DISSERTAÇÃO DE MESTRADO Nº 720

ARQUITETURA DE HARDWARE E SOFTWARE PARA SUPERVISÃO E CONTROLE DE UM CARRO AUTÔNOMO

Tiago Amadeu Arruda

DATA DA DEFESA: 13/02/2012

Laboratório de Sistemas Computacionais e Robótica
Departamento de Engenharia Elétrica
Universidade Federal de Minas Gerais
Av. Antônio Carlos 6627, 31270-901 Belo Horizonte, MG Brasil
Fone: +55 31 3409-3470

PDVA
GRUPO DE PESQUISA E DESENVOLVIMENTO DE VEÍCULOS AUTÔNOMOS

Arquitetura de Hardware e Software para Supervisão e Controle de um Carro Autônomo

Tiago Amadeu Arruda

Orientador: Prof. Dr. Guilherme Augusto Silva Pereira

Belo Horizonte, 13 de Fevereiro de 2012

Universidade Federal de Minas Gerais

Escola de Engenharia

Programa de Pós-Graduação em Engenharia Elétrica

ARQUITETURA DE HARDWARE E SOFTWARE PARA SUPERVISÃO E CONTROLE DE UM CARRO AUTÔNOMO

Tiago Amadeu Arruda

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Prof. Guilherme Augusto Silva Pereira

Belo Horizonte - MG

Fevereiro de 2012

"Arquitetura de Hardware e Software para Supervisão e Controle de um Carro Autônomo"

Tiago Amadeu Arruda

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Escola de Engenharia da Universidade Federal de Minas Gerais, como requisito para obtenção do grau de Mestre em Engenharia Elétrica.

Aprovada em 13 de fevereiro de 2012.

Por:

Prof. Dr. Guilherme Augusto Silva Pereira
DEE (UFMG) - Orientador

Leonardo A. B. Tôrres

Prof. Dr. Leonardo Antônio Borges Tôrres
DELT (UFMG)

Prof. Dr. Luiz Chaimowicz
DCC (UFMG)

Agradecimentos

Agradeço a Deus pela oportunidade de fazer um Mestrado, pela vida, saúde e paciência que foram constantemente renovadas por Ele para que eu pudesse completar este empreendimento.

Agradeço aos meus pais, Otávio e Aparecida, e aos meus irmãos Gustavo e Aline por me incentivar e me apoiar na execução do mestrado, sempre prontos a oferecer ajuda independente das circunstâncias.

Agradeço aos amigos e colegas de laboratório, Jullierme, Rogério, Érica, Marco Túlio Marlon, Plínio, Adriano, Luíz Guilherme, Lucas, Natália, Elias, Danilo e tantos outros com quem de uma forma ou outra, convivi nestes anos e que, sem dúvida, me ajudaram e torceram para que eu chegasse até aqui. Agradeço também ao Dimas que cedeu o código necessário para comunicação em tempo real com a EPOS e ao Víctor Costa que me auxiliou na criação dos Drivers para o Player.

Ofereço um agradecimento especial ao meu orientador professor Guilherme Pereira, que me orientou de forma magnífica, sempre pronto a me auxiliar e instruir, e que mesmo diante das situações mais adversas me deu um voto de confiança, detalhe fundamental para que eu pudesse chegar ao fim deste trabalho. Agradeço aos demais professores com quem convivi pela disposição na transmissão do conhecimento e que me forneceram a base para crescer no campo acadêmico, profissional e pessoal.

Agradeço o apoio financeiro e o suporte do CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) por meio da bolsa de mestrado, da FAPEMIG (Fundação de Amparo à Pesquisa do Estado de Minas Gerais) pelo fornecimento dos materiais e recursos utilizados no projeto, e o apoio da CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) que financiou a viagem e as despesas para apresentação de um artigo.

Agradeço a GE, que flexibilizou o horário para que eu pudesse cumprir com as tarefas da atividade acadêmica.

"Se dá para imaginar, dá para fazer"
General Electric.

Resumo

Veículos autônomos são aqueles veículos capazes de realizar tarefas ou missões, sejam elas em terra, no ar ou na água, sem a intervenção de operadores humanos. Estes veículos possuem diversas aplicações, que incluem exploração espacial, mapeamentos geográficos ou ainda a condução de passageiros e cargas.

As pesquisas com veículos autônomos têm estado em foco nos últimos anos, principalmente devido aos avanços das tecnologias utilizadas nos mesmos. No entanto, dada a diversidade dos sistemas, cada um deles acaba por fornecer uma solução de hardware e software muito particular, inviabilizando por muitas vezes a reutilização da arquitetura em sistemas que sejam diferentes do original.

Nesta linha, esta dissertação aborda o projeto, desenvolvimento e validação de uma arquitetura hierárquica de hardware e software para controle e supervisão de um carro autônomo com características genéricas que permitam a extensão da mesma para outros tipos de veículos autônomos terrestres. A arquitetura proposta possui três níveis denominados Nível de Controle, Nível de Processamento e Nível Usuário. Esta foi baseada no estudo de arquiteturas desenvolvidas em vários trabalhos disponíveis na literatura, visando a obtenção de uma solução de compromisso que, de um lado ofereça um sistema capaz de atender as demandas exigidas por um carro autônomo, e por outro, que ofereça um modelo de fácil acesso para envio de comandos e recepção de informações por um usuário final.

A solução encontrada foi implementada no carro autônomo da UFMG. A validação se deu por testes realizados nos três níveis por meio de um controle de velocidade, localização do veículo e envio de comandos remotos ao veículo. Os resultados mostraram a robustez, estabilidade e viabilidade da solução proposta, levando a conclusão de que a mesma atende de forma satisfatória os requisitos exigidos para a implementação em um veículo autônomo e ao mesmo tempo fornece uma interface amigável.

Abstract

Autonomous vehicles are vehicles able to perform tasks or missions, on land, in the air or water, with no human intervention. These vehicles have numerous applications such as outer space exploration, geographic mapping or even passenger and freight transportation.

Researches with autonomous vehicles have been in focus for the last years, mainly due to the technological advances in this area. However, because of the variety of systems, each one brings a very particular solution of hardware and software, making the application in different systems most of times impracticable.

With this background, this dissertation covers the project, the development and the validation of a hierarchical architecture of hardware and software for control and supervision of an autonomous car with generic characteristics aiming the application in various terrestrial autonomous vehicles. The proposed architecture has three levels named Control Level, Processing Level and User Level. This proposal was developed based on the study of architectures from several studies available in the literature. The solution targets a compromise between fulfilling all the demands of an autonomous car and offering an easy access model for the transmission of commands and reception of information by an end user.

The chosen solution was implemented in the autonomous car of the UFMG. The validation was performed with tests in the three levels by using a speed control, a vehicle localization system and remote transmission of commands to the vehicle. The results testify the robustness, stability and feasibility of the proposed solution, concluding that it complies satisfactorily with the demanded requirements for implementation in an autonomous car, providing yet a user friendly interface

Sumário

Lista de Figuras

Lista de Siglas

API Interface de Programação de Aplicativos - *Application Programming Interface*

ASCII - *American Standard Code for Information Interchange*

ATA Aquisição Automática de Alvo - *Automatic Target Acquisition*

AUV Veículo Autônomo Subaquático - *Autonomous Underwater Vehicles*

CADU Carro Autônomo Da UFMG

CAN Rede de Controle de Área - *Controller Area Network*

CaRINA Carro Robótico Inteligente para Navegação Autônoma

CRC Checagem de Redundância Cíclica - *Cyclic redundancy check*

DARPA - *Defense Advanced Research Projects Agency*

ECU Unidade de Controle Eletrônica - *Electronic Control Unit*

GPS Sistema de Posicionamento Global - *Global Positioning System*

HD Disco Rígido - *Hard Disk*

IMU Unidade de Medição Inercial - *Inertial Measurement Unit*

INS - *Inertial Navigation System*

LIDAR Detecção e Medição por Luz - *Light Detection and Ranging*

MIT - *Massachusetts Institute of Technology*

OCU Unidade de Controle do Operador - *Operator Control Unit*

PC Personal Computer - *Computador Pessoal*

PDVA Grupo de Pesquisa e Desenvolvimento de Veículos Autônomos

RAM Memória de Acesso Aleatório - *Random Access Memory*

RCS Sistema de Controle de Tempo Real - *Real-Time Control System*

RCV Veículo Controlado Remotamente - *Remote Controlled Vehicle*

ROS Sistema Operacional de Robô - *Robot Operating System*

RTAI - *Real Time Application Interface*

SBAI Simpósio Brasileiro de Automação Inteligente

SHF Super Alta Frequência - *Super High Frequency*

SSD Disco de Estado Sólido - *Solid State Disk*

STESCA - *Strategic-Tactical-Execution Software Control Architecture*

SUAV Pequeno Veículo Autônomo Aéreo - *Small Unmanned Aerial Vehicle*

TTL Lógica Transistor-Transistor - *Transistor-transistor Logic*

UAV Veículo Autônomo Aéreo - *Unmanned Aerial Vehicle*

UFMG Universidade Federal de Minas Gerais

UGV Veículo Autônomo Terrestre - *Unmanned Ground Vehicle*

USB Barramento Serial Universal - *Universal Serial Bus*

USP Universidade de São Paulo

UTC Coordenada Universal de Hora - *Universal Coordinated Time*

UUV Veículo Autônomo Subaquático - *Unmanned Underwater Vehicle*

1 Introdução

Robótica, segundo o dicionário Aurélio [Aurélio, 2011], é a ciência e técnica de concepção e construção de robôs. Os robôs, por sua vez, são máquinas capazes de realizar tarefas de forma autônoma ou não, e estão distribuídos em duas grandes classes denominadas robôs manipuladores, como os utilizados nas indústria em geral (vide Figura 1.1), ou robôs móveis, para os quais o objeto de estudo deste trabalho se enquadra.

Figura 1.1: Robô Smart Six da Comau [Comau, 2011].

Os robôs móveis compreendem aqueles que podem deslocar todos os seus pontos materiais em seu espaço de trabalho ao longo do tempo. Nesta categoria tem-se vários tipos como, por exemplo, humanóides (Figura 1.2a), aqueles que mimetizam movimentos de animais, como o Big Dog mostrado na Figura 1.2b e outros que se utilizam de asas, hélices ou rodas para se locomover.

Os veículos autônomos são robôs móveis capazes de executar tarefas sem a intervenção humana, sendo utilizados para transporte de passageiros, transporte de cargas, ou, até mesmo, exploração em ambientes que não podem ser facilmente alcançado por seres huma-

Figura 1.2: (a) Robô móvel humanóide [Xie et al., 2009]; (b) Robô móvel mimetizando um cão [Dynamics, 2011].

nos, como outros planetas e águas profundas. Nesta mesma classe ainda se encontram os veículos semi-autônomos e os carros inteligentes, que não são capazes de realizar missões completamente controladas por computador, mas que possuem sistemas de auxílio ao motorista como assistência para estacionamento, mudança de faixa e até mesmo controle de cruzeiro que permite ao veículo manter velocidade constante e distância segura de outros veículos [Jones, 2001].

Para o correto funcionamento de um veículo autônomo, seja ele de qualquer natureza, é necessária a utilização de sensores e atuadores. Os sensores, tais como medidor de velocidade, aceleração, atitude e detecção de obstáculos, são os dispositivos por meio dos quais a informação do espaço de trabalho e do próprio veículo é captada e enviada para um computador. Os atuadores são os elementos pelos quais a central de processamento, ou computador, realiza ações no mundo real. São exemplos de atuadores um motor, um relé ou até mesmo uma sirene. A Figura 1.3 mostra um veículo autônomo terrestre e alguns de seus diversos sensores.

Cada sensor ou atuador precisa se comunicar com uma central de processamento, de modo a repassar ou receber uma informação. Em alguns casos, várias centrais de processamento também precisam se comunicar entre si para executar as diversas tarefas necessárias ao controle do veículo. Ao conjunto de sensores, atuadores e centrais de processamento, juntamente com todas as funções e interfaces necessárias para o seu correto funciona-

Figura 1.3: Carro Autônomo do MIT referenciado em [Fletcher et al., 2008].

mento dá-se o nome de arquitetura e, devido a grande diversidade de veículos autônomos existentes, também é grande a variedade das arquiteturas presentes nos mesmos, sendo a maioria delas concebidas para um veículo em particular.

1.1 Motivação

A principal motivação para a realização deste trabalho foi necessidade de atualização da arquitetura presente no CADU (Carro Autônomo da UFMG), mostrado na Figura 1.4. Este veículo já possuía um sistema completo de controle que, apesar de funcional, estava sujeito a uma série de limitações, principalmente devido à comunicação por meio de portas USB (*Universal Serial Bus*), à utilização do Sistema Operacional Microsoft, e à complexidade existente tanto para adicionar, quanto para modificar os programas existentes. Além disso não era possível ter boa precisão temporal devido ao não determinismo imposto pelo sistema operacional, exigindo o uso de leis de controle menos elaboradas.

Uma segunda motivação para a realização deste trabalho é o preenchimento da lacuna causada pela ausência de uma arquitetura padronizada e facilmente adaptável para um carros autônomo qualquer. As arquiteturas carros autônomos encontradas na literatura tendem a minimizar o universo onde as mesmas possam ser aplicadas devido as especificidades envolvidas que por muita das vezes inserem o uso de elementos nativos dos veículos, inviabilizando a reutilização da solução em outros sistemas.

Na etapa de desenvolvimento de um veículo autônomo algumas arquiteturas possuem a fragilidade de não permitir alterações isoladas no sistema. Ou seja, quando um desen-

Figura 1.4: Carro Autônomo da UFMG.

volvedor altera algum subsistema do veículo, esta mudança frequentemente se propaga de forma a exigir de alterações até mesmo na interface de aplicação.

Em universidades e centros de pesquisa, o desenvolvedor da aplicação, muitas vezes um aluno ou pesquisador temporário, já pode ter deixado o projeto, o que causa muito trabalho para o restante da equipe na adequação do hardware e do software, levando até mesmo a inutilização deles. Com uma arquitetura hierárquica, constituída de vários níveis de hardware e software, quando um nível da hierarquia é alterado, geralmente poucas alterações são feitas somente no primeiro nível logo acima, sendo que níveis superiores a este e níveis inferiores ao alterado não precisam sofrer modificações.

Um problema enfrentado por arquiteturas mal projetadas é a complexidade exigida para operação pelo usuário final que necessita conhecer por completo o sistema de forma a utilizá-lo adequadamente. Este tipo de exigência faz com que o número de pessoas capacitadas para a operação de um veículo autônomo seja reduzido, limitando assim o uso do sistema.

Assim, nesta dissertação propõe-se uma arquitetura que possa ser utilizada em carros autônomos e semi-autônomos de forma a tornar o seu desenvolvimento e utilização simplificados.

1.2 Objetivos e Contribuições

Pretendeu-se que este trabalho servisse de base para diversos projetos envolvendo a utilização de carros autônomos ou semi-autônomos na realização das mais diversas tarefas.

Este trabalho visou cumprir os objetivos listados abaixo:

1. Desenvolvimento de uma arquitetura para instrumentação eletrônica de um veículo que seja robusta, determinística, e que, principalmente, seja de fácil operação, manutenção e expansão.

2. Desenvolvimento de uma arquitetura de software flexível e multi-plataforma para interação com o hardware do veículo desenvolvido no item 1.

3. Instanciação do hardware e software desenvolvido no carro autônomo da UFMG.

O presente trabalho propõe uma arquitetura para ser aplicada a carros autônomos que permita o seu uso em vários tipos diferentes carros, sendo necessária pouca ou nenhuma integração com os sistemas pré-existentes no veículo. Esta arquitetura é dividida em três níveis para permitir que, nível a nível, a abstração das informações se torne cada vez maior possibilitando atingir aos dois primeiros objetivos propostos na seção anterior.

No primeiro nível ou Nível de Controle, onde as informações são as de mais baixo nível, com a mais alta frequência, e disponibilizadas em tempo real, a arquitetura propõe a coleta de dados e envio dos comandos considerando-se que todas as informações dos sensores e atuadores são obtidas por ligação direta aos sinais do veículo e comandos, ou seja, sem passar pela central de dados e processamento nativa do veículo. Com esta abordagem a necessidade de se conhecer o protocolo de fábrica do veículo é eliminada tornando os conceitos desenvolvidos neste trabalho aplicáveis a qualquer sistema veicular.

No segundo nível ou Nível de Processamento se encontram os elementos necessários para gerar informação de navegação como os sistemas de localização, câmeras e sensores *laser* que, apesar da sua importância não necessitam de informações com uma frequência tão alta quanto a do Nível Inferior.

O nível superior ou Nível Usuário da arquitetura hierárquica foi projetado para fornecer ao usuário um acesso aos dados e comandos do veículo de uma forma intuitiva e muito semelhante ao controle utilizado em robôs móveis comerciais. Para este objetivo, é utilizada a interface Player [Gerkey et al., 2003] que permite, além da facilidade de programação por parte do usuário, a possibilidade de simulação do veículo caso o modelo do

mesmo seja construído.

A arquitetura proposta foi implementada e testada no CADU. Isto permitiu a criação de um sistema no qual tanto a inserção quanto a modificação de qualquer um dos sistemas existentes no carro seja feita de maneira fácil e intuitiva. Juntamente com a arquitetura, foram desenvolvidos uma série de manuais e especificações que permitem aos usuários o aprendizado do sistema sem a necessidade de explicações adicionais por meio do idealizador do sistema.

1.3 Organização da Dissertação

O restante desta dissertação está organizada da seguinte forma: no próximo capítulo é feita uma revisão da literatura com os conceitos envolvidos no trabalho e os trabalhos nos quais esta dissertação se inspirou. No Capítulo 3 são introduzidos os requisitos desejados para uma arquitetura de veículo autônomo e no capítulo posterior a este são colocadas as soluções adotadas para alcançar os resultados. Este capítulo também apresenta alguns experimentos realizados no CADU. O Capítulo 5 apresenta as conclusões e propostas para trabalhos futuros.

2 Revisão da Literatura

2.1 Veículos Autônomos

Os Veículos Autônomos vêm sendo amplamente utilizados tanto pelo setor civil quanto militar. Estes veículos são classificados basicamente em três categorias sendo elas UGV *(Unmanned Ground Vehicle)* ou veículos autônomos terrestres, UAV *(Unmanned Aerial Vehicle)* ou veículos autônomos aéreos e UUV *(Unmanned Underwater Vehicle)* ou veículos autônomos subaquáticos. Alguns sistemas são semi-autônomos, exigindo alguma intervenção humana para seu correto funcionamento [Long et al., 2007]. As aplicações destes veículos variam desde tarefas onde a presença de seres humanos é inviável até tarefas que visam segurança e conforto ao usuário. Cabe ressaltar que as siglas UGV, UAV e UUV são também utilizadas para referência a veículos não tripulados tele-operados, no entanto, ao longo do texto o uso das mesmas se refere unicamente aos veículos autônomos.

Os UGV's compreendem os mais variados tipos de veículos terrestres como carros, trens e caminhões. No entanto, grande parte das pesquisas publicadas envolvem carros de passeio que são, por meio de alterações em seu sistema, convertidos em veículos autônomos. As Figuras 2.1 (a) e (b), mostram, respectivamente, um veículo autônomo desenvolvido pelo Google [Teck, 2011] e o veículo CaRINA (Carro Robótico Inteligente para Navegação Autônoma) baseado em carro elétrico desenvolvido pela USP (Universidade de São Paulo) [Fernandes, 2011]. Uma das explicações para a pesquisa neste seguimento em particular é mostrada em [Kolski et al., 2006], onde os autores introduzem o seu artigo apontando os altos índices de acidentes de automóvel na Europa e ressaltam que 95% deles são devidos à falha humana. Assim, o uso sistemas automáticos embarcados têm de sido grande valia na redução do número de acidentes pela diminuição da responsabilidade do operador na condução de um veículo. Exemplos destes sistemas são freios ABS, acionamento *Drive-by-Wire* e Sistema de Controle de Tração que, juntamente com o avanço tecnológico dos últimos anos, levaram aos estudos e desenvolvimento dos veículos terrestres autônomos.

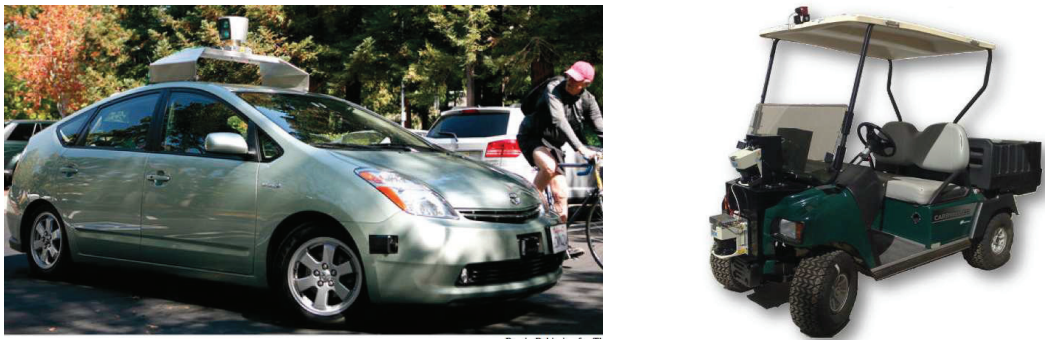

Figura 2.1: (a) Carro Autônomo do Google [Teck, 2011]. (b) Carro Autônomo da USP [Fernandes, 2011].

Nos Estados Unidos, em 2003, foi realizada a primeira grande competição para veículos autônomos. Este desafio, lançado pela Agência de Pesquisa e Projetos Avançados para Defesa (DARPA), consiste em desenvolver automóveis autônomos capazes de navegar pelo deserto em trilhas e estradas em alta velocidade, e percorrer um caminho pré-definido sem a intervenção humana [Urmson et al., 2008]. A mesma competição ocorreu novamente por duas vezes em 2005 e 2007 sendo que em 2007 sofreu uma alteração importante quando o desafio, ao invés de ocorrer em um deserto como na primeira competição, se deu em meio ao trânsito urbano de uma cidade.

Outra classe de veículos autônomos largamente pesquisada é a de UAV's, ou mais especificamente SUAV's*(Small Unmanned Aerial Vehicles)* - veículos autônomos aéreos pequenos - normalmente compreendidos entre 15 cm até algumas unidades de metros de envergadura [Fu et al., 2008]. Os SUAV's são uma subclasse dos UAV's, e a preferência por sua utilização se deve principalmente pela portabilidade, baixo custo e existência de aplicações onde o uso de aeronaves grandes não é adequado, tais como:

- Coleta de Dados como na aplicação mostrada em [Chiabrando et al., 2011] onde o UAV Pelicano foi equipado com uma câmera digital empregado para tirar fotos de sítios arqueológicos na Itália.
- Mapeamento e Monitoramento como em [Goktogan et al., 2010] que utiliza UAV's para localizar ervas daninhas em plantações aquáticas.
- Segurança como em [Maza et al., 2011] onde se propõe a utilização de UAV's em gerenciamento de desastres para procurar vítimas em áreas potencialmente perigosas neste tipo de ocorrência.

O uso de SUAV's ainda facilita a execução de missões que exijam múltiplos veículos

para a realização de uma tarefa, como no trabalho de [Semsch et al., 2009] em que se faz o monitoramento e vigilância em tempo real de um ambiente urbano complexo utilizando-se coordenação de vários SUAV's, evitando assim o problema de oclusão comumente encontrado neste tipo de ambiente. Ou no trabalho de [Merino et al., 2006] em que se utiliza de uma estratégia de múltiplos UAV's para detecção automática de incêndios em florestas.

Os UUV's ou AUV's *(Autonomous Underwater Vehicles)* tem aplicações tais como exploração de petróleo, verificação de tubulações e fibras óticas marinhas, operações navais e localização de minas marítimas [Xie et al., 2010]. Também em pesquisas, este tipo de robô pode ser largamente aplicado como na exploração e mapeamento oceânico ou mesmo em pesquisa em áreas inexploradas do Ártico, onde é necessário navegar por debaixo do gelo [Xiao-xu et al., 2010]. Em [Incze, 2009] os autores utilizam UUV's comerciais para a exploração de águas costeiras, enquanto em [McDowell et al., 2002] é explorada uma estratégia para movimentação de multi UUV's baseado em sistemas biológicos.

Outras aplicações nas quais os veículos autônomos são de grande utilidade é na exploração espaciais. Neste tipo de missão é inviável o controle em tempo real por um operador humano devido ao tempo de atraso existente entre o envio e recebimento da informação e comando de ambos os lados. Nesta linha [Elfes et al., 2008] discorre sobre o uso de robôs para exploração aérea da lua Titan de Saturno, por meio de uma arquitetura otimizada para este tipo de tarefa.

2.2 Arquiteturas

Juntamente com o desenvolvimento de um veículo autônomo, faz-se necessário a escolha de uma arquitetura adequada ao uso que se fará deste sistema.

O termo arquitetura na robótica móvel, pode ser aplicado à estratégia com a qual se deseja realizar uma tarefa, como em [Correia and Steiger-Garcao, 1994], onde é proposta uma arquitetura baseada em comportamento na qual, dependendo dos sinais recebidos nos sensores, é adotada uma diretiva para realização de uma tarefa designada. Outro exemplo desta mesma interpretação do termo é mostrado em [Widyotriatmo and Hong, 2008] que desenvolve um conceito baseado em decisões para garantir o cumprimento de um objetivo, que é sair de um ponto e chegar a outro, partindo de um planejamento inicial e tomando decisões baseadas em quantidades probabilísticas para o cumprimento do objetivo ao longo do trajeto. Um terceiro exemplo é obtido em [Cesetti et al., 2010] que propõe uma

arquitetura baseada em visão para navegação e pouso de um UAV. Em todos estes casos, a arquitetura é tratada do nível mais alto para o nível mais baixo, de modo que o sistema inteligente, aquele que atenderá a missão designada por um usuário do sistema, necessite de um grande volume de informação desde os níveis mais baixos, como os sensores e atuadores, até os sistemas mais complexos como câmeras e radares.

A arquitetura tratada neste trabalho se refere ao modo como os diversos componentes de um veículo autônomo devem se integrar, tanto ao nível de hardware quanto de software, para fornecer uma solução orientada ao operador e que possa ser aplicada a diferentes tipos de carros. A orientação ao operador visa a criação de um padrão que torne a programação e operação de veículos autônomos e semi-autônomos intuitiva e simplificada de modo que o usuário final possa operar o sistema como se estivesse operando um sistema robótico comercial.

Cada veículo autônomo desenvolvido tem suas particularidades, tais como redes utilizadas, sensores e atuadores presentes, tipo de comunicação entre os diversos instrumentos, além de diferentes dinâmicas, comportamentos e aplicação propriamente dita. Deste modo a proposição de uma arquitetura que não sofra grandes mudanças em função das particularidades destes veículos é de difícil obtenção.

Algumas arquiteturas para aplicação em veículos robóticos já foram propostas com objetivos semelhantes, ainda que menos abrangentes. Entre elas pode-se destacar aquelas com características distribuídas, tais como a *Open Agent Architecture* [Martin et al., 1999], onde cada elemento da arquitetura é denominado um agente e onde se busca minimizar o impacto de novos elementos na estrutura. Outras têm características de reusabilidade como o *Framework* e Software apresentados em [Alur et al., 2001] que visam a reusabilidade de uma arquitetura hierárquica para utilização em multi-robôs na realização de tarefas que variam de resgate e segurança, até manipulação de objetos, mapeamento, localização e controle de formação. Ambos com a característica de serem aplicáveis em alto nível. No artigo [Koralalage and Yoshiura, 2009] é proposta uma arquitetura de software em alto nível denominada *iRail* para aplicação em locomotivas autônomas. Nesse trabalho, a arquitetura é descentralizada e fortemente voltada para a comunicação por rádio frequência como meio de transmissão de informação entre os elementos do sistema.

Em [Nelson, 1998] é proposta uma arquitetura de software denominada STESCA (*Strategic-Tactical-Execution Software Control Architecture*) que provê uma base sobre a qual pode ser desenvolvido um software completo de controle de um veículo autônomo

qualquer, e em [Zammit and Zammit-Mangion, 2010] de modo um pouco mais limitado é apresentada uma arquitetura e um projeto conceitual de uma estrutura computacional leve e de baixo custo para aplicação em navegação de veículos autônomos pequenos.

Algumas arquiteturas se aproximam mais da solução objetivo buscada neste trabalho, como [Szabo et al., 1992] que descreve uma arquitetura voltada para um sistema que permite a operação de veículos terrestres tanto no modo autônomo, quanto no modo tele-operado. A arquitetura utilizada é denominada RCS *(Real-Time Control System)*. A RCS é uma arquitetura hierárquica na qual cada módulo controla um ou mais módulos do nível inferior. A RCS foi desenvolvida para aplicação em veículos terrestres baseados em sensores e atuadores cuja missão envolvia a localização de determinados alvos em áreas não mapeadas. A Figura 2.2 mostra os elementos presentes na arquitetura e a disposição dos mesmos.

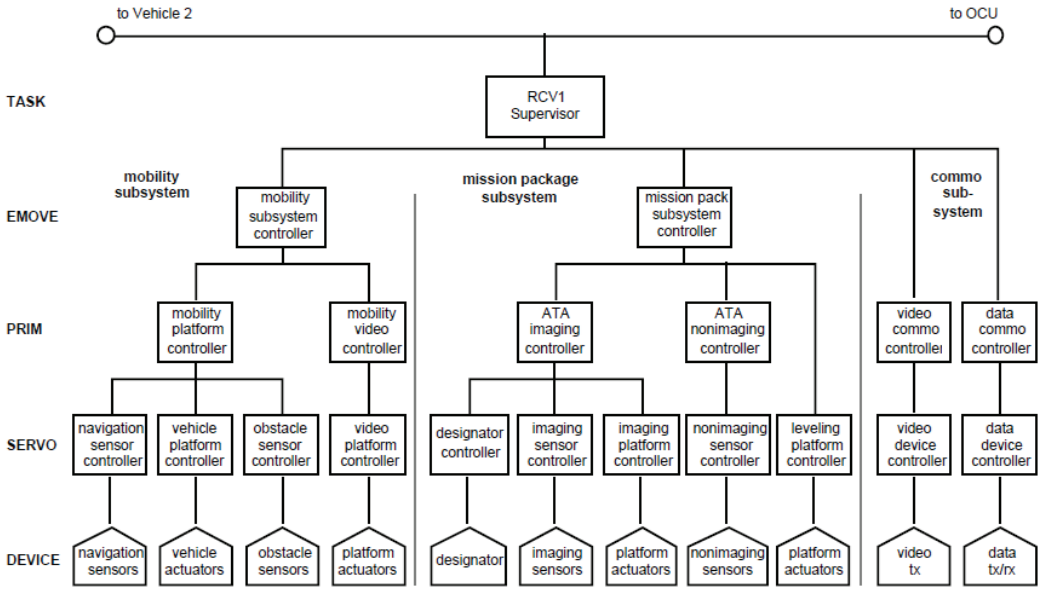

Figura 2.2: Arquitetura Padrão Proposta por [Szabo et al., 1992] baseada em RCS.

Nesta arquitetura, o mais alto nível de controle se encontra no Supervisor RCV (*Remote Controlled Vehicle*), responsável por coordenar mobilidade, pacote de missão e comunicação de todos os elementos inferiores na arquitetura. Abaixo do RCV, o subsistema de Mobilidade é responsável por dirigir o veículo, o subsistema de pacote de missão é responsável por gerenciar os alvos, e o subsistema *Common* gera informações tanto para os outros subsistemas quanto para o observador no último nível. Esta estrutura é dividida em cinco níveis: No primeiro nível *Device Level* (DEVICE) os elementos do mesmo correspondem a todos os atuadores e sensores utilizados no sistema. No *Servo Level*

(SERVO), a plataforma de controle no subsistema de mobilidade do veículo comanda os atuadores de posição do volante, do pedal de acelerador, freio e outros motores cada um deles independentemente, ao mesmo tempo em que nos outros subsistemas é controlada a aquisição de informação dos sensores de imagem e das câmeras. O *Primitive Level*(PRIM) no subsistema de Mobilidade controla a geração de trajetórias suaves em coordenadas convenientes baseado, ou nos objetivos repassados pelo operador, ou em caminhos gravados em memória. No subsistema de pacote de missão o ATA (*Automatic Target Acquisition*) automaticamente detecta, localiza e direciona um laser para os alvos, tanto os estáticos quanto os em movimentos. O *Elemental Level* (EMOVE) é o responsável pelo caminho e por desviar de obstáculos sendo uma integração entre os três subsistemas neste nível. Esta arquitetura possui a estrutura de uma árvore invertida e a informação proveniente do OCU (*Operator Control Unit*) pode ser repassada tanto a um veículo quanto a múltiplos por meio do *link* de comunicação disponibilizado no último Nível da arquitetura.

Também em [Vermaas et al., 2007] é apresentada uma pesquisa sobre os benefícios e aplicações dos veículos autônomos terrestres e uma descrição dos modelos mais comuns de sensores e atuadores presentes nesses veículos. Por meio destas informações é proposta uma arquitetura de software genérica voltada aos veículos terrestres podendo variar entre dois e quatro níveis sendo eles:

- *Sensor Network* que contém um conjunto de sensores como câmeras, LIDAR, radar, velocidade, posição, IMU, etc, responsáveis por capturar dados do ambiente.
- *Environment Identification* responsável por processar as informações recebidas e prover informações como detecção de obstáculos, pedestre, probabilidade de colisão e reconhecimento de padrões.
- *Situation and Behavior Control* responsável por reconhecer a situação do veículo e por meio de modelos determinar o comportamento e os próximos passos do mesmo.
- *Low Level Control* responsável por converter os comandos recebidos em controle efetivo no veículo por meio de ação nos atuadores que controlam a volante, aceleração, frenagem e outros elementos responsáveis pelo movimento do veículo.

A Figura 2.3 mostra a arquitetura desenvolvida para uso genérico em veículos tanto autônomos, quanto semi-autônomos. As quatro camadas da arquitetura se encontram a direita na figura.

No artigo [Pastor et al., 2006] é apresentada uma arquitetura de Hardware e Software

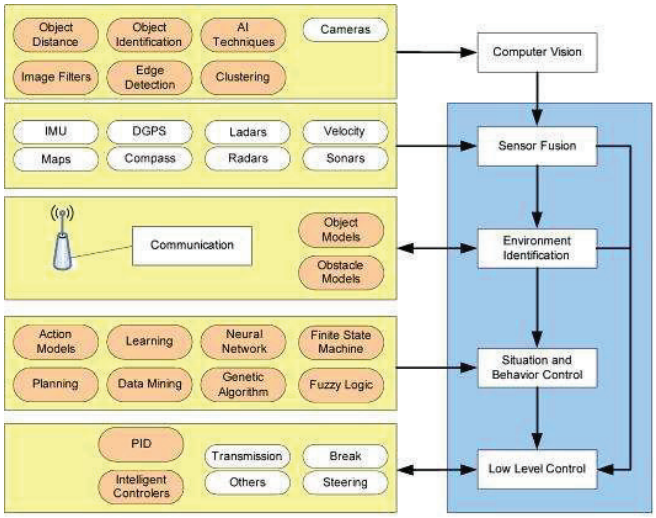

Figura 2.3: Arquitetura Padrão Proposta por [Vermaas et al., 2007] para UGV's.

para aplicação em SUAV's. Neste artigo, o SUAV é divido em seis sistemas principais denominados *UAV Airframe, Flight Computer, Payload, Mission/Payload Controller, Base Station e Communication Infrastructure* que são dispostas de forma a oferecer modularidade e flexibilidade em sua aplicação nos SUAV's. O principal objetivo desta arquitetura é oferecer um esquema de ligação de fácil adaptação a qualquer tipo de veículo SUAV. As arquiteturas de Hardware e Software são mostradas respectivamente nas Figuras 2.4 e 2.5.

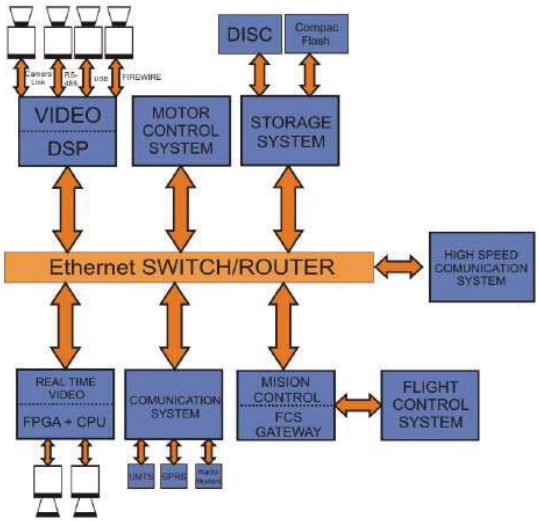

Figura 2.4: Arquitetura de Hardware Proposta por [Pastor et al., 2006] para SUAV's.

Nesta arquitetura são identificados os três principais componentes de um sistema

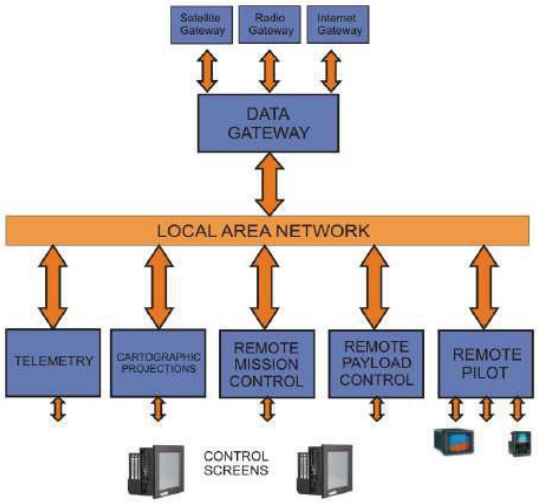

Figura 2.5: Arquitetura de Software Proposta por [Pastor et al., 2006] para SUAV's.

autônomo sendo eles o Sistema de Movimentação composto pelo *Monitor Control System*, o Sistema de Localização composto pelo *Communication System* e o Sistema de Percepção identificado nas caixas *DSP* e *Real Time Video*. Todo o sistema se comunica via rede de alta velocidade com a central de processamento de mais alto nível chamada *Mission Control*. Além destes elementos, observa-se o bloco denominado *Storage System* responsável por guardar informações de todas as mensagens que passam pela rede. A arquitetura de Software é a chamada SOA (*Service-oriented architecture*) onde cada elemento de processamento na rede pode subscrever a mensagem de um elemento presente na rede e que forneça este serviço. Esta arquitetura apresenta a vantagem de permitir uma rápida troca de informações entre os diversos sistemas e, apesar de possui um computador de processamento de alto nível, parte do processamento é realizado também nos sub-blocos que podem subscrever os serviços e todas as informações presentes na mesmas podem ser guardadas pelo bloco de armazenamento. No entanto, neste tipo de arquitetura se torna difícil a separação dos elementos em níveis tanto em Hardware quanto em Software uma vez que não há uma distinção clara de quais elementos estão em um nível mais alto e quais estão em níveis mais baixos.

No artigo [Martinez et al., 2010] é apresentada uma arquitetura de hardware e software genérica para UUVs. Este trabalho foi desenvolvido pelo *Hidrographic Research Center* de Cuba em conjunto com o a Universidade Central de Villas que visaram principalmente a obtenção de uma arquitetura baseada em uma plataforma de baixo custo tanto do ponto de vista de sensores quanto atuadores. Apesar de sua aplicação marítima e dos sensores e atuadores diferenciados, a arquitetura apresentada se aproxima em muito

das utilizadas em veículos aéreos e terrestres como mostram as Figuras 2.6 e 2.7 que apresentam respectivamente a arquitetura de Hardware e Software do sistema.

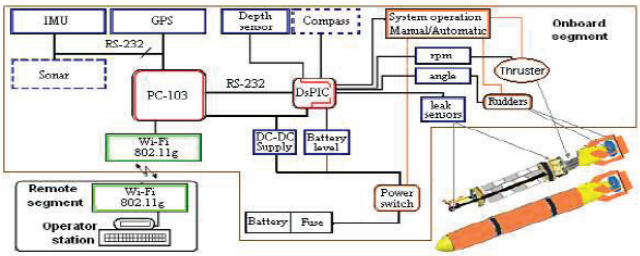

Figura 2.6: Arquitetura de Hardware para UUV's proposta em [Martinez et al., 2010].

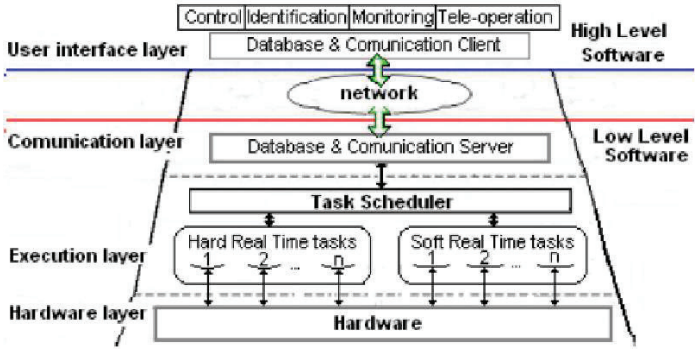

Figura 2.7: Arquitetura de Software para UUV's proposta em [Martinez et al., 2010].

Esta arquitetura é centralizada do ponto de vista de hardware e software e na mesma podem ser identificados apenas dois dos três componentes básicos de um veículo autônomo sendo eles: o Sistema de Movimentação, composto pelos sensores de ângulo e RPM, e atuadores leme e hélices, e o Sistema de Localização, composto pelo GPS, IMU, sensor de profundidade e Bússola. Todos estes elementos, juntamente com o sistema de segurança composto por sensores de gás e medidor de nível da bateria, estão ligados ao PC-103 que é a central de processamento. Alguns são ligados diretamente enquanto outros passam por um microcontrolador, para evitar sobrecarga nas interfaces do PC Central. A Nível de Software, pode-se ver uma separação em tarefas de *Hard Real Time*, *Soft Real Time* e *Non-Real Time*, que estão divididas hierarquicamente e onde é possível observar que quanto mais alto o nível, maior a quantidade de dados e processamento necessários. Nota-se a ausência de um elemento de supervisão, ao mesmo tempo em que não é possível uma divisão clara da Arquitetura de Hardware em níveis devido ao modo de ligação dos diversos elementos.

Outros trabalhos importantes foram fonte de inspiração para o desenvolvimento deste

trabalho, como o sistema utilizado no AqVS de [Iscold et al., 2010] e no veículo Little Ben [Bohren et al., 2009].

O AqVS desenvolvido pelo grupo de Pesquisas e Desenvolvimento de Veículos Autônomos da UFMG (PDVA) é um pequeno veículo aéreo autônomo (SUAV), projetado para reconhecimento de alvos no solo. Este veículo foi especialmente projetado de forma a reduzir o número de sensores e atuadores necessários sem o comprometimento da execução de sua missão. Sua arquitetura de hardware é centralizada de forma que todos os elementos presentes no sistema se comunicam diretamente com a central de processamento, no caso um PDA, conforme mostrado na Figura 2.8. A arquitetura de software, mostrada na Figura 2.9, é divida em programação de baixo nível, para sensores e atuadores, e alto nível para o computador central que recebe as informações para realização da missão. Esta arquitetura é prática do ponto de vista do número de elementos envolvidos, mas está sujeita a falhas por não permitir redundância.

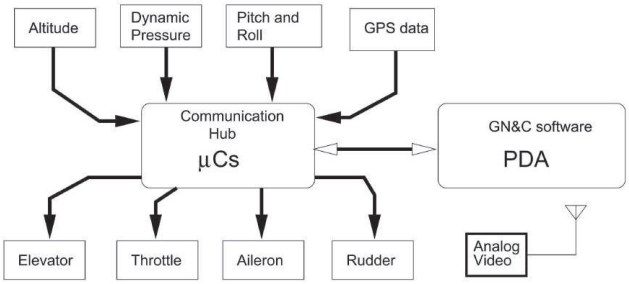

Figura 2.8: Arquitetura Hardware Utilizada no AqVS [Iscold et al., 2010].

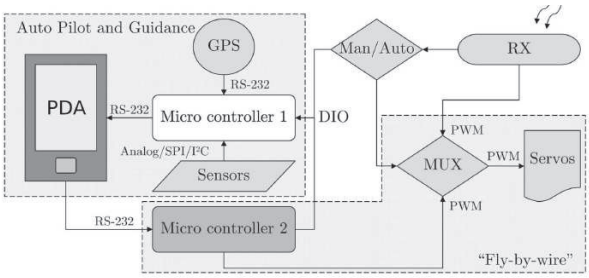

Figura 2.9: Arquitetura Software Utilizada no AqVS [Iscold et al., 2010].

O Little Ben foi construído para o desafio urbano DARPA de 2007 e sua arquitetura de software é mostrada na Figura 2.10. Nesta arquitetura pode-se verificar uma separação em dois níveis sendo o nível mais baixo denominado *Driving Module* responsável pela

interface com o veículo, controle de hardware e verificação da correta operação do volante, acelerador, freios e transmissão. Neste mesmo nível também foram inseridos um GPS e uma INS (*Inertial Navigation System*) capaz de enviar informação a uma frequência de 100Hz. No nível mais alto, denominado *Mission Planing Module*, se encontram os elementos necessários para determinar a sequencia ótima de pontos de passagem para completar um objetivo. No centro da arquitetura se encontra o *MapPlan* que processa tanto as informações recebidas do *Driving Module* e do *Mission Planning Module* quanto informações recebidas diretamente neste módulo como câmeras e LIDAR. Todo o sistema está sob supervisão de um gravador ou *Logger* responsável por coletar todas as informações trafegadas no sistema. A Arquitetura de Hardware é multiprocessada, sendo formada por vários PC's Mac Mini Core2Duo com sistema operacional Linux, e cada um destes PC's é responsável pelo cumprimento de uma tarefa específica no sistema. Não há uma divisão clara de camadas, apesar de se observar o agrupamento de dispositivos com funções semelhantes. O sistema de comunicação é fortemente baseado em Ethernet, além de usar padrões RS-232 e RS-422 para interface entre o computador e os PCs. O esquema de Hardware é mostrado na Figura 2.11. Observar-se nesta implementação que o nível mais baixo da arquitetura exigiu a interligação com o sistema do veículo, ou seja, parte do sistema de controle passa pela ECU (*Electronic Control Unit*) do sistema nativo do carro. Este tipo de abordagem tem a desvantagem de restringir a utilização desta arquitetura a veículos semelhantes ao Little Ben, além de exigir um profundo conhecimento do protocolo da ECU do veículo que por muitas vezes está indisponível, ou cujo domínio é muito dispendioso.

Outros dois veículos nos mesmos padrões do Little Ben são o Junior e a Caroline que, igualmente, tem informações interessantes acerca de arquitetura para contribuir com o objetivo deste trabalho.

No Veículo Junior de Stanford [Montemerlo et al., 2008] utilizou-se de uma estratégia baseada na arquitetura por função, separando os equipamentos hierarquicamente de acordo com a informação que iriam repassar. Cada um dos blocos presentes nesta arquitetura é designada por serviços. Além dos blocos funcionais de interação com o veículo como o *Vehicle Interface* e *Sensor Interface* e Blocos de processamento e interação com o usuário, como *Navigation*, *Perception* e *User Interface*, um terceiro tipo de bloco é introduzido na arquitetura para gerar os relatórios de informação e diagnóstico, além de manter a integridade do sistema. Este bloco é chamado *Global Services* e está presente em todos os níveis. A comunicação entre os blocos é feita por meio de publicação/subscrição anônima de mensagens. A arquitetura deste sistema é mostrada Figura 2.12.

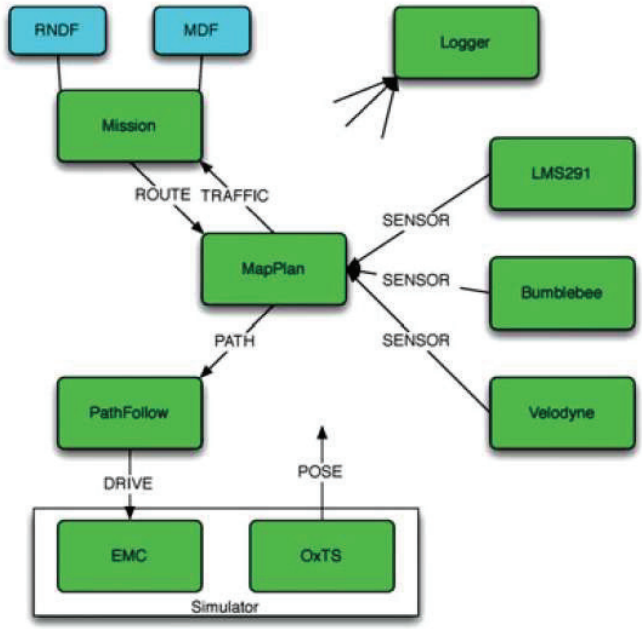

Figura 2.10: Arquitetura Software do Little Ben [Bohren et al., 2009].

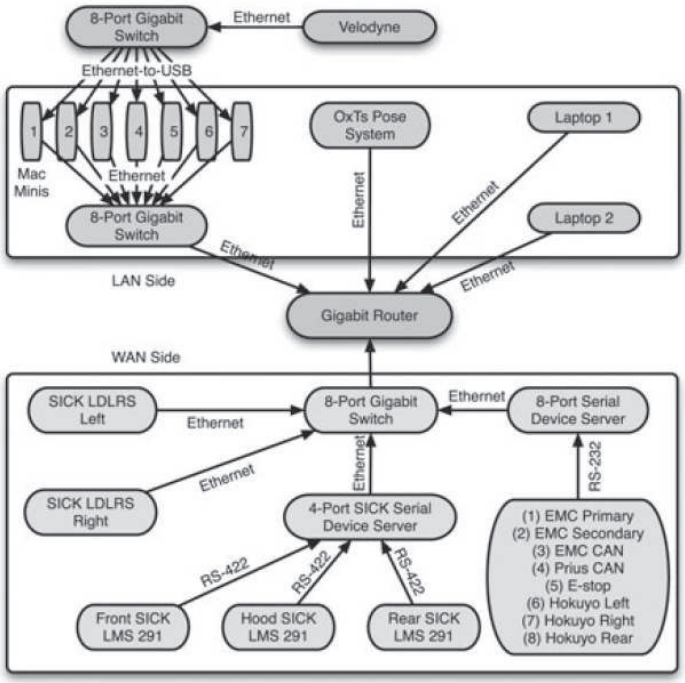

Figura 2.11: Arquitetura de Hardware do Little Ben [Bohren et al., 2009].

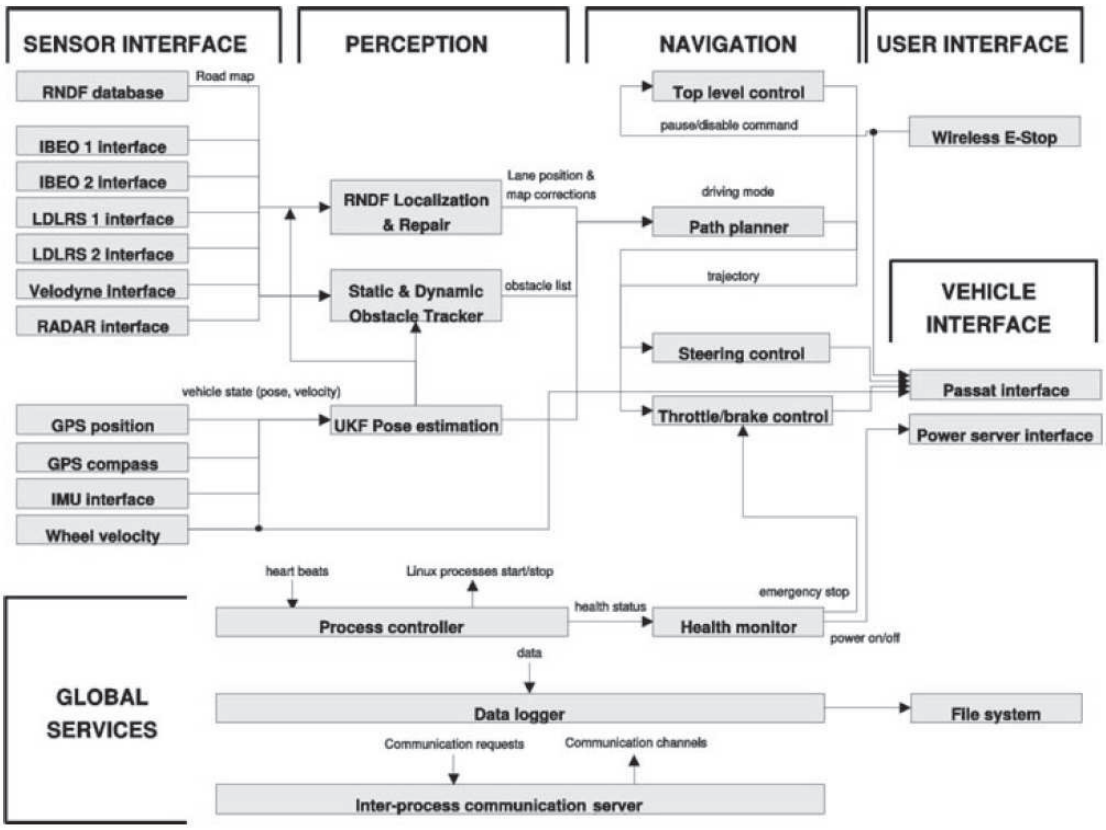

Figura 2.12: Arquitetura de Junior [Montemerlo et al., 2008].

A arquitetura de Caroline [Rauskolb et al., 2008] é mostrada na Figura 2.13 e possui algumas características interessantes, tanto do ponto de vista de controle, quanto do ponto de vista de diagnóstico. O sistema é subdivido em três níveis de software sendo que quanto maior o nível maior a complexidade da informação. No primeiro nível se encontram os sensores e atuadores que, de acordo com a referência, são nativos do veículo, ou seja, a rede CAN (*Controller Area Network*) utilizada neste projeto é a própria rede CAN utilizada no veículo para obtenção das informações e funcionamento dos sistemas do veículo como Freio ABS, Sistema de Alarme, Airbag e outros. No segundo nível, todas as informações são fundidas e repassadas ao terceiro nível que é o responsável pelo planejamento e execução da trajetória desejada.

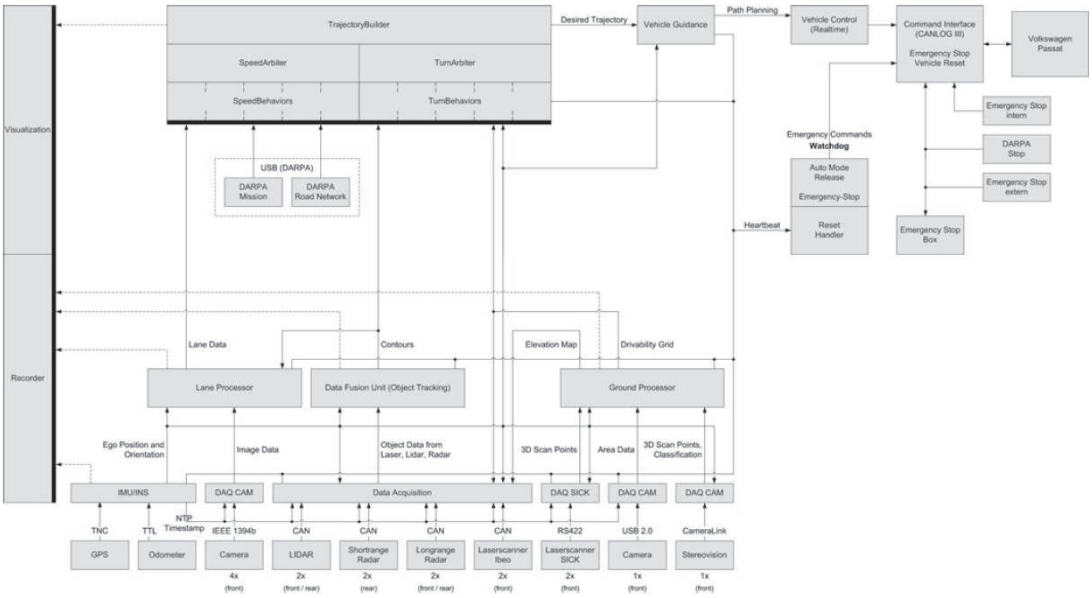

Figura 2.13: Arquitetura de Caroline [Rauskolb et al., 2008].

O trabalho apresentado nesta dissertação de mestrado usa algumas das ideias previamente publicadas para compor uma arquitetura que permita ao usuário comandar um carro autônomo por meio de um programa de computador ou por meio de interfaces simples com o usuário. As arquiteturas apresentadas foram escolhidas dentre muitas existentes na literatura para ressaltar a importância de alguns elementos presentes no próximo capítulo onde é desenvolvida a arquitetura de Hardware e Software proposta neste trabalho. Dentre eles observou-se:

- A similaridade existente entre arquiteturas concebidas para veículos diferentes.
- A divisão em vários níveis para atividades complexas e processamento centralizado para atividades mais simples.

- A falta do elemento de supervisão na maioria das arquiteturas apresentadas.
- A ausência de uma preocupação com o usuário final em todas elas.

A arquitetura foi implementada em um carro, devido a sua facilidade de operação e disponibilidade e o detalhamento da metodologia é mostrado no próximo capítulo.

3 Arquitetura Proposta

Com base nas referências citadas no capítulo anterior e no estudo de alguns sistemas autônomos atualmente existentes, propôs-se uma arquitetura de Hardware e Software para controle e supervisão de um carro autônomo subdivida em três níveis supervisionados, sendo eles:

1. Nível de Controle: No qual se localizam os elementos de controle e sensoriamento em tempo real do carro e cujas características envolvem taxas de amostragem da ordem de dezenas de Hertz e pequena quantidade de informação a ser transmitida.

2. Nível de Processamento: Onde se encontram os elementos que não necessitam de processamento em tempo real, e cuja transmissão é realizada com taxas da ordem de Hertz, com uma quantidade maior de dados a serem transmitidos se comparada às transmissões de informações do nível anterior.

3. Nível Usuário: É a camada pela qual o usuário final envia comandos e recebe informações de alto nível do veículo. Programas, como por exemplo, planejamento de trajetória, seguimento de caminho, desvio de obstáculos e execução de missões serão processados no nível usuário utilizando as informações obtidas e processadas nos níveis inferiores.

Em todas os níveis existe uma ligação direta com um elemento de supervisão responsável por gerar relatório de toda informação gerada na arquitetura. O processo pelo qual se chegou a esta solução é descrito nas próximas seções, juntamente com o processo de construção da arquitetura e a descrição detalhada da mesma.

3.1 Sistemas de Carros Autônomos

Um carro autônomo é um veículo terrestre composto por vários sistemas e subsistemas que permitem ao mesmo executar as tarefas assinaladas. Alguns destes sistemas são

considerados chave por serem a essência de um carro sem motorista, tal qual o sistema de movimentação que permite ao mesmo se deslocar sem a intervenção humana. Outros sistemas, apesar de importantes, são vistos como complementares para garantir um cumprimento otimizado de determinada tarefa.

Esta seção não pretende explorar todas as possibilidades de sistemas em um veículo autônomo terrestre, mas visa dar uma ideia geral dos sistemas comumente utilizados em carros autônomos e a necessidade dos mesmos estarem presentes na arquitetura de Hardware e Software desenvolvida.

3.1.1 Sistema de Movimentação

O Sistema de Movimentação permite ao veículo se deslocar. Este é um sistema chave pois a maioria absoluta das tarefas designadas a um determinado veículo autônomo terrestre envolve o deslocamento do mesmo de um ponto ao outro sem intervenção humana. Do sistema de movimentação de um veículo podem fazer parte dentre outros:

- Direcionamento: A direção do veículo permite que seja escolhida a trajetória pela qual o veículo vai percorrer. Em carros, o sistema de direcionamento é o volante, responsável pelo esterçamento das rodas e consequente mudança de direção.
- Frenagem: Permite ao veículo reduzir a velocidade ou mesmo parar em determinadas condições.
- Aceleração: Permite ao veículo alterar sua velocidade ao longo do tempo.
- Sensoriamento: Compreende as informações obtidas por meio dos sensores do próprio veículo e inclui todos os sinais que de uma forma ou de outra fornecem o estado atual de alguma grandeza do veículo, como velocidade, temperatura, rotação do motor e outros. Geralmente os sensores são proprioceptivos, ou seja, fazem medições internas ao veículo.

3.1.2 Sistema de Localização

O Sistema de Localização é o complemento do Sistema de Movimentação. Um veículo que se move deve, preferencialmente, ter ciência de sua origem e do seu destino. O sistema de localização permite ao veículo se movimentar de maneira inteligente saindo de um ponto e chegando a outro previamente determinado. Dos sensores de um sistema de localização pode-se citar:

- GPS (*Global Positioning System*): O GPS pode ser considerado o principal sensor na localização de um veículo que se movimenta em um meio externo. O GPS utiliza informações recebidas via satélite para dar a posição no globo em termos de latitude e longitude. A precisão de um GPS comum é da ordem de metros. No entanto, com a utilização de um GPS diferencial é possível ter uma precisão de centímetros para se localizar um veículo no mapa. O GPS é um bom equipamento em ambientes abertos, mas seu sinal pode sofrer cortes abruptos quando o veículo se encontra sobre área de sombra, como em túneis, áreas com árvores e garagens.

- IMU (*Inertial Measurement Unit*): A IMU é uma unidade inercial que permite por meio dos seus dados a obtenção dos perfis de velocidade angular, aceleração e as atitudes do veículo. Sua utilização é de extrema importância para detectar variações que interessam ao sistema de controle como presença de aclives ou declives e também o tipo de solo em que o veículo se encontra, ou os movimentos realizados pelo veículo em algum meio. Por meio dos dados da IMU é possível inferir os perfis de velocidade do veículo e até mesmo o deslocamento do mesmo.

- Bússola: A bússola é um importante elemento na atualização de determinadas variáveis pois serve como corretor na atitude do veículo, uma vez que a mesma informa com relativa precisão como o veículo está posicionado em relação ao campo magnético da Terra.

3.1.3 Sistema de Percepção

O Sistemas de Percepção de um veículo é de grande importância principalmente para a sua utilização em ambientes não controlados como um ambiente externo. Este sistema permite ao veículo, dentre outras coisas, desviar de obstáculos, reconhecer sinalizações e outro padrões de imagem que podem tanto auxiliar no sistema de localização quanto propriamente construir mapas e identificar áreas desconhecidas. Fazem parte do Sistema de Percepção, dentre outros:

- Câmeras: Podem ser dos mais variados tipos. São utilizadas para filmar o ambiente em 2D ou 3D, no espectro visível ou infravermelho, e permitem por meio de processamento das imagens captadas inferir posições de obstáculos, distâncias, sobreposição, reconhecimento de padrões e uma série de outras informações que são de grande utilidade para melhorar o desempenho na movimentação e navegação de um veículo autônomo.

- LIDAR: LIDAR é o acrônimo de *Light Detection And Ranging* (Detecção e Medição por meio de Luz) e compreendem os dispositivos que se utilizam normalmente de *Lasers* ou feixes infra-vermelho para medir distâncias de alguns objetos a si mesmos. Os LIDAR's podem ser uni, bi ou tri-dimensionais, e a sua utilização permite a detecção de objetos que por vezes não são detectados por câmeras ou outros dispositivos.

- Ultrassom: Têm a mesma finalidade dos LIDAR's, porém utilizam-se de som como elemento medidor. Este tipo de sensor permite ao veículo processar informações relativas a obstáculos pequenos, paredes e meio-fios de ruas para auxílio no estacionamento, desvios, localização e remapeamento de rota. Sua utilização também é de grande importância para veículos autônomos em ambientes sub-aquáticos.

- Radares - Um Radar é composto por uma antena transmissora e receptora de sinais para Super Alta Freqüência (SHF). Por meio da transmissão e recepção deste sinal é possível a detecção de obstáculos.

3.2 Requisitos da Arquitetura

Baseado nos sistemas mostrados na seção anterior, é possível a criação de uma estrutura que comporte estes sistemas e, simultaneamente, forneça algumas garantias mínimas quando se trata de um veículo autônomo terrestre, ou mais especificamente um carro autônomo. Para tanto, são definidos alguns requisitos necessários tanto para hardware quanto para software e, a partir destes requisitos, são criadas as soluções apresentadas na arquitetura obtida.

3.2.1 Requisitos de Hardware

De modo a comportar todos estes sistemas, a arquitetura de hardware de um veículo autônomo deve permitir uma flexibilização ao mesmo tempo em que mantém a integridade de todo o sistema. Isto significa na prática que o sistema não deve sofrer interferências drásticas em seu desempenho a medida em que forem adicionados novos elementos na estrutura. A ausência de alguns sistemas também não pode levar a situações como mal funcionamento do sistema ou até mesmo desligamentos inesperados.

Baseando-se nesta necessidade, e apoiando-se nos estudos previamente realizados, chegou-se a conclusão de que uma arquitetura de Hardware robusta o suficiente para

permitir o uso simultâneo dos sistemas necessita de uma divisão em diferentes camadas. Da mesma forma, para que não haja prejuízo da arquitetura, quando há uso dos sistemas separadamente, utiliza-se um computador (Unidade Central de Processamento) para cada nível da arquitetura, responsável pela gerência das informações, integridade do sistema e passagem de dados para os outros níveis. Complementam esta conclusão as premissas citadas abaixo:

- O Hardware utilizado no sistema de movimentação deve permitir uma comunicação determinística entre os elementos constituintes deste sistema e a unidade central de processamento. Esta característica implica que no nível mais baixo da arquitetura é possível a troca de mensagens entre seus elementos sem a introdução da incerteza associada a uma comunicação estocástica.
- O Hardware deve ser compatível com as dinâmicas do veículo para evitar perda de informação por incapacidade de processamento ou transmissão de dados.
- O Hardware deve ser robusto, não sendo afetado por ruídos e vibrações mecânicas, assim como não deve ser afetado por ruídos eletromagnéticos presentes no veículo.
- O Hardware deve permitir o monitoramento e controle do veículo à distância.
- O Hardware não pode interferir no sistema de funcionamento normal do veículo de modo que, se este for desligado, o veículo deve voltar ao seu estado original.
- O Hardware deve oferecer segurança e suporte a sistemas de emergência.
- O Hardware deve permitir a comunicação com uma central responsável por monitorar todos os sistemas e gerar dados de relatório para ensaios, diagnóstico e detecção de falhas.
- O Hardware deve permitir que cada sistema ou subsistema do veículo possua sua própria unidade central de processamento para favorecer o paralelismo e a clarificar as divisões entre cada sistema.

3.2.2 Requisitos de Software

De forma a permitir a integração entre os sistemas, a arquitetura de Software necessita garantir a integridade do sistema. Por meio dos estudos de algumas arquiteturas de softwares existentes, foram levantadas algumas características importantes que conduzem

a proposição de uma Arquitetura de Software robusta e integrada. Estas características são mostradas a seguir:

- O Software deve permitir o isolamento entre os níveis, prevenindo que o sistema de um nível interfira destrutivamente nos outros.
- O Software deve processar a informação para facilitar o seu entendimento para todos os subsistemas de interesse.
- O Software deve ser capaz de, em cada subsistema, atender as demandas de tempo requeridas.
- O Software deve permitir uma comunicação entre os diversos sistemas e níveis diferenciados por meio de interfaces bem definidas.
- O Software deve catalogar todas as informações enviadas e recebidas por todos os sistemas da arquitetura com registro ou *tag* de tempo, quando pertinente e necessário.
- O Software deve ser tolerante a perturbações e falhas.

A junção destas características é feita de forma a permitir uma arquitetura de software reutilisável. Esta característica permite que um desenvolvedor possa utilizar a arquitetura proposta em qualquer plataforma, desde que os requisitos de tempo, garantias de entregas de mensagens e escalabilidade do sistema sejam cumpridas.

3.3 Apresentação da Arquitetura Proposta

Com base nos requisitos, chegou-se a estrutura mostrada na Figura 3.1. Esta arquitetura, dividida em três níveis com supervisão visa satisfazer as premissas de Hardware e Software assumidas na seção anterior.

Por meio da Figura 3.1 ainda é possível observar a distinção entre as três camadas da arquitetura, sendo cada uma possuidora das características inerentes do seu funcionamento, e a supervisão, que por construção se encontra em todas as camadas.

- A primeira camada é a de mais baixo nível é chamada Nível de Controle.
- A segunda camada é chamada Nível de Processamento.

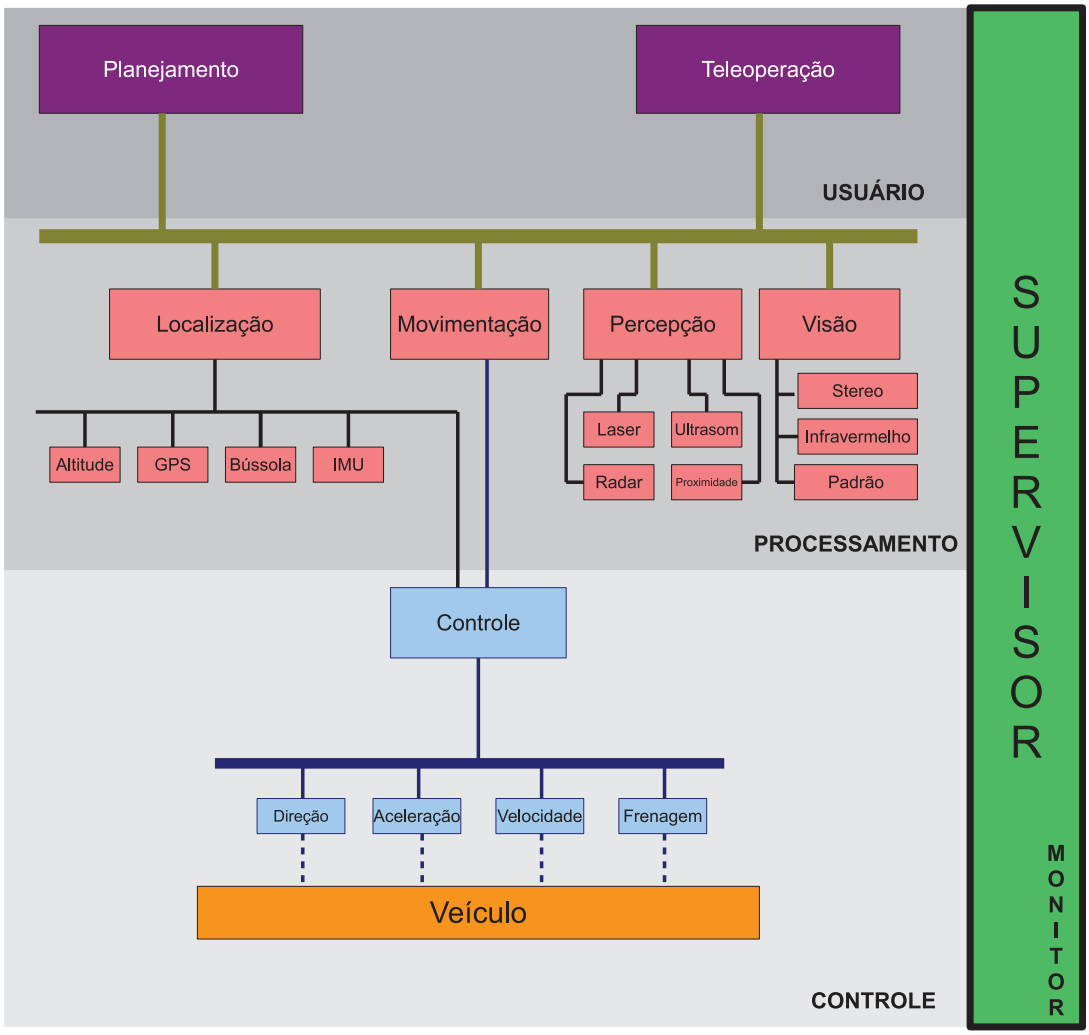

Figura 3.1: Arquitetura Proposta. As linhas contínuas indicam ligação entre os elementos da arquitetura e as pontilhadas indicam interfaces com o veículo. Caixas da mesma cor pertencem aos mesmos níveis.

- A terceira camada é chamada Nível Usuário.
- A supervisão é colocada em paralelo aos outros níveis e é chamada Monitor.

Cada um dos sistemas pode ter o subsistema da camada inferior como parte de sua formação. As próximas subseções detalham as características de cada um dos níveis.

3.3.1 Nível de Controle

O Nível de Controle, do ponto de vista de arquitetura, é o nível no qual os sensores e atuadores mais próximos ao hardware estão acoplados. Para este nível as principais características adotadas é o determinismo das informações trafegadas, e a capacidade de processamento que deve ser superior a demanda de mensagens enviadas e recebidas pelos sensores e atuadores presentes no nível.

Uma das características destes elementos é a frequência em que eles trabalham, que é ordem de dezenas de Hertz, podendo chegar até centenas de Hertz. Neste nível, o hardware proposto é capaz de absorver e processar todas as informações necessárias para o bom funcionamento do veículo, para o desenvolvimento de malhas de controle de tempo real, e para obtenção de dados confiáveis em etapas de modelagem e validação do sistema. Assim as dinâmicas do veículo em questão devem ser observadas na construção do hardware de forma a fornecer um mínimo de informação para a construção de um bom sistema de controle.

Outra necessidade atendida pela arquitetura apresentada é relativa ao período de amostragem dos sensores. Por utilizar sistemas determinísticos, a arquitetura é capaz de fornecer períodos de amostragem constantes ou com desvios mensuráveis de forma a permitir a criação de estratégias de controle digital simples e de qualidade.

Toda lógica de controle de baixo nível é feito exclusivamente neste nível, de forma que para os níveis superiores estão disponíveis apenas informações trabalhadas e processadas. Os comandos recebidos pelos níveis superiores são desmembrados de forma a executar as ações de controle demandadas.

O Nível de Controle contém uma Unidade Central de Processamento responsável por se comunicar com os demais elementos de atuação no veículo. É necessário que haja uma ligação lógica entre o Nível de Controle e o Nível de Processamento, que será descrito a seguir, podendo ser esta ligação física, separada por Hardware ou virtual, separada por software.

Para o Nível de Controle, do ponto de vista de Software, o principal requisito a ser atendido está relacionado à restrição de tempo imposta para que se tenham o tráfego de informações de forma determinística e confiável, permitindo-se assim o uso de técnicas clássicas de controle digital na construção dos algoritmos desta camada. Um dos principais requisitos exigidos para implementação de um controle digital satisfatório é obtenção de taxa de amostragem constante. Por meio do uso de um sistema operacional de tempo real é possível obter taxas de amostragem com um desvio temporal não significativo se levado em consideração as dinâmicas envolvidas no sistema.

As informações coletadas no Nível de Controle são processadas e enviadas ao Nível de Processamento em um modelo de interface pré-estabelecido. Isto permite que a arquitetura concebida sofra modificações no Nível de Controle sem implicar em modificações nos níveis superiores.

Pode-se identificar quatro interfaces neste Nível:

- Fornecedor de Dados: Interface para envio de informação sensoriais para a Unidade Central de Processamento;
- Consumidor de Dados: Interface para recebimento de comandos enviados da Unidade Central de Processamento;
- Concentrador Tempo Real: Interface necessária para gerenciar as informações do Fornecedor de Dados e do Consumidor de Dados.
- Servidor Real-Time de Dados de I/O: Interface responsável por para o Nível de Processamento os dados recebidos e processados pela Unidade Central de Processamento.

Estas interfaces são mostradas na Figura 3.2.

3.3.2 Nível de Processamento

No Nível de Processamento, devido a natureza das medições, a frequência de trabalho é menor que a frequência no Nível de Controle, enquanto que a quantidade de informações é maior em cada medição dos sensores, o que leva a priorização, neste nível, da capacidade de processamento da informação.

Os dados recebidos pelos sensores visam permitir a inferência de informações como a localização do veículo, os obstáculos detectados, reconhecimento de padrões, dentre

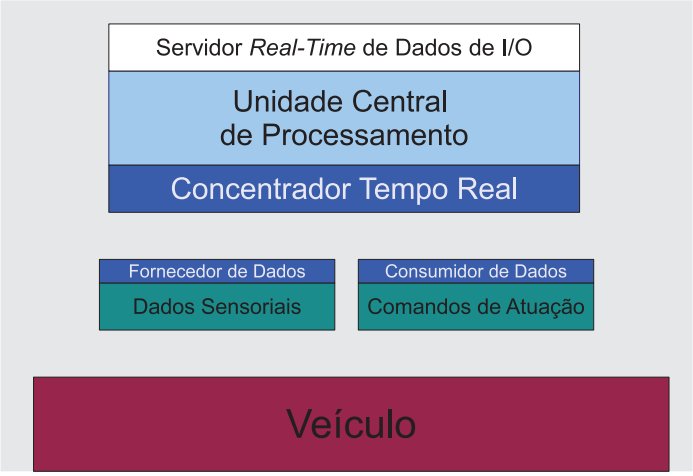

Figura 3.2: Interfaces de comunicação e processamento no Nível de Controle

outras. O hardware, com os requisitos desejados, é capaz de processar estas informações e entregá-las em tempo hábil ao próximo nível.

Da mesma forma que no Nível de Controle, as informações passadas para o nível Superior devem estar devidamente processadas e simplificadas, não permitindo que dados brutos subam pelo sistema.

Além destas características, sugere-se que sistemas com objetivos diferentes sejam ligados em computadores diferentes para uma melhor separação entre os mesmos e, simultaneamente, permitir que executem tarefas em paralelo. Isto significa na prática que, por exemplo, o Sistema de Localização devem estar separado do Sistema de Percepção, tendo cada um sua Unidade Central de Processamento.

Os elementos do Nível de Processamento podem ter como "sensores" a unidade central de processamento do Nível de Controle. Isto dá uma flexibilidade na arquitetura que permite a junção de informações compatíveis que se encontram em níveis diferentes da arquitetura.

Uma vez que o Nível de Processamento não está diretamente ligado ao Veículo propriamente dito, não se faz necessária a utilização de um sistema de tempo real para a coleta de informações, apesar de que a etiqueta de tempo deve estar presente em cada pacote coletado, para permitir uma melhor avaliação e uso dos dados.

Os dados coletados neste nível são processados de forma a eliminar informações desnecessárias, redundantes ou danificadas, e permitir que o próximo nível obtenha somente

as informações pertinentes.

A característica mais importante do Processamento é fornecer uma interface bem definida para permitir que diferentes elementos do Nível Usuário possam se conectar aos elementos do Nível de Processamento. Esta interface bem definida se caracteriza por permitir que sejam feitas alterações nos elementos de um nível sem que estas modificações sejam repassadas ao nível superior. Além disto, por meio de definições de interface é que se introduz a "facilidade de operação" indicada na proposta de elaboração deste trabalho e que permite um uso do sistema por qualquer um como se fosse utilizado um sistema robótico comercial.

Na Figura 3.3 pode ser observada a existência de quatro elementos importantes sendo eles:

- Cliente RT: Interface capaz de conversar com a Unidade Central de Processamento do Nível de Controle por meio da interface Servidor Real-Time de Dados de I/O do mesmo;
- Concentrador: Interface capaz de obter informação dos sensores presentes neste nível;
- Fornecedor de Dados: Interface para envio da informação sensorial às Unidades Centrais de Processamento do nível;
- Servidor de Dados I/O: Interfaces bem definidas de interação com o Nível Usuário.

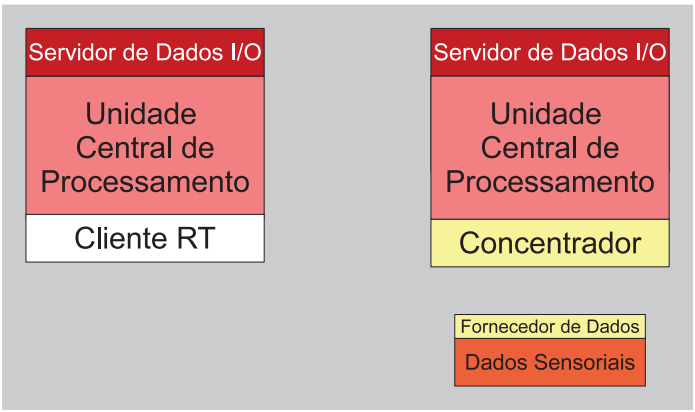

Figura 3.3: Interfaces de comunicação e processamento no Nível de Processamento

O Nível de processamento é escalável, ou seja, podem ser introduzidos quantos elementos forem necessários desde que respeitados os limites do Hardware. Isto se dá pois

cada um dos seus elementos possui sua própria unidade central de processamento e, deste modo, não há sobrecarga de informações na rede.

Uma vez que o nível de Usuário não tem acesso às informações cruas obtidas dos diversos elementos deste sistema, todos os cálculos, como por exemplo determinação de posição, filtros de Kalman e detecção de obstáculos, devem ser processados e a informação obtida pós-processamento é a única a ser repassada ao próximo nível. Isto é feito para não haver contaminação de um sistema pelo outro, e para permitir a flexibilidade buscada na construção desta arquitetura.

3.3.3 Nível de Usuário

O Nível de Usuário, do ponto de vista de Hardware, existe apenas por meio das redes necessárias para permitir a interação entre o usuário e os níveis inferiores. O hardware do usuário é qualquer elemento capaz de receber informações e enviar comandos aos elementos da arquitetura por meio desta rede, podendo ser um PC, um Tablet ou qualquer outro elemento capaz de se comunicar com a rede do nível inferior. A principal característica do Hardware de Nível usuário é a completa ausência de ligação física com qualquer sensor ou atuador presente nos outros níveis, sendo esta ligação feita apenas a nível lógico.

Do ponto de vista de Software, o Nível Usuário é o que conterá os algoritmos computacionalmente mais intensivos como planejamento de trajetórias, processamento de imagens, tomada de decisões e definição de comportamentos a serem seguidos pelo veículo. No entanto, a comunicação do nível de Usuário com o nível de Processamento se dá única e exclusivamente por meio das interfaces de comunicação. Não há interação entre o Nível de Usuário e o Nível de Controle em nenhuma circunstância.

Portanto, é necessário um estudo para se averiguar quais interfaces devem ser criadas e disponibilizadas de forma a permitir o uso contínuo e irrestrito do sistema sem a necessidade de se recorrer a subterfúgios para obtenção de dados que não sejam disponibilizados por meio das interfaces já definidas.

Do ponto de vista de usuário final, não há nenhum sistema entre o veículo e o Nível de Usuário, uma vez que para se alcançar a compatibilidade buscada e a facilidade de manuseio, não deve haver necessidade do usuário final conhecer os sistemas intermediários entre o veículo e o seu computador.

A única interface existente no Nível Usuário é chamada de Cliente, e a mesma pode ser utilizada para comunicação com qualquer sistema do Nível de Processamento, fazendo

uso da interface de Servidor de Dados I/O do mesmo.

Acima desta interface, na Unidade Central de Processamento, se encontram todos os algoritmos de alto nível que resultarão nos comandos enviados ao veículo. A Figura 3.4 mostra a interface Cliente e a Unidade Central de Processamento da mesma.

Figura 3.4: Interfaces de comunicação e processamento no Nível Usuário

3.3.4 Monitor

O Monitor não possui nenhuma hierarquia relativa aos outros três níveis, uma vez que na prática, a nível de hardware, ele está ligado nos três níveis.

Esta camada contém como principal elemento uma Unidade Central de Processamento, responsável por gerar registros ou *log* para toda e qualquer comunicação que ocorra em qualquer um dos níveis, desde que esta informação seja disponibilizada de alguma forma. Este sistema é importante, tanto para a geração de dados para coleta, estudo e identificação do veículo, quanto para determinar falhas e possíveis outras causas de não conformidades nos sistemas.

Fisicamente, o Monitor possui uma conexão com cada uma das redes ou elementos ao qual o mesmo deseja se conectar, como pode ser visto na Figura 3.1, onde é mostrada a arquitetura completa com todos os níveis e o sistema de supervisão e, do ponto de vista de Software, ele possui comunicação com todas as interfaces descritas anteriormente. O Monitor não interfere no restante do sistema sendo um agente passivo na coleta de dados.

3.4 Resumo do Capítulo

Neste capítulo foram identificadas as necessidades observadas para o desenvolvimento de uma arquitetura satisfatória e proposta uma arquitetura capaz de suprir a todas elas. A arquitetura proposta tem três níveis (Controle, Processamento e Usuário), e uma descrição detalhada de cada um destes níveis foi fornecida. O próximo Capítulo mostra a implementação desta arquitetura de Hardware e Software em um veículo autônomo do tipo UGV e a discussão acerca dos resultados.

4 Implementação e Resultados

No Capítulo 3 foi levantada uma série de requisitos para formatar a construção de uma Arquitetura de Sistemas aplicável a carros autônomos terrestres. Neste capítulo, com base na arquitetura apresentada, é feita a sua aplicação no CADU, um veículo autônomo terrestre desenvolvido na UFMG. As próximas seções descrevem o CADU e suas características, a aplicação da arquitetura de Hardware e de Software neste veículo e, por fim, os resultados obtidos.

A introdução da arquitetura proposta no Capítulo 3 é aplicada ao CADU de forma a manter todos os sistemas funcionais da arquitetura anterior principalmente na camada de controle, onde o sistema é o mais crítico, ou seja, sem a necessidade de criação de novos nós sensores ou atuadores ou até mesmo retrabalho nas adaptações introduzidas no veículo como os sistemas de motores para acionamento dos freios e do câmbio, o *bypass* de controle de aceleração e o sensor de velocidade das rodas presente no veículo. Durante o desenvolvimento deste trabalho, também foi buscada em todo momento a compatibilidade entre o que já existia e o que estava sendo criado de forma a não inviabilizar o uso do veículo neste intervalo.

4.1 CADU

O CADU é um veículo Chevrolet Astra ano 2003/2004 equipado com câmbio automático e direção hidráulica, que desde 2007 vem sofrido adaptações para se tornar um veículo autônomo. As adaptações, que incluem intervenção no sistema de aceleração e introdução de motores para controle dos sistemas de Câmbio, Direção e Freios, são mostradas em [Freitas et al., 2009]. O principal elemento nesta estrutura é a placa de desenvolvimento baseada em Microcontrolador PIC que, por meio de suas interfaces, permite tanto o controle dos atuadores no veículo quanto a coleta de sinais. Esta placa é mostrada na Figura 4.1. O conjunto formado pelos elementos de intervenção do veículo, juntamente com o responsável pelo processamento, constitui a Arquitetura de Hardware

do CADU. Os protocolos de comunicação, os softwares de controle e de fluxo de dados são a arquitetura de Software do CADU.

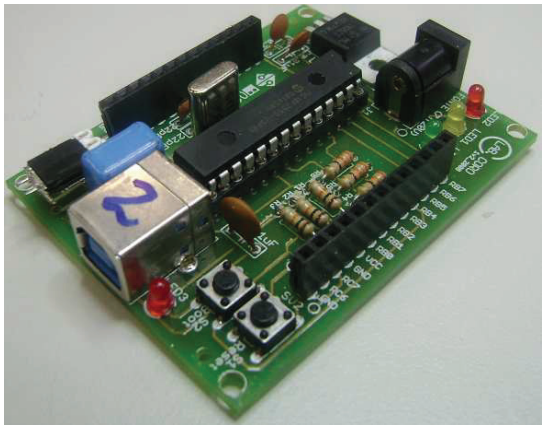

Figura 4.1: Primeira versão da Placa de Desenvolvimento baseada no Microcontrolador PIC18F2550 com interface USB, desenvolvida por Elias para os primeiros testes com o Carro Autônomo.

4.1.1 Arquitetura de Hardware do CADU

A Arquitetura de Hardware do CADU, antes da execução desta dissertação, é mostrada na Figura 4.2 e é baseada na centralização das informações e comunicação serial entre os dispositivos e um único computador central.

Esta arquitetura, apesar de funcional, possui falhas de estrutura que a tornam frágil do ponto de vista de robustez e integridade. O principal problema do ponto de vista de Hardware é a limitação imposta pela utilização de um modelo de comunicação baseado em Serial/USB. Este tipo de arquitetura torna o sistema instável a medida em que novos elementos são adicionados. É necessário um grande número de portas USB no sistema e uso de concentradores ou *Hubs* externos para permitir as ligações. Também é grande a quantidade de cabos necessários a tantas ligações, o que leva a redução do espaço disponível e dificulta alterações no sistema.

Outro ponto importante é que neste modelo de arquitetura, todo o sistema, desde os elementos de níveis mais baixos até os de maior nível estão ligados diretamente ao Computador Central dificultando assim a inserção de um elemento adicional para monitoramento e geração de diagnóstico ou até mesmo redundância de alguns sistemas.

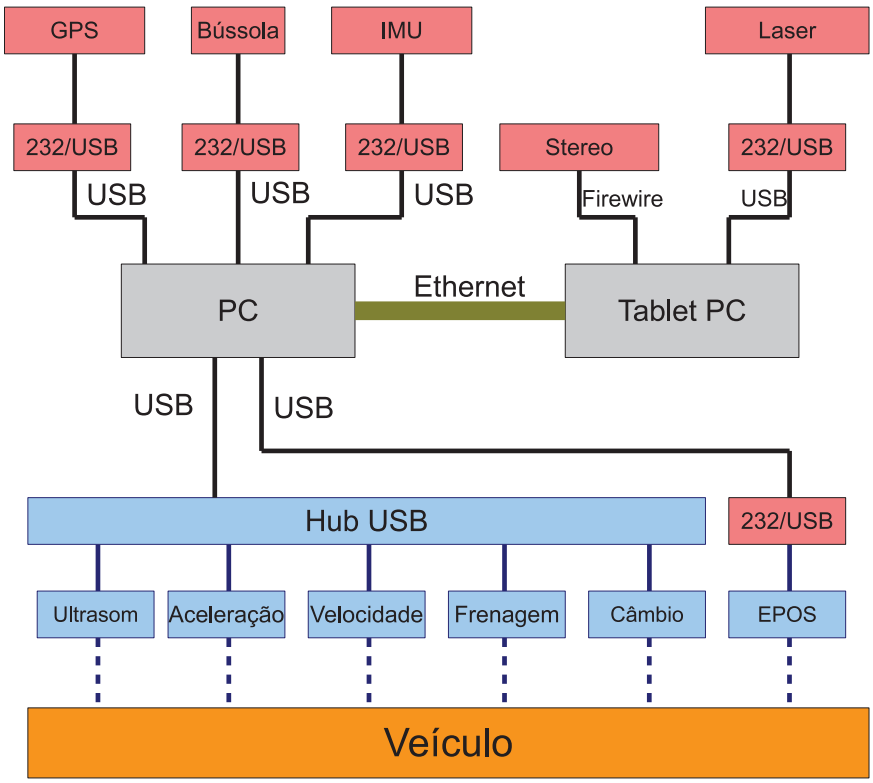

Figura 4.2: Antiga Arquitetura de Hardware do CADU obtida de [Freitas et al., 2009, de Lima, 2010].

4.1.2 Arquitetura de Software do CADU

A arquitetura de software antiga é mostrada na Figura 4.3 e baseava-se no envio autônomo de informações. Na prática, cada um dos sensores enviava periodicamente a sua informação para o computador central e este poderia ler ou simplesmente ignorar estes dados, enquanto que para os atuadores os comandos eram enviados na forma de valores de referência codificados em caracteres hexadecimais compreendidos entre 0x20 e 0x7F devido a necessidade dos mesmos serem caracteres imprimíveis reconhecidos no padrão ASCII. Esta escolha foi feita para permitir o uso de um programa de comunicação serial tipo terminal de digitação, como o Microsoft Windows Hyperterminal ou outro similar, quando do desenvolvimento dos primeiros sistemas do carro autônomo. Devido a este padrão de arquitetura de software, o sistema como um todo estava sujeito a uma série de limitações listadas abaixo:

- Comunicação Não-Determinística devido ao modelo de comunicação USB.
- Impossibilidade de utilização de Sistema Operacional de Tempo Real para controle

de Baixo Nível dado que a maioria dos sistemas operacionais de tempo real não tem suporte a USB.

- Instabilidade da arquitetura de software devido ao efeito cascata na falha de um dos gerenciadores ou *drivers* de comunicação USB/Serial.
- Sobrecarga do Computador Central pela dificuldade da divisão de tarefa com outros computadores.
- Comunicação serial baseada em caracteres ASCII sem verificação de erro, tornando a comunicação ineficiente.
- Difícil alteração no sistema devido ao modelo centralizado de processamento.

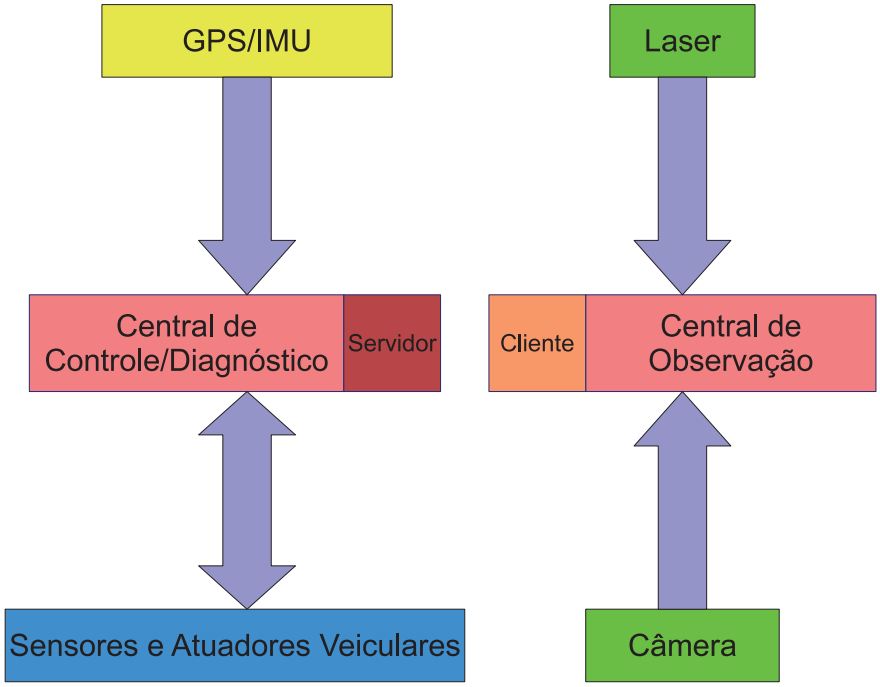

Figura 4.3: Antiga Arquitetura de Software do CADU, obtida de [de Lima, 2010]

4.2 Nova Arquitetura de Hardware

Esta seção apresenta as modificações realizadas no hardware do CADU para implementação da arquitetura proposta. O Hardware do CADU, assim como a arquitetura proposta, foi dividido em três níveis sendo que cada uma das subseções seguintes explica de forma detalhada o que foi implementado e quais os elementos dispostos em cada parte da arquitetura.

4.2.1 Nível de Controle

O Nível de Controle pode ser considerado o nível mais crítico para o funcionamento de um sistema autônomo. Isto porque é neste nível que se encontram os atuadores necessários para o efetivo controle do veículo. Fazem parte do Nível de Controle do CADU o Sistema de Frenagem, Aceleração, Câmbio, Direção e Velocidade.

Para a interação entre os nós de sensoriamento e atuação e o carro, nenhuma alteração foi feita no Hardware. No entanto, para a comunicação entre a central de processamento e os nós sensores e atuadores do veículo foi escolhida a rede Modbus sobre camada física RS-485. O Projeto Final de Curso [Arruda, 2009] mostra em detalhes as características e o funcionamento do protocolo Modbus sobre camada Física RS-485.

Esta escolha permitiu a introdução de um hardware capaz de oferecer determinismo, ao mesmo tempo em que o cabeamento RS-485 permite a ligação de até 31 nós em rede *Daisy Chain* com grande rejeição a ruídos eletromagnéticos. De forma a manter a compatibilidade entre o sistema anterior e o atual, foi desenvolvido o Nó Modbus/485 utilizando-se o mesmo microcontrolador e a mesma base de placa mostrada na Figura 4.1 sendo, somente, acrescentados conectores de comunicação e o conversor TTL/RS-485, como pode ser visto na Figura 4.4. O esquema de conversão de sinais utilizado pela placa é mostrado na Figura 4.5 na qual o sinal recebido por meio de alguma entrada no Microcontrolador é processado e preparado para envio à Rede Modbus/485 por meio do conversor TTL/485.

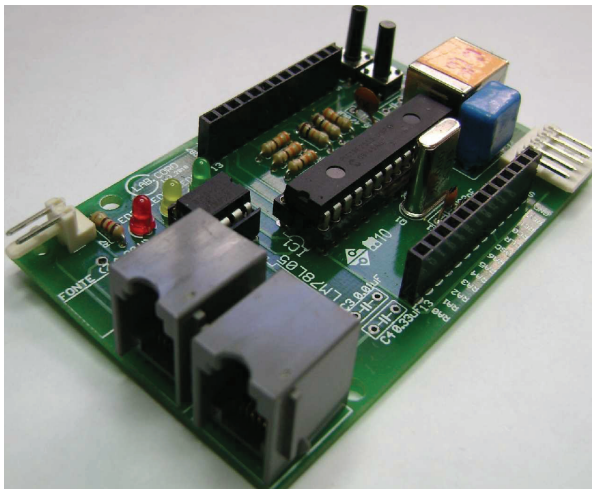

Figura 4.4: Placa Microporocessada com interface para rede Modbus/485.

A rede apresentada foi montada com cabo de par trançado dois fios e o modo como as placas foram montadas permite que qualquer um dos elementos seja o terminador da

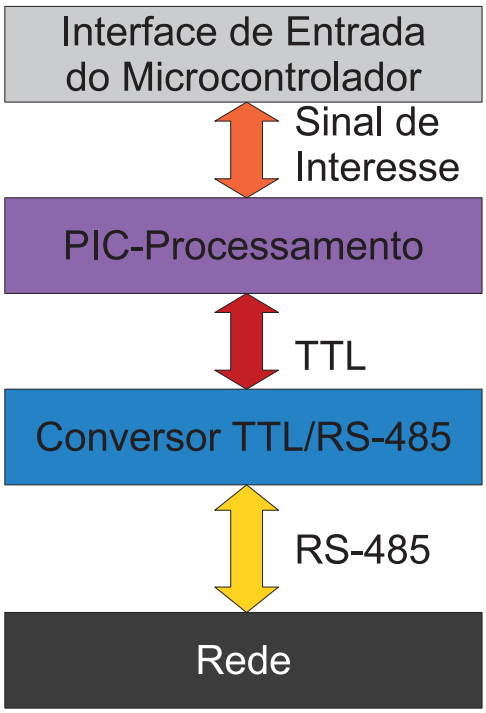

Figura 4.5: Caminho dos sinais de comunicação na Placa de Desenvolvimento. O sinal passa, tanto de recepção quanto de envio passa por dois elementos principais que correspondem a recepção do sinal de interesse e conversão da resposta final de Serial TTL para RS-485.

rede necessitando apenas fechar a conexão ou *jumper* terminador na placa.

Nesta camada nem todos os nós estão ligados em rede. Este é o caso da integração do sistema EPOS [Maxon, 2011], sistema este responsável por movimentar o volante do carro, que foi ligado em uma das portas seriais nativas do Computador Central.

O PC central do Nível de Controle da Arquitetura de Hardware é um Intel Atom de 1Gb de RAM e 40 Gb de SSD *(Solid State Disk)* que utiliza um conversor RS-232/485 para comunicação com elementos da rede Modbus, interface RS-232 para comunicação com a EPOS, conversor USB/485 para fornecer dados ao Nível de Processamento e interface Ethernet para comunicação com o Nível Usuário da arquitetura.

Do modo como o sistema é montado, para que o veículo passe de autônomo a normal, basta que se desabilite os nós atuadores, seja por hardware ou até mesmo por software.

O Nível de controle em detalhe é mostrado na Figura 4.6.

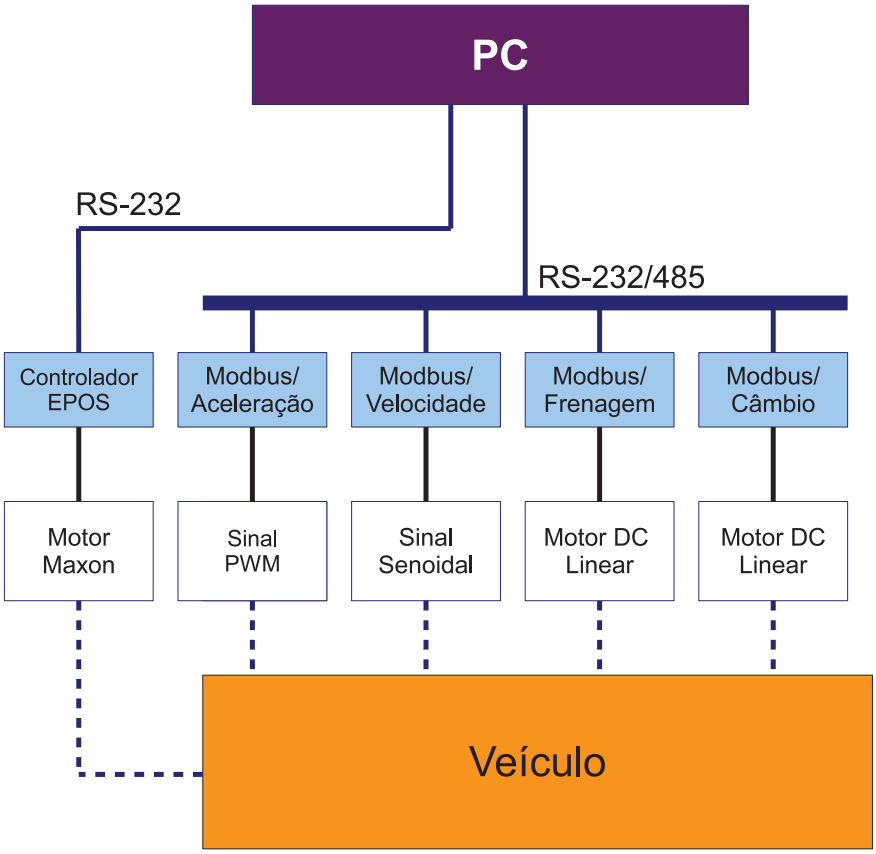

Figura 4.6: Nova Arquitetura de Hardware do Nível de Controle do CADU, mostrando em linha tracejada os elementos diretamente em contato com o veículo e em linha contínua a ligação serial RS-232 e RS 232/485 e o restante dos elementos.

4.2.2 Nível de Processamento

O Nível de processamento do CADU é estruturado em três sistemas sendo eles o de Movimentação, Localização e Reconhecimento.

No Sistema de Movimentação, por questão de conveniência, o PC utilizado é o mesmo do Nível de controle, sendo sua separação visível apenas a nível Lógico. As informações que este sistema obtém do nível de Controle são processadas e preparadas para envio ao nível de Usuário.

No Sistema de Localização o processamento é realizado por um PC com características idênticas ao do Nível de Controle. Trata-se de um PC Intel Atom de 1Gb de RAM, SSD de 40Gb e Interfaces Serial, Ethernet e USB. Esta central é a responsável por receber informações da Rede Modbus/485 que contém como elementos os nós responsáveis pela coleta da informação do GPS, da IMU e da Bússola.

De forma a aproveitar as vantagens de uma comunicação em rede evitando os inconvenientes causados por múltiplas ligações seriais em um único PC, todos os elementos do sistema de localização da camada de Processamento foram interligados na Rede Modbus/485. No entanto, uma vez que a comunicação do GPS e demais elementos é serial nativa, para que fosse possível a comunicação entre eles e a Rede Modbus/485 foi necessário a confecção de um novo tipo de nó, que por meio de dois microcontroladores, obtém os dados serialmente destes elementos e envia a informação a um outro microcontrolador que enviará estas informações a Rede. Este procedimento foi necessário pois os microcontroladores utilizados possuem somente uma interface serial de hardware, e esta, é utilizada na Rede Modbus/485. Outra opção seria a utilização de Serial emulada em software, no entanto esta solução não é tão robusta quanto uma solução de Hardware por não permitir o uso de interrupções para controle do fluxo de dados. O esquema deste sistema é mostrado na Figura 4.7 e consiste em receber as informações por meio de um nó com serial e repassar esta informação por meio do protocolo I2C para a placa de comunicação responsável por conversar com a rede. A Figura 4.8 mostra o resultado final deste nó denominado Modbus/485/I2C.

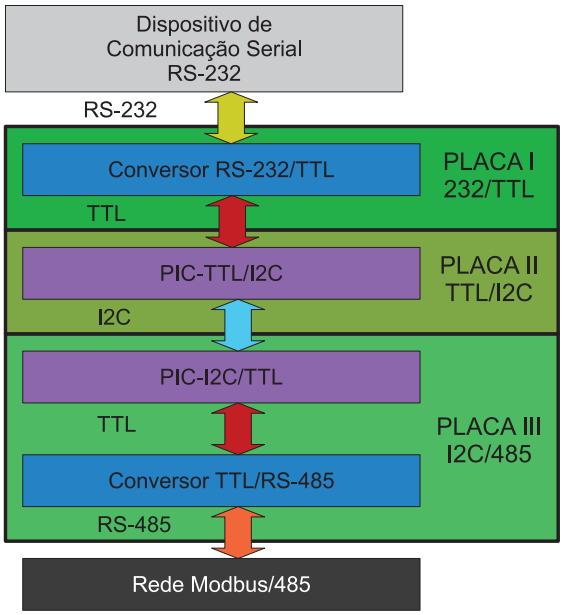

Figura 4.7: Caminho dos sinais de comunicação na Placa de Desenvolvimento com Nó I2C. Inicialmente o sinal recebido é RS-232 que é convertido em TTL na Placa I, processado e enviado via I2C pela Placa II e por fim processado e enviado a rede Mobus/485 por meio da Placa III.

Além destes elementos, o Sistema de Localização utiliza as informações de Sensor de Velocidade e Posição da Direção fornecidas pelo Nível de Controle por meio de uma ligação

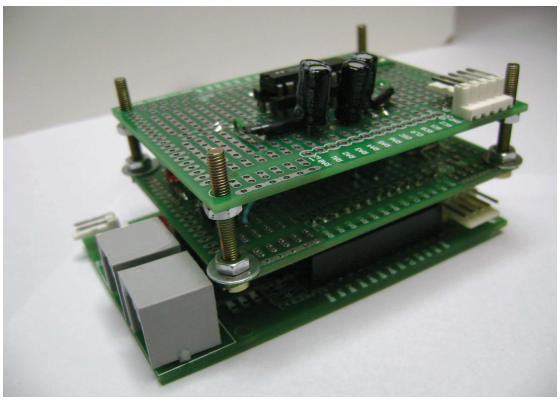

Figura 4.8: Placa de Desenvolvimento com comunicação entre o dispositivo de comunicação serial e a rede Modbus/485, passando pelas conversões RS-232/TTL, TTL/I2C, I2C/TTL, TTL/RS-485.

Modbus/RS-485 que torna o PC do Nível de Controle um Nó do Nível de Processamento.

Para o Sistema de Reconhecimento o Hardware implementado consiste atualmente apenas de um PC com a mesma configuração dos anteriores e um Laser SICK [SICK, 2011] ligado ao mesmo por meio de comunicação RS-422.

O esquema final para o Nível de Processamento é mostrado na Figura 4.9. Todos os elementos processadores neste nível necessitam ter uma interface de comunicação com o Nível de Usuário que no caso do CADU é a interface Ethernet fornecida por meio de um Roteador. Neste nível é possível se introduzir tantos subsistemas quanto a interface Ethernet permite, uma vez que eles são independentes, e, como será visto na arquitetura de Software, não se corre o risco de sobrecarregar a rede apenas introduzindo novos elementos de Hardware.

4.2.3 Nível de Usuário

O Nível de usuário, do ponto de vista de arquitetura de hardware, é o menos crítico pois sua especificação e utilização são dependentes exclusivamente da atividade que os usuários vão realizar. E, como será mostrado adiante, a arquitetura de software torna a escolha deste hardware independente de qualquer parâmetro utilizado nos níveis abaixo.

Como a arquitetura de Hardware utiliza um roteador sem fio, a ligação entre o nível Usuário e o Nível de processamento pode ser feita por cabos ou via rádio e pode utilizar quaisquer sistemas que permitam a comunicação entre dispositivos por meio de acesso via telnet, ou fazendo uso de algum outro protocolo cliente-servidor como será

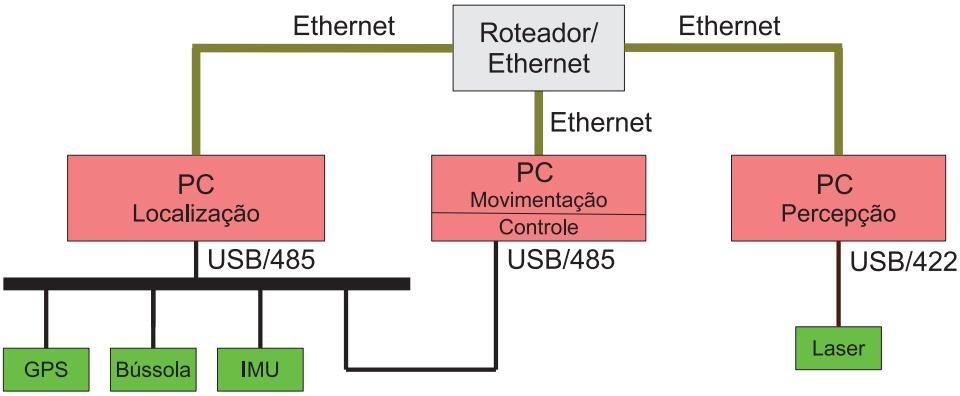

Figura 4.9: Nível de Processamento contendo os sistemas de Localização, Movimentação e Reconhecimento, juntamente com as interfaces RS-485, RS-422 e Ethernet. Não são mostrados o nós responsáveis por converter o sinal RS-232 dos sensores em sinal RS-485 da Rede Modbus.

mostrado adiante. Neste nível podem ser utilizados PDA's, iPads, Tablets, PCs e até mesmo Smartphones para processar os dados provenientes da camada de processamento e enviar comandos ao veículo. A limitação para a utilização destes dados é dada pela arquitetura de Software que estabelece um padrão específico para comunicação entre os elementos.

Para o CADU, o Hardware atual do Nível Usuário corresponde ao TabletPC Toshiba, Core2Duo com 2Gb de RAM e 80Gb de HD.

4.2.4 Monitor

Como indicado no Capítulo 3, a arquitetura de Hardware permite a introdução de um sistema de supervisão para coleta de dados e diagnóstico. O Monitor introduzido no CADU está inserido de modo a obter informação das fontes, sendo elas a Rede Modbus do Nível de Controle, a Rede Modbus do Nível de Processamento e as informações disponibilizadas na Ethernet. O computador responsável por esta coleta tem a mesma configuração do PC de Controle e possui dois conversores USB/485. Estes conversores são a conexão física existente entre as redes e o Supervisor.

A Figura 4.10 mostra o Hardware de Supervisor juntamente com a arquitetura Completa de Hardware aplicada ao CADU.

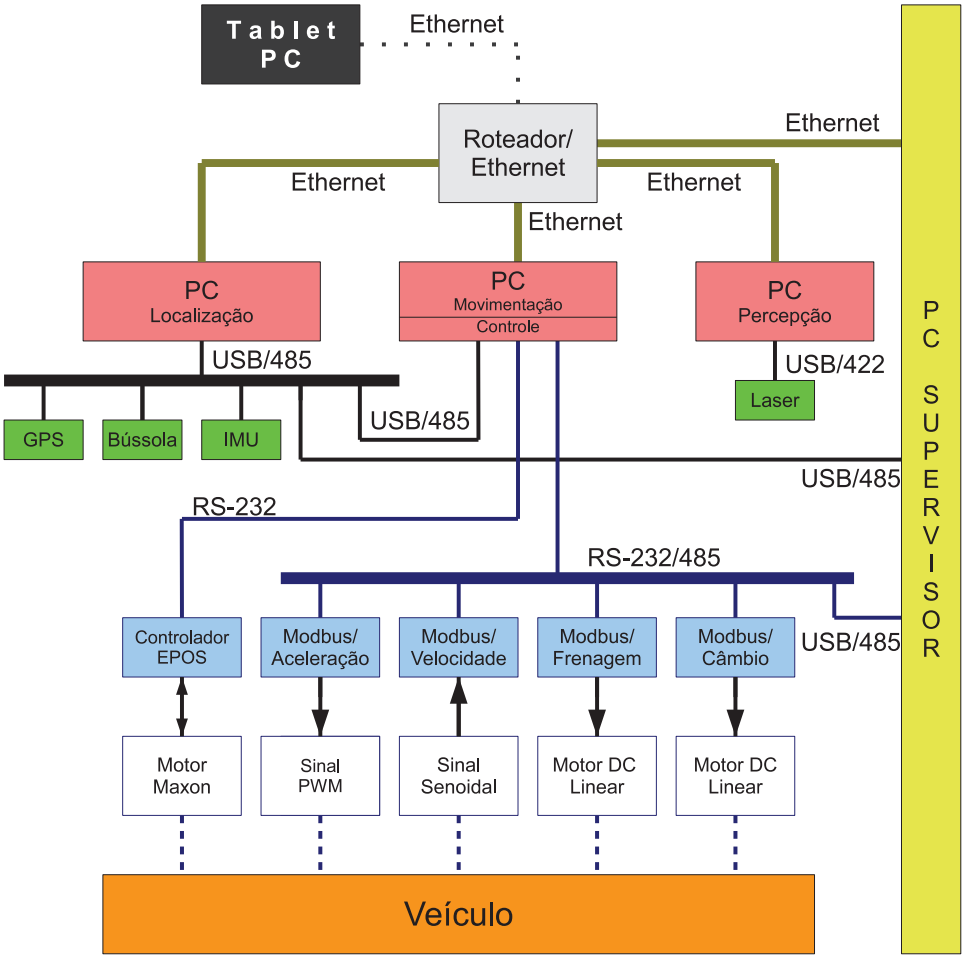

Figura 4.10: Nova Arquitetura Completa de Hardware para o Carro Autônomo da UFMG. As linhas contínuas representam comunicação via cabo, as linhas tracejadas indicam contato direto com o veículo e a linha pontilhada representa comunicação sem fio.

4.3 Nova Arquitetura de Software

Para o Software implementado no CADU também houve, de acordo com a arquitetura de software proposta, uma divisão em três Níveis. As próximas subseções descrevem detalhadamente cada um dos níveis, as características e os algoritmos implementados. As linguagens utilizadas foram C para os módulos de Tempo Real uma vez que o mesmo não possui suporte para C++, e para o restante dos programas foram utilizadas as linguagens C/C++.

4.3.1 Nível de Controle

O Nível de Controle da arquitetura de software foi desenvolvido para se adequar às propostas feitas no capítulo anterior. Para tanto, foram elaborados algoritmos que visam, além de uma flexibilidade do ponto de vista de inserção e remoção de funcionalidades, o cumprimento dos requisitos temporais que este nível exige. Há duas classes de algoritmos sendo executadas no nível de Controle:

- Software Mestre-Escravo para execução das rotinas de Controle do Carro Autônomo.
- Software de Publicação de Mensagens para o nível de Processamento.

A primeira classe de software corresponde a implementação do protocolo da rede nos nós Modbus/485, na qual foi possível construir um software determinístico do tipo mestre escravo por meio das funcionalidades oferecidas pelo protocolo Modbus. Este protocolo e sua implementação como rede de instrumentação é explicada no trabalho [Arruda, 2009] e os requisitos temporais impostos pela arquitetura se mostram favoráveis na adoção desta rede como mostrado no trabalho [Arruda et al., 2011] onde é levantado um modelo de atraso para o sistema baseado em rede Modbus, juntamente com as frequências máximas que podem ser obtidas com esta implementação em função dos parâmetros adotados. Os principais pontos deste estudo se encontram no Apêndice A. O protocolo de Mensagens Mestre/Escravo para os elementos presentes na rede Modbus é mostrado na Figura 4.11.

Para cada um dos nós das rede foi desenvolvido um algoritmo específico visando a obtenção de um software que permitisse a fácil inserção e remoção de funcionalidades mesmo no nível de controle. Todos os nós microcontrolados da rede Modbus do Nível de Controle implementam o algoritmo escravo e possuem um endereço único na rede para comunicação individual com o Mestre e um endereço de *broadcast*. Uma hierarquia de

Quadro de Mensagem do Mestre					
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Endereço	Função	Parâmetro1	Parâmetro2	CRC_MSB	CRC_LSB

Quadro de Mensagem Sem Parâmetro do Escravo			
Byte 1	Byte 2	Byte 3	Byte 4
Endereço	Função	CRC_MSB	CRC_LSB

Quadro de Mensagem Com Parâmetro do Escravo							
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Endereço	Função	Parâmetro1	Parâmetro2	Parâmetro3	Parâmetro4	CRC_MSB	CRC_LSB

Figura 4.11: Estruturas de Dados Mensagens de Rede do CADU.

funções existe por meio de uma tabela criada na qual são estabelecidas que funções cada um dos nós pode executar e quais as solicitações os mesmos podem responder. Esta tabela é mostrada em formato resumido na Figura 4.12.

Este tipo de abordagem permite que os novos sensores ou atuadores criados sejam facilmente incorporados ao sistema bastando que para isto sejam programados com as funções comuns a todos os nós como capacidade de recebimento de mensagens do tipo *broadcast*, resposta diante de erros, e a função de *reset*.

O desenvolvimento de novas funcionalidades no nível de controle é facilitada pois as mensagens enviadas aos nós foram padronizadas e a sua utilização em um sistema operacional de tempo real permite o desenvolvimento de sistemas de controle em malha fechada. Isto é, a criação de um método de controle específico como o de velocidade ou direção é facilitado devido a disponibilidade tanto das medidas quanto dos comandos necessários fornecidos por meio de mensagens padronizadas.

A EPOS, que é o dispositivo controlador do atuador da direção do CADU, ponto de vista do hardware, não se encontra na rede Modbus mas sim em uma outra porta serial do computador. No entanto, do ponto de vista de software, a mesma está no mesmo nível que os nós Modbus, inclusive, sendo acessada periodicamente com o mesmo intervalo de tempo que todos os outros elementos presentes na rede.

O Mestre da Rede é um PC cujo sistema operacional é de tempo real. Sua função é se comunicar ciclicamente com todos os nós, tanto da rede, quanto outros que não estejam na rede, estabelecendo assim o circuito fechado de pergunta/resposta sobre o qual o controle do sistema deve ser construído.

Protocolo do CADU					
MESTRE					
Função	Tipo	Quem Implementa	Parametro1	Parametro2	Resposta do Escravo
0x00	RESERVADA				
0x01	RESET	TODOS	N	N	Não Há
0x1F	RESERVADA				
0x20	Solicita Velocidade	Sensor de Velocidade (ModVelo)	N	N	Velocidade 2 Rodas em 32 bits/Nack
0x30	Envia Set-Point Acelerador	Atuador de Acelerador(ModAcel)	SetPoint(8bits)	N	Ack/Nack
0x40	Envia Set-Point Freio	Atuador de Freio(ModFreio)	SetPoint(8bits)	N	Ack/Nack
0x41	Envia EMERGENCIA	Atuador de Freio(ModFreio)	N	N	Ack/Nack
0x50	Envia Set-Point Cambio	Atuador de Cambio(ModCambio)	SetPoint(8bits)	N	Ack/Nack
0x54	Solicita Roll/Pitch	IMU (ModIMU)	N	N	Roll/Pitch
0x55	Solicita Yaw/Accel_x	IMU (ModIMU)	N	N	Yaw/Accel_x
0x56	Solicita Accel_y/Accel_z	IMU (ModIMU)	N	N	Accel_y/Accel_z
0x60	Solicita Hora UTP	GPS(ModGPS)			Hora UTP
0x61	Solicita Latitude	GPS(ModGPS)			Latitude
0x62	Solicita Longitude	GPS(ModGPS)			Longitude
0x63	Solicita N. Satelies	GPS(ModGPS)			N. Satelies
0x64	Solicita Diluição GPS	GPS(ModGPS)			Diluição GPS
0x65	Solicita Altitude	GPS(ModGPS)			Altitude
0x66	Solicita Velocidade X	GPS(ModGPS)			Velocidade X
0x67	Solicita Velocidade Y	GPS(ModGPS)			Velocidade Y
0x68	Solicita Velocidade Z	GPS(ModGPS)			Velocidade Z
0x6F	Solicita Setpoint Generico	Placa Setpoint (ModSetpoint)	N	N	Valor entre 0x0000 e 0xFFFF
0x70	Solicita Status	Todos	N	N	Status
0x71	Ativa Modulo	Todos os Atuadores	N	N	Ack/Nack
0x72	Desativa Módulo	Todos os Atuadores	N	N	Ack/Nack
ESCRAVO					
0x80	Ack	Todos os Atuadores	**Todas as Respostas do Escravo contidas entre 0x80 e 0x9F correspondem a**		
0x81	Nack - CRC Errado	Todos	**pacotes de 4 Bytes do Modbus.**		
0x82	Nack - Ocupado	Pertinentes			
0x83	Nack - Sem Implementação	Pertinentes			
0x84	Nack - Mal Funcionamento	Pertinentes			
0x85	Nack - Inexplicável	Pertinentes			
0x90	Status - ATIVO	Todos			
0x91	Status - INATIVO	Todos os Atuadores			
0x92	Status - OCUPADO	Pertinentes			
0x93	Status - EMERGENCIA	Pertinentes			
0x94	Status - INDETERMINADO	Pertinentes			

Figura 4.12: Tabela de Funções do CADU.

Para o CADU a escolha do sistema operacional recaiu sobre o Linux e o sistema de tempo real escolhido foi o RTAI (*Real Time Application Interface*) [RTAI, 2011] por ele ser de código aberto ou *open-source*, por ser melhor em termos temporais que o Xenomai [Barbalace et al., 2008] e para manter a compatibilidade com outros sistemas desenvolvidos no laboratório. O sistema operacional utilizado foi o Ubuntu 10.04.1 com kernel 2.6.31.8 e *patch* RTAI 3.8. Ao contrário de um programa comum, o algoritmo de tempo real é denominado módulo, e sua execução ocorre a nível de Kernel ciclicamente, por tempo indeterminado, até que o usuário o remova do sistema.

A Figura 4.13 mostra um diagrama simplificado com a sequência de execução dos algoritmos Mestre-Escravos utilizados no Nível de Controle, onde o Mestre, ao executar alguma rotina como comando de velocidade ou ângulo do volante, necessita enviar esta mensagem ao nó escravo ou a EPOS. O início da execução do algoritmo do mestre se dá pela ocorrência de uma interrupção de tempo avisando ao módulo de tempo real que é hora de executar o fluxograma. No nó escravo é executada uma função padrão que pode ser desde a leitura da velocidade das rodas até o envio de comandos para o acelerador, câmbio ou freios. O nó escravo sofre uma interrupção de Hardware todas as vezes em que o Mestre deseja se comunicar com este nó. Diante deste cenário, o nó escravo pára momentaneamente de executar a rotina principal para atender ao Mestre e responder a sua requisição.

A segunda Classe de Algoritmos corresponde aos softwares responsáveis pelo trânsito da informação entre o Nível de Controle e o Nível de Processamento. Este conjunto de processos tem duas funções bem definidas:

- Recepção de Comandos - Corresponde a receber comandos do Nível Usuário e decodificar esta informação para enviar os comandos necessários aos nós do Nível de Controle. Esta função é executada pelo software de Movimentação rodando em Nível de Processamento que transmite ao Nível de Controle os comandos de Velocidade desejada e ângulo do volante desejado.

- Envio de Informações - Equivale a receber as informações dos Nós no nível de controle e disponibilizar esta informação para que o Nível de Processamento tenha acesso. As informações disponíveis no nível de Controle são de velocidade do veículo (obtido por meio de sensor localizado nas rodas) e de ângulo do volante atual. As mesmas são transmitidos ao nível de Processamento tanto para a o bloco de Movimentação quanto de Localização.

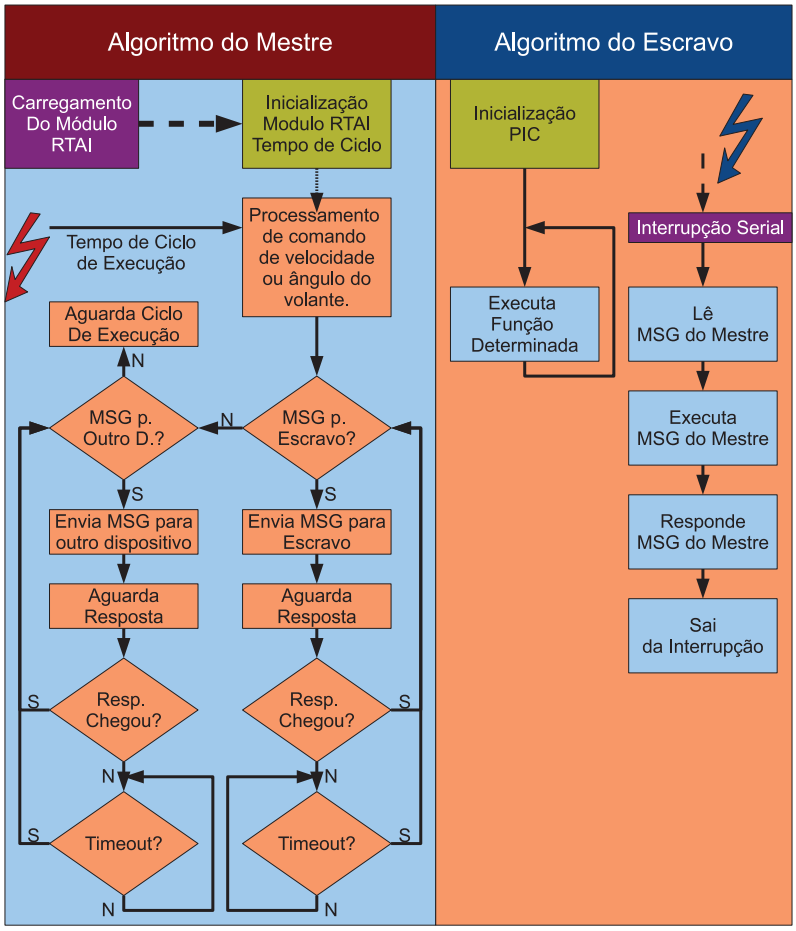

Figura 4.13: Algoritmo Mestre-Escravo do Nível de Controle. No escravo, a seta em forma de raio representa uma interrupção de Hardware responsável por atender a solicitação do mestre. No Mestre, o raio indica uma interrupção de Software indicando que é hora de executar a rotina contida no fluxograma.

Apesar de intuitiva, tanto a recepção quanto o envio de dados para um sistema de tempo real necessitam de um artifício denominado *pipe*. O *pipe* é o meio pelo qual é possível a transferência de dados entre um módulo de tempo real e um programa não-tempo real dentro do RTAI sem que as características de tempo real sejam perdidas. O espaço no qual o módulo de tempo real executa é denominado *Kernel Space* enquanto o local onde algoritmos comuns são executados é denominado *User-Space* [Corbet et al., 2005]. A Figura 4.14 mostra o esquema completo para o nível de controle do tráfego de informações. Neste esquema, os valores de referência do acelerador, freio, câmbio e ângulo do volante são lidos de um *pipe* de leitura e as informações são registradas dentro do módulo de tempo real. De igual forma, este mesmo módulo de tempo real escreve as informações de velocidade e ângulo do volante em um *pipe* de escrita que é a interface com o programa executado em *User Space*. O algoritmo executado em *User Space* é responsável pelo processamento destes dados e pelo controle da interface existente entre este Nível (Controle) e o Nível de Processamento na troca de informações.

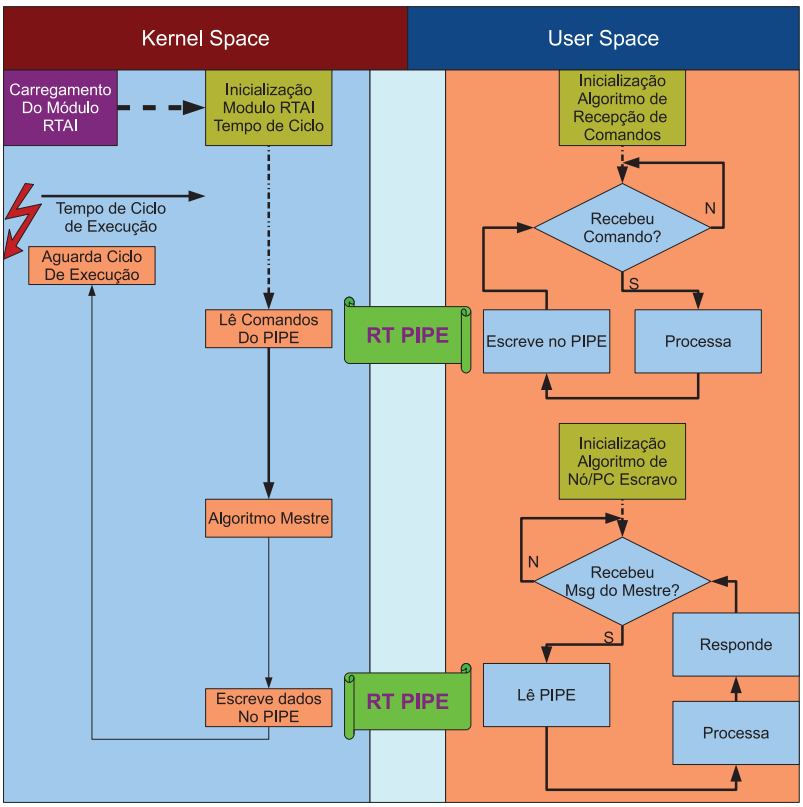

Figura 4.14: Tráfego de dados no Nível de Controle. Há dois algoritmos sendo executados, um em *Kernel Space* e outro em *User Space* que trocam informações por meio de *pipes* de leitura e escrita. A seta em forma de raio indica uma ação iniciada por interrupção de tempo.

4.3.2 Nível de Processamento

O Nível de Processamento é o responsável pela interface entre todos os sistemas do CADU e o usuário final, e é caracterizado por necessitar da transferência de mais dados, ao mesmo tempo em que requer uma frequência menor de transmissão, dada a natureza das informações e a sua utilização.

De forma a tornar a interface intuitiva, e permitir ao usuário o controle do Veículo Autônomo como se fosse um sistema robótico comercial foi introduzido no Nível de Processamento da Arquitetura de Software do CADU a API(*Application Programming Interface*) Player [Gerkey et al., 2003].

O Player, ou "*The Player Project*" corresponde a um arcabouço de software livre que auxilia pesquisas em robôs e sistemas de sensores [Stoy and Hedges, 2010]. O servidor Player é provavelmente uma das interfaces de robôs mais utilizadas no mundo. O modelo Cliente/Servidor do Player permite o desenvolvimento de programas de controles de robôs em diversas linguagens para rodar em qualquer computador que possua uma conexão de rede com o robô. Além disso, o Player permite o suporte a múltiplas conexões de clientes aos dispositivos, criando novas possibilidades para sensoriamento e controle, distribuído e colaborativo. O Player provê uma interface de rede com uma grande variedade de sensores e robôs e, neste trabalho, utiliza-se esta mesma interface para o Carro Autônomo. Isto permite que por meio de interfaces conhecidas, requisições e comandos possam ser facilmente enviadas ao sistema do veículo.

Utilizando-se desta interface e visando a transferência e processamento de dados, cada um dos subsistemas do veículo foi concebido conforme descrito a seguir.

Para o sistema de Movimentação do Nível de Processamento, o Computador Central do ponto de vista de Hardware é o mesmo do Nível de controle. No entanto, sob a ótica de Software, eles são independentes um do outro, exceto pela informação que trafega entre eles. Isto se dá pois o driver Player para comunicação com o nível de usuário é um programa que é executado em paralelo com o módulo controlador, mas não no mesmo fluxo de programa. As informações que trafegam do Nível de usuário para o Nível de Processamento consistem somente dos dados associados a uma interface do Player denominada *position2d* que envia ao veículo os comandos de velocidade e ângulo do volante. Já do Nível de Processamento para o Nível de Controle a implementação feita no CADU corresponde as conversões dos dados recebidos de valor de referência de velocidade e ângulo do volante.

Por meio do Player, também é possível habilitar ou desabilitar quaisquer sensores e atuadores da arquitetura desde que esta implementação seja feita na interface dos subsistemas do nível de Processamento, permitindo assim que mesmo estando no Nível Usuário, o usuário tenha a opção de escolher quais dispositivos não serão utilizados em algum experimento. A Figura 4.15 mostra o fluxo de informações e o papel do algoritmo implementado na transmissão de informação de um nível a outro. Este algoritmo inicia um Servidor Player e em seguida entra em um *loop* no qual aguarda um determinado comando do Nível de Usuário para enviá-lo ao Nível de Controle.

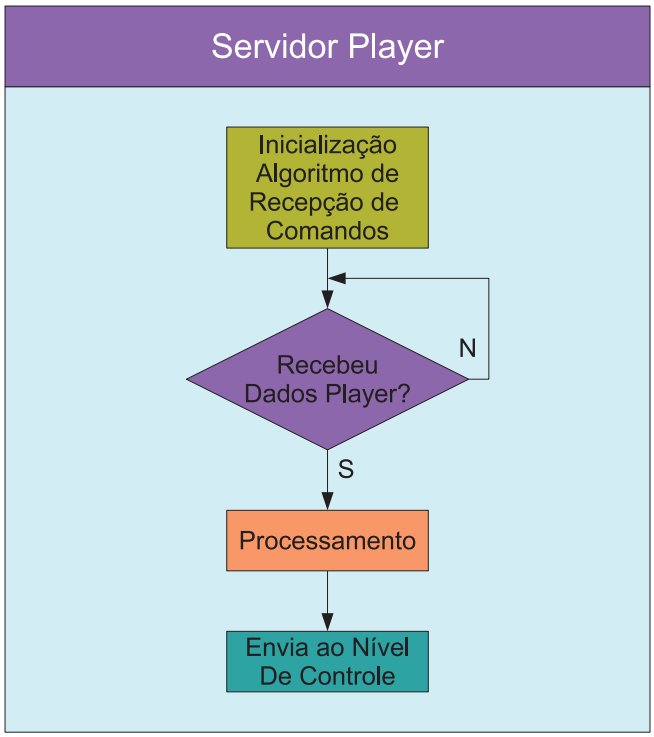

Figura 4.15: Software presente no Nível Processamento, no sistema de Movimentação. O algoritmo inicia um Servidor Player e em seguida entra em um *loop* no qual aguarda um determinado comando do Nível de Usuário para enviá-lo ao Nível de Controle.

Para o sistema de Localização, o algoritmo é semelhante ao descrito para o sistema de Movimentação. O algoritmo de localização é responsável por coletar as informações provenientes do GPS, da Bússola, da IMU, do encoder e da posição do volante de forma a fornecer uma informação consistente da velocidade e da posição atual do veículo. Esta informação, por não depender de um controle restrito de tempo, não executa em tempo real, mas apesar disto, a comunicação baseada em Modbus/485 de tipo Mestre-Escravo foi escolhida devido as suas vantagens de controle da informação, robustez e escalabilidade fornecidos. A Figura 4.16 mostra o diagrama completo dos algoritmos que são executados

tanto no Mestre quanto nos escravos. A figura está divida em três algoritmos sendo eles da direita para a esquerda:

- Algoritmo do Escravo I2C: executa a função de coleta e processamento dos dados recebidos na interface serial e prepara os mesmos para serem enviados ao Mestre I2C quando da solicitação do mesmo. Esta solicitação ocorre por meio de uma interrupção I2C.
- Algoritmo do Mestre I2C e Escravo Modbus: executa duas funções. A primeira delas é solicitar quando necessário as informações disponíveis no Escravo I2C. A segunda função é processar estes dados e disponibilizá-los de forma que quando o Mestre Modbus requisitar alguma informação, esta esteja disponível para envio imediato. A requisição enviada pelo Mestre Modbus é detectada no Escravo Modbus por meio de interrupção de serial.
- Algoritmo do Mestre e Servidor Player: executa duas funções. A primeira delas é requisitar, quando necessário, dados do Escravo Modbus e processá-los para obtenção de alguma informação. A segunda função consiste em preparar esta informação obtida para ser enviada por meio de uma interface Player para o nível de Usuário.

4.3.3 Nível de Usuário

O processamento no nível usuário depende exclusivamente das funções que serão executadas pelo mesmo. O único requisito do ponto de vista de arquitetura de software para que haja integridade no sistema é que os elementos do Nível Usuário sejam capazes de instanciar um cliente Player e estabelecer comunicação com ele.

O Algoritmo do Nível de Usuário é chamado Cliente Player e o modelo implementado permite que em um determinado servidor sejam ligados quantos clientes se fizerem necessários. A Figura 4.17 mostra como se comporta um algoritmo de usuário final que realiza alguma tarefa.

4.3.4 Monitor

Conforme previsto na arquitetura é necessária a presença de um elemento supervisor. Este elemento corresponde a um outro PC responsável por coletar e catalogar todas as informações presentes no sistema. O supervisor foi adicionado na arquitetura do CADU e

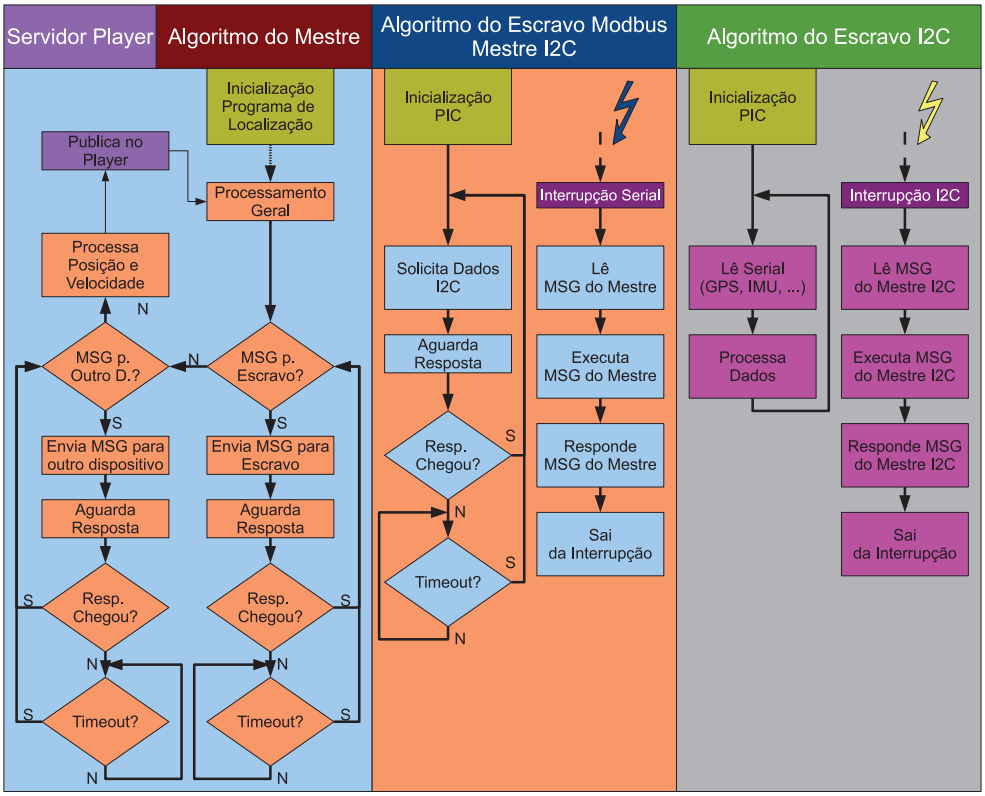

Figura 4.16: Algoritmos de Comunicação do Nível de Processamento. Nesta sequência de programas, da direita para a esquerda, se encontram: o responsável por receber os dados da serial, processá-los e disponibilizá-los por meio da I2C, o responsável por receber os dados da I2C, processá-lo e disponibilizá-lo para a rede Modbus e o último, responsável por requisitar as mensagens do nó escravo Modbus, processá-la e enviá-la ao nível de usuário por meio da interface Player. Todos os algoritmos marcados por setas em forma de raio, indicam uma ação iniciada por uma interrupção.

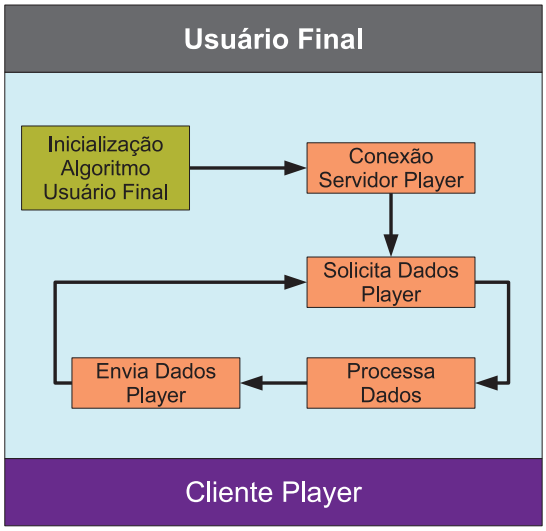

Figura 4.17: Software presente no Nível Usuário. Este algoritmo inicia uma conexão com o Servidor Player do Nível de Processamento e faz a transferência de dados tanto recebidos quanto enviados para o Nível de Processamento.

sua função é coletar os dados presentes nas Redes Modbus do Sistema (Nível de Controle e Sistema de Localização do Nível de Processamento) e também da Ethernet. Do ponto de vista de arquitetura de software, a presença deste elemento se resume ao diagrama mostrado na Figura 4.18 para cada uma das redes a qual o sistema está ligado, no qual o algoritmo lê as diversas redes presentes no sistema e gera um relatório de mensagem contendo todos os dados trafegados nas respectivas redes. No CADU o Supervisor foi implementado como sendo um elemento passivo, não interferindo nos dados trafegados em nenhuma das redes.

A Figura 4.19 mostra o resultado final para a arquitetura de Software implementada no CADU. As interfaces estão representadas nos diversos blocos.

No Nível de Controle, as interfaces Fornecedor e Consumidor de Dados se comunicam com o Concentrador de Tempo Real. O Supervisor tem acesso a estas mensagens por meio do Observador RT. Ainda neste nível, o Servidor Real-Time de Dados I/O disponibiliza os dados para quem de interesse no Nível de Processamento e para o Supervisor tem acesso por meio da interface Cliente RT.

No Nível de Processamento, a interface Cliente RT do bloco de Movimentação obtém os dados disponibilizados pela interface Servidor Real-Time de Dados I/O, enquanto que o GPS, a Bússola e a IMU por meio das interface de Fornecedor de Dados e Concentrador, disponibilizam as informações para o bloco de Localização. O Laser se utiliza das mesmas

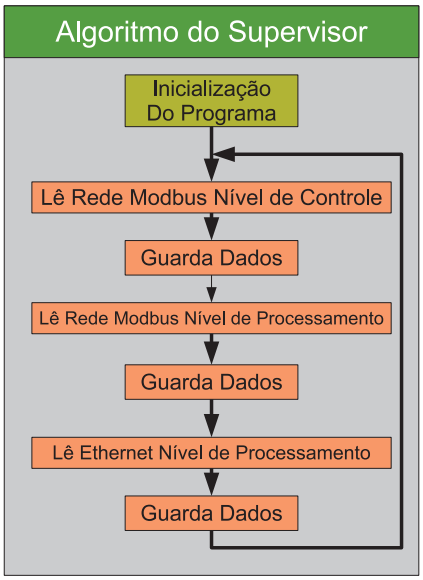

Figura 4.18: O algoritmo do Supervisor é responsável por ler os dados trafegados nas diversas redes presentes no veículo.

interfaces para se comunicar com o bloco de Reconhecimento. A interface Observador do Supervisor é a responsável por coletar os dados neste Nível. As informações geradas no Nível de Processamento são disponibilizadas ao Nível Usuário por meio da interface de Servidor Player disponível em cada bloco.

Por último, a interface Cliente Player do Nível de Usuário, se utiliza da mesma para enviar comandos e receber dados do Nível de Processamento.

4.4 Arquitetura Construída

O sistema final montado no CADU, contendo os quatro computadores, o sistema de alimentação e a ligação de todos os sensores e atuadores presentes neste capítulo é mostrado montado no porta malas do veículo, como pode ser visto na Figura 4.20. Nesta figura estão presentes todos os sistemas descritos na arquitetura, inclusive com a central de energia, responsável pela alimentação, e o sistema de comunicação responsável por conter os elementos tais como conversores USB/485, Rs-232/485 e o roteador da rede Ethernet.

Os computadores utilizados, apesar de possuírem placas mães comerciais, foram montados em laboratório para que pudessem ser empilhados um sobre o outro ocupando o menor espaço possível. O detalhe para este computador é mostrado na Figura 4.21. Este

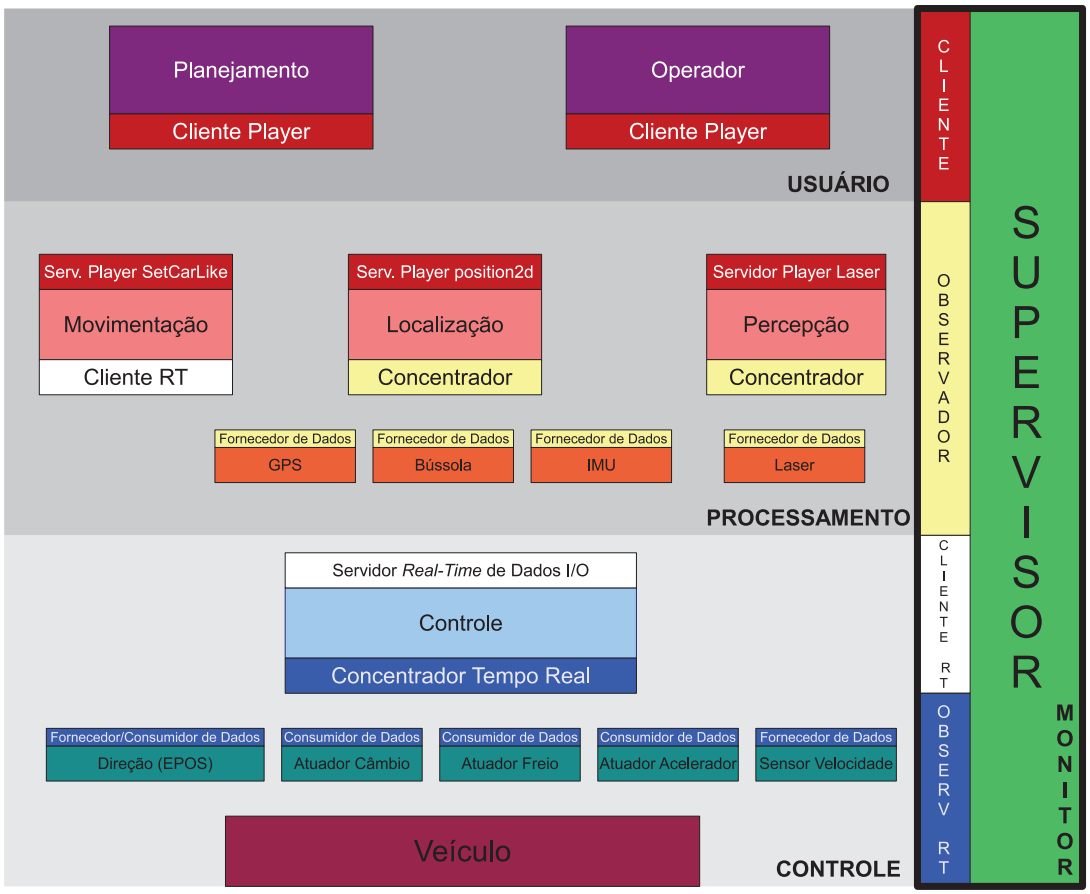

Figura 4.19: A arquitetura de Software do CADU é composta por vários blocos nos Níveis de Controle, Processamento e Usuário, além do bloco Monitor. Cada bloco possui uma ou várias interfaces, representadas pelas cores que se vêem na Figura. Cores iguais significam comunicação entre as interfaces.

Figura 4.20: Sistema completo de Hardware implementado no CADU, sendo mostrado o porta malas do CADU e da direta para a esquerda o sistema de Alimentação composto por conversores DC/DC e uma bateria, o Sistema de Comunicação e os PCs presentes em cada nível, e por último as placas responsáveis pela interface com o veículo.

mesmo estilo de caixa foi utilizado para acomodar a central de energia, a central de comunicação e os nós Modbus do Nível de Processamento e Controle.

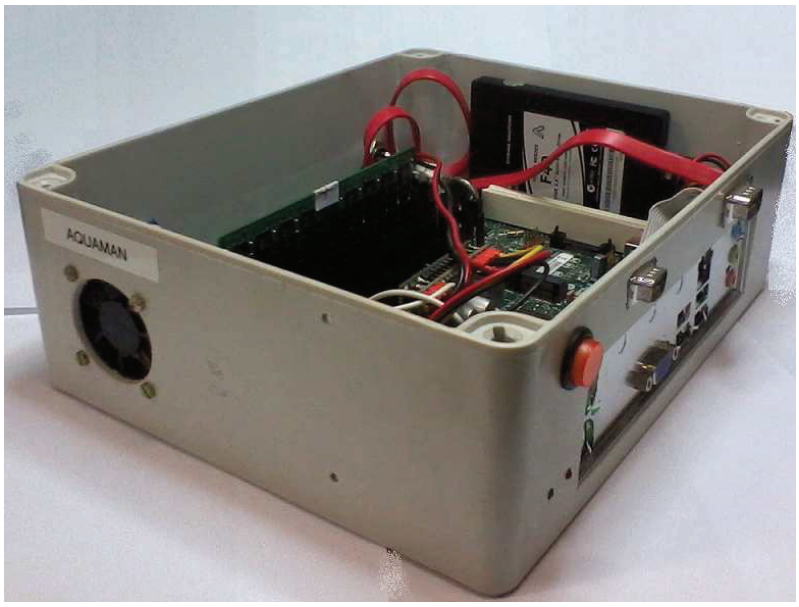

Figura 4.21: PC montado em laboratório para utilização no CADU.

4.5 Resultados

Os resultados apresentados nesta seção mostram uma série de experimentos realizados em todos os níveis da arquitetura do CADU para validar o seu funcionamento e sua integridade. Foram testados individualmente cada um dos sistemas e também o funcionamento do conjunto. As próximas subseções descrevem melhor os testes e o modo como a arquitetura foi importante para sua realização.

4.5.1 Controle de Velocidade do CADU

O Controle de Velocidade do CADU foi realizado utilizando-se o Nível de Controle da arquitetura, sendo para isto necessária a utilização da rede Modbus, juntamente com o seu mestre e os nós de aceleração, frenagem, o sensor de velocidade e um nó adicional chamado Setpoint, introduzido somente para indicar o valor de referência de velocidade sem a interferência dos níveis superiores da arquitetura. Neste teste foi criado, no próprio nível de controle, um programa executando no mestre que consistiu em:

1. Obter por meio de mensagens a velocidade atual do veículo medida pelo nó Sensor

de Velocidade das Rodas.

2. Obter o valor da referência fornecido pelo nó de *Setpoint*.

3. Fazer o processamento com base em um algoritmo de controle Fuzzy.

4. Enviar a mensagem de atuação para o nó Freio.

5. Enviar a mensagem de atuação para o nó Acelerador.

Para esta sequência, escolheu-se o período da tarefa de tempo real igual a 20ms no qual são executados os passos de 1 a 5, cumprindo-se assim a taxa de amostragem de 50Hz para cada um dos nós presentes na rede. O controle de Velocidade aqui executado é baseado em um Projeto de Fim de Curso realizado no CADU [Freitas, 2010].

O resultado é mostrado nas Figura 4.22 e ilustra a viabilidade do Nível de Controle para executar tarefas de controle em tempo real.

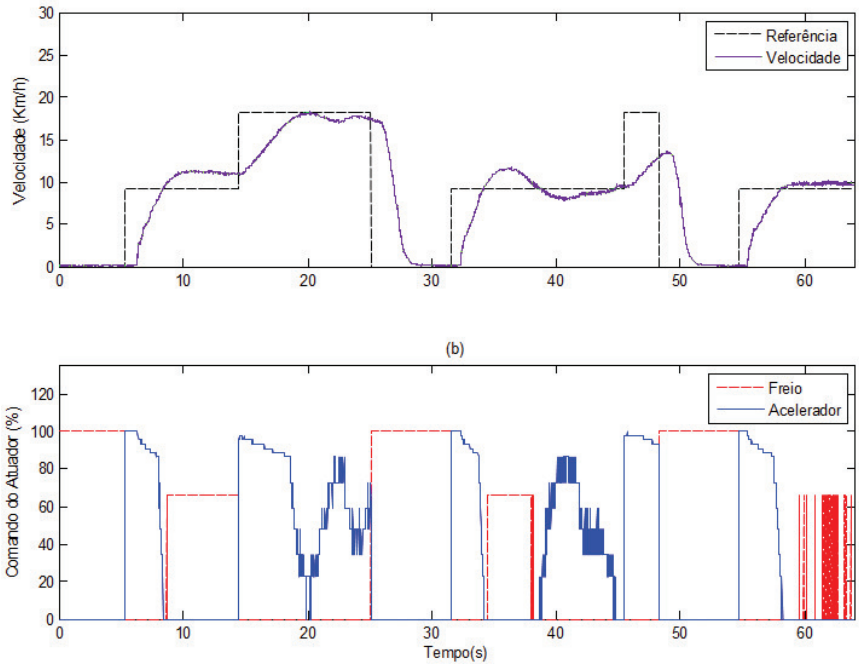

Figura 4.22: Controle Velocidade. Neste experimento foram utilizados quatro nós no Nível de Controle sendo um responsáveis respectivamente por enviar a referência de velocidade, obter velocidade do veículo, enviar comando de freio e enviar comando de acelerador. O gráfico (a) mostra a referência e a velocidade do veículo. O gráfico (b) mostra os comandos recebidos de aceleração e freio .

4.5.2 Determinação da Localização

O teste de localização foi realizado de forma a validar a rede Modbus do Sistema de Localização, juntamente com o Hardware de comunicação entre os sensores e a rede, além de permitir a verificação da integridade dos dados obtidos diretamente do Nível de Controle.

O Experimento consistiu em:

- Coletar dados do GPS Diferencial Hemisphere Modelo R120 existente no laboratório, enviando os dados a uma taxa de 5Hz.
- Coletar dados de Rolamento, Arfagem, Guinada e Aceleração em X, Y e Z do CADU por meio da IMU Microstrain 3DM-GX1. A informação da IMU é do tipo pergunta/resposta, e a solicitação da medida foi feita a uma taxa de 20Hz.
- Coletar dados do sensor de Velocidade das rodas, disponível por meio do computador do Nível de Controle.
- Teste do Supervisor nas redes do Nível de Controle e do Nível de Processamento.

Após todos os sistemas ligados, os programas foram iniciados e o carro foi conduzido por um determinado percurso para geração da informação de posição, velocidade e aceleração.

Cada conjunto de informações necessita de uma requisição específica do Mestre da rede e as respostas são limitadas a quatro bytes de acordo com o protocolo estabelecido. As informações recolhidas em cada requisição por meio de uma solicitação do Mestre da rede no Sistema de Localização foram:

- Hora UTC do GPS (3 bytes).
- Latitude do GPS (4 bytes).
- Longitude do GPS (4 bytes).
- Roll, Pitch da IMU (4 bytes).
- Yaw e Aceleração em X da IMU (4 bytes).
- Aceleração em Y e Z da IMU (4 bytes).

Neste experimento, escolheu-se um tempo fixo de 20ms entre cada uma das requisições. Isto faz com que a taxa de amostragem média de cada sensor seja inversamente proporcional ao número de mensagens na rede, de acordo com a seguinte equação: $A = n/T$ onde A é a taxa de amostragem, n é o número de requisições e T é o intervalo de tempo entre as requisições. Deste modo, para 7 mensagens com tempo entre requisições de 20ms leva a taxa 7,14 mensagens por segundo para cada uma das informações requeridas. As informações obtidas permitiram a sobreposição das coordenadas obtidas sobre uma imagem do google maps conforme mostrado na Figura 4.23 juntamente com os perfis de velocidade, atitude e aceleração do veículo também mostrados nesta figura.

Foram enviadas no total 12.188 mensagens pelo Mestre solicitando alguma medida num intervalo de 244 segundos. O servidor de velocidade recebeu 1.742 mensagens, enquanto a IMU e o GPS receberam cada um 5.233 mensagens no intervalo. As falhas de localização observadas no mapa se devem a perda de sinal pelo GPS. Para as requisições realizadas neste teste, 4 mensagens não foram respondidas. Dentre os fatores que podem levar a este erro se encontram:

- Falha na obtenção do dado pelo nó de comunicação com o dispositivo serial.
- Falha na transmissão de informação entre os nós I2C.
- Falha na resposta do Nó Escravo Modbus para o Mestre Modbus.

A ausência destas mensagens mostra que o sistema não imune a erros, mas possui uma boa taxa de confiabilidade uma vez que para os testes realizados mais de 99% das mensagens foram corretamente entregues.

4.5.3 Teste com o Player

Uma vez que não foi implementado um algoritmo para determinação da posição do veículo, o teste de Player envolveu o envio de comandos de velocidade e ângulo do volante para que o veículo desenvolvesse diferentes velocidades ao longo do teste, virasse o volante tanto para a direita quanto para a esquerda e parasse. Estes comandos foram enviados por um usuário conectado remotamente aos servidores do Nível de Processamento pela interface *setCarLike* do Player. O diagrama do programa executado como cliente no Player é mostrado na Figura 4.24.

Todos os comandos enviados ao veículo foram corretamente executados fazendo assim com que o veículo saísse de um ponto e chegasse a outro apenas se utilizando de uma

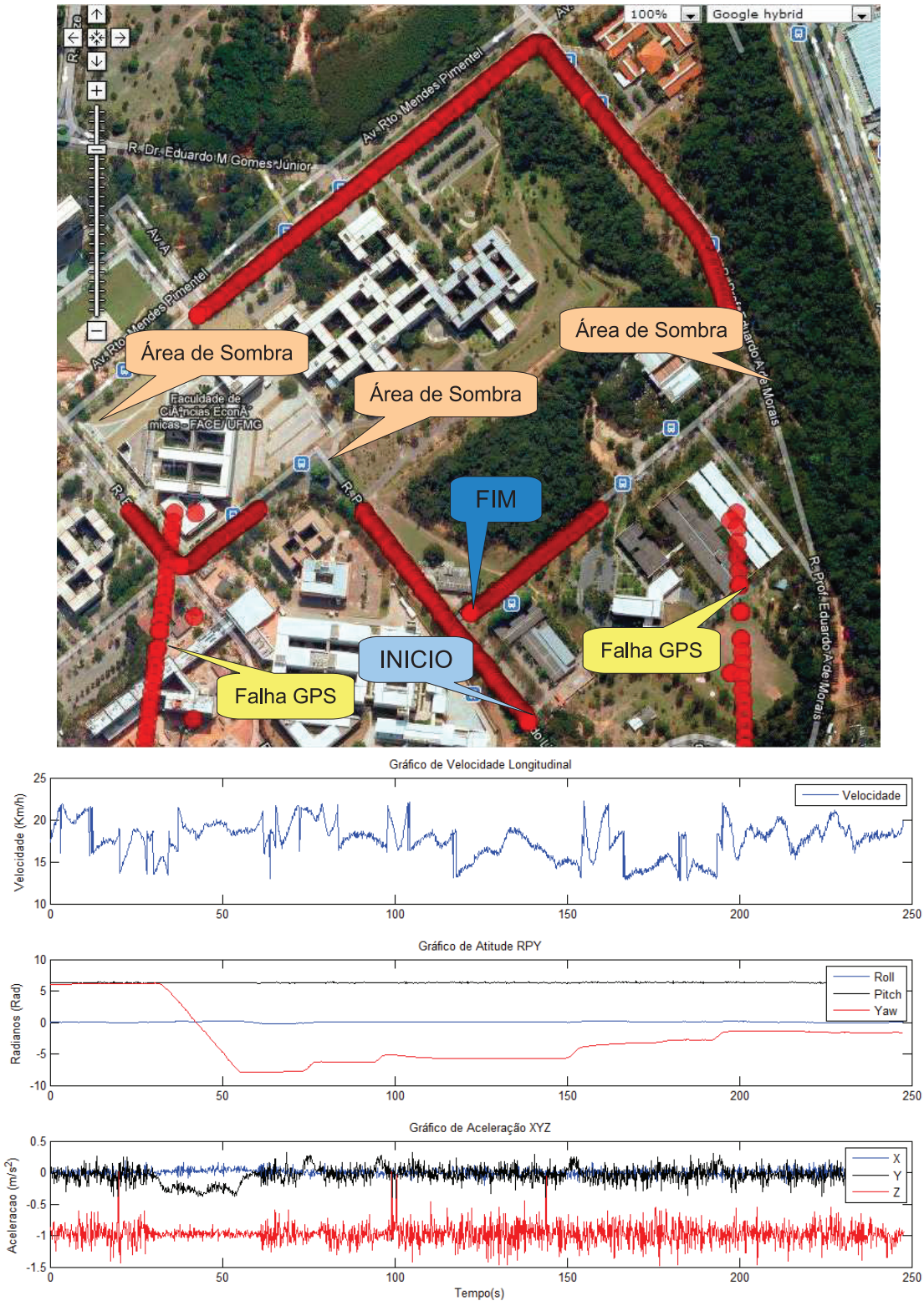

Figura 4.23: Teste de coleta de dados no Nível de Processamento. De cima para baixo é mostrado: Latitude e Longitude do GPS, representados no mapa; Velocidade do veículo obtida do Nível de Controle; Atitude do Veículo RPY e Aceleração do Carro XYZ enviados pela IMU.

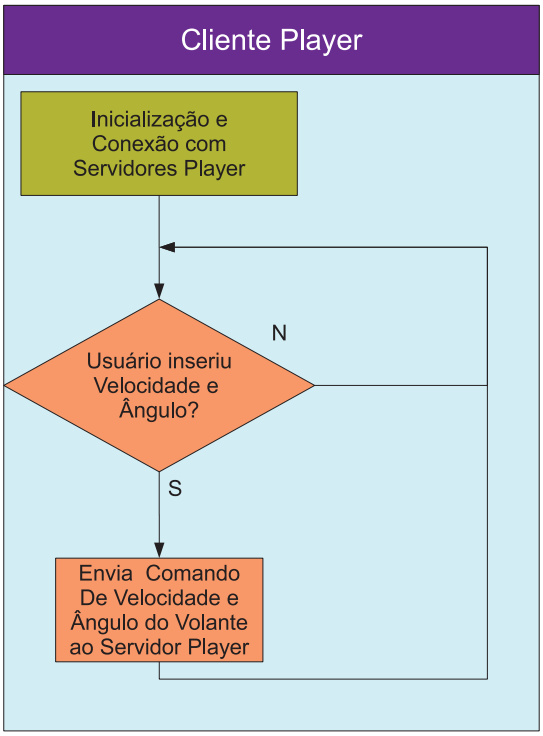

Figura 4.24: Algoritmo de Execução do Player. O programa aguarda que o usuário indique uma velocidade e um ângulo do volante entre -120 e 120 graus. Após pressionar ENTER o comando é enviado ao Sistema de Movimentação no Nível de Processamento que repassa os mesmos ao Nível de Controle para que sejam executados pelo veículo.

interface simples, a qual não exige nenhum conhecimento do usuário a respeito do restante do sistema.

4.5.4 Teste do Monitor

O Monitor foi testado durante a execução dos dois últimos testes e consistiu em catalogar simultaneamente as informações trafegadas tanto na Rede do Nível de Controle, quanto na Rede do Nível de Processamento. O Hardware e Software Implementados permitiram a coleta de dados de aproximadamente 200 mensagens por segundo em cada uma das redes durante 10 minutos com uma taxa de perda de mensagens próxima a 1% mostrando assim que o sistema, apesar do erro, é funcional e viável para o fim a que se dedica. Tais perdas são justificáveis porque o Monitor não é executado em tempo real. Desta forma, além da ausência de mensagens e mensagens com erro que são ignoradas, o Monitor corre o risco de perder mensagens que cheguem ao mesmo tempo em que alguma outra rotina é executada no mesmo.

4.6 Resumo do Capítulo

Neste capítulo a arquitetura de Hardware e Software foi aplicada ao CADU mostrando a sua viabilidade. Um carro autônomo é composto de vários sistemas complexos, que no entanto, sob a ótica da arquitetura implementada, se tornam simples em cada um dos níveis apresentados.

Os testes foram conduzidos de forma a mostrar o funcionamento do sistema após a mudança de arquitetura. As conclusões a respeito deste trabalho são discutidas no próximo capítulo.

Como resultado desta dissertação foi publicado o artigo intitulado “Rede de Instrumentação para Controle de Um Veículo Autônomo” no XI SBAI (Simpósio Brasileiro de Automação Inteligente) 2011, que aborda a questão de tempos de atraso e capacidades máximas da rede escolhida para fazer parte do sistema de controle em tempo real do CADU, nível mais baixo da arquitetura proposta.

5 Conclusões e Trabalhos Futuros

Neste trabalho propôs-se uma arquitetura de Hardware e Software para utilização em um carro autônomo. Esta é voltada ao operador, ou seja, visa a operação de um carro autônomo se dá da mesma forma que se opera um robô móvel comercial.

5.1 Síntese dos Resultados

A solução foi baseada no estudo de diversas arquiteturas, tanto de hardware quanto de software, utilizadas nos mais variados tipos de veículos autônomos. Ao longo do trabalho são mostradas as soluções encontradas para obtenção de uma arquitetura que atende aos requisitos levantados no Capítulo 3.

Para a arquitetura de Hardware foram propostos três níveis sendo eles:

- Nível de Controle - Onde são inseridos os sensores e atuadores de atuação direta no veículo e onde se faz necessário o maior controle de tempo e fluxo de mensagens a fim de se obter uma operação bem sucedida em um veículo autônomo. Dependendo da natureza do veículo os sistemas existentes neste nível podem variar, mas basicamente o que tem-se são os sistemas de Aceleração, Frenagem e Direção do veículo. Para este nível existe um computador central responsável por interligar todos os componentes e fazer a ligação física com os níveis superiores.

- Nível de Processamento - São onde se encontram os sensores que fornecem informações do ambiente externo ao veículo, ou mesmo o posicionamento do veículo em relação ao mundo, i.e. a sua localização, presença de obstáculos, atitude e outros. Os sistemas existentes neste nível são classificados em Sistema de Movimentação - responsável pela movimentação do veículo e interface com os sensores do Nível de Controle; Sistema de Localização - responsável por obter informações para a localização do Veículo; e Sistema de Reconhecimento - que fornece informações de distâncias e obstáculos que se encontram em volta do mesmo. Cada sistema pode

possuir um computador responsável por coletar as informações, interligado fisicamente ao nível de Usuário.

- Nível de Usuário - É onde se encontra a interface do Usuário Final com o sistema veicular. O hardware deste nível deve ser qualquer elemento capaz de processar ou produzir as informações que serão repassadas ou obtidas do Nível de processamento.

Para a arquitetura de Software uma divisão idêntica foi proposta, contudo, sob a ótica de software cada um dos Níveis tem uma funcionalidade específica:

- Nível de Controle - Neste nível cada um dos sensores possui o seu software próprio e entre eles e o computador central existe uma comunicação determinística. No PC é utilizado um software executando em um sistema operacional de tempo real para melhor controle temporal das informações. Tanto os dados que trafegam entre o PC e os demais elementos deste nível, quanto os dados existentes entre o Nível de Controle e o Nível de Processamento podem ser informação de sensores ou comandos que precisam ser processados e entregues a cada elemento de modo adequado.

- Nível de Processamento - Os algoritmos existentes neste nível devem ser capazes de obter as informações tanto dos elementos deste próprio nível, quanto informações provenientes do Nível de Controle, e formatar o dado de forma a entregar e receber informação simplificada para o nível de Usuário. Este Nível é responsável por tornar o uso do veículo autônomo como um todo semelhante ao uso de um robô comercial. Todos os elementos deste nível precisam de uma interface com o Nível de Usuário. A interface com o Nível de Controle é possível mas nem sempre é necessária.

- Nível de Usuário - Este nível, do ponto de vista de software, equivale a escolha das informações a serem enviadas ao veículo. As mesmas podem ter vindo de um complexo calculo de planejamento assim como pode ser comandos aleatórios enviados por um usuário qualquer. A decodificação e tratamento destas informações fica a cargo do Nível de Processamento.

Em ambos as arquiteturas (Hardware e Software) foi adicionado o sistema de supervisão, que do ponto de vista de hardware, corresponde a um computador responsável pela ligação física com todos os sistemas os quais se deseja obter os dados e, do ponto de vista de software, é o responsável por obter informações do sistema e, se necessário, realizar diagnóstico e até mesmo redundância em algum sistema como mostrado nos trabalhos futuros.

Esta arquitetura proposta foi implementada no CADU, carro autônomo desenvolvido na UFMG que, apesar de já operar em modo autônomo antes desta dissertação, estava sujeito a uma série de problemas causados pelo sistema de processamento centralizado e comunicação entre os diversos elementos baseados em Serial/USB. Dentre os problemas pode-se destacar: quantidade de fios necessários para ligação dos sensores e atuadores, necessidade de Hub-USB para dar suporte a todos os elementos, instabilidade para ligação do sistema, dificuldade para alteração dos sistemas devido a forte interdependência entre todos os participantes da estrutura, centralização do sistema sem possibilidade de diagnóstico por outro PC, e não determinismo dos dados trafegados nos elementos de controle.

5.2 Análise da Arquitetura

Baseado na arquitetura proposta, a arquitetura do CADU foi refeita e divida em três níveis, tanto da parte de software quanto da parte de hardware. Uma vez finalizado o processo de adaptação pôde-se fazer uma análise a respeito dos ganhos e perdas com a utilização do novo sistema. Em relação a arquitetura de Hardware as vantagens incluem:

- No nível de Controle foi reduzido o número de cabos utilizados uma vez que o sistema anteriormente em estrela devido a ligação direta do equipamento com o computador passou a ser uma rede Modbus sobre RS-485 ligada na configuração *Daisy Chain*.
- Os níveis de ruído ao qual o sistema estava sujeito foram reduzidos pois trocou-se uma ligação serial padrão RS-232 pela RS-485 de par trançado, sendo esta última a recomendada pela norma [Modbus, 2011] para ambientes sujeitos a ruídos eletromagnéticos e onde é necessária robustez no envio e recebimento da informação;
- Foi eliminada a necessidade de Hub-USB e portas USB adicionais no computador uma vez que agora toda a informação trafegada pela rede é enviada pelo mesmo barramento.
- A introdução do Nível de Processamento permitiu a separação física dos sistemas de localização e reconhecimento do veículo, possibilitando que cada um deles tenha um computador específico para realizar o processamento.
- A utilização de rede também no sistema de localização, composto pelo GPS, IMU e Bússola, além de reduzir o cabeamento necessário, permite a utilização das portas USB restantes no PC para outras funcionalidades.

- A separação do Hardware de Nível Usuário dos demais permite que o sistema seja mais robusto pois, ao final, a capacidade armazenamento, processamento e paralelismo do sistema como um todo se complementam para tornar a arquitetura de hardware atual melhor que a anterior. Além disso, a atualização e modernização do sistema pode ser feita pela troca do hardware deste nível sem a intervenção nos demais níveis.

Para a arquitetura de software foram observados os seguintes melhoramentos:

- No nível de controle as informações deixaram de ser estocásticas e passaram a ser determinísticas, tanto graças a propriedade existente em uma comunicação do tipo mestre escravo entre os nós e o computador central, quanto pela utilização de um sistema operacional de tempo real;

- Em relação às informações de controle propriamente ditas, houve um considerável ganho em seu tratamento, uma vez que a introdução da rede permitiu a comunicação por meio de um pacote bem definido para todos os nós ao invés da comunicação baseada em caracteres ASCII que, além de ter poucos recursos, não era padronizada entre os elementos de mesmo nível.

- Foram eliminados problemas anteriormente encontrados como falha de instalação das portas seriais emuladas, travamentos e corrupção de dados aos quais o sistema estava sujeito com o uso de muitos conversores seriais/USB inseridos em ambiente Windows.

- A divisão em níveis permite que o computador responsável pelo controle de baixo nível tenha uma maior capacidade de processamento direcionada diretamente para este fim.

- No nível de processamento, devido a utilização de todos os dispositivos seriais na forma de Rede Modbus/485, eliminou-se o problema de conflito de gerenciadores de dispositivos existente na utilização de alguns elementos, como GPS, que necessitavam ser inseridos e retirados várias vezes até serem reconhecidos pelo sistema.

- Também se adicionou a vantagem de um processamento exclusivo para cada um dos sistemas existentes no nível de controle;

- A necessidade de se tratar os dados vindos e enviados pela camada de Usuário de forma especial, cria a facilidade almejada na proposta deste trabalho que é de

simplificar o uso do sistema para o usuário final, e esta capacidade é obtida por meio da utilização do pacote computacional Player para ser a interface entre o veículo e usuário final.

A todas estas vantagens adiciona-se a possibilidade de monitoramento por um elemento externo de todos os dados que trafegam pelo veículo. Esta capacidade é fornecida pelo sistema de supervisão existente na arquitetura e que foi implementado no CADU para geração de relatório das informações da rede Modbus do Nível de Controle e da Rede Modbus do Nível de Processamento.

No entanto, mesmo para esta arquitetura dividida em vários níveis e que permite tanto a modularização quanto um ganho de desempenho foram observadas algumas desvantagens que são listada a seguir:

- O uso da rede Modbus impõe a necessidade da comunicação mestre-escravo impedindo que soluções como produtor-consumidor sejam adotadas.
- Por ser mais dividido, o novo sistema ficou mais complexo do ponto de vista de desenvolvedor. Isto ocorre devido a necessidade de se criar soluções que sejam compatíveis com as interfaces já definidas da arquitetura.
- Tanto o uso quanto o manuseio de um sistema operacional de tempo real exige mais esforço e conhecimento do que para um sistema operacional comum. Por isto, nesta arquitetura, é necessário um maior nível de conhecimento do desenvolvedor.
- O uso da rede Modbus limita o número de mensagens máximas que podem trafegar na rede devido as exigências da norma e também do limite de capacidade de transmissão do microcontrolador escolhido [Arruda et al., 2011]. Uma alternativa é o uso da Rede CAN *(Controller Area Network)* que, apesar de mais complexa do ponto de vista de desenvolvimento, oferece taxas de comunicação mais rápidas e permite superar algumas restrições de tempo impostas pela rede Modbus.
- Não é recomendável utilizar ponto flutuante em operações dentro do *kernel-space* [Corbet et al., 2005] exigindo do desenvolvedor um trabalho extra que desenvolva soluções sem o uso deste recurso;
- A elaboração de rotinas de tempo real é complexa pois exige um cuidado extra para garantir que o tempo máximo de execução de um ciclo não exceda o período cíclico estabelecido, e que os módulos de tempo real sejam devidamente carregados antes

de se executar o algoritmo de controle para que não haja riscos de travamento do sistema.

- Sem a utilização de programas para automatização de carregamento de módulos e programas, i.e.*scripts*, faz-se necessário iniciar muitos programas para que se tenha o CADU em ponto de utilização.

Apesar destas desvantagens, pode-se considerar que a aplicação da arquitetura é benéfica e trouxe melhorias ao sistema como um todo. Alguma destas desvantagens podem ser trabalhadas como apontado na proposição abaixo de possíveis trabalhos futuros.

5.3 Trabalhos Futuros

Para trabalhos futuros pode-se destacar algumas soluções possíveis ou alternativas tanto para melhoria da implementação da arquitetura no CADU quanto para complemento do que já foi feito.

De forma a aproximar a arquitetura proposta da implementação executada, pode-se utilizar o chamado nó I2C para utilização em conjunto com a EPOS, retirando-se assim a necessidade de uma ligação extra no Nível de Controle, e permitindo assim que a mesma seja efetivamente vista como um nó escravo da Rede Modbus/485, tanto do ponto de vista de software quanto de hardware.

A criação de redundância na rede permitiria que o monitor, em caso de falha de algum sistema, imediatamente assumisse o lugar deste nó. Por exemplo, ele poderia, ao ver que o mestre da rede de controle parou de enviar mensagens, assumir automaticamente o lugar deste. Ou outra aplicação interessante seria ele substituir o servidor de informações de localização, caso este viesse a falhar.

Com toda esta estrutura é possível implementar muitos sistemas de segurança de modo a manter a integridade do veículo nas mais variadas situações. O esquema de segurança pode ser colocado desde o sistema mais básico, como os nós microcontrolados, até os mais complexos, como os sistemas de tempo real e o servidor Player.

Além destes melhoramentos, três mudanças que podem trazer benefícios a arquitetura são sugeridas:

- O uso da rede CAN que permitiria uma velocidade maior na rede ao mesmo tempo

em que aumentaria o número de nós simultâneos possíveis de 31 para 255 na rede de controle de mais baixo nível, além da possibilidade de se utilizar comunicação do tipo Produtor/Consumidor;

- Troca do sistema de Player pelo uso do ROS (*Robot Operating System*) [ROS, 2011] uma vez que este também é um sistema capaz de atender satisfatoriamente aos requisitos impostos pelas arquiteturas propostas e parece estar mais ativo atualmente na comunidade de robótica;

- Troca do RTAI pelo Xenomai [Xenomai, 2011], uma vez que os softwares já estão definidos e esta troca envolveria em sua grande maioria mudanças em diretivas do sistema. Esta opção permitiria a comparação dos sistemas para verificação do desempenho dos mesmos para uso em veículos autônomos.

Referências

[Alur et al., 2001] Alur, R., Das, A., Esposito, J., Fierro, R., Grudic, G., Hur, Y., Kumar, V., Lee, I., Ostrowski, J., Pappas, G., Southall, B., Spletzer, J., and Taylor, C. (2001). A framework and architecture for multirobot coordination. In Rus, D. and Singh, S., editors, *Experimental Robotics VII*. Springer Verlag.

[Arruda, 2009] Arruda, T. A. (2009). Rede de instrumentação para aplicação em controle embarcado. *Projeto Final de Curso, Escola de Engenharia/UFMG*, 1(1):1–54.

[Arruda et al., 2011] Arruda, T. A., Campedelli, E. R., and Pereira, G. A. S. (2011). Desenvolvimento de uma rede modbus para o controle de um veículo autônomo. *X Simpósio Brasileiro de Automação Inteligente*, 1:1–6.

[Aurélio, 2011] Aurélio (acesso em 20/09/2011). Dicionário aurélio online. `http://www.dicionariodoaurelio.com/`.

[Barbalace et al., 2008] Barbalace, A., Luchetta, A., Manduchi, G., Moro, M., Soppelsa, A., and Taliercio, C. (2008). Performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application. *Nuclear Science, IEEE Transactions on*, 55(1):435 –439.

[Bohren et al., 2009] Bohren, J., Foote, T., Keller, J., Kushleyev, A., Lee, D., Stewart, A., Vernaza, P., Derenick, J., Spletzer, J., and Satterfield, B. (2009). Little ben: The ben franklin racing team's entry in the 2007 darpa urban challenge. In Buehler, M., Iagnemma, K., and Singh, S., editors, *Darpa Urban Challenge: Autonomous Vehicles In City Traffic*, volume 56 of *Springer Tracts in Advanced Robotics*, pages 231–255. Springer-Verlag Berlin.

[Cesetti et al., 2010] Cesetti, A., Frontoni, E., Mancini, A., Zingaretti, P., and Longhi, S. (2010). A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 57(1-4, SI):233–257.

[Chiabrando et al., 2011] Chiabrando, F., Nex, F., Piatti, D., and Rinaudo, F. (2011). Uav and rpv systems for photogrammetric surveys in archaelogical areas: two tests in the piedmont region (italy). *Journal Of Archaeological Science*, 38(3):697–710.

[Comau, 2011] Comau (acesso em 01/11/2011). Comau website. `http://www.comau.com`.

[Corbet et al., 2005] Corbet, J., Rubini, A., and Kroah-Hartman, G. (2005). *Linux device drivers*. O'Reilly.

[Correia and Steiger-Garcao, 1994] Correia, L. and Steiger-Garcao, A. (1994). A model of hierarchical behavior control for an autonomous vehicle. In *From Perception to Action Conference, 1994., Proceedings*, pages 396–399.

[de Lima, 2010] de Lima, D. A. (2010). Navegação segura de um carro autônomo utilizando campos vetoriais e o método da janela dinâmica. *Dissertação de Mestrado - UFMG*.

[Dynamics, 2011] Dynamics, B. (acesso em 01/11/2011). The most advanced rough-terrain robot on earth. `http://www.bostondynamics.com/robot_bigdog.html`.

[Elfes et al., 2008] Elfes, A., Hall, J. L., Kulczycki, E. A., Clouse, D. S., Morfopoulos, A. C., Montgomery, J. F., Cameron, J. M., Ansar, A., and Machuzak, R. J. (2008). Autonomy architecture for aerobot exploration of saturnian moon titan. *IEEE Aerospace And Electronic Systems Magazine*, 23(7):16–24.

[Farsi et al., 1999] Farsi, M., Ratcliff, K., and Barbosa, M. (1999). An overview of controller area network. *Computing & Control Engineering Journal*, 10(3):113–120.

[Fernandes, 2011] Fernandes, L. C. (2011). Desenvolvimento de sistemas de navegação para robôs móveis e veículos inteligentes em ambientes externos. *Exame de Qualificação - PUC/SP*.

[Fletcher et al., 2008] Fletcher, L., Teller, S., Olson, E., Moore, D., Kuwata, Y., How, J., Leonard, J., Miller, I., Campbell, M., Huttenlocher, D., Nathan, A., and Kline, F.-R. (2008). The MIT-Cornell Collision and Why It Happened. *Journal Of Field Robotics*, 25(10):775–807.

[Freitas, 2010] Freitas, E. J. R. (2010). Controle longitudinal de um veículo autônomo. *Projeto Final de Curso, Escola de Engenharia/UFMG*, 1(1):1–60.

[Freitas et al., 2009] Freitas, E. J. R., Vinti, M. N. W., Santos, M. M., Iscold, P., Torres, L. A. B., and Pereira, G. A. S. (2009). Desenvolvimento de automação embarcada para um robô móvel baseado em um carro de passeio. *IX Simpósio Brasileiro de Automação Inteligente*, 1:1–6.

[Fu et al., 2008] Fu, X., Zhou, Z., Xiong, W., and Guo, Q. (2008). Mems-based low-cost flight control system for small uavs. *Tsinghua Science & Technology*, 13(5):614 – 618.

[Gerkey et al., 2003] Gerkey, B., Vaughan, R. T., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proc. of the 11th Int'l. Conf. on Advanced Robotics*, pages 317–323.

[Godoy et al., 2010] Godoy, E. P., Lopes, W. C., Sousa, R. V., Porto, A. J. V., and Inamasu, R. Y. (2010). Modelagem e simulação de redes de comunicação baseadas no protocolo CAN - Controller Area Network. *Sba: Controle & Automação Sociedade Brasileira de Automatica*, 21:425 – 438.

[Goktogan et al., 2010] Goktogan, A. H., Sukkarieh, S., Bryson, M., Randle, J., Lupton, T., and Hung, C. (2010). A rotary-wing unmanned air vehicle for aquatic weed surveillance and management. *Journal of Intelligent & Robotic Systems*, 57(1-4, Sp. Iss. SI):467–484.

[Incze, 2009] Incze, M. L. (2009). Optimized deployment of autonomous underwater vehicles for characterization of coastal waters. *Journal of Marine Systems*, 78(Sp. Iss. SI Suppl. S):S415–S424.

[Iscold et al., 2010] Iscold, P., Pereira, G. A. S., and Torres, L. A. B. (2010). Development of a hand-launched small uav for ground reconnaissance. *IEEE Transactions On Aerospace And Electronic Systems*, 46(1):335–348.

[Jeon et al., 2001] Jeon, J. M., Kim, D. W., Kim, H. S., Cho, Y. J., and Lee, B. H. (2001). An analysis of network-based control system using can (controller area network) protocol. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3577 – 3581.

[Jones, 2001] Jones, W. (2001). Keeping cars from crashing. *Spectrum, IEEE*, 38(9):40 –45.

[Kolski et al., 2006] Kolski, S., Ferguson, D., Bellino, M., and Siegwart, R. (2006). Autonomous driving in structured and unstructured environments. In *Intelligent Vehicles Symposium, 2006 IEEE*, pages 558–563.

[Koralalage and Yoshiura, 2009] Koralalage, K. and Yoshiura, N. (2009). irail: A novel architecture towards autonomous locomotives and intelligent infrastructures for developing countries. In *Intelligent Transportation Systems, 2009. ITSC '09. 12th International IEEE Conference on*, pages 1 –6.

[Long et al., 2007] Long, L., Hanford, S., Janrathitikarn, O., Sinsley, G., and Miller, J. (2007). A review of intelligent systems software for autonomous vehicles. In *Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007. IEEE Symposium on*, pages 69–76.

[Martin et al., 1999] Martin, D., Cheyer, A., and Moran, D. (1999). The open agent architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128.

[Martinez et al., 2010] Martinez, A., Rodriguez, Y., Hernandez, L., Guerra, C., and Sahli, H. (2010). Hardware and software architecture for auv based on low-cost sensors. In *Control Automation Robotics Vision (ICARCV), 2010 11th International Conference on*, pages 1428 –1433.

[Maxon, 2011] Maxon (Acesso em 01/11/2011). Maxon website. `http://www.maxonmotorusa.com`.

[Maza et al., 2011] Maza, I., Caballero, F., Capitan, J., Martinez-de Dios, J. R., and Ollero, A. (2011). Experimental results in multi-uav coordination for disaster management and civil security applications. *Journal Of Intelligent & Robotic Systems*, 61(1-4, SI):563–585.

[McDowell et al., 2002] McDowell, P., Chen, J., and Bourgeois, B. (2002). Uuv teams, control from a biological perspective. In *Oceans '02 Mts/IEEE*, volume 1, pages 331 – 337 vol.1.

[Merino et al., 2006] Merino, L., Caballero, F., Martinez-de Dios, J. R., Ferruz, J., and Ollero, A. (2006). A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23(3-4):165–184.

[Modbus, 2011] Modbus (Acesso em 15/11/2011). Modbus over serial line especification. http://www.modbus.org/.

[Montemerlo et al., 2008] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A., Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. (2008). Junior: The stanford entry in the urban challenge. *J. Field Robot.*, 25:569–597.

[Nelson, 1998] Nelson, M. L. (1998). A software control architecture for autonomous vehicles. *Hawaii International Conference on System Sciences*, 3:226.

[Pastor et al., 2006] Pastor, E., Lopez, J., and Royo, P. (2006). A hardware/software architecture for uav payload and mission control. In *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, pages 1 –8.

[Rauskolb et al., 2008] Rauskolb, F. W., Berger, K., Lipski, C., Magnor, M., Cornelsen, K., Effertz, J., Form, T., Graefe, F., Ohl, S., Schumacher, W., Wille, J.-M., Hecker, P., Nothdurft, T., Doering, M., Homeier, K., Morgenroth, J., Wolf, L., Basarke, C., Berger, C., Gülke, T., Klose, F., and Rumpe, B. (2008). Caroline: An autonomously driving vehicle for urban environments. *J. Field Robot.*, 25:674–724.

[ROS, 2011] ROS (Acesso em 01/12/2011). Ros website. http://www.ros.org/wiki/.

[RTAI, 2011] RTAI (Acesso em 01/10/2011). Rtai - real time application interface. https://www.rtai.org/.

[Semsch et al., 2009] Semsch, E., Jakob, M., Pavlicek, D., and Pechoucek, M. (2009). Autonomous uav surveillance in complex urban environments. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, volume 2, pages 82 –85.

[SICK, 2011] SICK (acesso em 01/11/2011). Sick. http://www.sick.com/.

[Stoy and Hedges, 2010] Stoy, K. and Hedges, R. (2010). The player project. http://playerstage.sourceforge.net/.

[Szabo et al., 1992] Szabo, S., Szabo, O., Scott, H. A., Murphy, K. N., Legowik, S. A., and Bostelman, R. V. (1992). High level mobility controller for a remotely operated unmanned land vehicle. *Journal of Intelligent and Robotic Systems*, 5:63–77.

[Teck, 2011] Teck (acesso em 01/12/2011). Google automated self driving car test drove successfully. teck.in/google-automated-self-driving-cars-tested-successfully.html.

[Urmson et al., 2008] Urmson, C., Anhalt, J., Bae, H., Bagnell, J. A. D., Baker, C., Bittner, R. E., Brown, T., Clark, M. N., Darms, M., Demitrish, D., Dolan, J., Duggins, D., Ferguson, D., Galatali, T., Geyer, C. M., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T., Kolski, S., Likhachev, M., Litkouhi, B., Kelly, A., McNaughton, M., Miller, N., Nickolaou, J., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Sadekar, V., Salesky, B., Seo, Y.-W., Singh, S., Snider, J. M., Struble, J. C., Stentz, A. T.,

Taylor, M., Whittaker, W. R. L., Wolkowicki, Z., Zhang, W., and Ziglar, J. (2008). Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(1):425–466.

[Vermaas et al., 2007] Vermaas, L. L. G., de Mello Honório, L., de Jesus, E. O., Freire, M., and Barbosa, D. A. (2007). Intelligent vehicle survey and applications. In Lambert-Torres, G., Abe, J. M., da Silva Filho, J. I., and Martins, H. G., editors, *LAPTEC*, volume 186 of *Frontiers in Artificial Intelligence and Applications*, pages 205–235. IOS Press.

[Widyotriatmo and Hong, 2008] Widyotriatmo, A. and Hong, K.-S. (2008). Decision making framework for autonomous vehicle navigation. In *SICE Annual Conference, 2008*, pages 1002–1007.

[Xenomai, 2011] Xenomai (Acesso em 01/12/2011). Xenomai website. `http://www.xenomai.org/index.php/Main_Page`.

[Xiao-xu et al., 2010] Xiao-xu, D., Bao-wei, S., and Guang, P. (2010). Dps motion simulation of uuv based on fuzzy controller. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, volume 2, pages 157 –160.

[Xiaodong and Yanjun, 2002] Xiaodong, N. and Yanjun, Z. (2002). Determining message delivery delay of controller area networks. In *Proceedings. of the IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, pages 767 – 771.

[Xie et al., 2009] Xie, M., Wang, L., Linbo, X., Li, J., Yang, H., Song, C., and Zhang, L. (2009). Developing hardware capability for mobile manipulation by low-cost humanoid robot (LOCH). *Industrial Robot-An International Journal*, 36(5):428–440.

[Xie et al., 2010] Xie, P., Kang, F., and Wang, Y. (2010). Cooperative navigation for multi-uuv using relative observations. In *Image and Signal Processing (CISP), 2010 3rd International Congress on*, volume 7, pages 3191 –3194.

[Zammit and Zammit-Mangion, 2010] Zammit, P. and Zammit-Mangion, D. (2010). A proposed computational framework for autonomous vehicles. In *MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*, pages 1188 –1191.

APÊNDICE A – Análise da Rede Modbus sob camada Física RS-485

As redes industriais podem ter diferenciadas aplicações dependendo de onde e como as mesmas serão implementadas. Para a rede Modbus desenvolvida neste trabalho levantou-se uma série de parâmetros importantes para se determinar a viabilidade ou não de sua utilização para a aplicação em veículos autônomos. Na sequência são feitas as devidas considerações em relação aos atrasos da rede e da carga máxima suportada utilizando um sistema operacional de tempo real.

A.1 Atrasos de Rede

Uma característica importante a ser analisada do ponto de vista da redes em geral, são os atrasos envolvidos na comunicação.

Em [Jeon et al., 2001], [Xiaodong and Yanjun, 2002] e [Godoy et al., 2010] são desenvolvidos modelos de atrasos para análise da rede CAN. A rede CAN é multimestre e é baseada no modelo produtor consumidor [Farsi et al., 1999]. O que propõe-se aqui é uma adaptação dos modelos de atrasos desenvolvidos por estes autores ao protocolo Modbus que é do tipo Mestre-Escravo e permite somente um Mestre por barramento. A Equação (A.1) mostra que o atraso total da mensagem (T_{delay}) equivale ao tempo percorrido no início da transmissão de mensagem (T_{start}) até a completa interpretação, processamento e envio de resposta pelo nó de destino (T_{end}).

$$T_{delay} = T_{start} - T_{end} \tag{A.1}$$

Pode-se dividir os atrasos em algumas partes para melhor analise de suas dependências com os elementos da rede. Eles são:

Atraso de Geração – O atraso de geração (A_G) é o tempo gasto entre o início da tarefa e o momento efetivo em que a mesma é enviada ao barramento pelo mestre. Este valor é tipicamente pequeno e depende do *hardware* e do modo como o *software* foi implementado. Os valores deste parâmetro são usualmente obtidos por meio de experimentos.

Atraso de Transmissão – O atraso de transmissão (A_T) corresponde ao atraso no momento do envio dos *bytes* pelo mestre, considerando o *overhead* causado pelos *bits* de *start bit*, *stop bit* e paridade. No protocolo Modbus não há *stuff bits* sendo assim o número de *bits* por *byte* fixados em determinado valor. A Equação (A.2) fornece o atraso de transmissão.

$$A_T = \tau \times (bHeader + bData + bError) \times (starB + stopB + parityB + 8)\,, \tag{A.2}$$

onde *bHeader* é o cabeçalho do quadro Modbus RTU que possui dois *bytes* correspondentes ao endereço e a função do nó, *bData* corresponde aos *bytes* de dados que podem variar entre 0 e 247 e os *bytes bError* correspondem a dois *bytes* para cálculo do CRC16. Os parâmetros *starB*, *stopB* e *parityB* são respectivamente o *start bit*, *stop bit* e a paridade. A variável τ corresponde ao tempo de transmissão de um *bit*, e pode ser obtido por $\tau = \frac{1}{Baudrate}$.

Atraso de Entrega – O atraso de entrega (A_E), assim como o atraso de geração equivale ao tempo necessário para que o nó escravo processe por completo a mensagem recebida. Este valor também depende tanto do *hardware* quanto do *software* envolvidos. Quanto mais rápido o processamento da informação, menor é o tempo de entrega.

Atraso de Controle – O Atraso de Controle (A_C) equivale a um atraso somado do tempo de envio da resposta do escravo ao mestre e o silêncio necessário indicado na norma Modbus que exige que um intervalo mínimo de 1,75ms entre a transmissão de mensagens consecutivas para que as mesmas sejam corretamente identificadas pelos nós escravos como sendo duas mensagens diferentes [Modbus, 2011]. O tempo de resposta do escravo varia de nó para nó dependendo do número de bytes transmitidos e do atraso na transmissão de cada byte, sendo portanto obtido por meio de experimentos ou cálculos explícitos de tempo do firmware do escravo.

Atraso de Fila – Uma rede que se utiliza o paradigma Mestre-Escravo não possui fila pois somente o Mestre pode iniciar uma comunicação. Uma vez que haja somente um mestre, não existe risco de colisão neste protocolo. O Modbus também especifica que cada mensagem do mestre só pode ser direcionada a um escravo, que pode ou não responder a

solicitação. Em mensagens do tipo *broadcast*, nenhum nó escravo pode responder. Deste modo, podemos considerar o Atraso de Fila (A_F) igual a zero para o protocolo Modbus.

A Figura A.1 mostra os tempos encontrados desde a sua inicialização até o momento final no qual o escravo recebe e interpreta e responde a mensagem do mestre.

O atraso total T_{delay} pode ser resumido como:

$$T_{delay} = A_T + A_G + A_E + A_C; \tag{A.3}$$

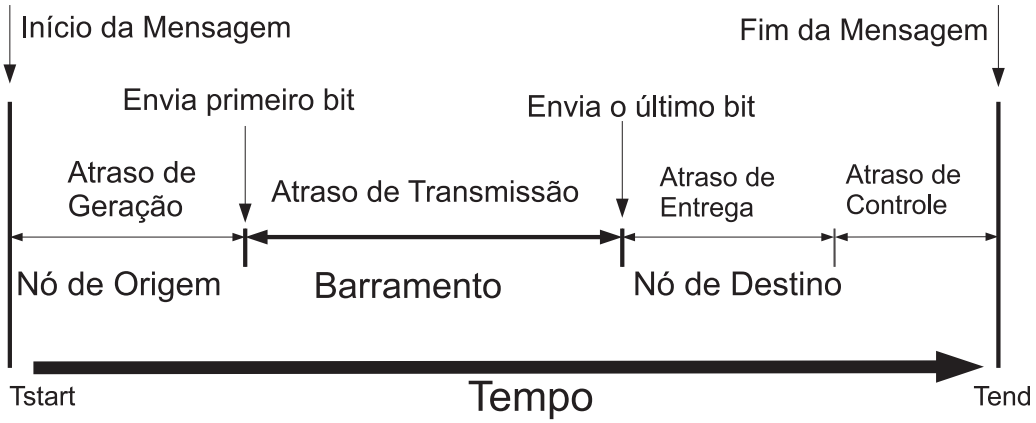

Figura A.1: Diagrama temporal do envio de mensagem Modbus

A.2 Carga máxima da rede e taxa de utilização

A carga máxima da rede equivale a quantidade máxima mensagens por segundo possíveis de serem geradas pelo mestre sem que haja problemas na comunicação. A carga máxima pode ser calculada como:

$$Cmax = \frac{1}{T_{Delay}}\,. \tag{A.4}$$

Obtida a carga máxima da rede, sua taxa de utilização pode ser calculada como:

$$Tx = \frac{\sum f_a}{Cmax}\,,$$

onde f_a é a frequência de amostragem do nó em questão.

A frequência máxima de mensagens por nó (f_M) é inversamente proporcional ao

número de nós sendo esta proporção dada por:

$$f_M = \frac{Cmax}{n}\,, \tag{A.5}$$

onde n é o número de nós escravos da rede.

A.3 Desenvolvimento da Rede Modbus

Na rede desenvolvida, os nós escravos foram construídos em uma placa-protótipo que contém um Microcontrolador PIC modelo 18F2550, um Regulador de Tensão para alimentação, conector USB para programação *in-circuit* do PIC e interface de entrada e saída dos dados RS-485. Esta placa pode ser vista na Figura A.2.

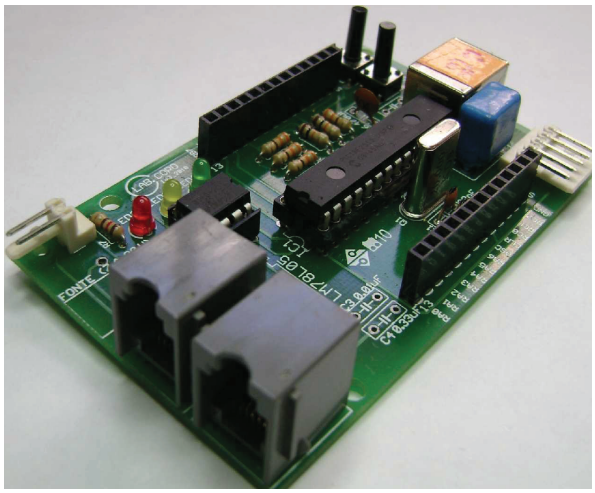

Figura A.2: Placa Protótipo para desenvolvimento da Rede

As configurações escolhidas para a rede foram limitadas aos componentes utilizados tendo sua velocidade de transmissão definida em 115,2Kbps e sem paridade. A transmissão é *Half-Duplex* e o modo de ligação das placas é *Daisy-Chain*.

O nó Mestre corresponde a um PC Mini-ITX com processador Intel Atom 330 Dual Core de 1,6GHz. Para que este nó atendesse aos requisitos temporais, optou-se por escolher um sistema operacional de tempo real. Em [Barbalace et al., 2008] são comparados quatros sistemas, sendo eles o RTAI, o Linux, VxWorks do Windows e o Xenomai, e seus respectivos tempos de latência, *jitter* e reescalonamento. Baseado nos resultados deste artigo, a escolha pelo RTAI se deu por ele ser *open-source*, por ser melhor em termos temporais que o Xenomai e para manter a compatibilidade com outros sistemas desenvolvidos no laboratório. O sistema operacional utilizado foi o Ubuntu 10.04.1 com kernel 2.6.31.8

Quadro Modbus RTU

Start Frame	Endereço	Função	Dados	CRC	End Frame
>= 3,5 chars	1 byte	1 byte	n x 1 byte	2 bytes	>= 3,5 chars

Figura A.3: Quadro de Mensagem Modbus/RTU.

e patch RTAI 3.8.

As mensagens do Mestre trafegadas pela rede estão conforme o quadro de mensagem Modbus RTU mostrado na Figura A.3, onde o Endereço e a Função possuem um *byte* cada uma, os bytes de dados foram fixados em dois e o CRC possui também dois *bytes* totalizando um quadro Modbus de 6 *bytes*.

Com base neste desenvolvimento, foram levantados os valores experimentais para as equações propostas na seção anterior que têm como base a rede utilizada no veículo autônomo como será mostrado na próxima seção. Os resultados mostram que o Atraso de Geração é menor que 1μs e varia junto com o *schedule* e o *jitter* do sistema operacional de tempo real [Barbalace et al., 2008].

O Atraso de Transmissão é calculado considerando que tem-se um quadro Modbus do Mestre composto por dois *bytes* de cabeçalho contendo o endereço e a função ($bHeader$), dois *bytes* de dados ($bData$) e dois *bytes* de CRC ($bError$). Cada byte contém um *start bit* ($starB$) e um *stop bit* ($stopB$) pois não há bit de paridade ($parityB$). A Velocidade de comunicação de 115,2Kbps que fornece um tempo τ de $8,6805\mu$s. O valor encontrado para o atraso de transmissão, utilizando-se a Equação (A.2) é de 521μs. O valor obtido no osciloscópio, dadas as limitações de medição do mesmo, foi de 520μs que é bem próximo ao valor calculado.

O Atraso de Entrega é um valor obtido experimentalmente. Nesta rede, o maior valor para este atraso foi de 355μs nas rotinas necessárias para interpretação da mensagem enviada por parte do nó escravo.

O Atraso de Controle utilizado foi de 2,75ms valor este que comporta o envio de resposta por parte do escravo com o maior quadro Modbus da rede e o silêncio de 1,75ms necessário para uma correta comunicação.

Os valores indicados acima foram obtidos por meio de medições com um osciloscópio digital. A Figura A.4 mostra o formato das mensagens Modbus no barramento 485 como

visto na tela do osciloscópio.

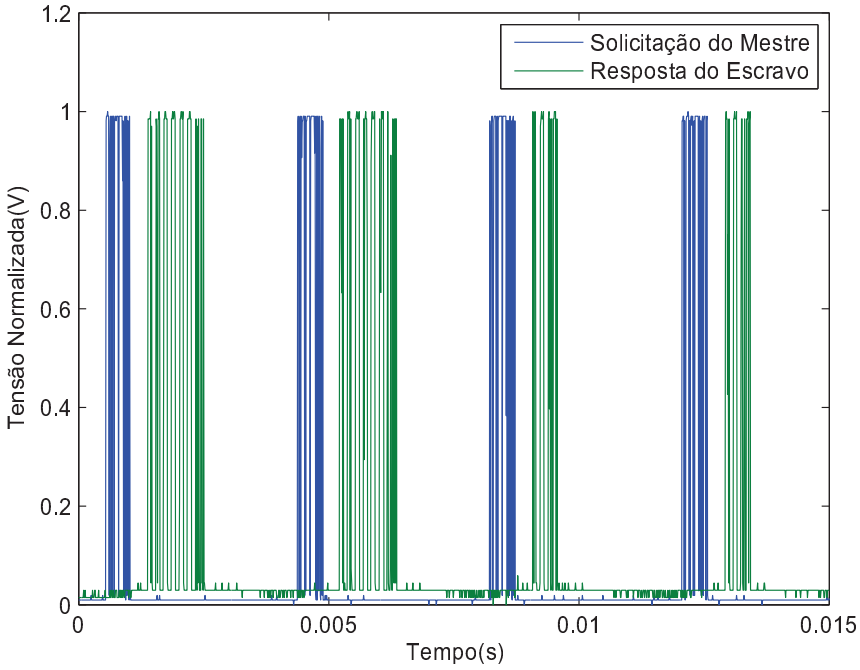

Figura A.4: Sequência de Mensagens da Comunicação Modbus.

Com base nestes valores obtidos, a capacidade suportada pela rede, de acordo com a Equação (A.4) é de 275 mensagens por segundo. Em experimentos realizados com nós escravos mais simples, foram obtidas taxas de até 380 mensagens por segundo.