Novembro 2000.

[5] A. Elfes, S. S. Bueno, M. Bergerman, e J. J. G. Ramos. "A semi-autonomous robotic airship for environmental monitoring missions". *IEEE International Conference on Robotics and Automation*, pp. 3449–3455, Leuven, Bélgica, Maio 1998.

[6] A. Elfes, S. S. Bueno, M. Bergerman, J. J. G. Ramos, e S. B. V. Gomes. "Project AURORA: Development of an autonomous unmanned remote monitoring robotic airship". *Journal of the Brazilian Computer Society*, 4(3):70–78, Abril 1998.

[7] E. Gamma, R. Helm, R. Johnson, e J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company, Primeira edição, 1994.

[8] S. B. V. Gomes e J. J. G. Ramos. "Airship dinamic modeling for autonomous operation". *IEEE International Conference on Robotics and Automation*, pp. 3462–3467, Leuven, Bélgica, Maio 1998.

[9] T. Hansen, A. Kahn, S. Kannan, R. Peon, e F. Tapia. "Georgia Tech entry for the 1997 International Aerial Robotics Competition". *Association for Unmanned Vehicle Systems, 1997 International Aerial Robotics Competition Papers*, 1997.

[10] *Herley Vega Systems - Company Overview and MAGIC2 Briefing.* Company Presentation, 1999.

[11] J. How. *GPS Formation Flying Blimps.* Web Page, 1998. http://www.mit.edu/people/jhow/ff/blimps/blimps.html.

[12] IEEE. "The Portable Application Standards Committee of the IEEE Computer Society, P1003.13 draft standard for information technology, posix realtime application suport". Relatório técnico, 1998.

[13] H. Jones, E. Frew, B. Woodley, e S. Rock. "Human-robot interaction for field operation of an autonomous helicopter". *Mobile Robots XIII*, Boston, MA, EUA, Novembro 1998.

[14] S. Kannan, J. Hur, G. Saroufiem, e I. Yavrucuk. "Georgia Tech UAV Software Systems". Relatório técnico, UAV Research Facility, School of Aerospace Engineering, Georgia Institute of Technology, Junho 1999. Apresentado como relatório técnico na "1999 International Aerial Robotics Competition".

[15] S. Kannan, J. V. R. Prasad, D. P. Schrage, I. Yavrucuk, L. Wills, e C. Restrepo. "Simulation and Flight Control Integration using the Open Control Platform for Unmanned Aerial Vehicles". *18th AIAA Digital Avionics Conference*, 1999.

[16] S. Lacroix. "Toward autonomous airships: research and development at LAAS/CNRS". *Third International Airship Convention and Exhibition*, Friedrichshafen, Alemanha, Julho 2000.

[17] D. L. Levine, S. Flores-Gaitan, e D. C. Schimidt. "Measuring OS Support for Real-Time CORBA ORBs". *Fourth IEEE International Workshop on Object-oriented Real-Time Dependable Systems (WORDS'99)*, Santa Barbara, CA, EUA, Janeiro 1999.

[18] *Linux Router Project.* Web Page, Maio 2001. http://www.linuxrouter.org/.

[19] D. C. MacKenzie e R. C. Arkin. "Evaluating the usability of robot programming toolsets". *The International Journal of Robotics Research*, 17(4):381–401, Abril 1998.

[20] S. M. Maeta. *Desenvolvimento da Infra-Estrutura Embarcada do Projeto AURORA.* Dissertação de mestrado, Instituto de Computação - UNICAMP, Campinas, SP, Brasil, Julho 2001.

[21] S. M. Maeta, J. J. G. Ramos, L. G. B. Mirisola, S. S. Bueno, e M. Bergerman. "Arquitetura de Hardware do Projeto Aurora". *XIII Congresso Brasileiro de Automática*, pp. 937–942, Florianópolis, SC, Brasil, Setembro 2000.

[22] N. McGuire. "minirtl - A Minimum Realtime Linux System". *I Real-Time Linux WorkShop*, Viena, Áustria, Dezembro 1999.

[23] N. McGuire. "minirlt - Hard Real Time Linux For Embedded Systems". *I Real Time Operating Systems Workshop and II Real Time Linux Workshop*, Orlando, FL, EUA, Novembro 2000.

[24] P. Montegazza. "DIAPM-RTAI for Linux: WHYs, WHATs and HOWs". *I Real Time Linux Workshop*, Viena, Áustria, Dezembro 1999.

[25] E. Mowforth. *Introduction to the Airship*. Número 3 em Airship Association Publication. The Airship Association Ltd, 1991.

[26] J. J. G. Ramos, S. S. Bueno, S. M. Maeta, E. C. Paiva, K. Asanuma, L. G. C. Nascimento, M. Bergerman, A. Elfes, e J. A. R. Beiral. "Project AURORA: autonomous unmanned remote monitoring robotic airship". *2nd International Airship Convention and Exhibition*, pp. 91–103, Bedford, Reino Unido, Junho 1998.

[27] J. J. G. Ramos, E. C. de Paiva, J. R. Azinheira, S. S. Bueno, S. M. Maeta, L. G. B. Mirisola, M. Bergerman, e B. G. Faria. "Autonomous Flight Experiment With a Robotic Unmanned Airship". *IEEE International Conference on Robotics and Automation*, pp. 4152–4157, Seul, Coréia do Sul, Maio 2001.

[28] J. J. G. Ramos, E. C. de Paiva, S. M. Maeta, L. G. B. Mirisola, J. R. Azinheira, B. G. Faria, S. S. Bueno, M. Bergerman, C. S. Pereira, C. T. Fujiwara, J. P. G. Batistela, R. da Rocha Frazzato, R. P. Peixoto, G. C. Martins, e A. Elfes. "Project AURORA: a status report". *3rd International Airship Convention and Exhibition*, Friedrichshafen, Alemanha, Julho 2000.

[29] J. J. G. Ramos, S. M. Maeta, M. Bergerman, S. S. Bueno, A. Bruciapaglia, e L. G. B. Mirisola. "Development of a VRML/Java unmanned airship simulating environment". *International Conference on Intelligent Robots and Systems*, pp. 1354–1359, Kyongju, Coréia do Sul, Outubro 1999.

[30] J. J. G. Ramos, S. M. Maeta, L. G. B. Mirisola, M. Bergerman, S. S. Bueno, G. S. Pavani, e A. Bruciapaglia. "A software environment for an autonomous unmanned airship". *International Conference on Advanced Intelligent Mechatronics*, pp. 1008–1013, Atlanta, GA, EUA, Setembro 1999.

[31] *Simon Fraser University Aerial Robotics Group*. Web Page, 1998. http://www.sfu.ca/~arg.

[32] D. H. Shim. "Complete Guide for Building UAVs". *IEEE CCA UAV Workshop*, Anchorage, AK, EUA, Setembro 2000.

[33] R. Simmons. "Structured control for autonomous robots". *IEEE Transactions on Robotics and Automation*, 10(1), Fevereiro 1994.

[34] A. S. Tanenbaum. *Computer Networks*, capítulo 3. Prentice Hall PTR, Terceira edição, 1996.

[35] E. Torun. "UAV Requirements and Design Consideration". *NATO Research and Technology Organization Meeting Proceddings 44 - Advances in Vehicle Systems Concepts and Integration*, Ancara, Turquia, Abril 1999.

[36] J. White, C. McCrerie, e C. Miles. "An Evaluation of Input Devices and Menu Systems for Remote Workstations". *NATO Research and Technology Organization Meeting Proceddings 44 - Advances in Vehicle Systems Concepts and Integration*, Ancara, Turquia, Abril 1999.

[37] K. Wong, D. Newman, P. Gibbens, D. Auld, S. Wishart, H. Stone, J. Randle, K. Choong, D. Boyle, e P. Blythe. *Maturing UAV Capabilities - Stepping from Technology Demonstrators to Mission-Specific Systems.* Web Page, 1996. http://www.aero.usyd.edu.au/wwwdocs/uav1096.html.

[38] K. C. Wong. *Unmanned Aerial Vehicles (UAVs) - Are They Ready This Time? Are We?* . Apresentado na Sydney Royal Aeronautical Society, 1997. http://www.aero.usyd.edu.au/wwwdocs.

Apêndice A

Instalação do Sistema

A.1 *Software* necessário

Este apêndice apresenta os passos necessários para instalar e configurar todo o *software* necessário na estação de terra e no sistema embarcado. O computador da estação de terra deve ser no mínimo um Celeron ou Pentium II com 64MB de RAM, e o sistema embarcado normalmente é executado em um *hardware* PC-104, correntemente um Pentium 133 MHz, com 32 MB de RAM. O sistema embarcado simulado pode ser executado localmente na estação de terra, ou em outro computador conectado por um cabo serial à estação de terra.

A estação de terra utiliza a distribuição Linux Red Hat 6.2, com *kernel* versão 2.4.0-test1 e RT-Linux versão 3.0. A distribuição Red Hat 7.0 pode apresentar problemas na execução do RT-Linux (maiores detalhes podem ser encontrados nos arquivos da lista de discussão do RT-Linux). Espera-se que nas próximas versões da distribuição os problemas estejam corrigidos.

O sistema embarcado utiliza a distribuição do minirtl2.2, utilizando o *kernel* 2.2.14 e RT-Linux 2.2.

Os seguintes pacotes de *software* devem ser instalados na estação de terra:

1. Sistema operacional Linux, *kernel* 2.4.0-test1, libc 6. Utilizamos a distribuição Red Hat 6.2, porém utilizando outra versão do *kernel*, que deve ser obtida via download. Cada versão do RT-Linux necessita de uma versão específica do *kernel* do Linux;

2. Real Time Linux v 3.0. Disponível em `ftp://ftp.rtlinux.com/pub/rtlinux/v3/`. Basta seguir as instruções de instalação, e executar os exemplos para se certificar que o sistema foi instalado corretamente. Normalmente esta etapa envolve compilar o *kernel* (veja o Apêndice A.2);

3. *rt_com serial port driver* (distribuído juntamente com o RT-Linux, e disponível em `http://rt-com.sourceforge.net/`);

4. Biblioteca GNU pthread (utilizamos a versão 13.0.27, disponível em `http://www.gnu.org/software/pth/pth.html`);

5. Tcl/Tk 8.0, com extensões BLT e Xtcl (distribuído juntamente com o Red Hat Linux). Disponível em `http://www.scriptics.com/` e `http://www.tcltk.com/blt/`;

6. Biblioteca MESA 3.1 (implementação em código aberto da biblioteca gráfica OpenGL para Linux, distribuído juntamente com o Red Hat Linux, e disponível em `http://www.mesa3D.org/`);

7. Máquina Virtual Java 1.2.X ou superior (disponível em `http://java.sun.com/j2se/`);

8. Biblioteca Qt (utilizamos a versão 2.2.2. Disponível em qualquer distribuição Linux que utilize o gerenciador de janelas KDE, inclusive o Red Hat Linux, e disponível em `http://www.trolltech.com/`);

9. LKM da estação de terra.

10. Demais programas componentes da estação de terra. Está disponível na estação de terra um *script* que insere o LKM do item anterior com os parâmetros apropriados e a seguir inicia todos os programas da estação de terra.

Se, durante a instalação da distribuição Linux os os itens 5, 6 e 8 não foram instalados, basta instalar os respectivos arquivos RPM, contidos no CD da distribuição do Linux. Estes itens também estão disponíveis livremente na Internet.

Os seguintes pacotes de *software* devem ser instalados no sistema embarcado:

1. minirtl2.2 (disponível para download em `http://www.thinkingnerds.com/projects/minirtl/minirtl.html`). É possível utilizar o kernel contido na distribuição do minirtl, não sendo necessário recompilá-lo;

2. LKM específico para a placa de rede utilizada, dentre aquelas suportadas pelo Linux (alguns são distribuídos juntamente com o minirtl). O módulo deve ser copiado em `/usr/lib/modules`, e deve-se editar o arquivo `/dev/modules` (Figura A.1) instruindo o sistema para carregar o módulo na inicialização do sistema, com os parâmetros corretos, que dependem da placa de rede específica (consulte a documentação da

placa ou do LKM correspondente). É necessário também editar os arquivos de configuração dos serviços de rede para habilitar a utilização da rede, especialmente os arquivos `/etc/network`, `/etc/network.conf`, `/etc/resolv.conf`;

3. *rt_com serial port driver*: o apêndice A.2 mostra como instalá-lo;
4. cliente NFS (*Network File System*). Supondo que o suporte para clientes de NFS está habilitado no *kernel* (veja Apêndice A.2), como é o caso do minirtl, então a instalação do NFS consiste em:
   (a) editar `/etc/exports` no servidor NFS (um outro computador Linux no laboratório), permitindo que um cliente NFS monte um diretório do servidor que será utilizado para transferência de arquivos entre os dois computadores;
   (b) editar `/etc/fstab` no cliente (a CPU embarcada), instruindo o sistema embarcado a montar o diretório exportado pelo servidor. A documentação do Linux (man pages e NFS HOW-TO) mostra como editar estes arquivos;
   (c) copiar para `/usr/sbin` no sistema embarcado os arquivos `portmap`, `rpc.mountd` e `rpc.nfsd`, e executá-los nesta ordem. Isto pode ser inserido nos *scripts* de inicialização do sistema operacional para habilitar o NFS sempre que o sistema for inicializado.
5. Inserir o LKM do sistema embarcado, editando o arquivo `/etc/modules`. O arquivo `/etc/modules` utilizado no projeto AURORA é apresentado na Figura A.1 como exemplo (as linhas iniciando com # são comentários).

A.2 Configurando o *kernel* e o *rt_com*

Durante a compilação do *kernel* é necessário habilitar as seguintes opções:

- RT-Linux (opção CONFIG_RTLINUX do arquivo .config do *kernel*). Dependendo da versão do RT-Linux utilizada, pode existir a opção de habilitar ou não a utilização do RT-Linux no *kernel* gerado.
- Suporte a LKMs (opção CONFIG_MODULES). Absolutamente necessário.
- *Loopback device* (opção CONFIG_BLK_DEV_LOOP). É recomendável habilitar esta opção na estação de terra. Permite que se monte um arquivo imagem (o disquete do minirtl pode ser copiado para o disco rígido como um arquivo imagem do disco flexível), facilitando a tarefa de modificar o conteúdo do disquete do sistema embarcado.

```
# lista de LKMs a serem inseridos no kernel
# durante a inicialização do sistema

# driver da placa de rede, com os parâmetros apropriados
cs89x0 io=0x300 irq=10
# LKMs do RT-Linux
rtl_time
rtl_sched
rtl_posixio
rtl_fifo
# rt_com serial driver
rt_com
# LKM do sistema embarcado
rt_serial
```

Figura A.1: Um exemplo de arquivo `/dev/modules`.

- *Driver* serial padrão. Deve ser modular (opção CONFIG_SERIAL=m). Dessa forma, retirando-se o LKM serial.o do *kernel* pode-se deixar todas as portas livres para o *rt_com*, e inserindo-o no *kernel*, pode-se utilizar as seriais normalmente se necessário. A partir do *kernel* 2.2, pode-se desabilitar cada porta individualmente, mas mesmo assim instalar o *driver* serial como um LKM deve facilitar a configuração do sistema.

- Suporte a clientes de NFS (Network File System) (opção CONFIG_NFS_FS). Deve-se habilitar esta opção no sistema embarcado, para permitir que o sistema embarcado monte um drive remotamente via rede, facilitando a manutenção do sistema. Usualmente o minirtl já apresenta esta opção habilitada.

O código fonte do *driver rt_com* é distribuído juntamente com o RT-Linux, porém o usuário necessita editar o código, indicando quais portas seriais devem ser controladas pelo *driver rt_com* e configurando-as. Caso necessário, o *driver* serial padrão do Linux pode ainda ser utilizado em portas seriais que não estiverem sendo controladas pelo *rt_com*. Neste caso deve-se utilizar o comando `setserial` para desabilitar as portas a serem utilizadas pelo *rt_com*, para que o *driver* serial libere os recursos utilizados pela porta (IRQ e endereço de I/O), deixando-os livres para o *rt_com*.

A seguir serão apresentadas as modificações necessárias no código fonte do *rt_com*, pois isto não está detalhado no manual do *rt_com*. Deve-se modificar a tabela `rt_com_table` no arquivo `rt_com.c` de forma que para cada porta serial a ser controlada pelo *rt_com* a tabela contenha uma entrada na forma:

```
{ <ÍNDICE>, BASE_BAUD, <I/O ADDRESS>,<IRQ>, STD_COM_FLAG, <ISR>},
```

onde <ÍNDICE> não é utilizado, porém recomenda-se que se mantenha uma numeração sequêncial de acordo com o número do item na tabela; <I/O ADDRESS> e <IRQ> são o endereço de I/O e o número da interrupção utilizadas pela porta serial; e a <ISR> (*Interrupt Service Routine*) é a função de tratamento da interrupção. A documentação do *rt_com* fornece informações sobre os outros campos.

A seguir deve-se inserir as ISRs necessárias no código fonte e inserir as suas declarações no arquivo de cabeçalho (`rt_comP.h`). Na verdade as ISR das portas apenas chamam uma outra função de tratamento passando como parâmetro o numero da porta serial específica. O código já possui um exemplo de declaração da ISR para uma porta serial que deve ser copiado e modificado.

Também deve-se configurar no arquivo `rt_com.h` o tamanho dos *buffers* utilizados para cada porta serial, de forma a conter com segurança o tamanho máximo de pacote a ser transmitido.

A seguir deve-se recompilar o *rt_com* utilizando-se o *makefile* fornecido pela distribuição para se obter o LKM a ser inserido no *kernel*, configurado para controlar as portas seriais desejadas.

Apêndice B

Aspectos Operacionais da Estação de Terra

Esta seção fornece algumas recomendações operacionais para o leitor que desejar desenvolver um projeto semelhante ao aqui descrito. Projetos de veículos aéreos, bem como muitos projetos de robótica de campo, costumam exigir uma considerável carga de trabalho e tempo dos pesquisadores para efetuar a manutenção do sistema, transportá-lo até a área de operação, e operá-lo em uma situação real. Recomenda-se fortemente que qualquer interessado no Projeto AURORA leia também a tese de mestrado de Silvio Mano Maeta [20], que descreve o sistema embarcado, inclusive diversos aspectos operacionais em relação ao sistema embarcado e ao dirigível em si.

- Para realizar um vôo, é necessário um operador para a estação de terra, um piloto para operar a RCU, e uma equipe de suporte de terra, para executar as manobras de decolagem e aterrisagem do veículo, de pelo menos duas pessoas. Durante o desenvolvimento dos algoritmos de controle ou da montagem mecânica do veículo, é aconselhável também filmar todos os vôos, o que requer mais uma pessoa.

- A equipe de suporte de terra deve ser instruída a prosseguir com a decolagem somente depois de autorizada pelo operador da estação de terra. Este deve verificar o funcionamento de todos os sistemas antes de autorizar a decolagem. Por exemplo, receptores GPS costumam necessitar de alguns minutos depois de ligados antes de começarem a transmitir dados.

- É bastante recomendável a utilização de *walkie-talkies* para comunicação entre os membros da equipe durante o vôo. O operador da estação de terra, o piloto, e pelo menos alguns outros membros da equipe de suporte de terra podem ter que permanecer a uma distância considerável durante os vôos, impossibilitando o contato

através da voz. No caso do operador da estação de terra, isto é particularmente importante porque ele está monitorando o veículo. Caso ocorra algum problema, ele deve avisar o piloto ou a equipe de suporte de terra para abortar uma decolagem ou um vôo. Existem no mercado *walk-talkies* utilizando fone de ouvido com microfone acoplado, que permitem manter as duas mãos livres enquanto se fala. Isto é útil para o piloto, que tem que operar a RCU.

- Tanto o rádio-modem quanto os *walkie-talkies* podem necessitar de autorização das autoridades competentes para serem utilizados. No Brasil, a ANATEL (Agência Nacional de Telecomunicações) emite certificados autorizando a utilização de transmissores e receptores de rádio-frequência. O usuário dos rádios deve portar os documentos durante a sua utilização.

- Para se realizar vôos com qualquer veículo aéreo, é necessário autorização específica ao tipo de veículo, altitude e local dos vôos, emitida pelas autoridades de proteção ao vôo (DAC, Departamento de Aviação Civil). Deve ser expedida uma NOTAM (*NOTice to AirMen*), um documento autorizando a realização dos vôos, e é necessário comunicar-se com o aeroporto mais próximo antes da realização de qualquer vôo. Este documento tem validade determinada, e deve ser renovado periodicamente.

- Para a expedição de uma autorização de vôo, as autoridades requerem a existência de um seguro cobrindo perdas materiais e humanas causadas por eventuais quedas do veículo aéreo (seguro contra bens e pessoas ao solo - código R.E.T.A. 3 e 4).

- Também é recomendável se adquirir um anemômetro para avaliação das condições metereológicas no momento dos vôos. Deve-se respeitar um limite máximo para a velocidade do vento para se realizar um vôo, cujo valor depende das características do veículo (tamanho, motorização, atuadores). Estão disponíveis comercialmente diversos tipos, como anemômetros manuais (formato de um multímetro digital, com um mostrador de cristal líquido mostrando a velocidade do vento medida em uma ventoinha) ou anemômetros que são montados em um mastro e enviam dados para um computador.

- Deve-se armazenar o equipamento de forma a facilitar o transporte para a área de operação: deve-se manter todo o equipamento necessário (computador, antenas, rádio-modem e *walk-talkies*, cabos, DGPS, baterias, anemômetro, RCU, mesa e cadeira, etc.) acondicionado no mesmo lugar e em poucos volumes. A Figura 2.1 mostra uma caixa de plástico utilizada para armazenar o DGPS juntamente com a sua bateria, antena, fios e um rádio-modem. Para colocar em funcionamento a estação DGPS basta levar esta caixa para o campo, numa posição que permita boa recepção dos satélites, e conectar a bateria de alimentação.

- Deve-se conferir as leituras de GPS obtidas em um mapa de boa resolução da área, para garantir que o receptor GPS e a estação de DGPS estão configurados adequadamente. Por exemplo, algumas estações DGPS necessitam ser colocadas em coordenadas fixas e configuradas previamente. Configurações erradas como, por exemplo, utilizar um modelo da geóide da Terra menos preciso, podem levar a erros consideráveis nas leituras de posição.

Apêndice C

Publicações do autor

Ramos, J.J.G; Maeta, S. M.; Mirisola, L.G.B.; Bergerman, M.; Bueno, S.S; Pavani, G.S. Bruciapaglia, A. "A Software Environment for an Autonomous Unmanned Airship". em *International Conference on Advanced Intelligent Mechatronics*, pp. 1008-1013, Atlanta, GA, EUA, Setembro, 1999.

Ramos, J.G; Maeta, S.M.; Bergerman, M.; Bueno, S.S; Bruciapaglia, A.; Mirisola, L.G.B. "Development of a VRML/Java Unmanned Airship Simulating Environment". em *International Conference on Intelligent Robots and Systems*, pp. 1354 - 1359, Kyongju, Coréia do Sul, Outubro, 1999.

Ramos, J.J.G. et al. "Project AURORA: a status report." *3rd International Airship Convention and Exhibition*, Friedrichshafen, Alemanha, Julho, 2000.

Mirisola, L.G.B.; Ramos, J.J.G.; Maeta, S.S.; Bergerman, M.; Bueno, S.S. "Um sistema de comunicação para um dirigível robótico não-tripulado". *XIII Congresso Brasileiro de Automática*, pp. 943-948, Florianópolis, SC, Brasil, Setembro, 2000.

Maeta, S.S.; Ramos, J.J.G.; Mirisola, L.G.B.; Bueno, S.S.; Bergerman, M. "Arquitetura de Hardware do Projeto AURORA". *XIII Congresso Brasileiro de Automática*, pp. 937-942, Florianópolis, SC, Brasil, Setembro, 2000.

Elfes, A.; Bueno, S.S.; Bergerman, M.; Ramos, J.J.G.; Maeta, S.M.;Mirisola, L.G.B.; Paiva, E.C.; Faria, B.G. "Tropical forest aerial inspection with an autonomous airship". In "The 21st Century: The Status Quo in 9 Important Regions of the Earth.", *video ed. FAW, EXPO 2000 Hannover - Millenium World's Fair Theme Park.* Theme: "The 21st Century". Hannover, Alemanha, de Junho a Outubro de 2000.

Bueno, S.S.; Ramos, J.J.G.; Bergerman, M.; Paiva, E.C.; Azinheira, J.R.; Maeta, S.M.; Mirisola, L.G.B.; Faria, B.G. "Um dirigível não tripulado para inspeção aérea robotizada". *Revista Robótica*, nº 39, pp. 8-10, Porto, Portugal, segundo trimestre de 2000.

Ramos, J.J.G.; Paiva, E.C.; Azinheira, J.R.; Bueno, S.S.; Maeta, S.M.; Mirisola, L.G.B.; Bergerman, M.; Faria, B.G. "Autonomous flight experiment with a robotic unmanned airship". *IEEE International Conference on Robotics and Automation*, pp. 4152-4157, Seul, Coréia do Sul, Maio 2001.