Osvaldo Hugo Bertone

Desenvolvimento de uma Plataforma Universal para aplicações em Robôs Móveis

CAMPINAS

2012

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

OSVALDO HUGO BERTONE

Desenvolvimento de uma Plataforma Universal para aplicações em Robôs Móveis

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Engenharia Elétrica, na área de Engenharia Eletrônica, Microeletrônica e Optoeletrônica.

Orientador: Prof. Dr. Marco Antônio Robert Alves

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA TESE
DEFENDIDA PELO ALUNO OSVALDO HUGO BERTONE
E ORIENTADO PELO PROF. DR. MARCO ANTÔNIO ROBERT ALVES
Assinatura do Orientador

_____

CAMPINAS

2012

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Bertone, Osvaldo Hugo
B462d Desenvolvimento de uma plataforma universal para aplicações em robôs móveis / Osvaldo Hugo Bertone. -- Campinas, SP: [s.n.], 1912.

Orientador: Marco Antônio Robert Alves.
Tese de Doutorado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Robôs - Móveis. 2. Robôs. 3. Mecatrônica. I. Robert Alves, Marco Antônio, 1964-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Development of the universal platform for mobile robots applications
Palavras-chave em Inglês: Mobile robots, Robots, Mecatronic
Área de concentração: Eletrônica, Microeletrônica e Optoeletrônica
Titulação: Doutor em Engenharia Elétrica
Banca examinadora: Marco Anônio Rober Alves, Evandro Mazina Martins, Evandro Luís Linhari Rodrigues, Edmundo da Silva Braga
Data da defesa: 09-10-2012
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Osvaldo Hugo Bertone

Data da Defesa: 9 de outubro de 2012

Título da Tese: "Desenvolvimento de uma Plataforma Universal para Aplicações em Robôs Móveis"

Prof. Dr. Marco Antônio Robert Alves (Presidente): Marco A. R. Alves

Prof. Dr. Evandro Mazina Martins: _____

Prof. Dr. Evandro Luis Linhari Rodrigues: _____

Prof. Dr. Edmundo da Silva Braga: _____

Prof. Dr. José Antonio Siqueira Dias: _____

Dedico este trabalho a meu pai Mateo Isabelo Bertone (Don Carlos, in memoriam) e a minha mãe Angélica Leonilda Perrée de Bertone (in memoriam) que foram os motores de toda minha vida. Também ao meu tio Julio Adolfo Perrée (in memoriam) que foi meu primeiro professor que me ensino a trabalhar e a tocar o violão e sempre esteve ao meu lado em todas as circunstâncias da minha vida, nas boas e nas não tão boas.

“Siempre el amigo mas fiel es una conducta honrada”

Martin Fierro

AGRADECIMENTOS

Ao meu orientador Prof. Dr. Marco Antônio Robert e Prof. Alberto Martins Jorge (in memoriam), ao Dr. Antonio Valério Netto de Cientistas Associados Desenvolvimento Tecnológico, ao Dr. Marcelo Prado da MultiCorpos Enenharia S/S de São Carlos, ao Prof. Douglas Eduardo Zampieri da Unicamp e aos colegas do DEMIC, Marco Aurélio Seluque Fregonezi e Carlos Roberto Mingoto Jr. e a minha família que me deu todo o apoio duranto todos estes longos anos.

Um agradecimento especial à equipe mulidisciplinar da empresa Cientistas Associados Desenvolvimento Tecnológico que permitiu com o seu trabalho o desenvolvimento de um robô que fez possível a validação da tese.

Também a FAPESP e o CNPq que facilitaram os recursos para o desenvolvimento deste projeto onde foram gastos entre equipamentos, materiais e bolsas por volta de R$750.000,00.

RESUMO

Este trabalho visa o desenvolvimento de uma plataforma básica com o propósito de servir de estrutura para aplicações em robôs móveis. Esta plataforma consta de módulos de comunicação para o meio exterior (Bluetooth, WiFi e ZigBee) e com um barramento de comunicação interior que permite a inclusão de módulos que controlam sensores e atuadores (sensores como de distância, câmeras de vídeo, GPS, Bussola digital, acelerômetros, umidade, temperatura, pressão, etc., e atuadores como motores, servos, válvulas, etc.).

A plataforma básica consta de protocolos de comunicação e um set de comandos tanto nas comunicações via rádio como no barramento interno o que permite o desenvolvimento de novas aplicações. O *software* interno, também é um *software* de código aberto que permite aos pesquisadores, hobbistas, profissionais, alunos e professores modificar e desenvolver qualquer tipo de aplicação tanto para que a plataforma seja montada em estruturas diferentes como também o desenvolvimento de *software* desktop para comando do robô via internet, celular ou qualquer tipo de tele-comunicação futura.

Com este trabalho se pretende preencher um vazio que fará viavel sua aplicação nas áreas de educação básica, esta não apenas nas disciplinas tradicionais, como Matemática e Geografia, mas também em disciplinas transversais, nas quais os alunos poderão aprender a programar rotinas para educação no trânsito, seleção de lixo reciclável, ecologia (reconhecimento de águas), inclusão social e reconhecimentos de MAPas, utilizando sistemas de programação em blocos. No caso dos pesquisadores e hobbistas, os produtos seriam utilizados para o desenvolvimento de novos módulos e outras aplicações mais sofisticadas, podendo tudo isso ser realizado via tele-operação.

Como foi desenvolvido um único sistema a ser usado tanto na educação básica quanto para pesquisas de alto nível tecnológico, o sistema é flexível, possuindo um *software* com programação em blocos, para as escolas de ensino fundamental e médio; e um *software* com programação em C, para pesquisadores e hobbistas.

Palavras-chave: Robótica Móvel, Robôs, Mecatrônica.

ABSTRACT

This study aims at developing a basic platform in order to provide a structure for applications in mobile robots. This platform consists of modules for communication to the outside world (Bluetooth, WiFi and ZigBee) and an internal communication bus that allows the inclusion of modules that control sensors and actuators (such as distance sensors, video cameras, GPS, digital compass , accelerometers, humidity, temperature, pressure, etc ... as motors and actuators, servos, valves, etc ...).

The basic platform consists of communication protocols and a set of commands as much as radio communications in the internal bus that allows the development of new applications. The internal software is also an open source software that allows researchers, hobbyists, professionals, students and teachers to modify and develop any type of application for both the platform is mounted on different structures as well as the development of desktop software for remote control of robot via the Internet, mobile phone or any type of tele-communication future.

This work is intended to fill a void that will make feasible its application in the areas of basic education, this not only in traditional disciplines such as math and geography, but also in cross-disciplines in which students can learn to program routines for traffic education, selection of recyclable waste, ecology recognition (water), social inclusion and recognition of MAPas using systems programming block. For researchers and hobbyists, the products would be used for the development of new modules and other more sophisticated applications, may all be performed via the tele-operation.

Was developed as a single system to be used both in basic education and for research on high technology, the system is flexible, having a software programming blocks to the elementary schools and middle, and a software programming in C for researchers and hobbyists.

***Keywords**: Mobile Robots, Robots, Mecatronic.*

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IFR	*International Federation of Robotics*
CAN	Controller Area Network
MCS	Módulo de Controle de Sessão.
MCR	Módulo de Controle Robótico.
MCC	Módulo de Controle de Comunicação.
MAP	Módulo de alta performance.
HAL	*Hardware Abstract Layer*
MMU	*Memory Management Unit*
SDK	*Software Development Kit*
MRS	Microsoft *Robotics Studio*
VPL	*Visual Programming Language*
PDA	*Personal Digital Assistants*
PTZ	*Pan-Tilt-Zoom*
EMI	*External memory interface*
UAT	*University of Arizona Tire*
CM	Centro de massa
CG	Centro de gravidade
MEMS	Micro Electro-*Mechanical System*
SSP	*Serial Syncronous Protocol*

SUMÁRIO

CAPÍTULO 1

INTRODUÇÃO

Dentre as áreas de Robótica, os robôs móveis de pequeno porte são os que possuem mercados em franca expansão. Segundo previsões do *World Robotic Report* 2005, da *International Federation of Robotics* (IFR), o mercado nos próximos anos terá crescimento de cerca de 21% a.a., ocupando o primeiro lugar em número de equipamentos dentro do total de robôs de serviços (cerca de 21,45%). As plataformas móveis mais sofisticadas possuem dispositivos como sensores de infra-vermelho e a laser, e visão computacional. Diante desse contexto, surgiu o trabalho proposto [01] cujo objetivo é desenvolver uma plataforma universal para um robô móvel que possua uma "arquitetura aberta", possibilitando que o usuário desenvolva, ele mesmo, uma nova aplicação. Além do desenvolvimento dessa plataforma móvel aberta, o trabalho desenvolveu três módulos a serem usados com tal estrutura: sistema de visão embarcado, pinça mecânica, e *software* de comunicação para tele-operação.

Este trabalho visou definir uma plataforma básica que é constituída por uma estrutura mecânica que possui interfaces para comunicação com o meio exterior e, internamente, constituída por uma "placa mãe" com um barramento de expansão. A estrutura mecânica dá suporte a todo o circuito elétrico interno que se comunica com o exterior por meio de várias interfaces onde são conectados os acessórios como pinça mecânica (manipulação), mecanismos de movimentação (locomoção), visão, sensores, etc. Nestas interfaces foram definidas: uma fixação mecânica, um conector, um barramento de comunicação, a função de cada linha do barramento e seus protocolos de comunicação. Internamente, a placa mãe contem, um processador com todos os periféricos necessários para o seu funcionamento, um *kernel* e um barramento que permite a instalação de placas de expansão (módulos) para implementação dos acessórios. Além disso, foi desenvolvido um ambiente modular de programação em "G" (blocos gráficos) para que qualquer pesquisador, estudante ou empresa, possa criar novos acessórios (módulos) com diferentes aplicações nas á-

reas de Educação e Pesquisa, inclusive com possibilidade, de adaptação da tecnologia gerada para exploração dos crescentes mercados de robôs para as áreas: médica (*home care*) e de segurança.

Neste trabalho, ademais do desenvolvimento deste robô com plataforma universal, foi também, desenvolvido três módulos: visão embarcada, pinça mecânica e tele-operação.

No caso dos pesquisadores e hobbistas, este trabalho permite o desenvolvimento de novos módulos e outras aplicações mais sofisticadas, podendo tudo isso ser realizado via tele-operação.

Como foi desenvolvido um único sistema a ser usado tanto na educação básica quanto para pesquisas de alto nível tecnológico, o sistema é flexível, possuindo um *software* com programação em blocos, para as escolas de ensino fundamental e médio; e um *software* com programação em C, para pesquisadores e hobbistas.

Sistemas com esse grau de sofisticação, flexibilidade e abrangência não existem na América Latina. Além disso, a proposta de se utilizar a Internet para tele-operação, isto é, controlar e programar o robô remotamente via WEB, possibilitará uma gama de sub-projetos a serem desenvolvidos em outras oportunidades, em áreas como tele-robótica e tele-medicina. A Interface remota via Internet também possibilita que o sistema robótico preparado para a educação seja utilizado como ferramenta de ensino à distância em escolas que não dispõem de recursos para construírem um laboratório de robótica. Estas escolas podem adquirir apenas o *software* e utilizar pela Internet os robôs disponíveis no Laboratório de Robótica à Distância que poderá estar disponível em um dos estabelecimentos de ensino.

CAPÍTULO 2

CONCEPÇÃO DO TRABALHO

2.1. OBJETIVO

O objetivo deste trabalho é apresentar uma plataforma básica com o propósito de servir de estrutura para aplicações em robôs móveis. Esta plataforma consta de módulos de comunicação para o meio exterior (como Bluetooth, WiFi e ZigBee [02]) e com um barramento de comunicação interior que permita a inclusão de módulos que controlam sensores e atuadores (sensores como de distância, câmeras de vídeo, GPS, Bússola digital, acelerômetros, umidade, temperatura, pressão, etc. e atuadores como motores, servos, válvulas, etc.). Para o desenvolvimento desta plataforma foi necessário a utilização de uma estrutura mecânica, ou seja, um projeto mecânico onde será alojada a plataforma. Este projeto mecânico não limita a plataforma que pode ser estendida para qualquer outra aplicação, mas a mecânica adotada serve para mostrar a viabilidade do proposto neste trabalho.

2.1.1. Especificações

Foram definidas as especificações da plataforma robótica como um ponto de partida, para ser atendidas por todas as partes envolvidas no trabalho [01]. Sendo estas especificações gerais as seguintes:

2.1.2. Especificações de Design

Definidas as especificações que determinam a aparência e condições de segurança conforme as normas ABNT [03].

1. Robustez
2. Segurança
3. Usabilidade
4. Meio ambiente
5. Não ter aparência de brinquedo
6. Não ter aparência de Robô Militar

2.1.3. Especificações de Características

Estas especificações comparam a plataforma robótica com um homem, relacionando cada parte do corpo humano com as parte da plataforma robótica.

1. Uma Unidade Básica = Locomoção = Chassi+Rodas = Tronco+Pés
2. Módulo Comunicação = Boca / Ouvidos
3. Módulo Braço/Pinça = Braços a Mãos
4. Módulo Visão = Olhos
5. Unidade Computacional= Cérebro

2.1.4. Especificações Técnicas gerais

As especificações técnicas definem os limites da plataforma robótica para os alvos que devem ser atingidos.

1. Autonomia diferenciada = 4 à 8h de funcionamento com recarga rápida
2. Precisão / repetibilidade nos movimentos / boa mobilidade
3. Apto para se locomover em ambientes fechados (*in door*): pisos lisos.
4. Ambientes externos (*out door*): terra batida e grama baixa
5. Subir rampa máxima de 30º
6. Peso para movimentação (Módulo Pinça): 400gr

7. Peso máximo da plataforma robótica: 18 kg
8. Dimensões básicas: 600 mm x 400 mm x 350 mm

2.2. Benchmark

Foram escolhidos os modelos de Robôs que são concorrentes da plataforma robótica e que foram considerados no projeto como *benchmark.*

1. K-Team = modelo Khepera/Koala. [04] Site http://www.k-team.com
2. ECA = CAMELEON LAB. [05] Site: http://www.eca.fr

2.3. Concepção do protótipo

Foi definido que a plataforma robótica será implementada num robô com quatro rodas com esterçamento independente e duas delas (rodas) são motoras. Consta das seguintes partes:

1. Sistema de alimentação
2. Sistema de locomoção
3. Chassi
4. Carenagem
5. Sistemas de sensores
6. Sistema de atuadores
7. Módulos lógicos e sistema de comunicação
8. Pinça mecânica
9. Sistema de Visão

2.4. Concepção lógica

Uma formação modular foi definida de maneira que cada módulo tenha certa autonomia e se comunique com os outros módulos por meio de um protocolo e uma biblioteca definida. A Figura -1, resume estes módulos e o detalhamento de cada módulo esta exposto nos capítulos seguintes.

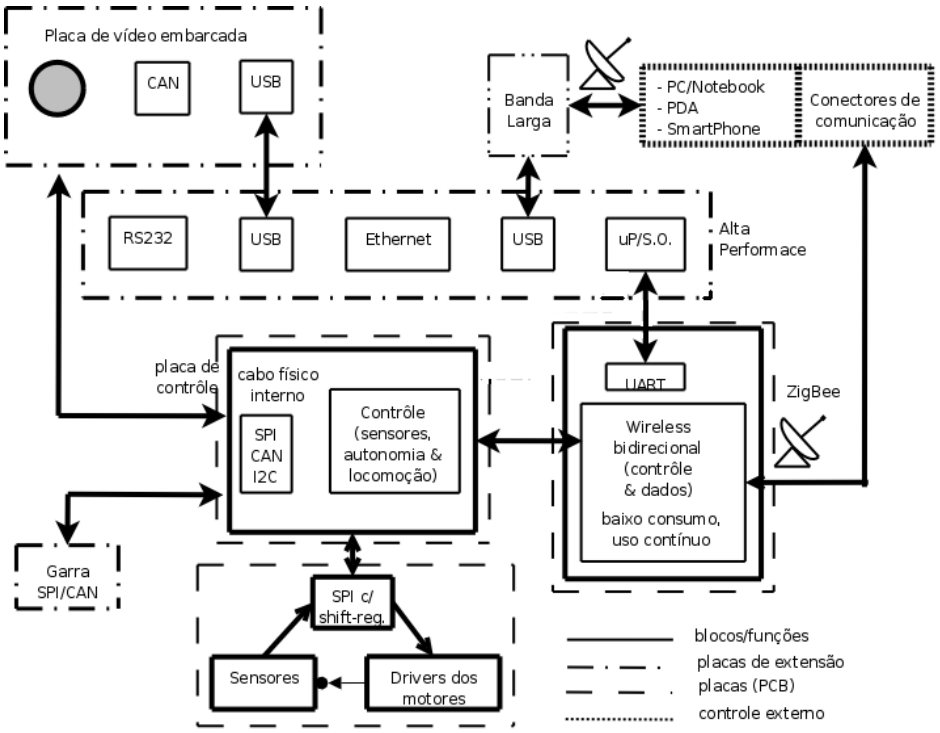

Figura -1 Módulos Lógicos [01]

CAN: Protocolo de comunicação digital serial Controller Area Network (CAN) tem sido elaborados e adotados para aplicações que utilizam eletrônica embarcada [06].

Os módulos do sistema são agrupados em duas diferentes categorias:

1. Módulos básicos: Implementam as funcionalidades essenciais do robô.

2. Módulos de extensão: Não são necessários para o funcionamento básico do robô, mas quando utilizados agregam funcionalidades extra.

A seguir são descritos os módulos dentro de suas respectivas categorias.

2.4.1. Módulos básicos

Esta categoria abrange todos os módulos essenciais ao funcionamento do robô. As placas eletrônicas que os compõem são fisicamente atreladas à estrutura física do robô. Ou seja, com estes módulos implementados o robô deve ser funcional.

É importante ressaltar que tendo apenas os módulos básicos o robô deverá atuar como um "escravo", executando passo-a-passo os comandos que lhe são fornecidos. Os módulos básicos não prevêem qualquer controle alto nível ou inter-relação entre informações e controle dos periféricos do robô. Este tipo de controle esta previsto com a utilização dos módulos da categoria de extensão, descritos mais adiante.

Os módulos básicos são:

1. **Módulo de controle da locomoção**. Implementa uma interface digital (SPI) para o controle digital dos motores e leitura dos sensores necessários à gerência da locomoção. Também é responsável por gerir de forma autônoma os *driver*s de potência dos motores e seus mecanismos de retenção.

2. **Módulo de comunicação**. Responsável por receber os comandos que controlarão o robô. Há duas interfaces previstas para a recepção de comandos por este módulo: física (UART) e remota (ZigBee). Independentemente da origem do comando, o módulo é responsável por formatá-lo para um protocolo conhecido do "núcleo robótico" que finalmente executará o comando. No caso de comandos remotos este módulo é responsável, também, por determinar a identificação do robô e situá-lo na rede de comunicações (ZigBee). Outra tarefa do módulo é decidir a quem (qual "*master*") o robô obedecerá. Obs: este módulo possui um micro-controlador que garante sua autonomia em relação aos outros módulos do robô.

3. **Módulo do núcleo robótico**. Possui o micro-controlador principal do robô responsável por gerenciar o controle digital básico de todos os periféricos do robô. Este módulo recebe comandos do módulo de comunicação e cuida para que estes comandos sejam executados pelos periféricos (locomoção e periféricos adicionais). Também tem a função de, a partir de comandos do módulo de comunicação, ler e repassar os estados dos periféricos ao solicitador.

2.4.2. Módulos de extensão

Os módulos de extensão não são necessários ao funcionamento básico do robô, no entanto ampliam suas capacidades. Duas foram as capacidades projetadas a serem estendidas: inteligência/alta-performance e periféricos.

Outro ponto importante é que estes módulos foram projetados para serem facilmente personalizados ou até mesmo substituídos por módulos desenvolvidos pelo próprio usuário. Como conseqüência, a concretização física (*hardware*) destes módulos passa a ter menos importância, para o projeto, do que a definição dos protocolos de comunicação (*hardware e software*) que os integra aos módulos básicos e do *software* que os gerencia.

A seguir são descritos os módulos de extensão do ponto de vista do robô.

Alta performance. *Hardware* embarcado, fisicamente conectado ao robô (Figura -1) através de uma porta serial (UART), capaz de suportar um sistema operacional para que os aplicativos que nele rodarão sejam independentes do *hardware* em questão. O papel deste módulo é o de executar as atividades que envolvam grande poder de processamento ou comunicação sem onerar o "núcleo robótico" que estará ocupado com a gerência básica do robô. Como exemplo, tem-se a captura e processamento de imagens. Outra função deste módulo é a de permitir a agregação de novas tecnologias ao robô através de interfaces padrão (ex: USB, *WiFi* e Bluetooth). Uma terceira função é a autonomia do robô. Este módulo é capaz de ser programado com uma aplicação de controle do robô que o faça independente de controles externos e o faça agir como um "peer" e não mais como um "escravo".

Periféricos adicionais. Os módulos periféricos representam *hardware*s que podem ser fisicamente adicionados ao sistema. Do ponto de vista eletrônico/computacional estes periféricos serão vistos como módulos capazes de se comunicar utilizando SPI (*hardware*) e seguindo um protocolo de *software* a ser definido ao longo do desenvolvimento do projeto, visto que alguns tipos de periféricos deverão ser suportados.

CAPÍTULO 3

DESENVOLVIMENTO DO PROJETO

3.1.Projeto de *Software* Embarcado

O Robô Universal foi concebido para ser um robô modular e que permitisse ao usuário expandir não só os seus comandos como também suas partes mecânicas, adicionando garras, braços, sensores ou outros periféricos robóticos que se deseje implementar.

Seguindo o conceito de modularidade, o próprio *hardware* básico de controle do robô é dividido em duas partes: o *hardware* essencial e o *hardware* opcional.

O *hardware* essencial é composto por dois microcontroladores que se comunicam por uma UART. Os *software*s que rodam neste *hardware* são escritos em linguagem C. Já o *hardware* opcional é composto por uma placa qualquer que seja capaz de rodar o sistema operacional Linux. Os *software*s que rodam no *hardware* opcional são escritos em linguagem C++ e são escritos tendo como base o sistema operacional Linux. A Figura 2, mostra as principais divisões de *hardware* e *software* do robô.

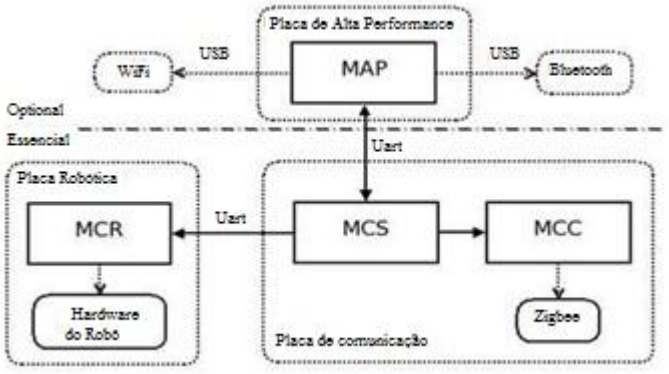

Figura 2: Visão geral da arquitetura de software do robô explicitando sua relação com o hardware essencial e opcional [01].

Os significados das letras dos módulos são:

MCS: Módulo de Controle de Sessão.

MCR: Módulo de Controle Robótico.

MCC: Módulo de Controle de Comunicação.

MAP: Módulo de alta performance.

As setas indicam a dependência entre os módulos

O primeiro microcontrolador da parte essencial do robô é responsável por gerir o *hardware* relacionado aos seus sensores e atuadores. O *software* implementado para rodar neste microcontrolador é chamado de Módulo de Controle Robótico (MCR). Este módulo age como um robô escravo, executando sequencialmente os comandos que a ele cheguem pela UART. Já o segundo microcontrolador do *hardware* essencial é responsável por gerir a comunicação de controle do robô e também por manter a sua identidade. A comunicação de controle é implementada com o protocolo ZigBee [02] pelo Módulo de Controle de Comunicação (MCC). A identidade do robô é implementada pelo Módulo de Controle de Sessão (MCS) que implementa o conceito de sessão aos comandos provenientes dos controladores do robô, de forma a impedir que o robô responda a dois controladores simultaneamente. Também o MCS se incumbe da identificação e autenticação dos controladores.

O *hardware* opcional do robô tem o objetivo de agregar autonomia e comunicação banda larga ao robô. O *software* que roda neste *hardware* é chamado de Módulo de Alta Performance (MAP). Para agregar autonomia ao robô, o MAP permite que os controladores carreguem aplicativos robóticos. Estes aplicativos podem, então, controlar o *hardware* essencial do robô através de comandos enviados diretamente ao MCS. Também, o MAP permite que os controladores se comuniquem com ele através de adaptadores USB para Bluetooth ou WiFi. Estes comandos podem ser apenas passados ao MCS, ou podem ser enviados aos aplicativos robóticos, permitindo a ampliação dos comandos aceitos pelo robô.

As sessões seguintes deste documento detalham as arquiteturas internas de cada um dos 4 módulos de *software* que compõem o robô(Figura 2).

Por último, é detalhada a arquitetura e funcionamento da biblioteca utilizada para a comunicação interna entre cada um dos módulos que se situam em processadores distintos e que precisam se comunicar pela interface serial UART.

3.1.1. Módulo de Controle de Sessão

O Módulo de Controle de Sessão (MCS) representa a identidade do robô. Ele é responsável por armazenar as informações sobre o robô, tais como identificador único, nome, versão do robô, versão do protocolo de comunicação e etc. Também, este módulo, se incumbe da autentificação dos controladores que queiram comandar o robô e pelo estabelecimento das sessões de controle.

Os comandos provenientes de um controlador só serão aceitos caso estejam dentro de uma sessão de controle previamente estabelecida entre o controlador e o robô. O MCS decide quem e quando as sessões podem ser abertas ou fechadas.

Todo comando que chega ao robô é primeiramente encaminhado ao MCS, independentemente da interface de origem (WiFi do MAP, Bluetooth do MAP ou ZigBee do MCC) e este decide qual parte do robô será encarregada pela execução do comando. Após executado o comando, quem o executou retorna a resposta para o MCS. Este, por sua vez, é encarregado de encaminhar a resposta ao controlador que originou o comando.

O MCS pode receber comandos provenientes do MCC ou do MAP. Os comandos recebidos podem ser delegados ao MCR ou ao MAP. Note que enquanto o MCR age apenas como escravo do MCS, pois apenas executa os comandos que a ele são atribuídos; e o MCC age apenas como mestre do MCS, pois apenas envia comandos ao MCS; o MAP pode agir tanto como escravo, pois pode implementar comandos robóticos, quanto como mestre, pois pode agir como um controlador ou repassar comandos provenientes de um controlador externo.

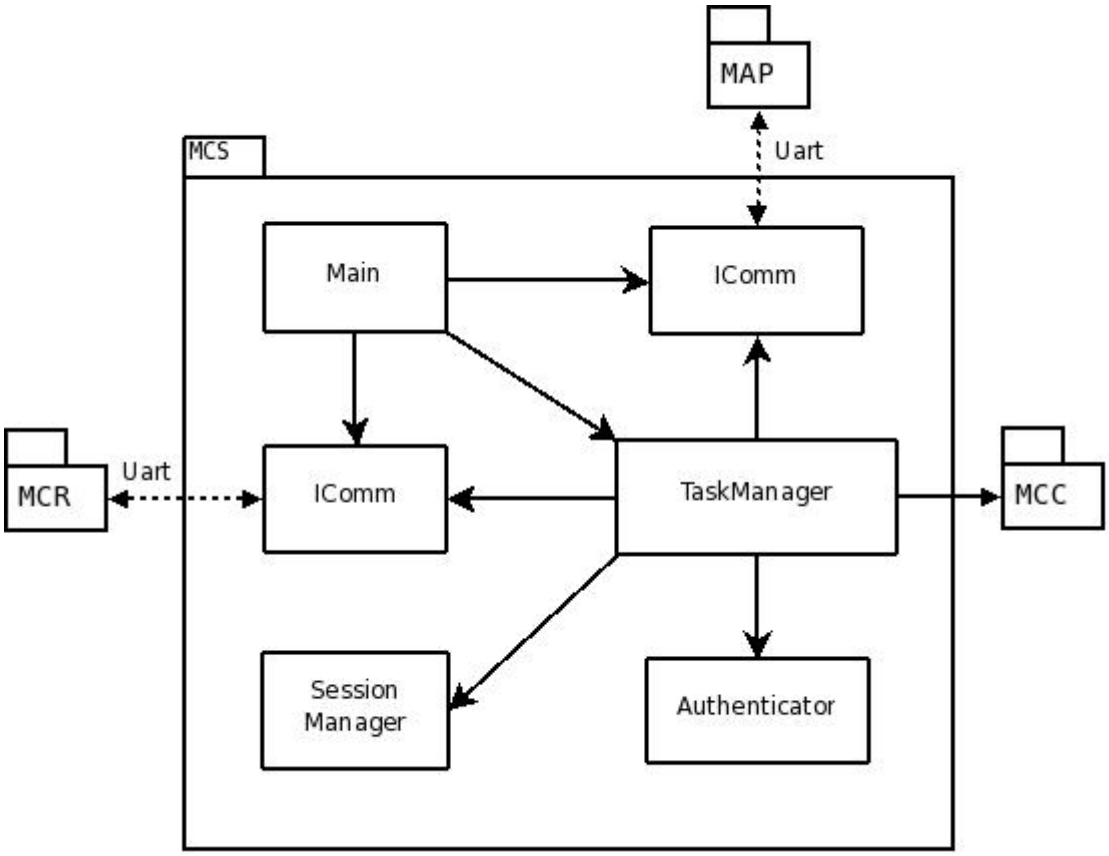

Figura 3: Módulo de Controle de Sessão e seus sub-módulos [01].

A Figura 3 mostra o MCS e seus sub-módulos. Além das duas instâncias do módulo IComm, utilizados para a comunicação serial com as outras placas (MCR e MAP) e do bloco que representa o programa principal (Main), os seguintes sub-módulos compõem o MCS:

Na figura as setas cheias indicam as dependências entre os sub-módulos e as setas pontilhadas indicam fluxos de dados.

TaskManager: Tem a função de gerenciar a execução dos comandos, ou seja, escolher o módulo que se encarregará da execução do comando, encaminhar o comando, aguardar a resposta e retorná-la ao solicitante;

SessionManager: Tem a função de criar sessões, manter as informações sobre as sessões abertas, estabelecer as prioridades entre os controladores com relação às sessões e decidir quando uma sessão deve ser finalizada;

Authenticator: Armazena as informações que identificam o robô e as informações necessárias à autenticação dos controladores.

Basicamente, o algoritmo do programa principal, além de iniciar o *hardware* específico do microcontrolador, pode ser resumido da seguinte forma [01]:

 # Iniciações

#

1. Inicia hardware específico do microcontrolador JN5139;

 # Trata as tarefas que estão chegando ao MCS:

#

2. Se puder inserir uma tarefa em TaskManager e houver tarefa a ser lida da IComm associada ao MCR, lê a tarefa e insere-a no TaskManager;

3. Se puder inserir uma tarefa em TaskManager e houver tarefa a ser lida da IComm associada ao MAP, lê a tarefa e insere-a no TaskManager;

4. Se puder inserir uma tarefa em TaskManager e houver tarefa a ser lida do MCC (ZigBee), lê a tarefa e insere-a no TaskManager;

 # Trata as tarefas que devem sair do MCS:

#

5. Se houver tarefa a ser lida do TaskManager, então:

a. Se puder enviar uma tarefa pela IComm associada ao MCR, lê a tarefa do TaskManager e insere-a na IComm associada ao MCR;

b. Se puder enviar uma tarefa pela IComm associada ao MAP, lê a tarefa do TaskManager e insere-a na IComm associada ao MAP;

c. Se puder enviar uma tarefa ao MCC (ZigBee), lê a tarefa do TaskManager e a envia ao MCC (ZigBee);

6. Retorne ao passo 2;

A seguir são detalhados os projetos de cada um dos sub-módulos do MCS.

3.1.1.1. TaskManager

O módulo TaskManager centraliza a recepção dos comandos provenientes dos controladores que chegam ao robô e decide qual módulo será responsável por tratá-los. Da mesma forma, o módulo tem a responsabilidade de receber de volta as respostas dos sub-módulos e encaminhá-las para o controlador que enviou o comando.

Os comandos do Robô Universal podem ser divididos em 5 categorias: comandos de configuração, comandos de baixo nível, comandos de alto nível, comandos de *slots* e comandos expandidos. Os comandos de configuração são de responsabilidade do próprio MCS. Os comandos de baixo nível, de alto nível e de *slots* são de responsabilidade do MCR. Já os comandos expandidos são de responsabilidade do MAP. Note que os comandos que chegam por uma interface deverão ter suas respostas retornadas pela mesma interface.

Para desempenhar seu papel, o módulo TaskManager, realiza as seguintes tarefas:

- Conhecer o formato do pacote utilizado pelo protocolo de comunicação externa com o Robô Universal;
- Conhecer a identidade do robô;
- Responder pelos comandos de configuração do robô;
- Validar os pacotes recebidos;
- Autenticar o remetente do pacote;
- Abrir e fechar sessões, decidindo sobre abertura de sessões prioritárias;
- Encaminhar comandos de baixo-nível para o Módulo de Controle Robótico (MCR) e receber as possíveis respostas ao comando;
- Encaminhar comandos de alto-nível para o MCR ou para o MAP, conforme seu tipo.
- Receber as respostas do MCR e do MAP, empacotá-las e encaminhá-las para o controlador que originou o comando utilizando a interface correta (MAP ou MCC).

O sub-módulo TaskManager está implementado como uma biblioteca que disponibiliza as seguintes funções [01]:

```
/*
* Inicia o TaskManager
*/
void task_init( void );
/*
* Retorna verdadeiro se houver espaço para inserir uma tarefa.
*/
Bool task_canPutTask( void );
/*
* Insere a tarefa 'taskData' proveniente do módulo 'mod'.

,*/
Bool task_putTask( Module mod, uint8 size, uint8 *taskData );
/*
* Retorna verdadeiro se houver alguma tarefa pendente a ser enviada ao
* referido módulo.
*/
Bool task_hasTask( Module *dstModule );
/*
* Lê a tarefa pendente a ser enviada ao referido módulo.
*/
uint8 task_getTask( Module *dstModule, uint8 *taskData );
```

Quando uma tarefa é inserida no TaskManager ela é um pacote (vetor de bytes) que pode seguir o formato definido pelo protocolo de comunicação externa do Robô Universal [01]. Este é o caso dos comandos provenientes dos controladores, ou o pacote pode seguir o formato definido pelo Protocolo de Comunicação Interno descrito no arquivo TaskManager.h. Este é o caso das respostas provenientes dos módulos MCR ou MAP.

Distinguir uma tarefa entre comando ou resposta é importante para encontrar seu destino quando ela for retirado do TaskManager. Enquanto uma resposta sempre irá para o lugar de onde veio o comando a ela associado, o destino de um comando dependerá da categoria a que o comando pertence. Sendo que comandos de sessão são executados no próprio MCS pelo sub-

módulo SessionManager; comandos de baixo nível, alto nível e de *slots* são enviados ao MCR e comandos de expansão são enviados ao MAP.

Outro ponto em que um comando difere de uma resposta é que os comandos, mesmo após serem retirados do TaskManager e retransmitidos ao seu destino, uma estrutura com algumas informações a cerca do comando (TaskInfo) é mantida pelo TaskManager para poder associar uma resposta que chegue ao comando que foi enviado. Ou seja, quando um comando é retirado do TaskManager é mantida uma estrutura TaskInfo com algumas informações. Já quando uma resposta é retirada, as informações do comando a ela associados é liberada do TaskManager.

3.1.1.2. SessionManager

O módulo SessionManager é responsável por armazenar as informações sobre a sessão corrente, assim como estabelecer as prioridades dos controladores para a abertura de sessões. Nesta versão apenas umas sessão poderá estar aberta por vez. Este módulo não possui dependências, ou seja, sua implementação se restringe às primitivas da linguagem.

Um exemplo da importância da centralização dos comandos pode ser observado na seguinte situação. Um controlador está com uma sessão de controle aberta e pede ao robô que vire à esquerda. Neste mesmo tempo outro controlador pede ao robô que vire à direita. O MCS vetará o segundo controlador, pois este está enviando um comando fora de sessão e, portanto, não tem controle sobre o robô. Por outro lado, se o segundo controlador pede para abrir uma sessão de controle o MCS deverá decidir qual dos controladores possui maior prioridade. Caso o segundo controlador tenha maior prioridade, então, é enviado um comando de finalização de sessão ao primeiro controlador e é estabelecida uma nova sessão de controle com o segundo controlador.

A seguir são descritos os métodos a serem exportados pelo módulo SessionManager [01].

```
typedef unsigned char Bool;
typedef char int8;
typedef unsigned short uint16;

typedef struct {
ushort session_id; // Session identification.
ushort controler_id; // Controler identification.
```

```
uchar connection; // From where the controler is connected.
} session_t;

/*
* Inicia o SessionManager.
*/

void session_init( void );
/*
* Abre uma nova sessão associada ao controlador 'controler_id'. 'connection'
* representa o módulo pelo qual o controlador está conectado.
* Se a sessão for aberta com sucesso, preenche 'sessionId' com o identificador
* da nova sessão.
* Se outra sessão foi fechada em consequência da abertura da nova sessão,
* preenche 'closed' com os dados da sessão finalizada.
*
* Retorna:
* -1: se a sessão foi aberta e outra sessão foi fechada;
* 0: se a sessão não pode ser aberta;
* +1: se a sessão foi aberta e nenhuma outra sessão foi fechada.
*/
int8 session_open( uint16 controllerId, Module connection, uint16 *sessionId,
Session *closed );
/*

* Fecha a referida sessão.
* Caso a sessão não exista ou o controlador da sessão fornecida não seja o
* mesmo da sessão armazenada, a sessão não é fechada e retorna 'falso'.
*/
Bool session_close( session_t *session );

/*
* Retorna verdadeiro se a sessão passada estiver aberta e pertencer ao
* referido controlador.
*/
Bool session_isOpened( uint16 sessionId, uint16 controllerId );

/*
* Retorna o módulo pelo qual a sessão está sendo mantida.
*/
Module session_getConnection( uint16 sessionId );
```

3.1.1.3. Authenticator

O módulo Authenticator é responsável por verificar a autenticidade dos controladores e por armazenar e fornecer a identidade do robô.

Nesta versão, o Authenticator não faz controle de senha, verificando apenas se a identificação do controlador é válida. Este módulo existe para que o mecanismo de autenticação com senha de acesso possa ser futuramente incluído de forma localizada. Este módulo não possui dependência, ou seja, sua implementação se restringe às primitivas da linguagem.

A seguir são descritos os métodos a serem exportados pelo módulo Authenticator [01].

```
typedef unsigned char uchar;
typedef unsigned short ushort;
void auth_init( void );
/*
* Modifica/Lê a identificação do robô.
* A identificação do robô é utilizada para endereçar o robô nos campos
* 'Src' e 'Dst' dos pacotes de comunicação externa com o Robô Universal.
*/
void auth_setId( uint16 robotId );
uint16 auth_getId( void );
/*
* Modifica/Lê o nome do robô.
*/
void auth_setName( char *robotName );
void auth_getName( char *robotName );
/*
* Verifica se o referido controlador tem permissão para abrir uma sessão.
*/
Bool auth_isValid( uint16 controllerId );
```

3.1.2. Módulo de Controle Robótico

O Módulo de Controle Robótico (MCR) é o *software* responsável pelo controle do *hardware* do robô. O MCR fornece uma interface serial (UART) capaz de receber pacotes de mensagens com os comandos robóticos relacionados ao *hardware*. O módulo então executa os comandos, quando possível, e retorna pacotes que representam as respostas aos comandos.

Os comandos aceitos pelo MCR pertencem a três categorias: comandos de baixo nível, comandos de alto nível e comandos de *slots*.

Os comandos de baixo nível são aqueles relacionados diretamente ao *hardware* do robô, ou seja, permitem a leitura e escrita nos periféricos fixos ao robô, tais como o acionamento individual dos motores e a leitura direta dos sensores.

Os comandos de alto nível são comandos construídos pela composição de comandos de baixo nível, e são disponibilizados para facilitar as tarefas cotidianas mais complexas. Por exemplo, o comando para fazer com que o robô realize uma curva exige a coordenação dos quatro servos-motores do robô. Cada servo-motor é utilizado para virar cada uma das quatro rodas do robô. Desta forma, o comando de alto nível utilizado para virar o robô como um todo é construído com base nos comandos de baixo nível que virar cada um dos 4 servos-motores de forma sincronizada.

Os comandos de *slots* são utilizados para a comunicação com os periféricos que por ventura estejam conectados ao robô. Estes são comandos genéricos que permitem a identificação do periférico opcional e também o envio de pacotes de mensagens a eles. A semântica dos pacotes enviados a um determinado *slot* depende do periférico a ele conectado.

A Figura 4 , mostra o MCR e seus sub-módulos. Este módulo está dividido em duas camadas: a camada de controle e a camada de abstração do *hardware*, ou *Hardware Abstract Layer* (HAL).

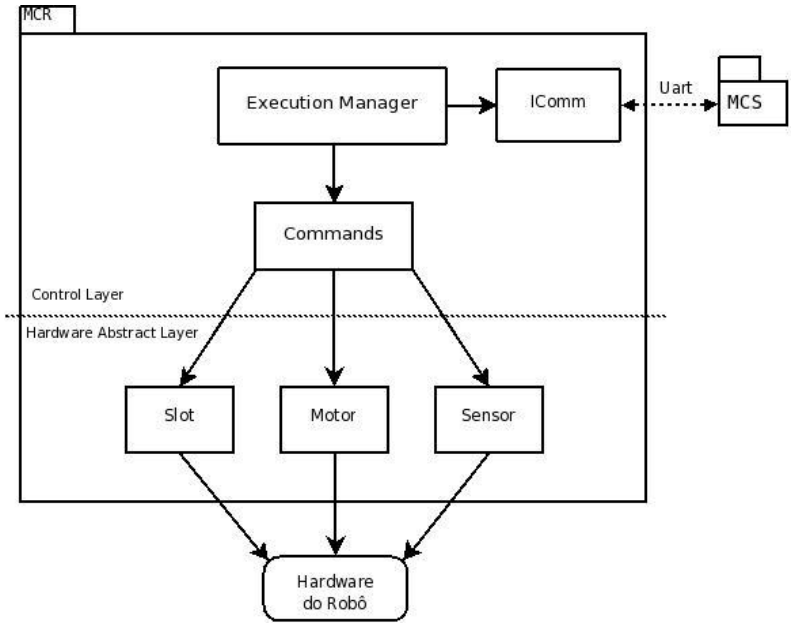

Figura 4: Módulo de Controle Robótico e seus sub-módulos [01]

As setas cheias indicam as dependências entre os sub-módulos.

As setas pontilhadas indicam fluxos de dados.

A camada de controle é logicamente ligada, e dependente, dos comandos implementados pelo MCR. Sua função é a de receber os pacotes contendo os comandos, decodificar os comandos e seus parâmetros, gerenciar a execução dos comandos, receber e serializar suas respostas transformá-las em pacotes e enviá-las de volta pela interface serial.

A camada de abstração do *hardware* (HAL) conhece os periféricos do robô, sabendo os *ports* e pinos pelos quais são acessados, seus modelos e seus comandos de controle.

A divisão do MCR em duas camadas permite que a implementação de um novo comando robótico que utilize o *hardware* corrente do robô seja feita modificando apenas a camada de controle. Da mesma forma, a mudança da forma de acesso a um periférico robótico pode ser realizada modificando apenas o *software* da HAL. Também, a substituição de um periférico por outro que possua a mesma semântica (sirva para o mesmo fim) pode ser realizada modificando apenas a HAL.

Além de uma instância da biblioteca de comunicação, IComm, utilizada para a comunicação serial do módulo, o MCR é composto pelos sub-módulos descritos a seguir.

3.1.2.1. ExecutionManager

Este sub-módulo recebe os pacotes provenientes da interface serial, decodificá-los, verifica se o comando é válido e, se for o caso, executa-o. Em seguida lê sua resposta, empacota-a e envia o pacote de volta pela serial, mantendo a associação da resposta com o comando pedido.

Para cada comando aceito pelo MCR é criada uma função específica que o executa. Esta função conhece os parâmetros do comando e seus respectivos tipos, assim como os valores de retorno e seus tipos. Cada uma destas funções decodifica os parâmetros do pacote, chama uma função que implementa o comando no sub-módulo Commmands e por último serializa os valores de retorno.

Quando um comando é recebido ele é executado sincronamente. Desta forma, enquanto a função que realmente implementa o comando (situada em Command) não retornar, nenhum outro comando será executado. Esta limitação faz com que o robô não atenda a nenhum outro comando enquanto não houver finalizado o comando que estiver sendo executado.

A seguir é mostrado o algoritmo para o fluxo principal do módulo.

1. Espera chegar um novo comando pelo módulo 'IComm';
2. Verifica se é um comando válido.
3. Se o comando não for válido, retorna uma mensagem de comando-inválido para 'IComm' e volta ao passo (1).
4. Verifica se o comando pode ser executado.
5. Se o comando não puder ser executado, retorna uma mensagem de “comando-não-pode-ser-executado” para IComm e volta ao passo (1).
6. Chama a função de 'Command' que executa o comando.
7. Informa o resultado da execução do comando para 'IComm'.

3.1.2.2. Commands

O sub-módulo Command implementa todos os possíveis comandos aceitos pelo MCR. Para isso, as funções deste sub-módulo utilizam-se das funções implementadas na camada HAL, como mostrado na Figura 4**: Módulo de Controle Robótico e seus sub-módulos [01]**.

Note que o sub-módulo ExecutionManager conhece apenas o módulo Commands e este, por sua vez, é o único que tem acesso ao HAL. Esta imposição de dependência é feita para que mudanças na natureza dos comandos com ganchos (*hooks*) ou verificações de permissão de execução possam facilmente ser implementadas no sub-módulo Commands sem alteração em nenhum outro sub-módulo.

As funções exportadas pela interface deste sub-módulo apresenta uma relação de "um para um" com os comandos exportados pelo MCR. Ou seja, com os comandos de baixo e alto nível e os comandos de *slots* do protocolo externo do robô. O AP C.1.2 mostra a interface deste sub-módulo.

3.1.2.3. Motor

Este sub-módulo implementa todas as funções associadas aos motores do robô. Estas funções são específicas para cada modelo e funcionalidade de cada um dos motores existentes no robô. As funções exportadas por este sub-módulo abstraem os detalhes técnicos específicos dos motores, concentrando-se na semântica que cada motor possa ter na composição do robô. O AP C.1.2.1 mostra a interface deste sub-módulo.

3.1.2.4. Sensor

Este sub-módulo implementa as funções associadas à leitura e configuração dos sensores do robô. As funções exportadas por este sub-módulo abstraem os detalhes técnicos específicos dos sensores, concentrando-se na semântica que cada um possa ter na composição do robô. O AP C.1.2.2 mostra a interface deste sub-módulo.

3.1.2.5. *Slot*

Este sub-módulo implementa as funções para a comunicação com os *slots* do robô. O robô possui 8 *slots*. Cada *slot* possui uma interface serial utilizada para a comunicação com algum periférico que possa estar conectado.

As funções exportadas por este sub-módulo permitem a leitura de informações que identificam o periférico conectado e o protocolo de comunicação utilizado por ele, permitindo o seu controle por parte de algum controlador do robô. O AP C.1.2.3 mostra a interface deste sub-módulo.

O protocolo utilizado por um periférico conectado a um *slot* é específico do periférico e é tratado como simples mensagens por este sub-módulo. Note que é de responsabilidade do protocolo especificado pelo periférico permitir a leitura de seus possíveis estados, assim como o acionamento de seus atuadores de forma a possibilitar seu controle.

3.1.3. Módulo de Controle de Comunicação

O Módulo de Controle de Comunicação (MCC) é responsável pela comunicação de controle do robô e é implementada com base no protocolo para comunicação sem fio chamado ZigBee [02] - .

O protocolo ZigBee é implementado por um *software* básico chamado ZigBeeStack [05] - [06] - [08] - disponibilizado pela empresa Jennic [06] - , fabricante do microcontrolador JN5139 [07] - que possui o *hardware* necessário para a comunicação sem fio.

Este módulo possui apenas um *wrapper* para adaptar as formas de chamar o ZigBeeStack ao modelo utilizado pelo MCS, como mostra a Figura 5.

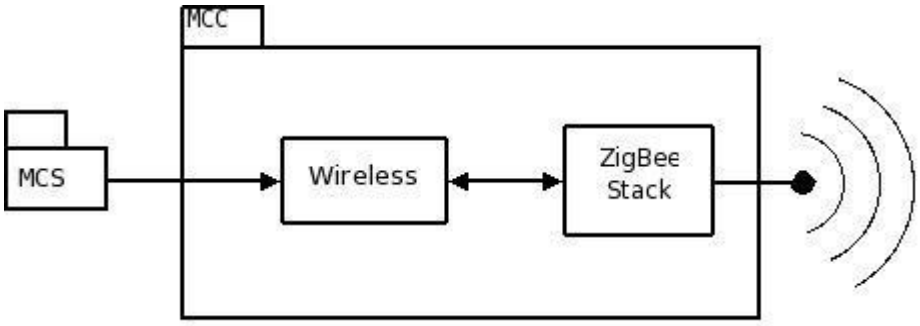

Figura 5: Módulo de Controle de Comunicação e seus sub-módulos [01].

As setas cheias indicam as dependências entre os sub-módulos.

3.1.3.1. Wireless

O sub-módulo Wireless desempenha o papel de *wrapper* do ZigBeeStack para o MCS, implementando as seguintes funções exportadas por sua interface [01]:

```
/*
* Inicia o controle da comunicação sem fio.
*/
void wireless_init( void );
/*
* Trata os possíveis eventos associados à formação da rede sem fio, tais como:
* - Varredura de canais finalizada. Utilizado na iniciação da rede;
* - Pedido para participação na rede recebido;
*/
void wireless_step( void );
/*
* Retorna verdadeiro se puder enviar uma mensagem pela rede sem fio.
*/
Bool wireless_canSend( void );
/*
* Envia a mensagem contida em 'data' com tamanho de 'size' bytes.
* Retorna o número de bytes realmente enviados.
*/
uint8 wireless_send( uint8 *data, uint8 size );
/*
* Se houver uma mensagem a receber, preenche 'data' com ela e retorna seu
* tamanho, em bytes.
```

```
*/
uint8 wireless_recv( uint8 *data );
```

Apesar de a interface do MCC, exportada pelo sub-módulo Wireless estar apta a utilizar todas as características do protocolo ZigBee, por simplicidade, para este trabalho optou-se por uma alternativa mais simples e ágil de implementação.

O conceito de proles fornecido pelo ZigBee não foi utilizado. Ao invés disso, o robô foi programado como Coordenador de uma rede IEEE 802.15.4 [09] - [010] - [011] - [012] - (camada subjacente à especificação ZigBee) e aceita a associação de outros nós à rede, que são interpretados como controladores do robô.

Toda a comunicação é realizada utilizando-se o envio e recepção de vetores de bytes. Caso o conceito de proles fosse utilizado poder-se-ia utilizar dados tipados na comunicação. Também poder-se-ia ter a publicação dos comandos aceitos pelo robô (proles) permitindo uma real integração em redes ZigBee criadas por terceiros. Para isso, também seria necessária uma modificação na topologia da rede atual. O robô deixaria de desempenhar o papel de *Coordinator* e teria que se conectar a uma rede pré-existente na condição de *Router*.

3.1.4. Módulo Robótico de Alta Performance

Os principais objetivos do Módulo de Alta Performance (MAP) são o de agregar poder de processamento e de comunicação de banda larga ao robô, assim como propiciar-lhe autonomia. Pode-se pensar neste módulo como um controlador embarcado, substituindo ou colaborando com o controlador externo no controle do robô.

Este é um módulo de extensão do robô e, portanto, o robô não dependerá dele para realizar suas tarefas básicas. Por ser um módulo opcional também se espera que o programador tenha a liberdade de desenvolver sua própria versão do módulo. Para facilitar o desenvolvimento por terceiros, tanto, o protocolo de comunicação, com o módulo de comunicação de controle, quanto, com o controlador, são robustos, abertos e bem documentados. Da mesma forma as bibliotecas de comunicação são disponibilizadas.

Uma característica importante das aplicações robóticas a serem executadas neste módulo é a independência em relação ao *hardware*. Para isso, as aplicações são desenvolvidas com o suporte de um sistema operacional e interagem com os outros módulos valendo-se de interfaces e conectores conhecidos no mercado e suportados pelo sistema escolhido, como por exemplo, USB, Ethernet, UART, *WIFI* e Bluetooth. Assim, espera-se que a portabilidade das aplicações desenvolvidas seja garantida para futuros *hardware*s permitindo que o núcleo robótico possa acompanhar o desenvolvimento tecnológico na área de processamento embarcado, prolongando sua vida competitiva no mercado. Como exemplo, pode-se pensar inclusive na utilização de celulares ou PDAs como substitutos deste módulo.

Outro ponto importante do isolamento entre o *hardware* e as aplicações deste módulo é a possibilidade de desenvolver, depurar e testar as aplicações em um desktop (PC) antes de compilá-las e carregá-las para o MAP. Esta característica deverá ser alcançada com o desenvolvimento de uma interface de controle robótico comum ao controlador e ao MAP.

A seguir é exposta a organização computacional do MAP, onde são definidos os módulos computacionais que o constituem; os requisitos de *hardware* necessários à implementação dos módulos computacionais; e, por último, as escolhas tecnológicas para a implantação neste projeto tanto para o *hardware* como para o *software*.

O Módulo Robótico de Alta Performance (MAP) do Robô Universal é um *software* desenhado para rodar em uma placa microprocessada qualquer, mas que seja capaz de rodar o sistema operacional Linux e que possua uma interface serial UART e duas USBs. Esta característica de generalidade da placa de alta performance permite que o robô seja atualizado conforme a tecnologia de circuitos integrados se desenvolva.

O objetivo de se adicionar a placa de alta performance ao robô é o de possibilitar que o robô seja autônomo e também o de adicionar comunicação banda larga.

O Módulo Robótico de Alta Performance permite que sejam desenvolvidos aplicativos robóticos capazes de atuar como controladores do robô, enviando comandos ao MCS. Os aplicativos robóticos também podem ser utilizados para expandir os comandos robóticos, ao implementar comandos que podem ser acessados pelos controladores do robô.

Um exemplo da utilização dos aplicativos robóticos é o da visão robótica. Depois de conectada uma câmera ao robô, um aplicativo robótico de captura de imagens pode ser criado e carregado para o robô de forma a permitir que um controlador leia as imagens enviadas pelo robô. A Seção 3.1.5.6 especifica tal aplicativo robótico.

O MAP executará como um *daemon* rodando na placa de alta performance e que centralizará as comunicações entre os controladores, o núcleo robótico (Módulo de Controle de Sessão - MCS) e os aplicativos robóticos. A Figura 6 ilustra estas relações.

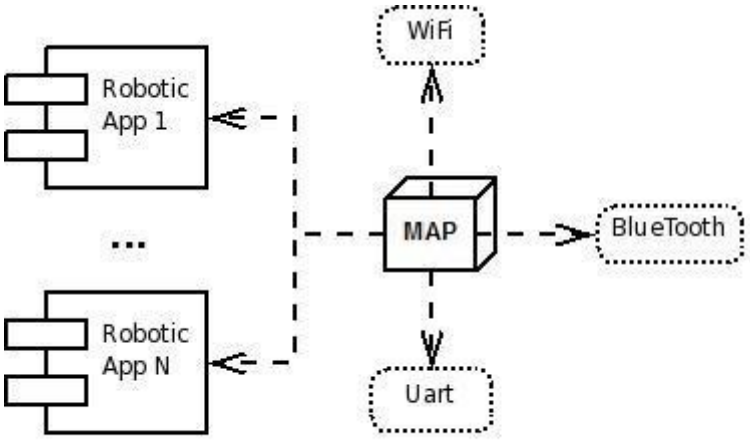

Figura 6: Módulo Robótico de Alta Performance contextualizado na placa de Alta Performance [01]

3.1.4.1. Estrutura do Módulo de Alta Performance

O MAP é formado por um núcleo e por componentes dinamicamente ligados ao núcleo, chamados de "aplicativos robóticos". Nesta seção são mostradas as principais estruturas que compõem o núcleo do MAP e também as principais estruturas que compõem os aplicativos robóticos.

3.1.4.2. Núcleo do MAP

O diagrama da Figura 7 explicita as principais classes que serão utilizadas na composição do MAP. Os métodos e atributos das classes ainda não são mostrados, pois serão especificados durante o refinamento do modelo.

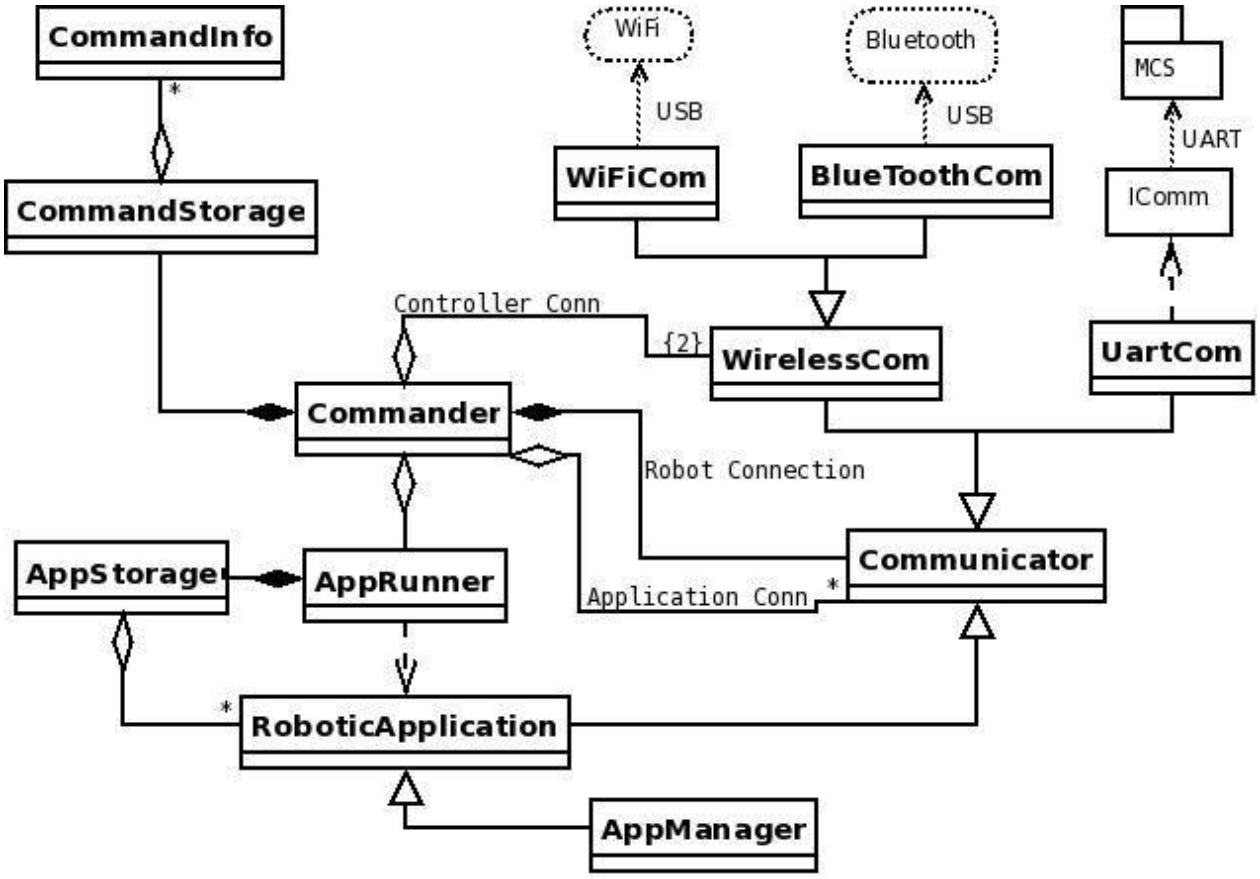

Figura 7: Diagrama de classes do Módulo de Alta Performance [01].

Commander: Esta classe implementa a coordenação geral do MAP. Ela é responsável por ler os pacotes que chegam pelas interfaces de comunicações, analisá-los e tomar as devidas providências. Resumidamente, esta classe é responsável por executar os comandos estendido do robô.

Communicator: Classe base para todas as classes de comunicação. Basicamente dispõe métodos para alertar quando uma mensagem chega, para ler e enviar mensagens.

WirelessCom: Especializa a classe Communicator para redes wireless. Basicamente, adiciona métodos relativos à configuração da rede, como conhecer os nós que estão acessíveis, saber seus endereços e retornar informações sobre eles.

BluetoothCom: Especializa a classe Wireless para redes Bluetooth. Lida com as características específicas para se comunicar com controladores utilizando o protocolo Bluetooth.

WiFiCom: Especializa a classe Wireless para redes WiFi. Lida com as características específicas para se comunicar com controladores utilizando o protocolo WiFi.

UartCom: Especializa a classe Communicator para se comunicar via interface serial UART. Esta classe é utilizada para a comunicação com o MCS do Robô Universal. A implementação da comunicação em si é feita com ajuda da biblioteca de comunicação interna IComm que já define o formato dos pacotes de dados e *acknowledge*.

CommandStorage: Armazena os estados dos comandos estendido sendo executados e, caso necessário, é responsável pelos *logs* das execuções dos comandos.

CommandInfo: Armazena as informações e os estados de um comando estendido que esteja sendo executado. Estas informações são utilizadas para conhecer o controlador responsável pela execução do comando, ou quantas respostas faltam para finalizar a execução, ou ainda qual o aplicativo que está executando o comando.

AppRunner: Classe responsável por iniciar, manter e finalizar os aplicativos robóticos. Os comandos estendidos são executados pelos aplicativos robóticos. Para tanto é necessário que o aplicativo esteja rodando. Esta classe garante que, se o aplicativo não estiver sendo executado, ele será iniciado. Também disponibiliza uma interface para que o sistema possa se comunicar com os aplicativos, lendo suas informações, enviando comandos para que possam ser executados e retornados suas respectivas respostas.

AppStorage: Armazena as informações sobre todos os aplicativos robóticos do sistema. Incluindo os aplicativos que estão sendo executados e os que não estão.

RoboticApplication: Classe base para todos os aplicativos robóticos. Implementa o mecanismo de carregamento dinâmico dos aplicativos e o mecanismo básico de comunicação com o controlador. Esta classe fornece os métodos que devem ser utilizados pelos aplicativos para enviarem comandos ao MCS do robô. Também, esta classe avisa o aplicativo da chegada de comandos que o aplicativo deverá implementar, fornecendo os parâmetros e permitindo ao aplicativo retornar os dados das respostas aos comandos sem se preocupar com o destinatário destes dados. Outra tarefa desta classe é definir a interface a ser implementada pelos aplicativos que permitirá ao sistema ler as informações sobre cada aplicativo específico, tais como a especificação dos comandos que o aplicativo implementa.

AppManager: Esta classe define o programa responsável pela carga dos aplicativos robóticos pelos controladores. A arquitetura do MAP impõe que a única forma de um controlador controlar o robô é enviando comandos estendidos. Estes comandos estendidos são executados, necessariamente, pelos aplicativos robóticos. Assim sendo, o próprio envio de aplicativos robóticos por parte do controlador é implementado por outro aplicativo. Para este trabalho este aplicativo permitirá apenas o recebimento de aplicativos, sendo que futuramente poderão ser implementados outros comandos que permitam o completo gerenciamento dos aplicativos. Como exemplo, os comandos de listar, remover, iniciar e finalizar aplicativos podem ser pensados.

3.1.4.3. Aplicativos robóticos

O diagrama da Figura 8 explicita as principais classes e componentes que servirão de base para a criação de um aplicativo robótico. Estes componentes e classes compõem o que se chamará de *Robotics*DK que é o *software* básico no qual os programadores devem se apoiar para desenvolver os aplicativos robóticos.

Note que os aplicativos robóticos serão criados pelos usuários do robô e podem seguir qualquer arquitetura computacional que seu desenhista queira. Porém, todo aplicativo robótico, para ser integrado ao MAP como tal, deve ter a classe RoboticApplication como classe base.

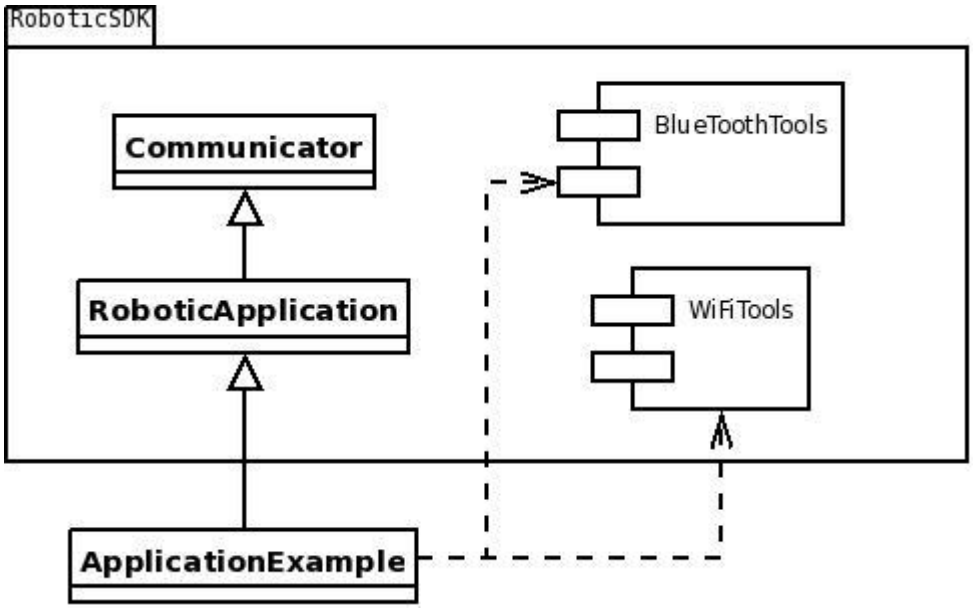

Figura 8: Diagrama estrutural do RoboticsDK, utilizado para o desenvolvimento dos aplicativos robóticos [01].

RoboticApplication: ver seção 3.1.4.2;

Communicator: ver a seção 3.1.4.2;

BluetoothTools: Componente com um conjunto de ferramentas para auxiliar os aplicativos a se comunicar com agentes externos, como os controladores, utilizando a tecnologia Bluetooth, caso esteja disponível.

WiFiTools: Componente com um conjunto de ferramentas para auxiliar os aplicativos a se comunicar com agentes externos, como os controladores, utilizando a tecnologia Bluetooth, caso esteja disponível.

3.1.5. Casos de uso

Para complementar a descrição do MAP, nesta seção, será mostrada os principais casos de uso do MAP. Ao longo da implementação do MAP estes casos de uso serão exercitados de forma a explicitarem a criação e a definição dos métodos e atributos que compõem as estruturas que formam o MAP, sendo utilizados para a criação de diagramas de colaboração.

3.1.5.1. Estabelecendo comunicação com o MAP

Antes de começar a se comunicar com o robô através do MAP, o controlador deve estabelecer um link de comunicação através da interface WiFi ou Bluetooth do MAP. Os passos necessários para isso são descritos a seguir:

1. O controlador escolhe uma sub-rede a ser utilizada para a comunicação entre WiFi e Bluetooth;

2. O controlador envia um broadcast perguntando quem está ouvindo;

3. Os robôs que receberem o pedido respondem enviando uma assinatura que os distingam como robôs juntamente com seus identificadores, endereços de rede e, possivelmente, outras informações relevantes para o estabelecimento da identidade de cada robô;

4. O controlador então escolhe o robô que deseja se comunicar e armazena seu endereço de rede, identificador e outras informações pertinentes.

3.1.5.2. Enviando um comando externo ao robô pelo MAP

Neste procedimento o MAP recebe um comando proveniente de um controlador externo e o repassa ao MCS do robô. Enquanto o comando não for finalizado todas as respostas por ele geradas são passadas de volta ao controlador que originou o comando.

1. O controlador estabelece comunicação com o MAP;

2. O controlador monta um pacote com o comando desejado;

3. O controlador envia o comando ao MAP;

4. Independentemente de a comunicação ser por Bluetooth ou WiFi, o *daemon* do MAP recebe o comando;

5. O comando e suas informações são registrados;

6. O comando é enviado ao MCS pela UART;

7. Quando chegar uma resposta do MCS pela UART ao MAP as informações sobre o comando que a originou são resgatadas e a resposta é transmitida ao controlador que a originou;

8. Caso seja a última resposta do comando, as informações sobre o comando são removidas.

3.1.5.3. Carregando um aplicativo robótico

Os aplicativos robóticos são programas que são executados na placa de alta performance. Portanto, o MAP permite que um controlador lhe envie um aplicativo robótico. Quando um aplicativo robótico é carregado ele pode publicar uma interface declarando os comandos robóticos expandidos implementados por ele.

1. O controlador envia um comando para carregamento de aplicativos, contendo informações sobre o aplicativo, tais como nome, tamanho, etc.;
2. O MAP repassa o comando ao MCS;
3. Caso o MCS negue a execução do comando, uma resposta é envia ao controlador (via MAP) negando o envio e o procedimento é finalizado;
4. O MCS repassa o comando ao MAP;
5. O MAP executa o aplicativo robótico de gerenciamento de aplicativos robóticos que abre um novo canal, no mesmo meio físico (WiFi ou Bluetooth) para o recebimento do aplicativo;
6. O MAP retorna uma resposta ao MCS com as informações sobre como se conectar ao canal aberto para a recepção do aplicativo;
7. O controlador, recebendo a resposta com as informações sobre o canal a ser utilizado para o envio do arquivo, abre o canal e inicia o envio do arquivo;
8. Finalizado o envio do arquivo o canal para envio é fechado;
9. O MAP executa o aplicativo e lê deste os dados necessários à publicação de sua interface.

3.1.5.4. Enviando um comando interno ao robô

Este caso de uso mostra os passos a serem executados quando um aplicativo robótico envia um comando ao MCS do robô.

1. O aplicativo monta um pacote com o comando desejado;
2. O aplicativo envia o comando ao *daemon* do MAP por algum tipo de IPC (provavelmente FIFO) utilizando-se de métodos disponibilizados pela classe RoboticApplication;

3. O comando e suas informações são registrados;
4. O comando é enviado ao MCS pela UART;
5. Quando chegar uma resposta do MCS pela UART ao MAP as informações sobre o comando que a originou são resgatadas e a resposta é transmitida ao controlador que a originou;
6. Caso seja a última resposta do comando, as informações sobre o comando são removidas.

3.1.5.5. Executando um comando expandido

Os aplicativos robóticos podem criar novos comandos do robô que podem ser executados por controladores externos. Estes comandos implementados pelos aplicativos robóticos são chamados de comandos expandidos. Este caso de uso descreve os passos para a execução de um comando expandido.

1. O controlador monta um pacote com o comando desejado;
2. O controlador envia o comando ao robô, seja através do MAP ou do MCC, o comando chega ao MCS;
3. O MCS, verificando que se trata de um comando expandido, envia-o ao MAP;
4. O MAP encontra o aplicativo que executa o comando em questão. Se não houver tal aplicativo, retorna erro ao MCS.
5. Se o aplicativo não estiver sendo executado, executa-o;
6. O comando e suas informações são registrados;
7. O comando é transmitido ao processo do aplicativo;
8. O aplicativo executa o comando envia a(s) resposta(s) ao *daemon* do MAP;
9. A(s) resposta(s) que chegam ao MAP é retornada ao MCS que as encaminhará de volta ao controlador requerente;

3.1.5.6. Aplicativo robótico para visão

Para ilustrar a utilização dos aplicativos robóticos foi criado um aplicativo para acrescentar visão ao robô.

O módulo de “Visão embarcada” é responsável por implementar a captura de imagens para o robô, além de realizar o processamento das imagens captadas de forma a permitir ao robô seguir algum tipo de objeto.

Este módulo é formado por uma componente de *hardware* e outra de *software*. O *hardware* tem a responsabilidade de capturar a imagem, digitalizá-la, possivelmente aplicar alguns filtros mais simples e transmitir as imagens resultantes ao módulo de alta performance. Já o *software* deste módulo tem a responsabilidade de capturar as imagens provenientes do *hardware*, aplicar alguma técnica de reconhecimento posicional (ex: reconhecimento de padrão ou de objeto) e informar o resultado ao aplicativo controlador responsável por transformar esta posição em comandos para o robô. É importante notar que este aplicativo controlador pode estar sendo executado tanto no controlador quanto no próprio módulo de alta performance. A Figura 9 ilustra o fluxo de dados principal deste módulo ressaltando seus componentes e suas respectivas responsabilidades.

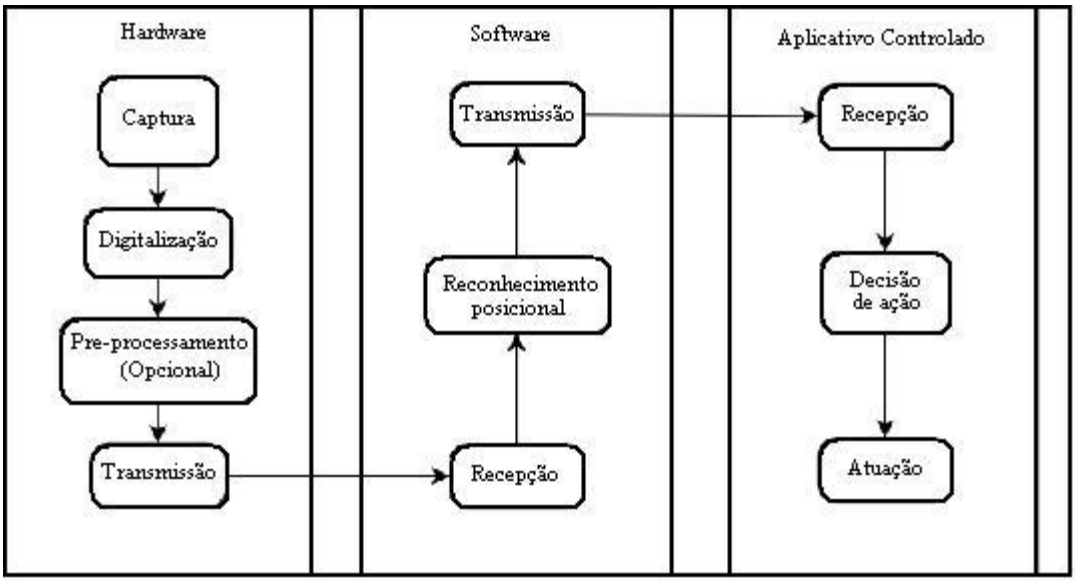

Figura 9: Fluxo de dados do módulo de Visão embarcada [01].

Com base na arquitetura adotada para o robô, tanto as responsabilidades do *hardware* quanto as do *software* podem ser atribuídas a módulos distintos, dependendo do poder de processamento e banda de comunicação exigida pela visão em confronto com as disponíveis nos módulos. Como exemplo, pode-se pensar em restringir o *hardware* a apenas uma câmera digital e implementar todo o processamento como um aplicativo no módulo de alta performance, ou pode-se pensar em desenvolver um *hardware* para o módulo de **Visão embarcada** que possua uma câmera e também um processador que realize todo o processamento.

O aplicativo robótico para a visão foi implementado em sua versão mais simples, fornecendo apenas os comandos para ligar e desligar o envio de imagens para o controlador. Outro ponto a ser notado é que o aplicativo só é capaz de enviar as imagens ao controlador utilizando o protocolo WiFi.

O caso de uso típico para o comando de início de leitura das imagens é o descrito a seguir.

1. O controlador envia ao MAP do robô, por WiFi, o comando para iniciar a captura de imagens;
2. O MAP envia o comando ao MCS do robô;
3. O MCS valida o comando e o controlador. Se o comando for negado, o MCS envia uma resposta negativa ao controlador, via MAP, e o procedimento é finalizado;
4. Com o comando validado, o MCS envia o comando de volta ao MAP;
5. O MAP descobre o aplicativo robótico que implementará o comando, executa-o e passa-lhe o comando;
6. O aplicativo de visão inicia a captura da webcam;
7. O aplicativo de visão cria um *socket* para transmitir as imagens;
8. O aplicativo de visão responde ao comando, para o MAP, passando o número da porta na qual enviará as imagens;
9. O MAP responde ao controlador;
10. O controlador recebe a resposta e cria uma conexão à porta a ser utilizada para o envio das imagens;
11. O aplicativo robótico, após aceitar a conexão do controlador, dá início ao envio das imagens.

3.1.5.7. Escolha Tecnológica para implementação do *software* do MAP

Foi utilizado um kit de desenvolvimento, o STR910-EVAL, com um STR91x para o desenvolvimento do *software* do módulo de alta performance. A placa utilizada foi a ARM9 LPC3180 (mostrado na Figura 10). Este microcontrolador permite que o linux seja instalado em sua forma completa (*ARM Linux*), incluindo o MMU "*Memory Management Unit*", dessa forma é possível portar o *driver* do *WiFi* para o ARM9.

Figura 10: Placa de desenvolvimento para o ARM9 - LPC3180

Para este microcontrolador também é necessário a comunicação através da UART com a placa de controle do robô, então foi utilizado o mesmo protocolo de comunicação descrito na secção do ZigBee e a forma de associar o endereço com o endereçamento dinâmico da rede também é o mesmo adotado para o ZigBee.

A grande vantagem dessa placa de desenvolvimento é que ela já possui os componentes de *WiFi* e Bluetooth integrados. Dessa forma existe a possibilidade de se utilizar tanto o *dongle* como o *hardware* integrado.

3.2. PROJETO DE *SOFTWARE* DESKTOP/PDA (PERSONAL DIGITAL ASSISTANTS)

3.2.1.*Software Development Kit – SDK* [01]

Para o controle da plataforma robótica, decidiu-se por dois meios tecnológicos: o computador e o celular. Para o usuário controlar o Robô-Universal pelo computador, ele terá a sua disposição um *Kit de Desenvolvimento de Software* (*Software Development Kit* – SDK), isto é um pacote de aplicativos que contém as ferramentas e bibliotecas de classes e funções usadas para desenvolver aplicações em linguagem C/C++. O controle usará tecnologia sem fio, através de rede ZigBee.

Como o Robô-Universal disponibiliza, através do módulo alta performance, tecnologia sem fio *Bluetooth* e *WiFi*, o usuário também poderá controlá-lo remotamente por telefone celular ou PDA. Dessa forma, um *Kit de Desenvolvimento de Software* para Java 2 *Micro Edition* (J2ME) [013] - que pode ser usado para desenvolver aplicações para dispositivos móveis.

Um ponto importante é que, independente, de qual SDK se escolha, eles terão basicamente as mesmas funcionalidades. Na Figura 11 têm-se os Casos de Uso preliminares e, em seguida, mais detalhes sobre os SDK´s..

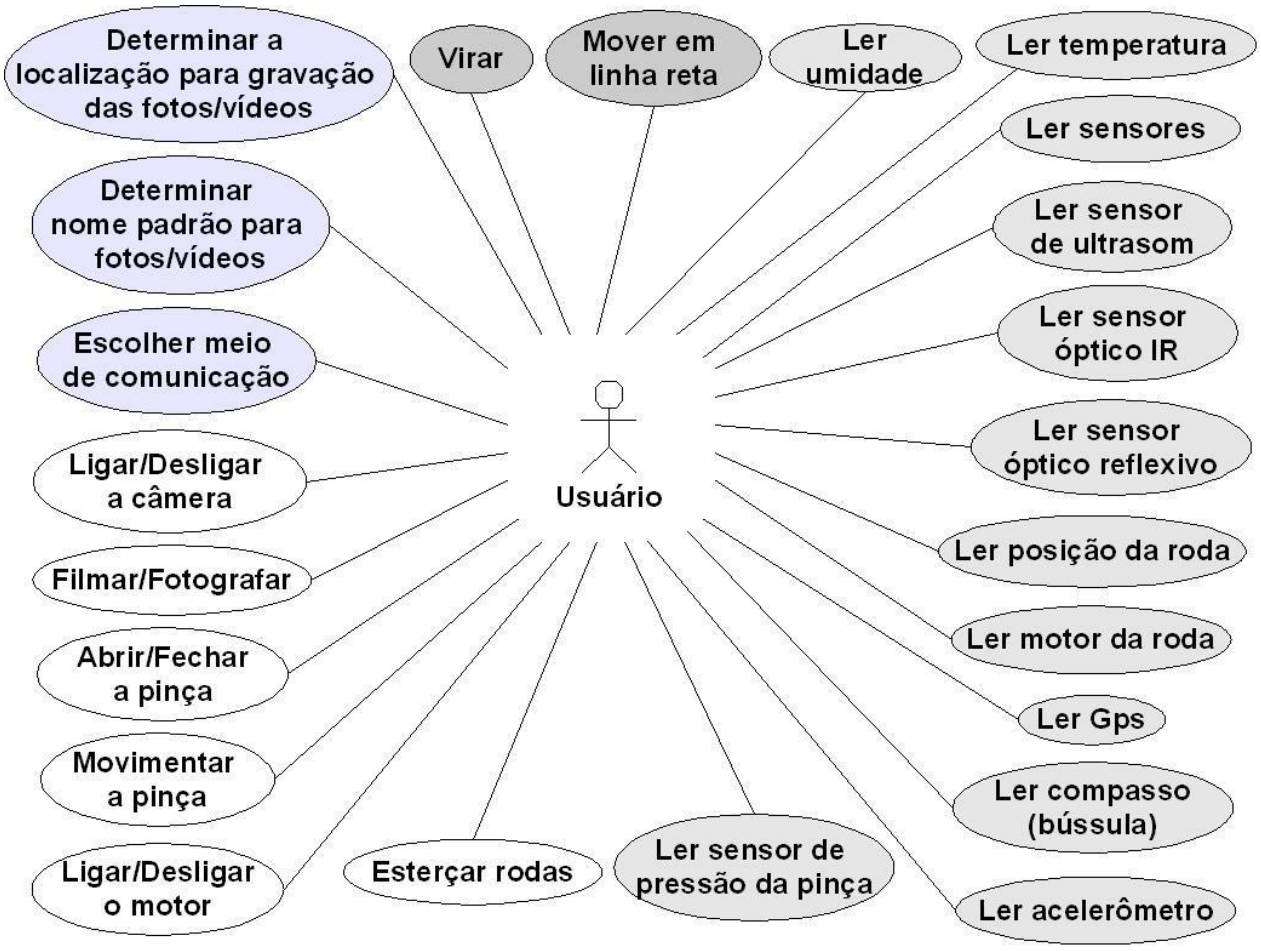

Figura 11: Caso de Uso da Biblioteca do Robô-Universal [01]

3.2.1.1. Robô-Universal SDK

O *software* MRS (Microsoft *Robotics Studio*) [014] - , desenvolvido pela Microsoft e lançado em 2006, é um ambiente gráfico para desenvolvimento, visualização e simulação de aplicações robóticas. Além disso, ainda conta com o recurso de controle de robôs por meio de interfaces via web graças ao "*run-time*" capaz de executar as aplicações geradas pelo *Robotics Studio*. Tem por objetivo não somente o desenvolvimento de *software*s para a área acadêmica, mas também para áreas comerciais e de entretenimento.

Robotics Studio foi elaborado de tal forma que permite a elaboração de programas para robôs das mais variadas plataformas de *hardware*. Já é compatível com as aplicações, serviços e robôs de empresas como CoreWare, iRobot, KUKA Robot Group, the LEGO Group, entre outras. Além disso, é possível que outros robôs possam ser controlados via *Robotics Studio*.

Para tanto, é necessário implementar os blocos de controle do robô na linguagem C# ou C/C++ e colocar o código gerado em diretório específico do Microsoft *Robotics Studio* para que os blocos sejam disponibilizados para utilização.

A construção de um programa em blocos no MRS é realizada no *Visual Programming Language* (VPL) [015] - , que possui todo um ambiente onde o usuário, por meio de blocos que representam atividades básicas e serviços, pode construir programas para seu robô.

Ao se implementar os blocos de controle do robô, que poderia ser feito em linguagem C/C++, verificou-se que haveria a possibilidade de disponibilizar para o usuário um Kit de Desenvolvimento de *Software* (SDK) contendo todos os mecanismos necessários para ele controlar o robô independe do *Robotics Studio*.

Dessa forma, ou Robô-Universal SDK, tem dois objetivos: primeiramente, deve dar suporte aos serviços para o controle da plataforma robótica através do *Microsoft Robotics Developer Studio*; depois, oferecer uma biblioteca independente do *Robotics Studio*, com os mecanismos necessários para controle do robô.

No Apêndice D, encontra-se os requisitos para elaboração do Robô-Universal SDK, as características consideradas na escolha da forma de comunicação, o modelo de comunicação, que exibe as estruturas e os processos envolvidos na comunicação com o robô e as classes do Robô-Universal SDK, além dos diagramas de sequência, usados para explicar os processos envolvidos na transmissão de um comando para o robô.

3.2.1.2. *Robô-Universal SDK Mobile*

Objetiva-se o desenvolvimento de um SDK para criação de *software*s que controlarão o robô remotamente utilizando dispositivos móveis. O projeto vislumbra a idéia de um SDK capaz de

fornecer recursos para arquitetar e gerar diversas aplicações em J2ME de maneira simples e consistente. Múltiplas funcionalidades serão cobertas pelo mesmo sistema, o que deverá ser relevante na idealização de diversos *softwares*.

Através do SDK será possível construir *softwares* que permitirão o controle de recursos físicos e até mesmo ter controle sobre dados e informações que o Robô-Universal venha a oferecer. Os *softwares* gerados utilizando o SDK serão executados em dispositivos móveis, inicialmente celulares, utilizando WiFi (IEEE 803.2b/g/n) e Bluetooth como meio de comunicação entre o Robô-Universal e a aplicação.

O SDK deverá oferecer um conjunto de classes, que permita o desenvolvimento de aplicações/*softwares* que controlem o Robô-Universal remotamente. Os seguintes recursos poderão ser controlados:

1. Motores de tração;
2. Motores direcionais;
3. Sensores;
4. Câmera.

O barramento de comunicação utilizado pelo SDK para se comunicar como Robô-Universal será o barramento wireless através dos recursos das tecnologias WiFi e Bluetooth que estão disponíveis no Robô-Universal.

Na comunicação utilizando WiFi, o robô estará configurado em modo de Infra-estrutura (*Access Point*) para que possa realizar comunicação com os *softwares* presentes nos dispositivos móveis e com outros robôs que estiverem próximos. Com essa tecnologia será possível, ao usuário, assistir ao que o robô vê, já que o robô dispõe de uma câmera.

Na comunicação Bluetooth, o robô estará sempre com o dispositivo ligado em modo *Listen* para receber instruções e não oferecerá o recurso de vídeo devido à taxa de transmissão de dados via Bluetooth ser muito baixa.

3.2.2. Módulo Controlador Externo

O Módulo de Controle Externo do Robô é responsável por suportar aplicativos que controlarão o robô por meio da leitura de seus estados e do envio direto de comandos. Outra função do módulo de controle é propiciar um framework para o desenvolvimento de aplicações robóticas a serem transferidas para o MAP, permitindo ao robô atuar de forma autônoma.

Com o avanço da tecnologia de dispositivos móveis microprocessados optou-se por não restringir o conceito do Módulo de Controle a computadores desktop. Assim sendo, este módulo foi concebido para poder ser implementado em dispositivos móveis como PDAs (*Personal Digital Assistants*), celulares ou *Smart Phones*.

3.2.2.1. Composição do Controlador

Módulo de Controle do Robô pode ser definido por sub-módulos (ou pacotes) de *software* que juntamente com algumas especificações do sistema operacional e do *hardware* nos quais será implementado permitem ao módulo atingir seus objetivos. A Figura 12 ilustra a composição do Módulo de Controle explicitando as dependências entre suas partes.

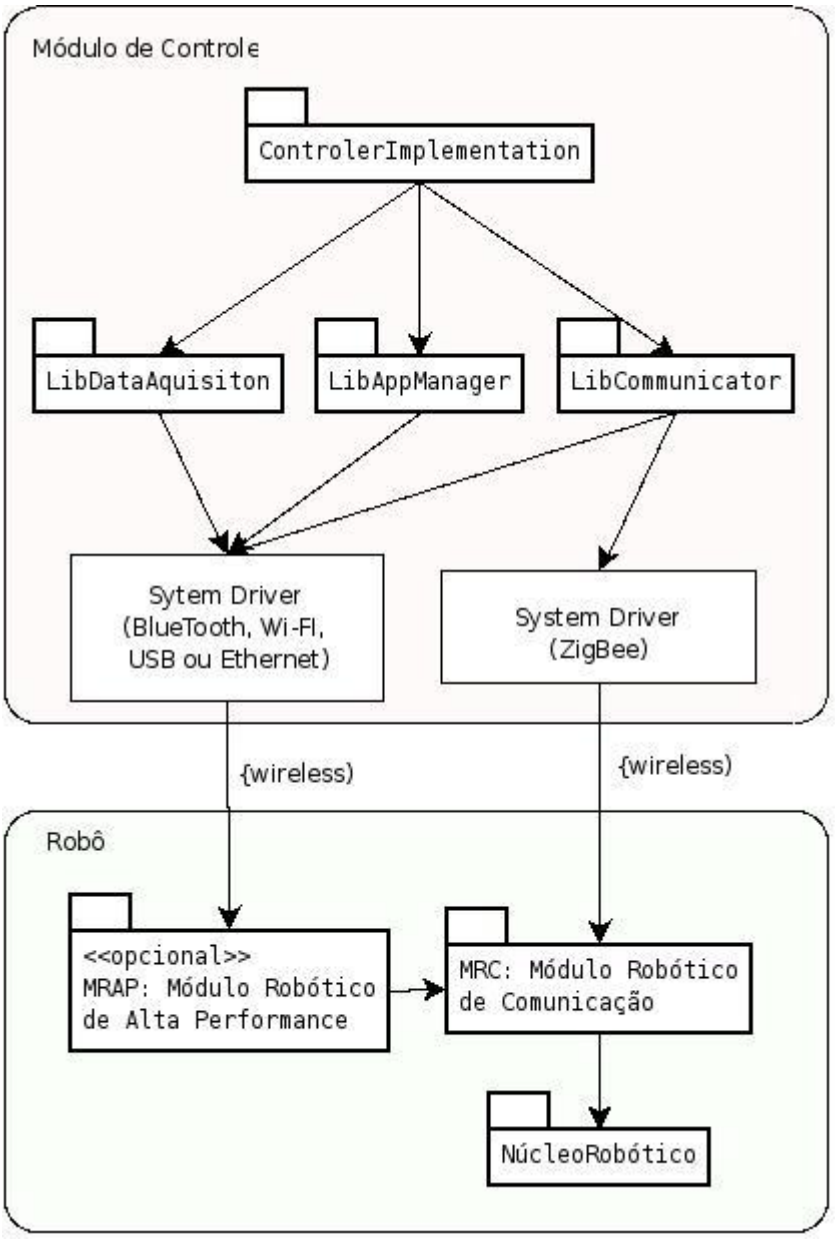

Figura 12: Visão geral dos Módulos de Controle [01]

3.2.2.2. Biblioteca de controle direto (LibCommunicator)

A LibCommunicator é a interface padrão para a interação entre as possíveis implementações dos aplicativos de controle (Controller Implementation) e o Núcleo Robótico. Assim sendo, o principal papel desta biblioteca é homogeneizar a forma de comunicação direta com o robô.

Quando o controlador se comunica diretamente com o robô, após estabelecer uma sessão, a relação entre o controlador e o robô é de mestre-escravo, ou seja, o robô reage de forma cega e imediata aos comandos do controlador.

As seguintes categorias de comandos são implementadas por esta biblioteca:

1. Envio de comandos ao robô (ex: movimentação);
2. Leitura dos estados do robô (ex: sensores);
3. Modificação das configurações do robô (ex: identificação);
4. Estabelecimento de sessões autenticadas;
5. Leitura das informações relativas aos periféricos robóticos (*hardware*s opcionais);

Um aspecto importante e que aumenta a complexidade desta biblioteca é que sua interface deve ser capaz de lidar com a adição de periféricos robóticos não conhecidos à priori. Ou seja, o robô possui um *hardware* básico já conhecido e um *hardware* que poderá estar presente ou não a cada reiniciação do sistema robótico. Assim sendo, as informações sobre os periféricos conectados ao robô devem poder ser lidas e também todos os outros procedimentos dispostos pela biblioteca devem ser capazes de referir-se aos periféricos opcionais que estejam conectados.

Outro aspecto importante é que como o Módulo Robótico de Comunicação (MRC) utilizará a tecnologia ZigBee, qualquer controlador que entre na rede poderá enviar comandos ao robô. Os comandos também podem ser enviados ao robô pelo MAP utilizando o canal de *broad-band* (Bluetooth, *WiFi*, etc), isto para que a comunicação entre o controlador não esteja restrita à utilização da tecnologia ZigBee, como ilustra a Figura 13. Para que o robô saiba a quem obedecer faz-se necessária a implementação do conceito de sessão. Assim, para que o robô se torne escravo de um controlador, antes é preciso que seja estabelecida uma sessão entre o controlador e o MRC do robô. O estabelecimento desta sessão deverá ser realizado de forma autenticada.

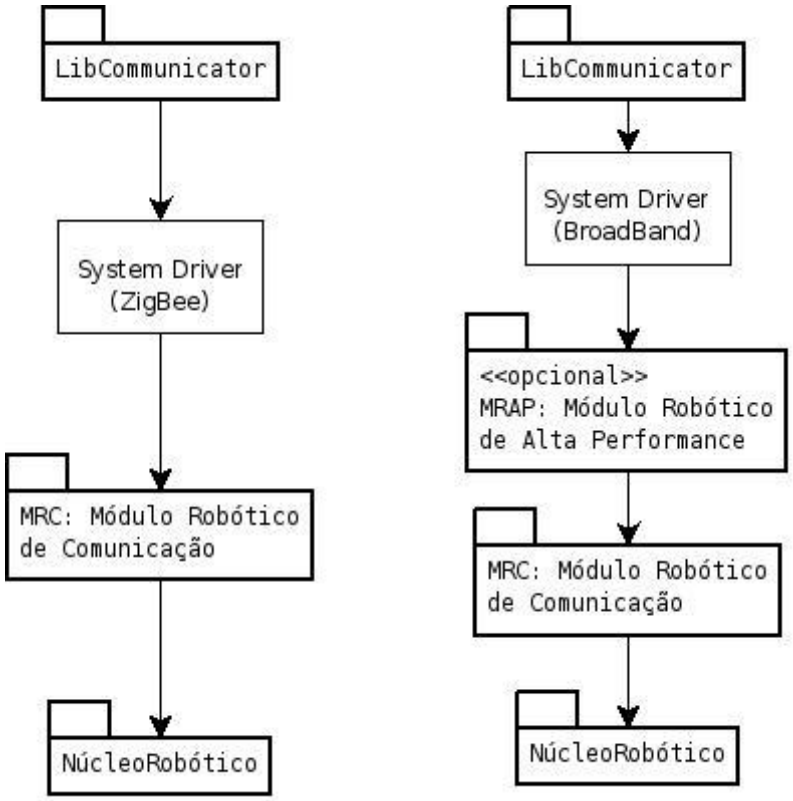

Figura 13: Possíveis formas de comunicação entre biblioteca LibCommunicator e o Núcleo Robótico [01]

Por ser uma biblioteca de comunicação, a LibCommunicator é apenas a interface programática para o acesso ao protocolo de comunicação. Este protocolo define o formato das requisições, respostas e do empacotamento dos dados a serem transmitidos e recebidos. Assim sendo, os agentes receptores (MAP e MRC) devem também conhecer o protocolo utilizado e prover também uma interface programática (biblioteca) para que os aplicativos possam interagir de forma coerente.

Sendo o objetivo estabelecer uma comunicação de controle com o robô e não de transferência de dados. Assim, não serão disponibilizados meios de troca de informações que exijam banda larga, como é o caso do recebimento de imagens. Este tipo de conexão é função da biblioteca LibDataAquision.

Por último, esta biblioteca deve prover meios de se comunicar com o Núcleo Robótico tanto por meio do MRC, caso haja uma interface ZigBee disponível no *hardware*, como por meio do

MAP, caso este esteja disponível no robô e o *hardware* do controlador disponha de alguma das interfaces também disponíveis no MAP. As complicações decorrentes da detecção destas situações e do estabelecimento do canal físico a ser utilizado para a comunicação são também de responsabilidade desta biblioteca.

3.2.2.3. Biblioteca para gerência de aplicativos (LibAppManager)

A gerência dos aplicativos que podem ser executados no MAP é feita pelo controlador visto que o MAP, quando existente, é um módulo embarcado integrado ao robô e não disponibiliza interface homem-máquina para o desenvolvimento de aplicativos. Esta biblioteca foi concebida para homogeneizar o acesso à interface computacional (protocolo de comunicação) que permite o desenvolvimento de um aplicativo robótico em um equipamento externo ao robô e que futuramente será transferido e executado no MAP. A Figura 14 ilustra o fluxo da comunicação entre a biblioteca e o MAP.

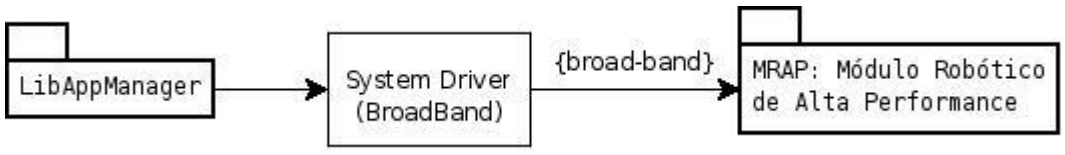

Figura 14: Comunicação entre a biblioteca LibAppManager e o MAP [01]

As seguintes categorias de comandos são implementadas por esta biblioteca:

Leitura das capacidades do sistema operacional (ex: se suporta linguagens interpretadas e quais, qual a família do microprocessador). Este item é importante para saber se o aplicativo de controle tem condições de gerar um aplicativo robótico capaz de ser executado na implementação atual do MAP;

Leitura das informações relativas aos aplicativos atualmente instalados no MAP e seus estados;

1. Transferência de aplicativos robóticos para o MAP;
2. Escalonamento da execução dos aplicativos no MAP;

3. Remoção de aplicativos do MAP;

3.2.2.4. Biblioteca para aquisição de dados (LibDataAquisition)

Sendo a biblioteca LibCommunicator voltada à comunicação de controle pode ser implementada via ZigBee que é uma tecnologia de comunicação de banda-estreita, e a biblioteca LibAppManager destinada à gerência dos aplicativos robóticos a serem executados no MAP, para completar os quesitos deste projeto robótico fez-se necessária a criação de uma biblioteca voltada à transferência de dados que exijam banda-larga.

Esta biblioteca destina-se à aquisição de dados que necessitem de uma conexão banda-larga como é o caso da transferência de imagens digitais. Imagine uma câmera embarcada captando imagens e transferindo-as ao controlador. A configuração do MAP para realizar este tipo de serviço e o mecanismo de transferência serão implementados por esta biblioteca.

3.2.2.5. Aplicação de controle *(Controller Implementation)*

O sub-módulo (pacote) Controller Implementation representa as possíveis implementações dos aplicativos que podem ser desenvolvidos para coordenar as ações do Módulo de Controle, incluindo as que provêm interface com o usuário. Por ser um aplicativo que pode interagir com o usuário sua implementação é extremamente dependente dos recursos disponibilizados pelo equipamento no qual será implementado.

Neste trabalho foram implementados dois aplicativos: uma versão para desktop (PC) e outra versão para um equipamento móvel.

Enquanto a versão desktop representará um controlador completo permitindo inclusive o desenvolvimento de aplicativo e transmissão de dados utilizando *broad-band*, a versão "móvel" do controlador, para este trabalho, utiliza apenas a LibCommunicator, demonstrando a capacidade do Robô-Universal em ser controlado por um equipamento móvel.

3.2.2.6. Protocolos de comunicação

Para garantir a interoperabilidade entre o controlador e o robô é necessário que o *hardware* utilizado para implementar o controlador possua uma interface ZigBee ou então que o Robô-Universal possua o módulo opcional MAP com ao menos uma interface de comunicação (Bluetooth, *WiFi*, USB ou Ethernet) e que o *hardware* do controlador também possua esta interface.

À parte da compatibilidade de conexão física e eletrônica também é necessário que os dois módulos compartilhem do mesmo protocolo de aplicação. Para a comunicação plena foram desenvolvidos três protocolos, cada um associado a uma das bibliotecas do módulo de controle. Ou seja, foi especificado um protocolo para comandos a ser implementado pela LibCommunicator e que também foi implementado no MAP (Módulo Robótico de Alta Performance) e no MRC (Módulo Robótico de Comunicação); outro protocolo para gerência de aplicativos robótico implementado pela LibAppManager e que também foi implementado no MAP; e um terceiro protocolo para comunicação de dados por banda-larga implementado pela LibDataAquisition e pelo MAP.

CAPÍTULO 4

PROJETO ELÉTRICO

4.1. SISTEMA DE ALIMENTAÇÃO [01]

O grande desafio colocado para a área de robótica móvel foi desenvolver um sistema de geração de energia com grande autonomia e naturalmente com custos compatíveis.

Diversas possibilidades foram estudadas até encontrar a que parece ser a mais satisfatória, considerando o binômio "Autonomia / Custos" e naturalmente as disponibilidades do mercado.

Soluções que incorporem situações não *standarizadas* ou não consolidadas tecnicamente foram descartadas. Dentro das baterias encontradas no mercado foi escolhida uma linha de aplicação industrial. Na linha de máquinas ferramentas *wireless* são utilizadas baterias que requerem grande potência, durabilidade, tolerância a sobrecargas, rápida recarga, facilidade em instalar e desinstalar da máquina e baixo custo. Todas essas condições são as requeridas para nossa aplicação, portanto foi decidido por a utilização de baterias da linha da Makita de Li-ion de 18V e 3.0A modelo BL1830, mostrada na Figura 15**.**

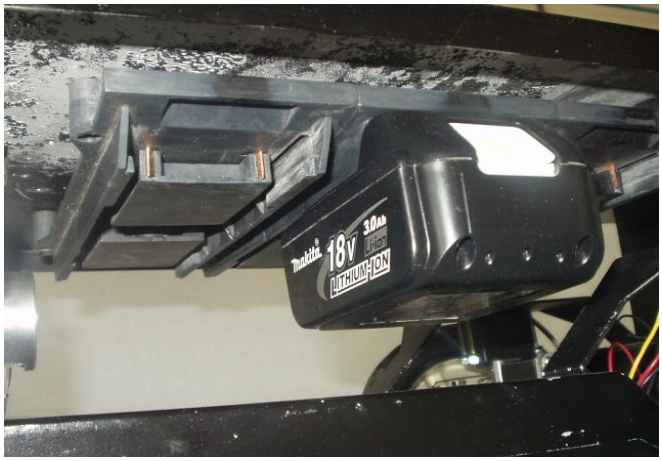

Figura 15: Bateria de Li-ion [01]

4.2. SISTEMA DE CONTROLE

4.2.1. Sistema de Controle Robótico

Para poder testar os circuitos e desenvolver o *software* foi necessário adquirir algumas placas, antes de definir o esquema final e a placa mãe que alojará a placa do microprocessador e os *driver*s e sensores. Na Figura 16 pode-se ver a placa STR910-Eval utilizada para o desenvolvimento do *software* da placa de controle.

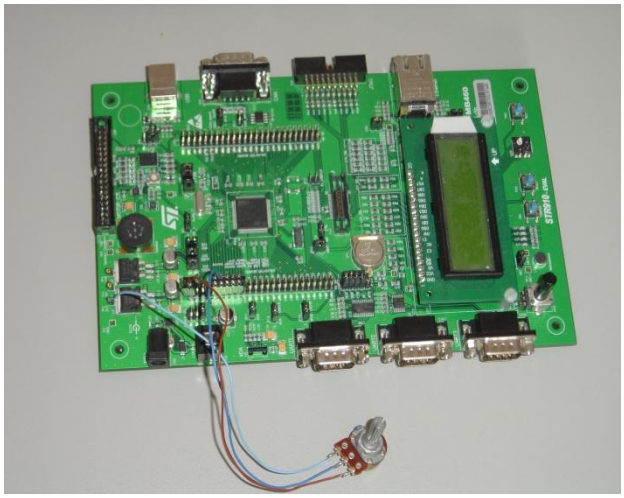

Figura 16: Placa de desenvolvimento STR910 - EVAL

4.2.2. Sistema de comunicação

Inicialmente antes de o *software* ser desenvolvido e necessário que um *hardware* seja determinado como plataforma para o desenvolvimento. O *hardware* de *ZigBee* escolhido foi uma plataforma da Jennic que utiliza o JN5139 como chip do *ZigBee* mostrado na Figura 17 até Figura 19.

Figura 17: Chip do ZigBee

Figura 18: Módulo do ZigBee

Esse *chip* é montado sobre um modulo com os componentes para recepção e transmissão de rádio frequência de 2.4GHz que e a frequência utilizada pelo *ZigBee*. Trabalha com um micro-processador da seguinte especificação:

1. *32-bit RISC processor sustains 16MIPs with low power*
2. *192kB ROM stores system code, including protocol stack*

3. *96kB RAM stores system data and optionally bootloaded program code*
4. *48-byte OTP eFuse, stores MAC ID on-chip, offers AES based code encryption feature*
5. *4-input 12-bit ADC, 2 11-bit DACs,*
   2 comparators
6. *2 Application timer/counters,*
   3 system timers
7. *2 UARTs (one for debug)*
8. *SPI port with 5 selects*
9. *2-wire serial interface*
10. *Up to 21 GPIO*

Figura 19: JN5139 IEEE802.15.4/JenNet Evaluation Kit

4.2.3. Sistema de Alta Performance

Para o sistema de alta performance, a idéia é utilizar uma placa do mercado e que no futuro possa ser substituída por novas placas com mais *features*. Para este trabalho escolhi a placa ARM9 LPC3180 mostrada na Figura 20. Este microcontrolador permite que o *linux* seja instalado em sua forma completa (*ARM Linux*), incluindo o MMU "*Memory Management Unit*", dessa forma e possível portar o *driver* do *Wi-fi* para o ARM9.

Figura 20: Placa de desenvolvimento para o ARM9 - LPC3180

Para este microcontrolador também e necessário a comunicação através da UART com a placa de controle do robô, então foi utilizado o mesmo protocolo de comunicação descrito na secção do *ZigBee* e a forma de associar o endereço com o endereçamento dinâmico da rede também e o mesmo adotado para o *ZigBee*.

A grande diferença dessa placa de desenvolvimento em relação a anterior e que ela já possui os componentes de *WiFi* e *Bluetooth* integrados. Dessa forma existe a possibilidade de se utilizar tanto o *dongle* como o *hardware* integrado.

4.2.4. Sistema de Visão

Para o sistema de visão adotei o uso de uma *webcam* com controle PTZ (*Pan-Tilt-Zoom*), a escolhida foi a QuickCam Orbit MP da Logitec mostradas nas Figura 21 e Figura 22. Outra câmera que também poderá ser utilizada e a QuickCam Chat da Logitec como uma opção de baixo custo sem PTZ. Estas Câmeras foram escolhidas por ter *drivers* para sistema operacional Linux disponíveis na internet.

Figura 21: Instalação da Webcam com PTZ

Figura 22: Instalação da Webcam vista lateral

4.2.5. Sistema de sensores

Para que uma plataforma robótica móvel tenha um mínimo de autonomia para se movimentar, sensores são requeridos. O nível de capacidade de independência de um robô pode ser, de certa forma, medida pela quantidade dos sensores disponível. Atualmente existe um vasto número de diferentes sensores sendo usado na robótica. Eles possuem funções diferentes para fins diversos.

Entre todos os tipos de sensores podemos enumerar aqui os mais conhecidos:

1. Sensores de luz: células solares, fotodiodos, fototransistores, tubos foto-elétricos, CCDs, radiômetro de Nichols, sensor de Imagem.
2. Sensores de som: microfones, hidrofone, sensores sísmicos, ultra-som.
3. Sensores de temperatura: termômetros, termopares, resistores sensíveis a temperatura (termístores), termômetros bimetálicos e termostatos.
4. Sensores de calor: bolômetro, calorímetro.
5. Sensores de radiação: contador Geiger, dosímetro
6. Sensores de partículas subatômicas: cintilômetro, câmara de nuvens, câmara de bolhas
7. Sensores de Resistência elétrica: ohmímetro
8. Sensores de corrente elétrica: galvanômetro, amperímetro
9. Sensores de tensão elétrica: electrômetro, voltímetro
10. Sensores de potência elétrica: wattímetro
11. Sensores magnéticos: compasso magnético, compasso de fluxo de porta, magnetômetro, dispositivo de efeito Hall
12. Sensores de pressão: barômetro, barógrafo, *pressure gauge*, indicados da velocidade do ar, variômetro
13. Sensores de fluxo de gás e líquido: sensor de fluxo, anemômetro, medidor de fluxo, gasômetro, sensor de fluxo de massa.
14. Sensores químicos: eletrodo ion-selectivo, eletrodo de vidro para medição de pH, eletrodo redox, sonda lambda

15. Sensores de movimento: arma radar, velocímetro, tacômetro, hodômetro, coordenador de giro, acelerômetros.
16. Sensores de posição: GPS.
17. Sensores de orientação: giroscópio, horizonte artificial, giroscópio de anel de laser
18. Sensores mecânicos: sensor de posição, selsyn, chave, *strain gauge*
19. Sensores de proximidade: Um tipo de sensor de distância porem menos sofisticado, apenas detecta uma proximidade específica. Uma combinação de uma fotocélula e um LED ou laser, sensores ultrasônicos, sensores infravermelhos, sensores mecânicos (bigode).
20. Sensores de distância (sem contato): sensores a laser, radar, sensores ultrasônicos.

O mais importante aqui é definir o conjunto de sensores apropriados para que a plataforma móvel execute todas as tarefas especificadas, isto envolve a técnica de medição correta relacionada com o tamanho, peso, faixa da temperatura de operação e consumo do dispositivo. Um ponto secundário a ser analisado na escolha do sensor adequado é quanto ao modo de leitura da informação pelo microcontrolador, ou seja, como os dados são transmitidos. Se o modo de transferência for analógico torna-se necessário um conversor AD para realizar a leitura e posterior processamento desta informação, consumindo recurso de processamento que poderia ser usado para o controle, por exemplo. O mais usual é que a transferência da informação lida seja feita digitalmente, com a conversão analógica e processamento realizados no próprio dispositivo de sensoriamento, mas dependendo do grau de complexidade da informação, custo do sistema sensor e disponibilidade de recurso de processamento, o projeto de sensoriamento pode ser embutido no circuito principal da plataforma robótica com o transdutor inserido no mesmo e todo o circuito de conversão.

Como já citado anteriormente, no caso de a transmissão da informação sensoreada ser feita digitalmente, ela poderá ser feita de modo contínuo com um período definido, nesse caso a leitura é feita através de interrupções, ou a informação é enviada somente quando requerida pelo microprocessador, nesse último caso, o sistema deverá checar se o dispositivo sensor está pronto para a

transmissão demandando tempo para verificar se o dispositivo está pronto para funcionar e enviar os dados.

Para o presente trabalho foi adotado uma série de sensores de modo que se possa prover a plataforma robótica de o máximo de autonomia possível para executar as tarefas munindo o usuário, que no caso é um programador para fins científicos, com recursos suficientes para desenvolver suas aplicações que necessitar. Os sensores pretendidos aqui são:

1. Um módulo acelerômetro de 3 eixos da marca Hitachi H48C;
2. *Encoders* em duas rodas sinal dos *drivers* dos motores *brushless*.
3. Um conjunto de transdutores de ultra-som: O receptor modelo 4093 e o transmissor modelo 4094;
4. Sensor tátil refletivo da Fairchild modelo QRB1134;
5. Sensor de campo magnético terrestre da Hitachi modelo HM55B;
6. Sensor de proximidade para obstáculos de médio alcance da Sharp modelo GP2D12;
7. Um módulo GPS fornecido pelo Hobby Engineering
8. Sensor de temperatura e umidade (*Sensirion SHT1x).*

A Figura 23 e Figura 24, ilustra as disposições dos sensores listados acima por toda a plataforma móvel robótica. Os detalhes e especificações dos sensores podem ser visto no anexo V.

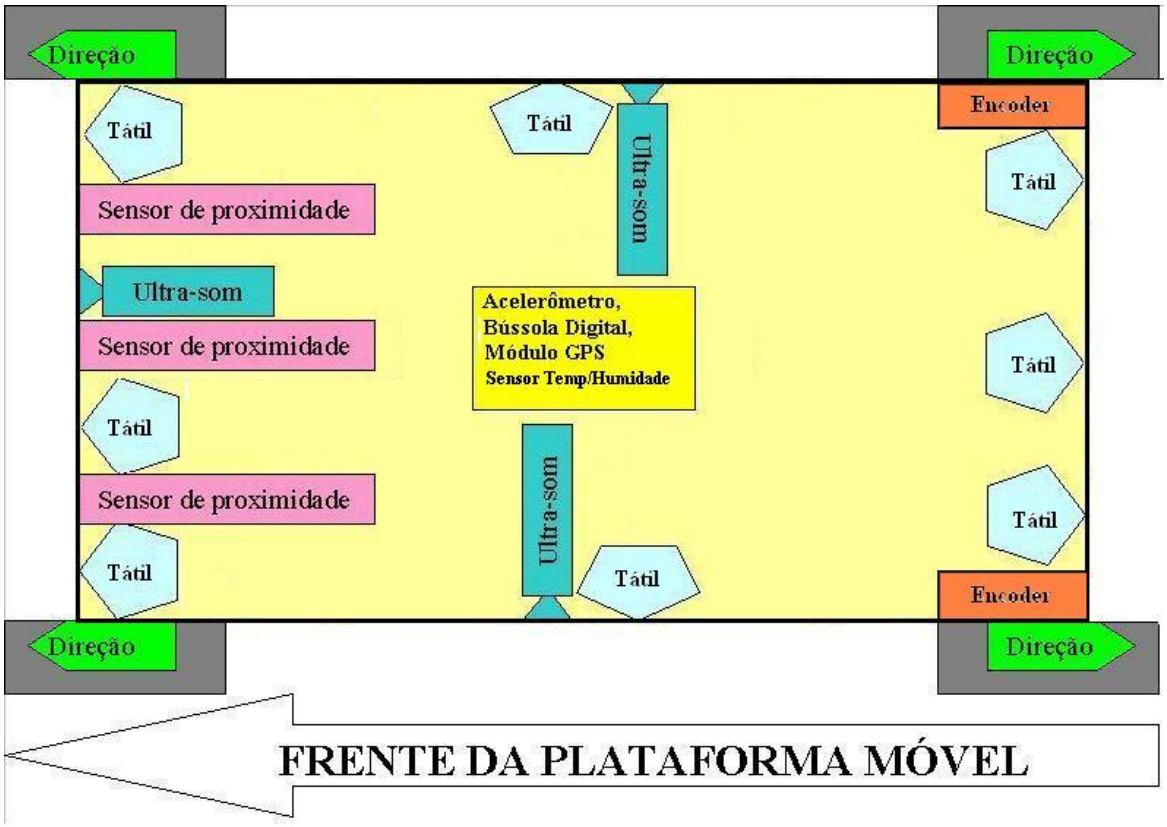

Figura 23: Possível disposição dos sensores na plataforma móvel robótica [01]

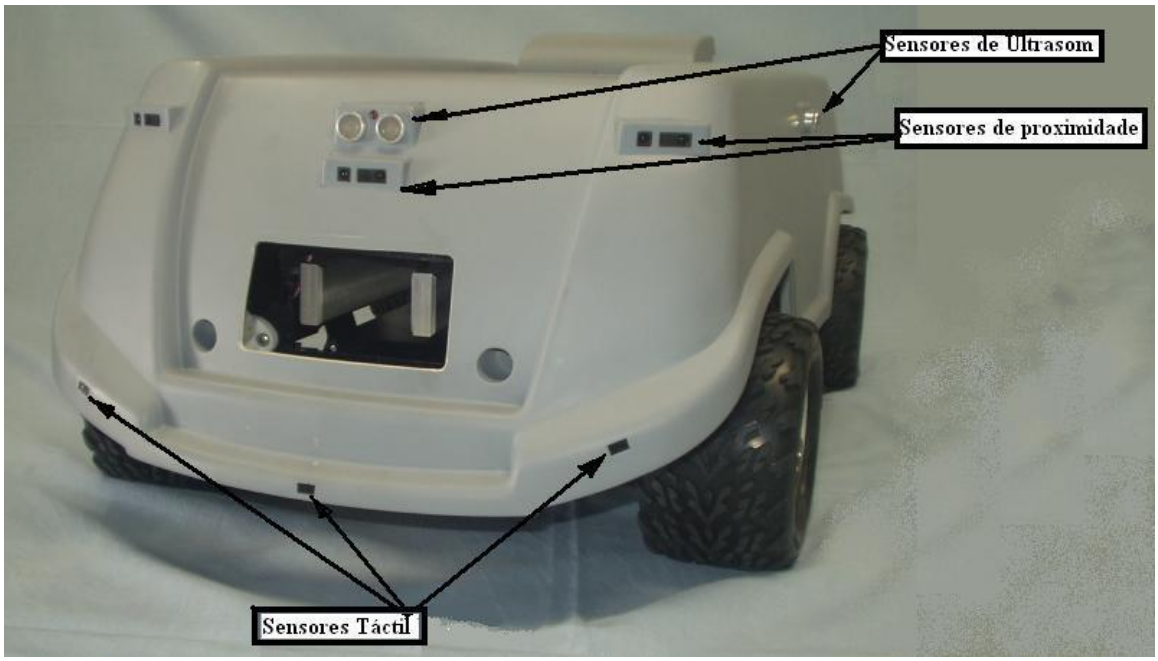

Figura 24: Disposição dos sensores na plataforma móvel robótica

4.2.6. Sistema de Atuadores

Detalhes dos atuadores podem ser visto no anexo V.

4.2.6.1. *Driver*s de Motores

No presente trabalho, para a tração do robô foi adotado dois motores *brushless*. Esses motores demandam *drivers* especiais, pois o acionamento de suas bobinas depende de uma série de fatores para que o motor funcione como a velocidade da rotação e o torque.

Foi escolhido dois *drivers* da Akiyama AKDBL12-30W, com alimentação de 12V e potência 30W para acionamento dos motores. Ver detalhes no Anexo I.

4.2.6.2. Servo-motor

Para direcionar as rodas foram usado quatro servos-motores da fabricante Futaba modelo S3003. Esses servos possuem sistema de controle interno onde as entradas são a alimentação de 5V (positivo e negativo) e o sinal de posicionamento angular do eixo.

CAPÍTULO 5

CIRCUITOS ELÉTRICOS

5.1. ESQUEMÁTICOS DAS PLACAS ELETRÔNICAS DO ROBÔ [01]

5.1.1. Esquemáticos dos circuitos da placa do microcontrolador

A seguir são apresentados os esquemáticos e breves descrições dos circuitos que compõem a placa do microcontrolador do Robô.

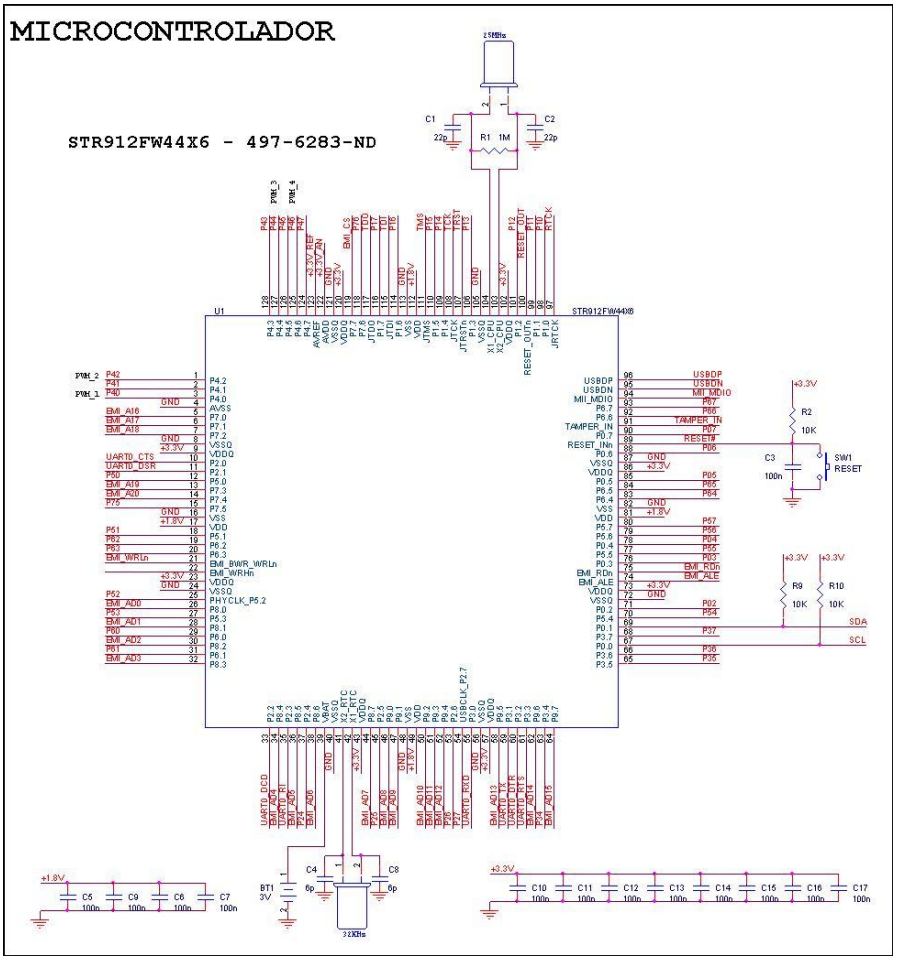

Figura 25: Microcontrolador STR912FW44X6 [01]

A Figura 25 , mostra o esquema do microcontrolador, contendo alimentações necessárias (3,3V e 1,8V), cristais para gerar as frequências necessárias para a CPU (25MHz) e para o relógio (32KHz). A bateria BT1 serve para alimentar o relógio e a SRAM interna em caso de falta de energia. A chave SW1 serve para provocar um *reset* no microcontrolador.

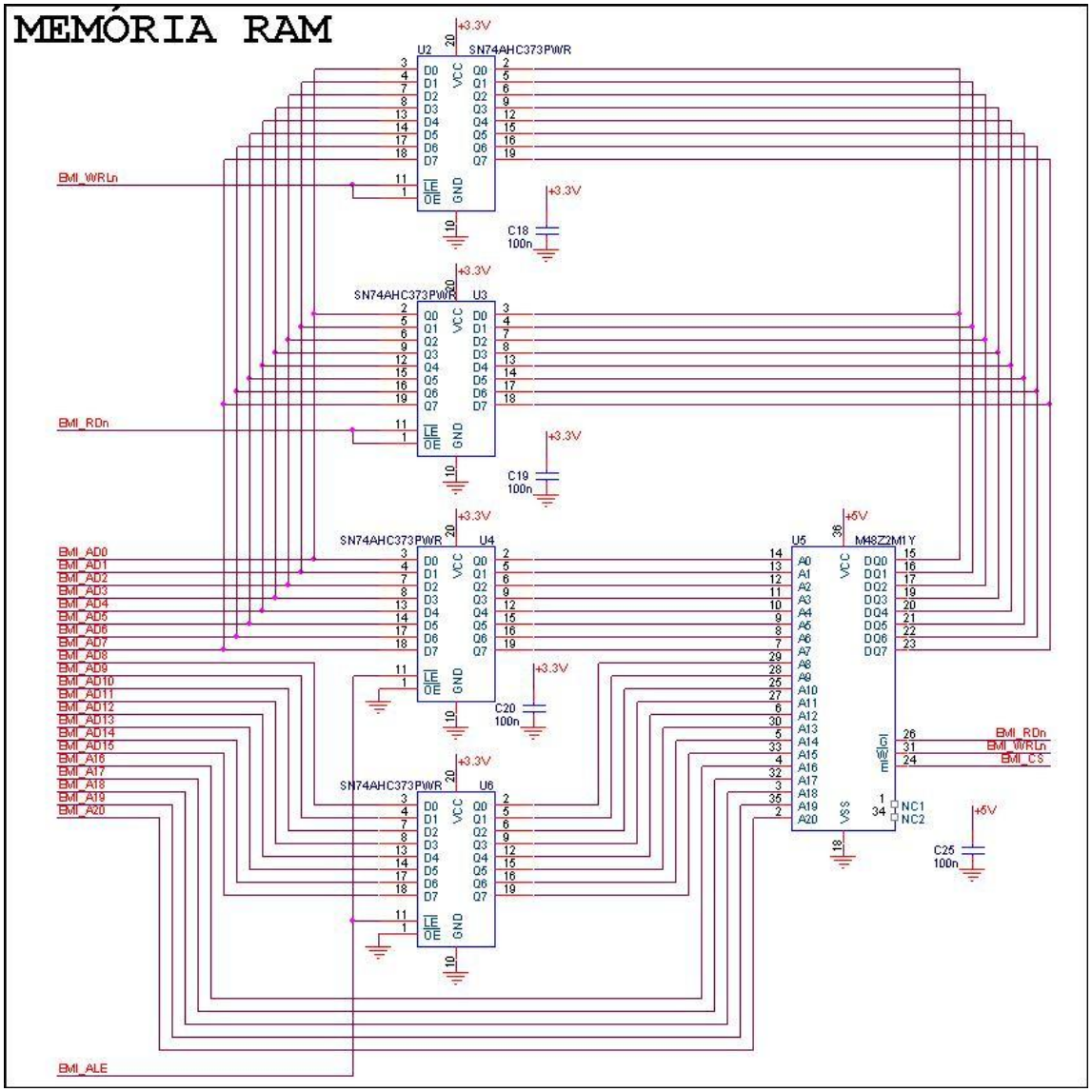

Figura 26: Circuito de interface da memória RAM [01]

A Figura 26 , mostra o circuito da memória RAM (M48Z2M1Y). Ela e conectada ao microcontrolador através da interface EMI (interface de memória externa, do inglês *External memory interface*). O *byte* menos significativo da interface EMI do microcontrolador e utilizado tanto para dados como para a parte mais baixa do endereçamento. A memória RAM e de 16MB, possuindo 8 bancos de 2MB cada. A Figura 27 , mostra o circuito de gravação do microcontrolador. O microcontrolador usa a interface JTAG, que possibilita sua gravação no próprio circuito. Alem disso a interface permite fazer um teste de todos os pinos do microcontrolador e *debug* da CPU via *software*.

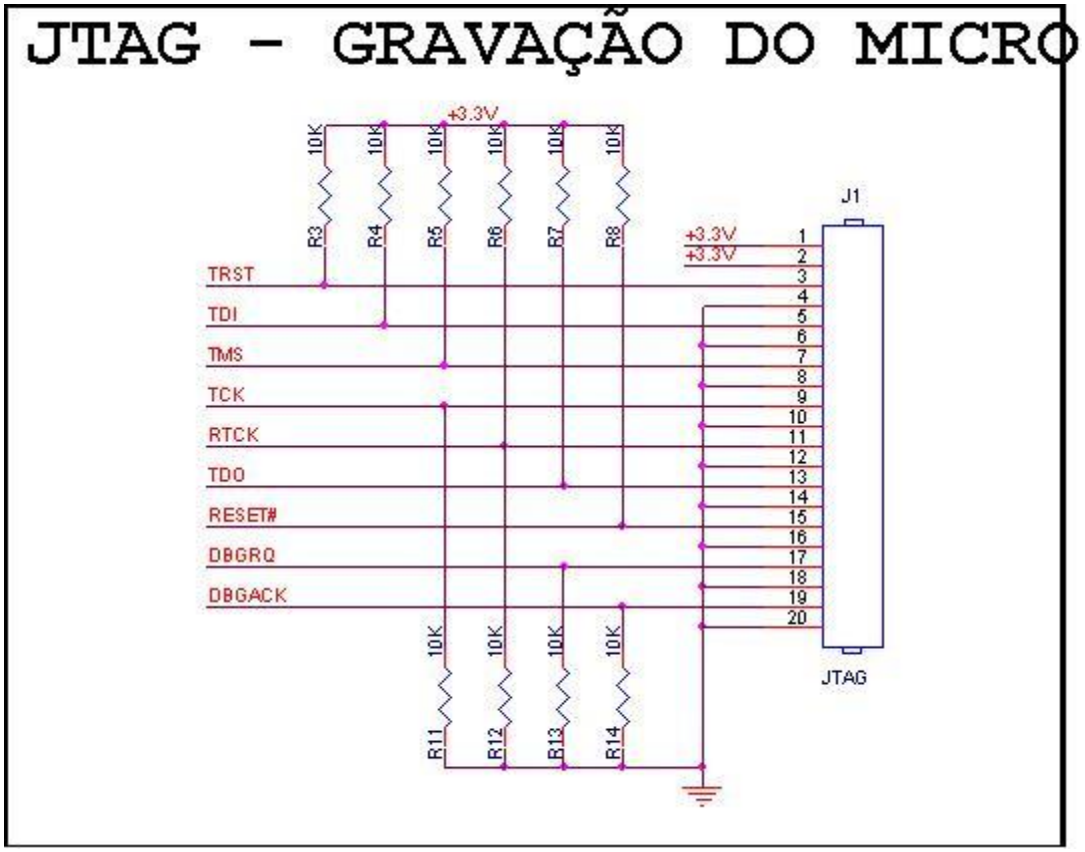

Figura 27: Interface de gravação do microcontrolador [01]

A Figura 28 , mostra o conector de acesso da placa do microcontrolador a placa mãe. A partir do conector, são fornecidos todos os sinais necessários a comunicação com os periféricos da placa mãe, tais como: sensores, *drives* de motores de direção e locomoção, *slots* de expansão, etc. Também são fornecidas as tensões de alimentação necessárias ao funcionamento do microcontrolador.

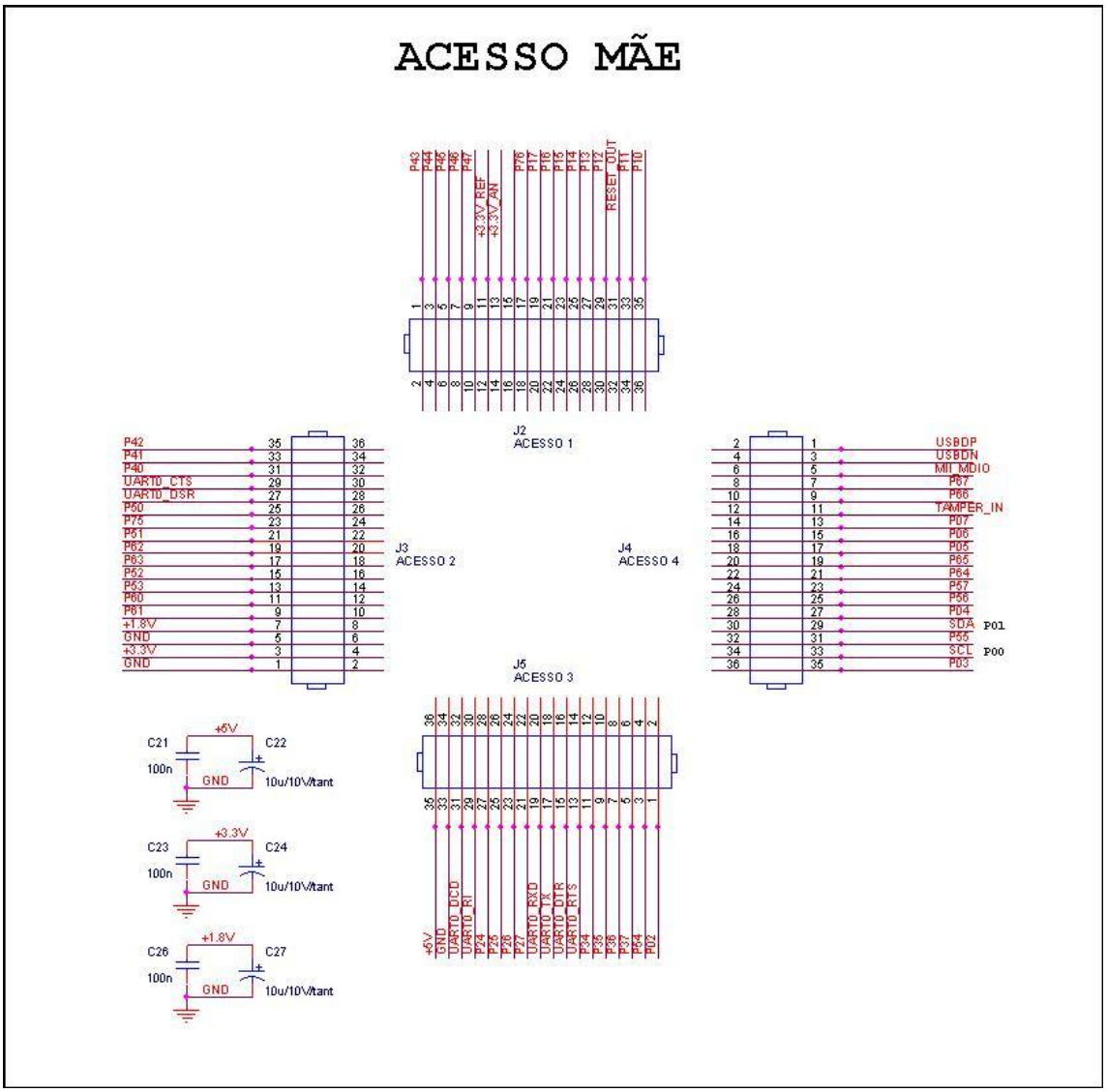

Figura 28: Conector de acesso da placa do microcontrolador à placa mãe [01]

5.1.2. Esquemáticos dos circuitos da placa mãe

São apresentados em seguida os esquemáticos e breves descrições dos circuitos que compõem a placa mãe do Robô A Figura 29 , mostra o conector de acesso da placa mãe a placa do microcontrolador. Este conector ira se encaixar ao conector da Figura 28.

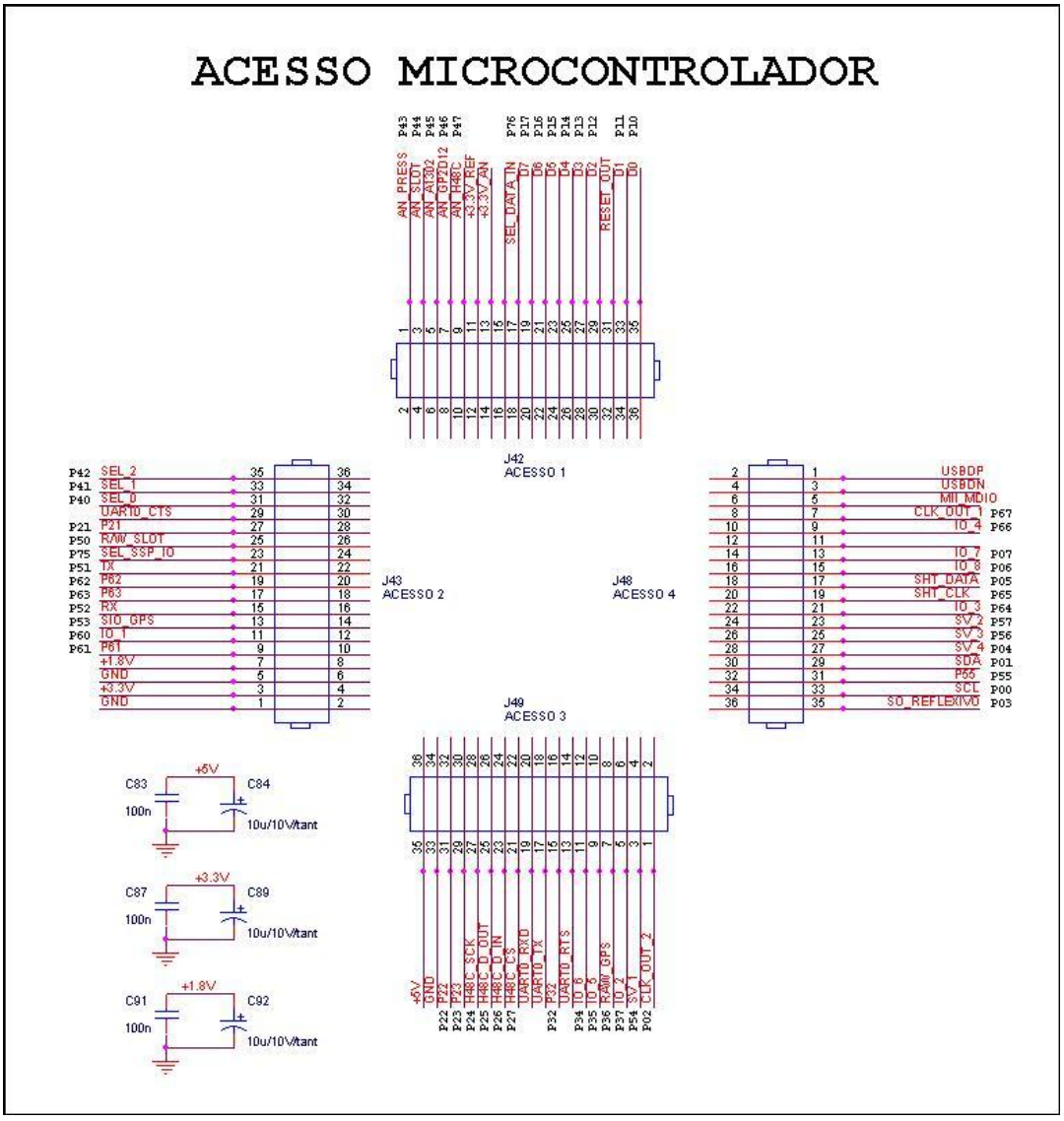

Figura 29: Conector de acesso da placa mãe à placa do microcontrolador [01]

A Figura 30 , mostra o circuito dos *slots* de expansão, necessários para comunicação com periféricos externos. São também utilizados para enviar os sinais de controle para os dois motores de locomoção. A placa mãe possui 8 *slots*, cada um contendo:

- Sinais de TX e RX para comunicação serial
- Duas Entradas e duas saídas digitais
- Duas entradas analógicas
- Uma porta de I/O configurável (dentre seus sinais, PWM's para os motores de locomoção)
- Alimentação de 5V e 3,3V

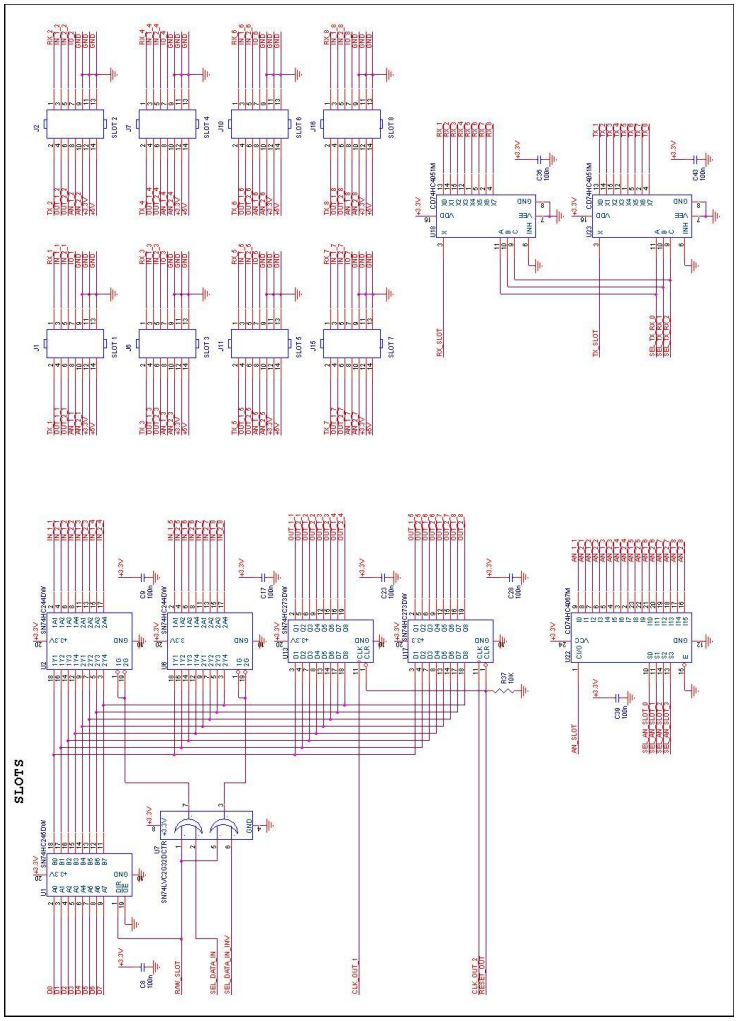

Figura 30:Circuito dos *slots* de expansão [01]

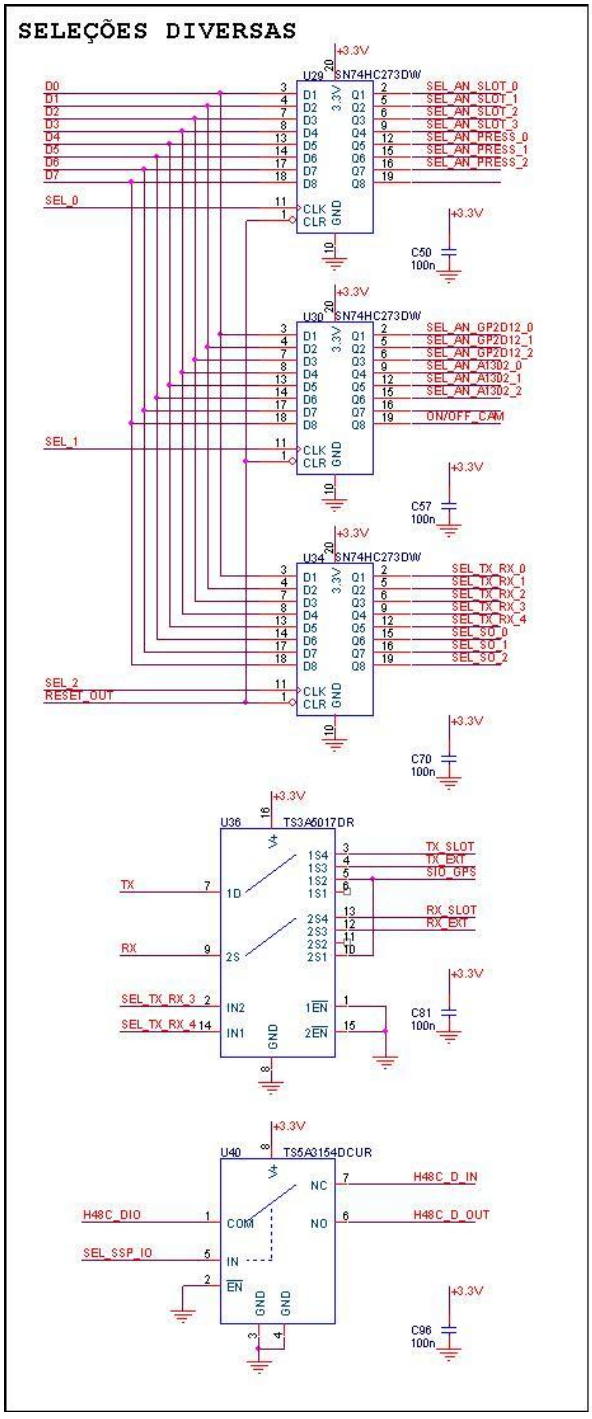

Figura 31: Circuito de seleções diversas [01]

A Figura 31, mostra o circuito de seleções diversas. Este circuito e utilizado para fazer a seleção de entradas analógicas e sinais de comunicação serial dos *slots* de expansão, de sensores de pressão, sensores infravermelho, sensores ótico reflexivo, câmera. Ademais disso, o circuito também chaveia os sinais TX e RX da porta UART2 do microcontrolador entre o *slot* de expansão, placa de comunicação *Zig Bee* e GPS.

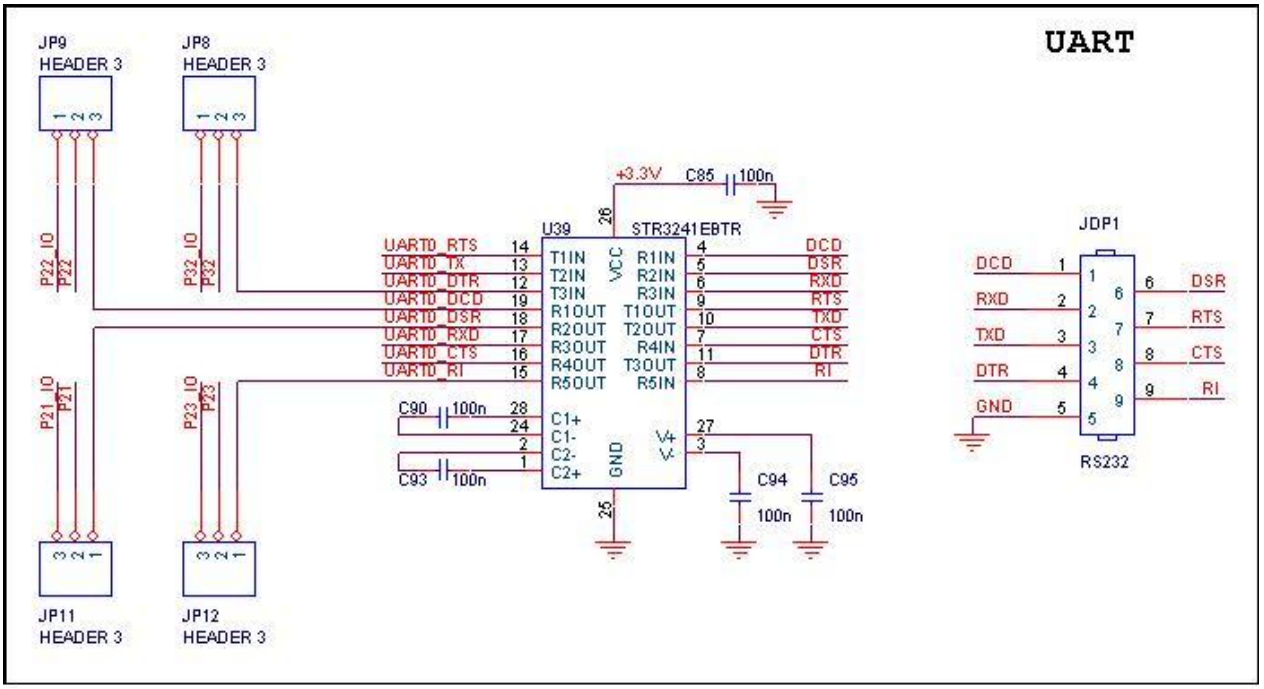

Figura 32: Circuito de interface serial (UART) [01]

A Figura 32, mostra o circuito de interface para comunicação serial via porta UART0 do microcontrolador. O circuito integrado U39 faz a conversão entre os níveis TTL e RS232. A placa de alta performance será conectada a placa mãe via interface RS232.

A Figura 33, mostra os sensores do Robô que são fixados diretamente a placa mãe. Dentre estes sensores, estão: acelerômetro, bússola, sensor de temperatura e umidade e GPS. Alem

disso, a figura mostra o circuito de acionamento e seleção de alimentação (5V ou 3,3V) da câmera, assim como um conector auxiliar que fornece alguns pinos do microcontrolador para aplicações externas.

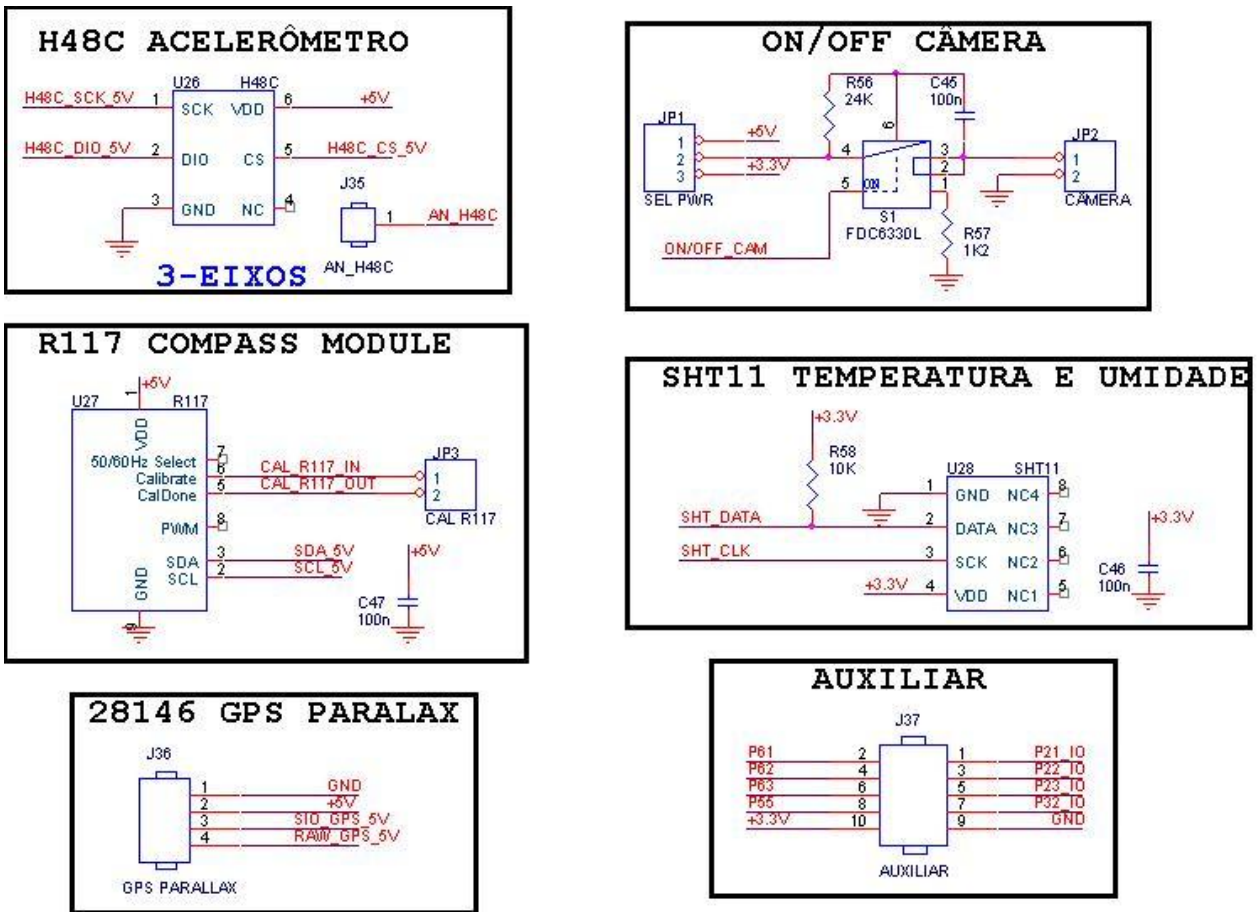

Figura 33: Conexões de sensores fixos na placa mãe [01]

A Figura 34, mostra o circuito de condicionamento dos sensores de distancia por infra-vermelho. O circuito integrado U14 e um multiplexador analógico, e a responsável por selecionar qual sensor a se fazer leitura.

A Figura 35, mostra o circuito de condicionamento dos sensores óticos reflexivos. A seleção de qual sensor se deseja fazer leitura se da através do multiplexador digital U24.

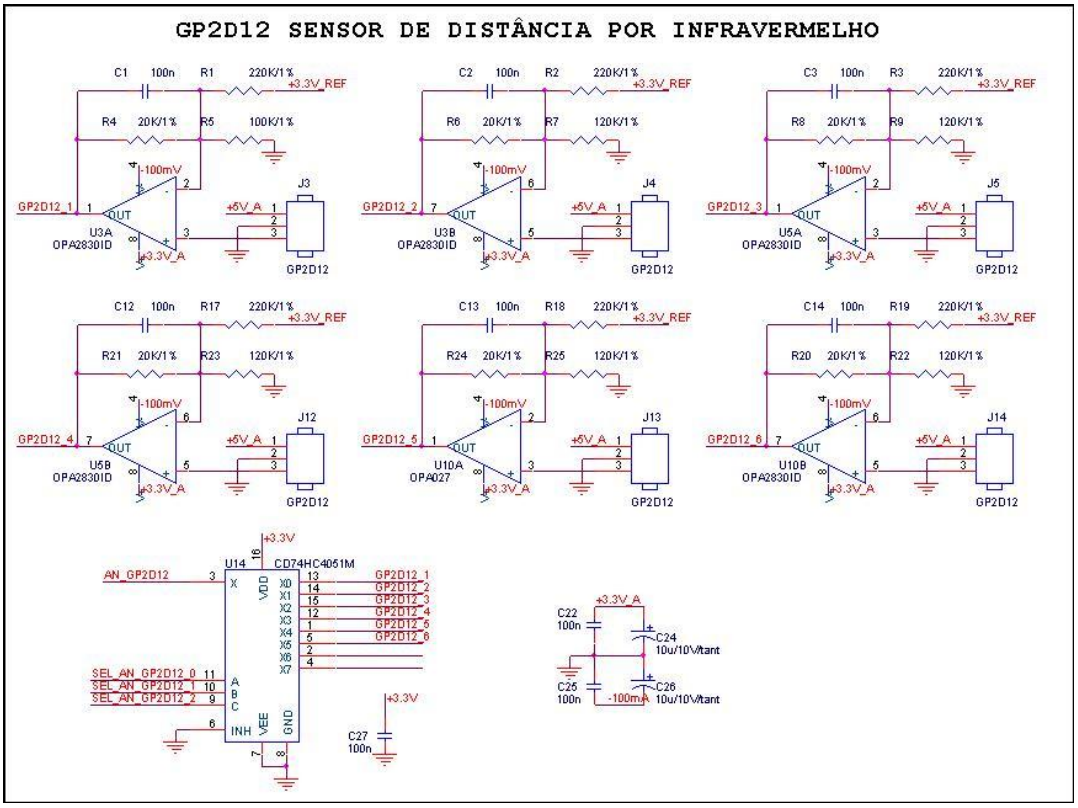

Figura 34: Circuito de condicionamento dos sensores de distância por infravermelho [01]

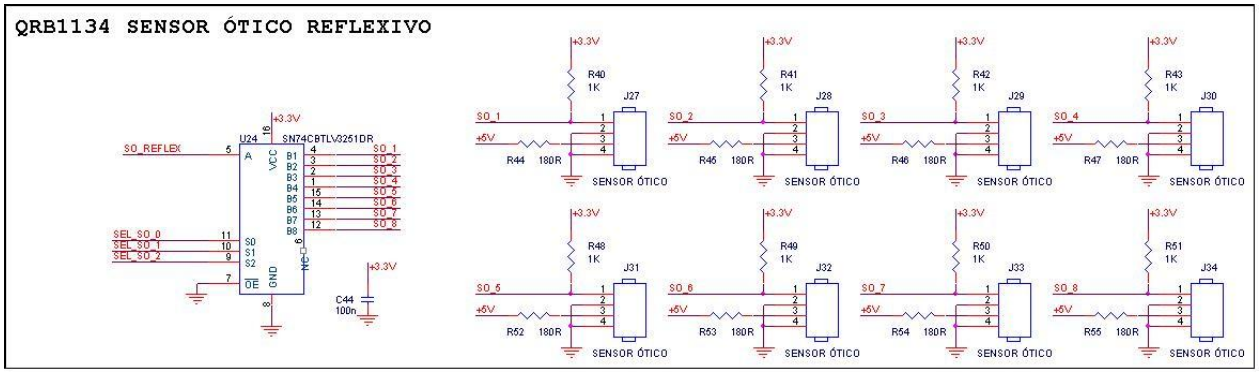

Figura 35: Circuito de condicionamento dos sensores óticos reflexivos [01]

A Figura 36, mostra o circuito de condicionamento dos sensores de pressão da garra, sendo que o circuito integrado U19 faz a seleção do sensor a se fazer leitura.

A Figura 37, mostra o circuito dos sensores de distancia por sem. Estes sensores se comunicam com o microcontrolador via interface I2C, assim sendo, todos compartilham o mesmo barramento de comunicação.

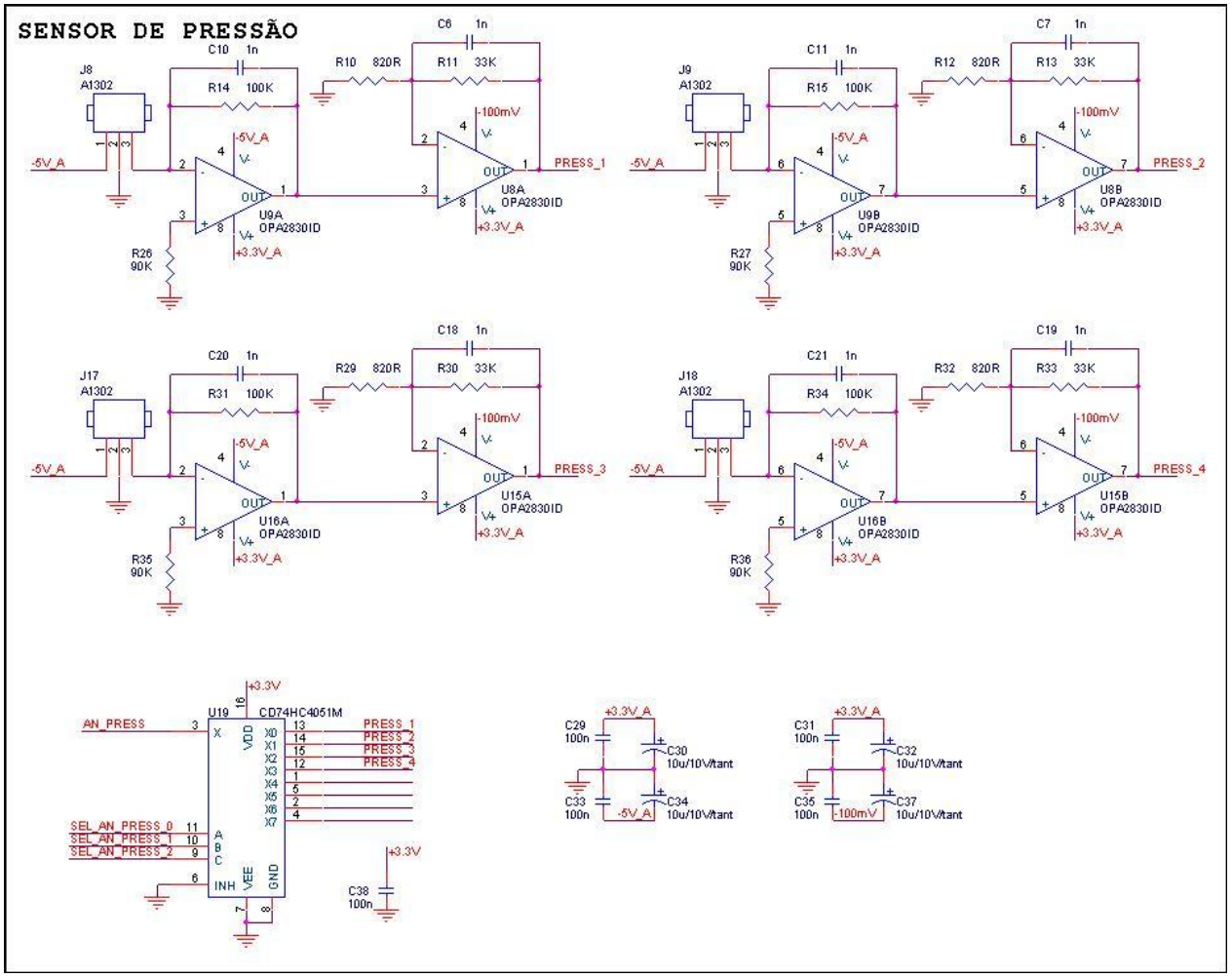

Figura 36: Circuito de condicionamento dos sensores de pressão [01]

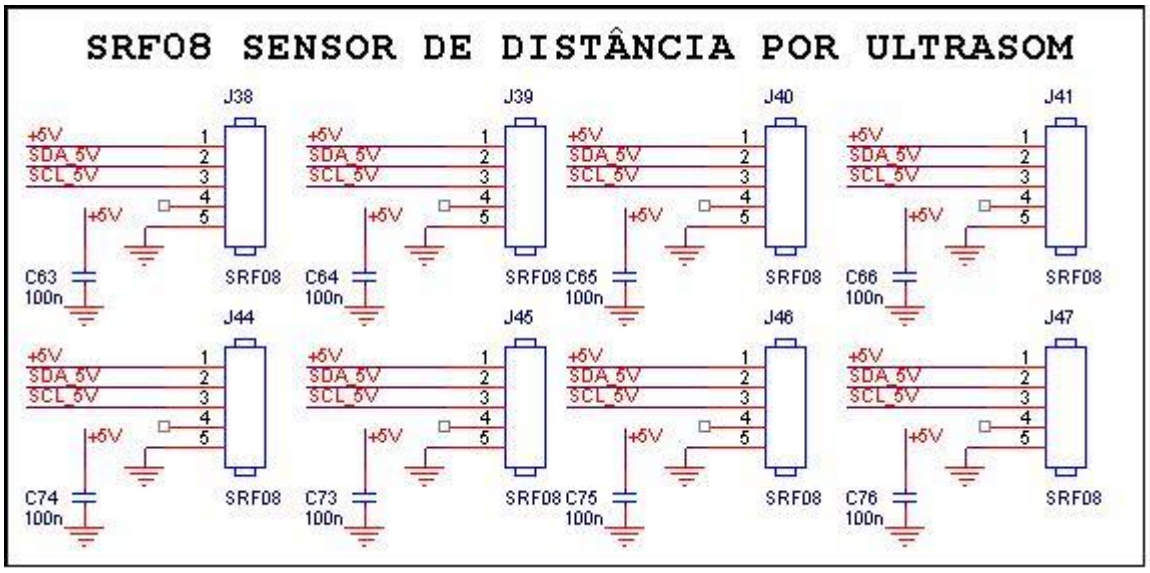

Figura 37: Circuito dos sensores de sem [01]

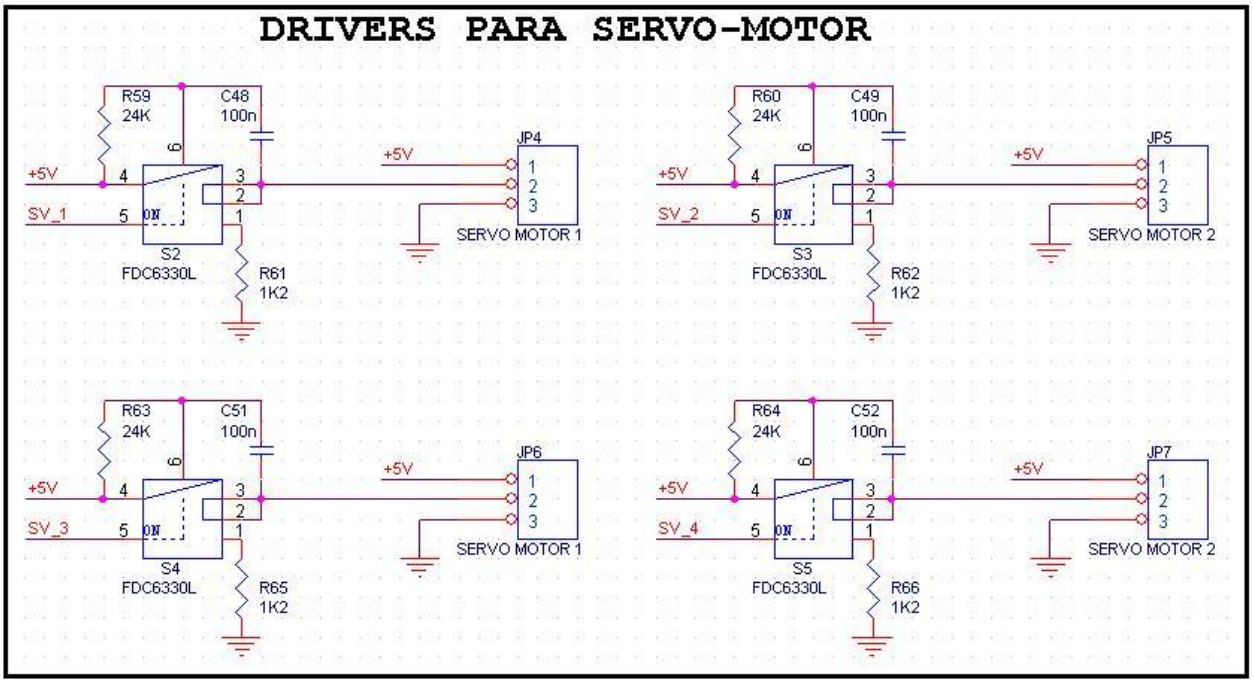

Figura 38: Circuito de acionamento dos motores de direção [01]

A Figura 38, mostra o circuito de controle independente dos 4 motores de direção. As chaves de estado sólido S2 a S5 são acionadas pelos sinais SV_1 a SV_4, vindos do microcontrolador.

A Figura 39, mostra o circuito dos reguladores de tensão. Ele fornece as tensões necessárias para o funcionamento de todos os circuitos da placa mãe, assim como a placa do microcontrolador.

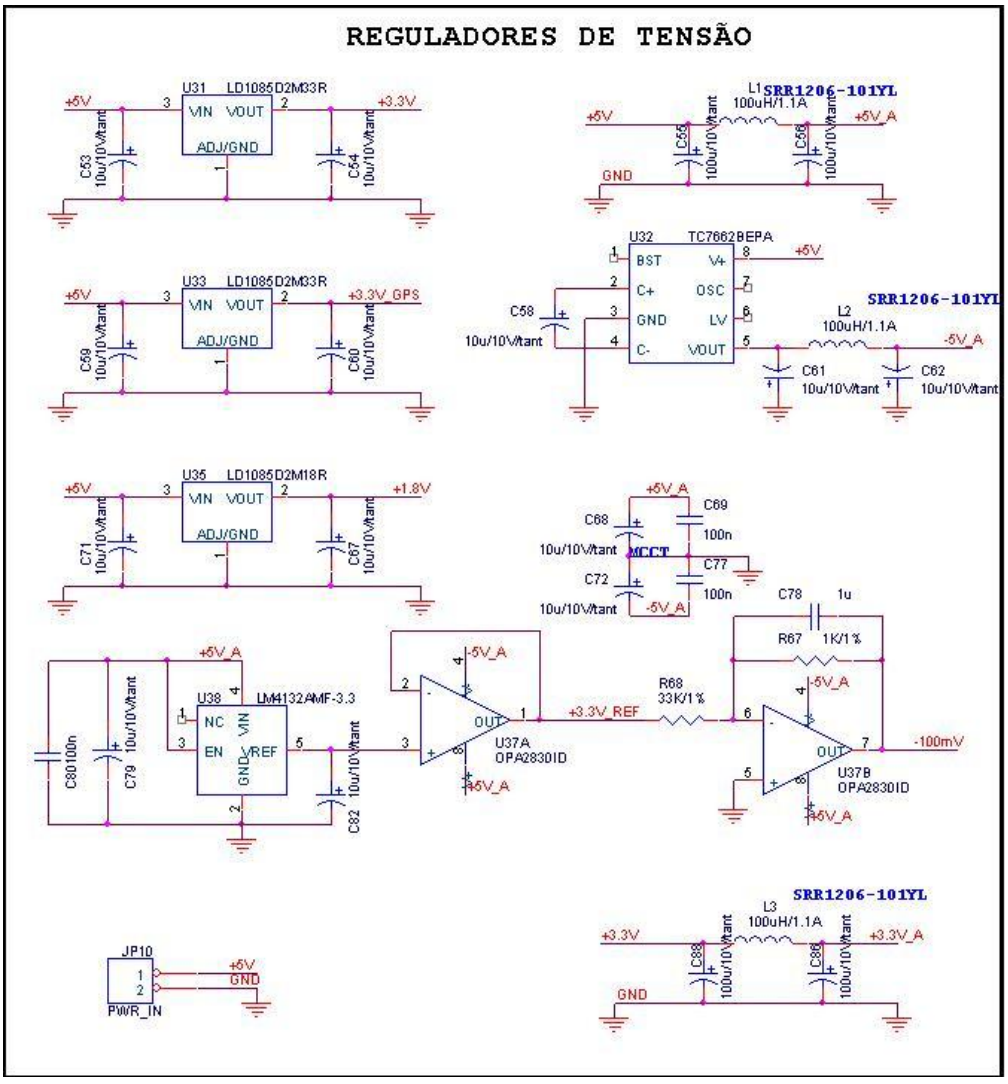

Figura 39: Circuito de reguladores de tensão [01]

O motor de tração possui um *driver* que permite controle de velocidade via PWM ou tensão analógica que varia de 0V a 5V, o microcontrolador do robô trabalha com 3.3V, por isso essa placa foi necessária para ser possível fazer o motor andar com velocidade máxima.

5.2. *LAYOUT* DAS PLACAS

5.2.1. *Layout* da placa do microcontrolador

A placa do microcontrolador e uma placa *multlayer* de quatro camadas, a Figura 40, mostra este *layout* com os componentes.

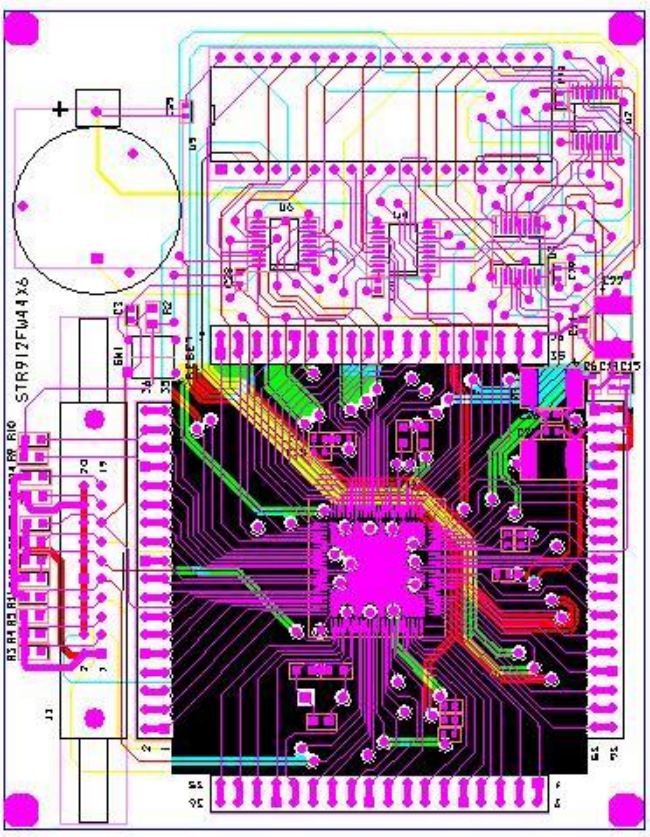

Figura 40: Layout da placa do microprocessador [01]

5.2.2. *Layout* da placa Mãe

A placa mãe é uma placa em fibra de vidro dupla faz. A Figura 41, mostra a distribuição de componentes.

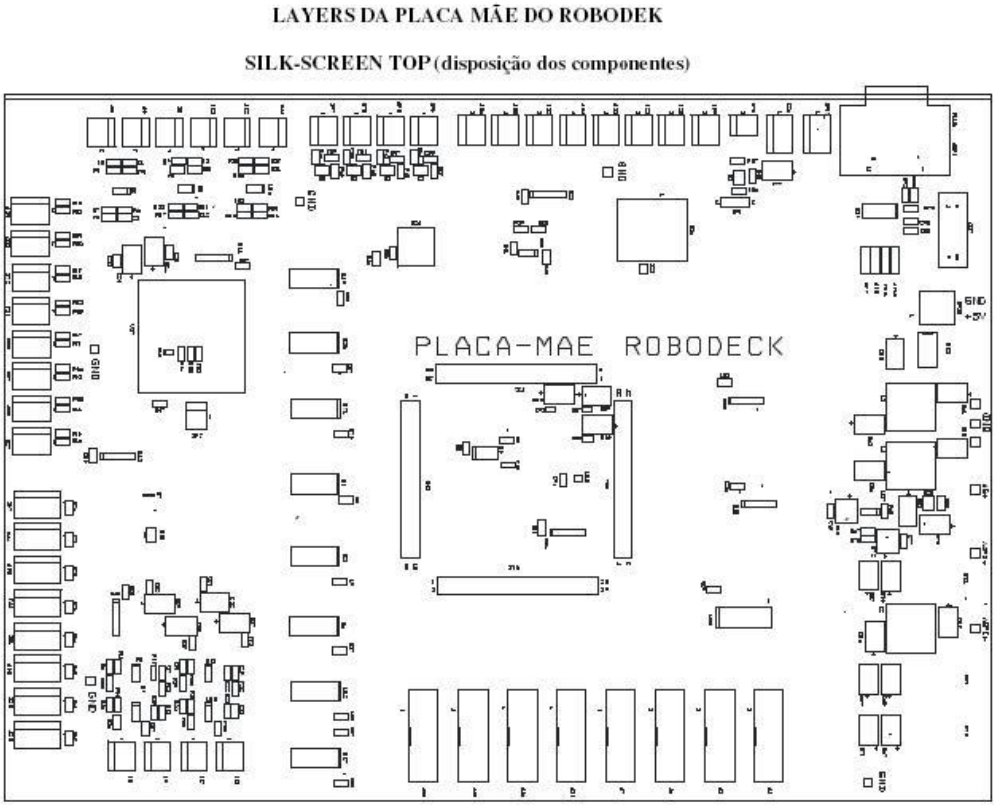

Figura 41: Placa Mãe – Silk Screen Top [01]

CAPÍTULO 6
PROJETO MECÂNICO

6.1.ESTUDO DO MODELO MECÂNICO [01]

O objetivo deste capítulo é o estudo dos conceitos do Robô através de análises da dinâmica do robô (comportamentos longitudinal, vertical e lateral). Também é foco do trabalho a análise de torque necessário para a movimentação do mecanismo (definição do motor a ser utilizado), e o estudo do ângulo de esterço permitido.

6.1.1. Dados Físicos do Robô

A Tabela 1, mostra a massa de cada parte do robô e a Figura 42, ilustra o modelo em multicorpos, feito em ambiente ADAMS/View®.

O modelo é dividido em quatro partes:

1. Mecanismo moto-redutor: motor;
2. Mecanismo de esterço: servo-motor + mecanismo quatro barras;
3. Chassis: estrutura de fixação das partes (compreende a pinça também);
4. Pneus e pista

Tabela 1: Tabela de massa das peças do Robô ´[01].

	Massa Unitária [Kg]	Quantidade	Total [Kg]
Estrutura Superior	6	1	6
Estrutura Inferior	1,7	1	1,7
Manga	0,08	4	0,32
Conj. Garra	0,6	1	0,6
Agregados	1,5	1	1,5
Conj. Pneus/Roda	0,59	4	2,36
Conj. Moto/Redutor	0,84	1	0,84
Baterias	0,5	6	3
Servo	0,37	4	1,48
Cubo de Rodas	0,03	4	0,12
Eixo de Saída	0,08	2	0,16
Semi-eixo	0,02	2	0,04
Ponta de Eixo	0,02	2	0,04
Total			18,16

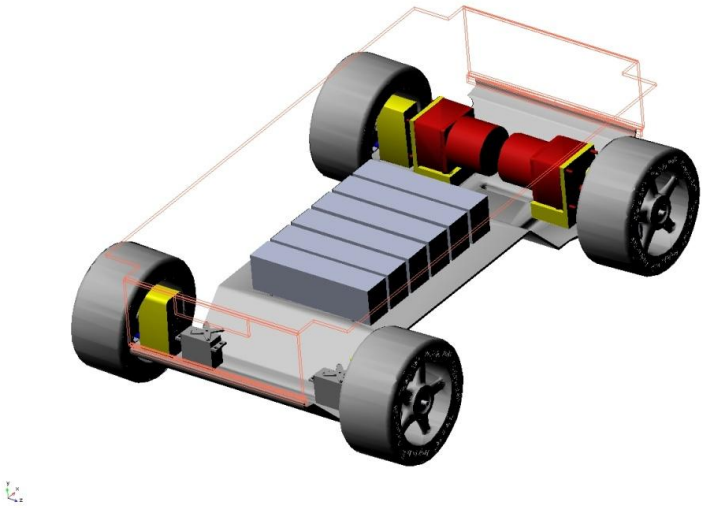

Figura 42: Modelo do Robô importado em ambiente Adams/View® [01].

O estudo aqui apresentado demonstra somente o comportamento dinâmico do robô, não analisando possíveis interferências entre as partes durante o funcionamento. Para o estudo da dinâmica se realizaram as seguintes análises:

- Análise de raio constante;
- Análise de subida de rampa;
- Análise de passagem em obstáculos.

6.1.2. Dados dos Motores

No estudo cinemático e dinâmico feito no ambiente ADAMS/View® considerei o análise de dois motores com torques diferentes. As curvas de torque de cada motor podem ser observadas na Figura 43 e Figura 44.

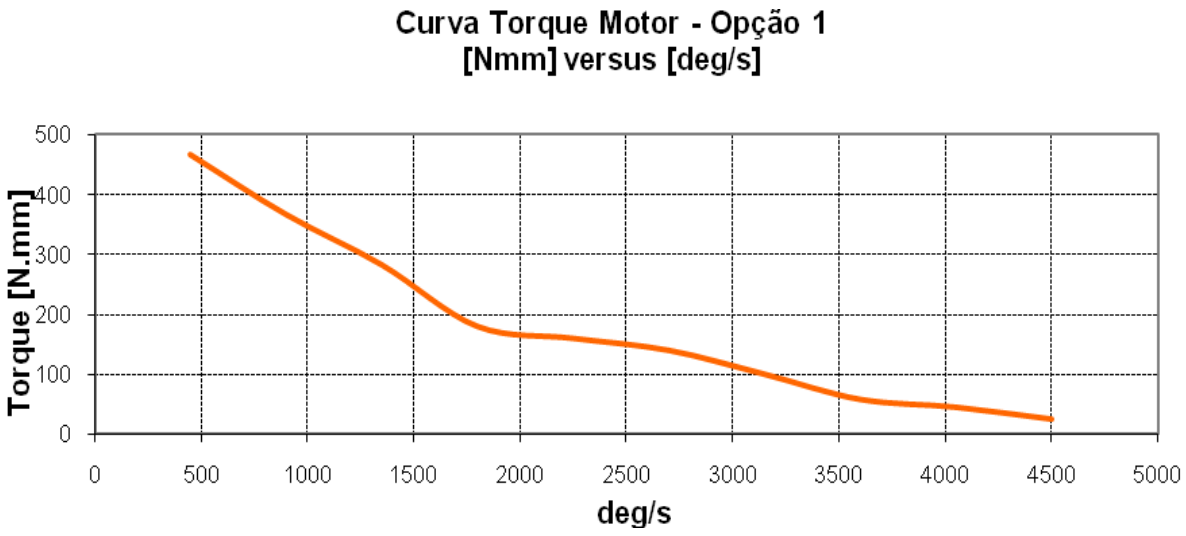

Figura 43: Curva de Torque do Motor – Opção 1 [01]

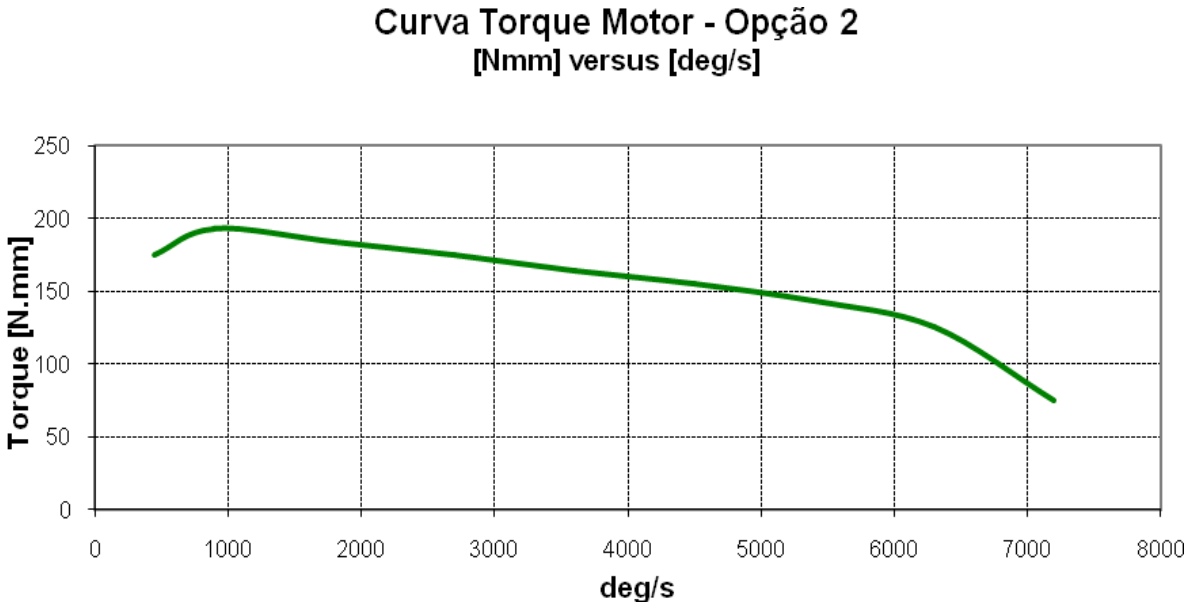

Figura 44: Curva de Toque do Motor – Opção 2 [01].

Os motores estão alocados na região posterior do robô (oposta à garra), sendo dois: um esquerdo e um direito, independentes entre si. Cada motor possui em seu eixo de saída um redutor acoplado, com relação de redução de 30:1. O torque (do motor) foi aplicado por uma equação contínua, levando em consideração a curva dos motores. Também foi aplicado um controle de tração simplificado para melhor aproveitamento dos motores.

6.1.3. Dados do Chassi

O chassi é compreendido pelas partes ilustradas na Figura 45. Essas partes não possuem movimento relativo entre si, podendo ser unidas em um único corpo.

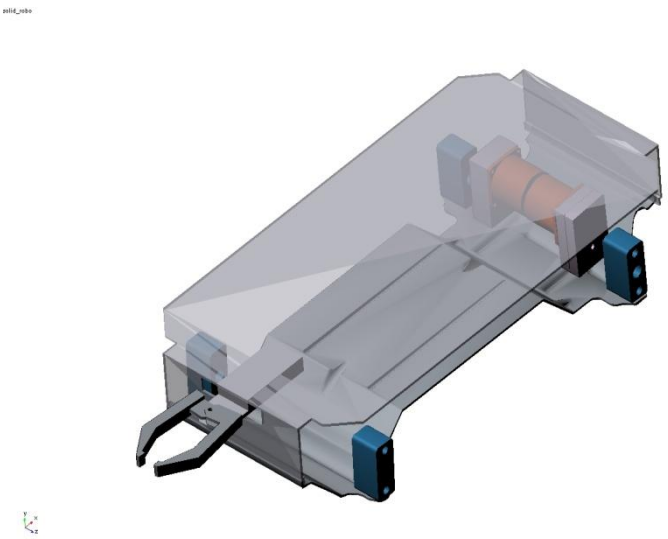

Figura 45: Chassi [01].

6.1.4. Dados do Pneus e Pista

O modelo de pneu utilizado para as análises foi o UAT (*University of Arizona Tire*), com os parâmetros ajustados conforme características do pneu utilizado no projeto.

As pistas utilizadas nas análises foram:

- Raio constante: *mdi_2d_flat.rdf*;
- Subida de rampa: *2d_ramp.rdf*;
- Passagem por obstáculos: *2d_pot_hole.rdf*.

6.1.5. Simulação e Comportamento Dinâmico

A seguir são descritas as análises realizadas no ambiente ADAMS/View®, e os resultados obtidos em cada uma delas. A velocidade de operação do robô foi considerada entre 400 mm/s e 600 mm/s.

6.1.5.1. Análise de Subida de Rampa

Para a análise de subida de rampa foi utilizada uma pista com inclinação de 30° (inclinação máxima desejada), e altura infinita. As análises foram feitas com o robô a uma velocidade longitudinal de 400 mm/s e 500 mm de *start* (distância percorrida até a rampa). A pista utilizada está ilustrada na Figura 46.

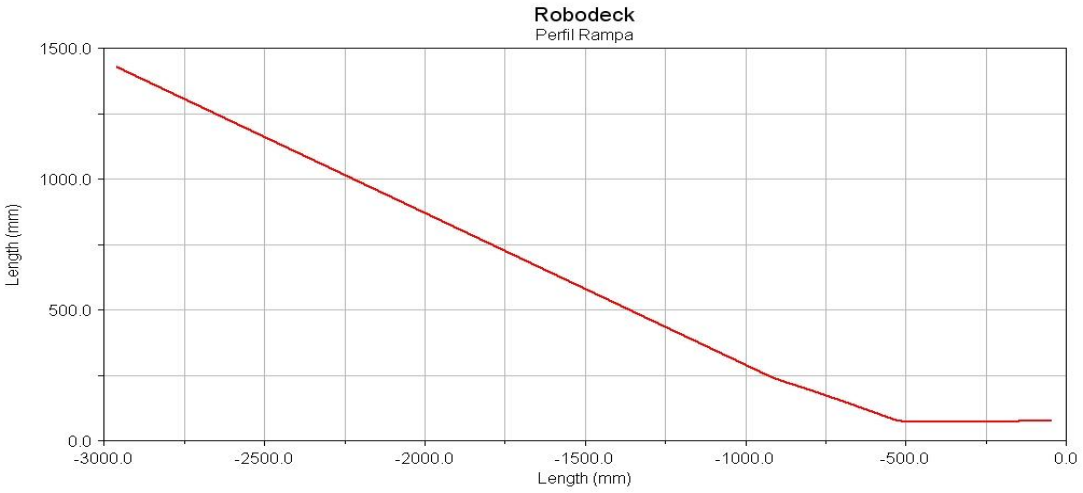

Figura 46: Análise de Subida de Rampa [01].

A Figura 47, mostra o comportamento da velocidade longitudinal do centro de massa do robô pelo tempo durante a subida da rampa. A perda de velocidade durante a subida da rampa para o motor 1 é de cerca de 40% e, para o motor 2, é de 88 %.

Ambos os motores superaram a rampa com inclinação de 30°. A Figura 48, mostra a variação dos torques aplicados em cada motor (motor 1 e 2), pelo tempo de subida.

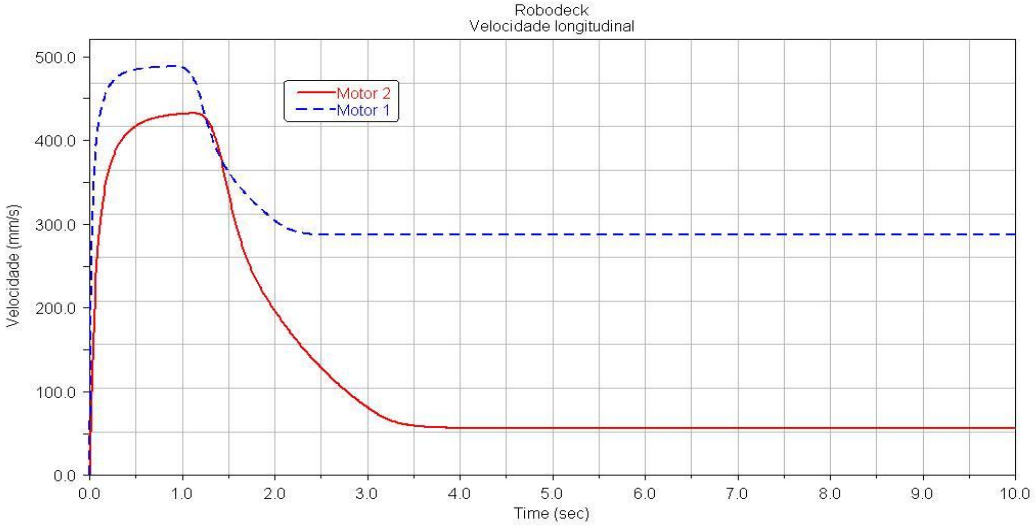

Figura 47: Gráfico da perda de velocidade durante a subida de rampa [1].

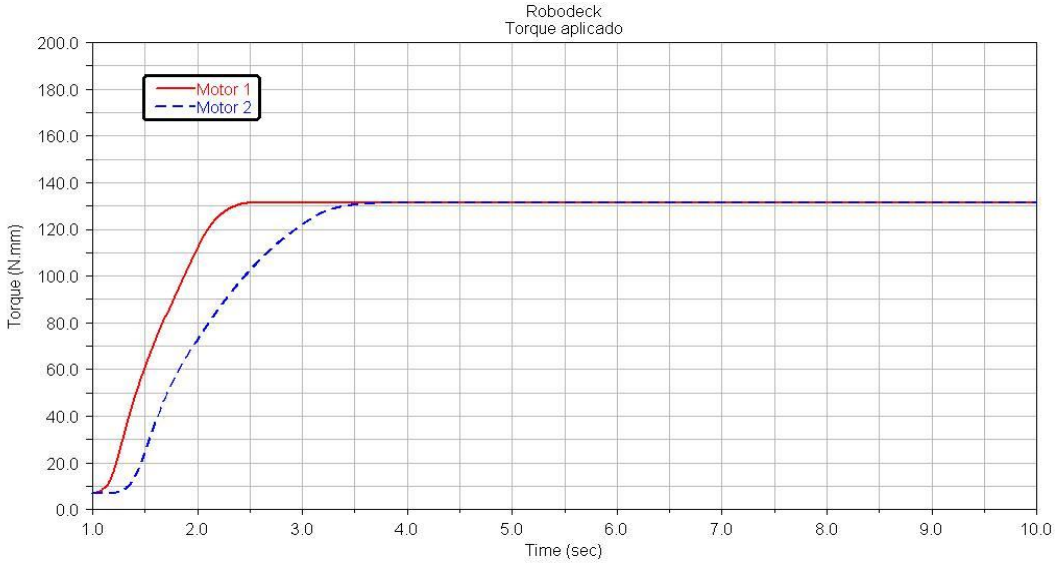

Figura 48: Torque aplicado pelos motores durante o evento [01].

6.1.5.2. Análise de Passagem por Obstáculos

A análise de passagem por obstáculo foi dividida em três partes:

- *Pot Hole* = passagem por depressão (retangular) no pavimento;

- Lombada 2 = passagem por saliência (arredondada) no pavimento.

A- Pot Hole

Nesta análise, o robô faz uma passagem em depressões com diferentes profundidades. É analisada aqui, a profundidade que ele consegue vencer. Foram estudadas três profundidades: 15mm, 25mm e 37mm, com duas velocidades de passagem: 400mm/s e 600mm/s. Os resultados obtidos para cada análise são mostrados na Tabela 2**.**

Tabela 2:Tabela de resultados das análises de pot hole [01].

	Motor 1		Motor 2	
	400 mm/s	600 mm/s	400 mm/s	600 mm/s
Profundidade: 15 mm Torque máx. (N.mm)	13,24	10,54	12,81	9,88
Profundidade: 25 mm Torque máx. (N.mm)	18,09	13,2	17,17	17,22

Na Figura 49 e Figura 50 é possível observar o deslocamento dos eixos dianteiro e traseiro do robô na análise com obstáculo de 15 e 25 mm de profundidade, respectivamente, e 400mm/s. Na Figura 51 e Figura 52, é possível observar o deslocamento dos eixos dianteiro e traseiro do robô na análise com obstáculo de 15 e 25 mm de profundidade, respectivamente, e velocidade longitudinal de 600mm/s.

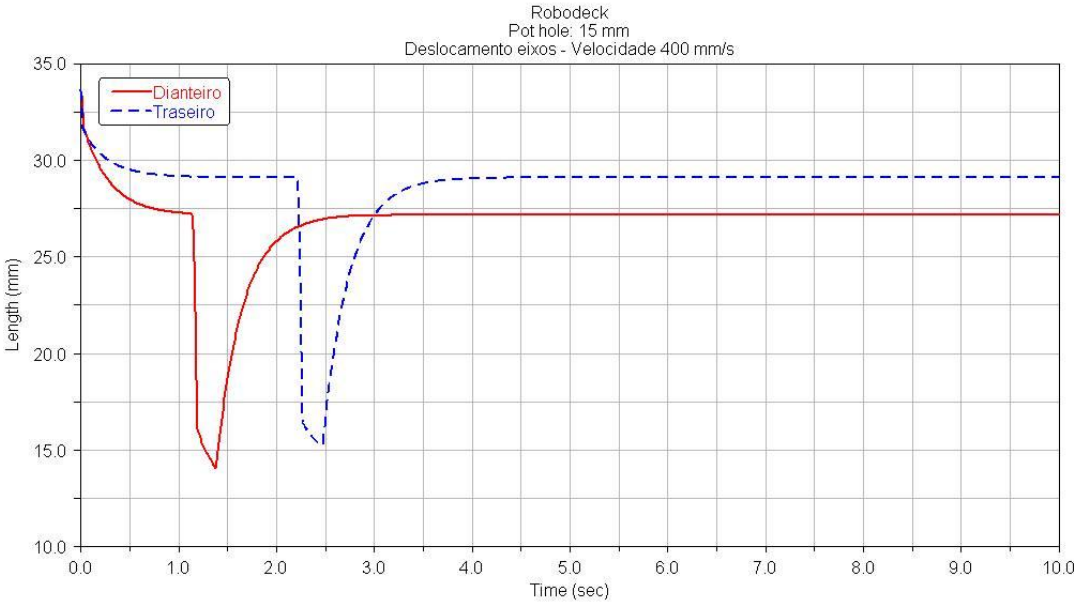

Figura 49: Deslocamento dos eixos durante evento (15 mm) [01].

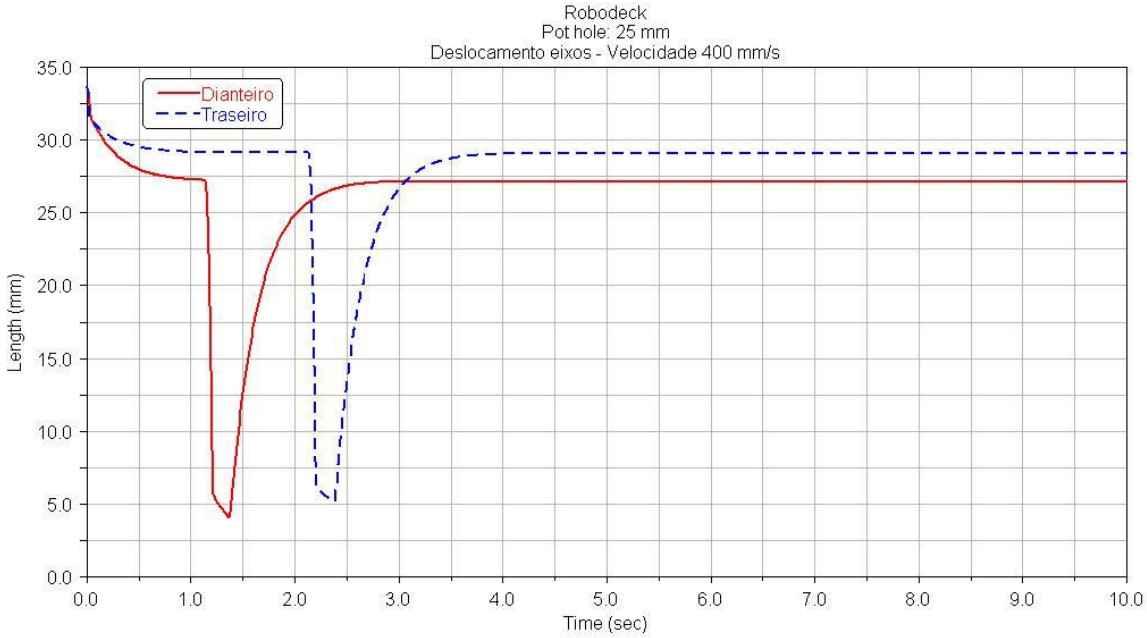

Figura 50: Deslocamento dos eixos durante evento (25 mm) [01].

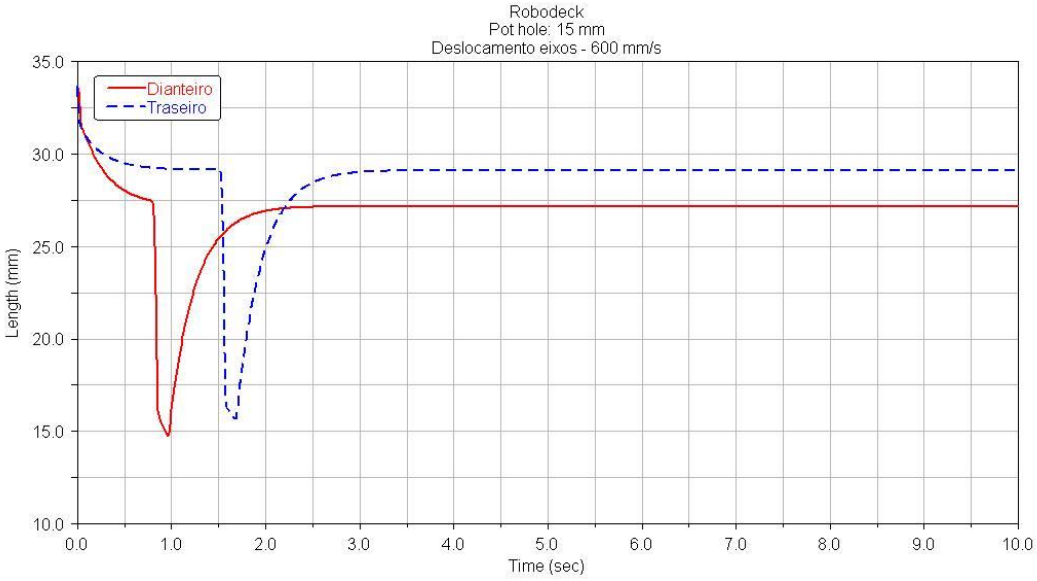

Figura 51: Deslocamento dos eixos durante evento (15 mm) [01].

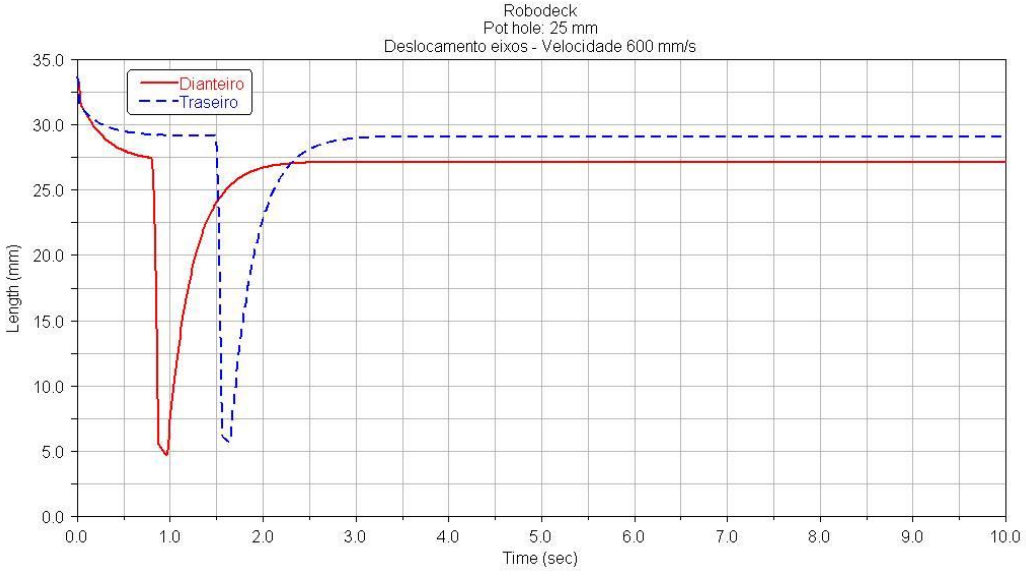

Figura 52: Deslocamento dos eixos durante evento (25 mm) [01].

O gráfico, para uma depressão de 15 mm, de Torque *versus* tempo pode ser visto na Figura 53, para o Motor 1, e na Figura 54, para o Motor 2. O torque necessário para vencer o obstáculo é inferior ao torque máximo disponível em ambos os motores (Motor 1 e Motor 2).

A velocidade longitudinal do robô ao longo da análise, para as velocidades de 400mm/s e 600mm/s, é mostrada na Figura 55. A variação da velocidade longitudinal é semelhante em todos os casos, e por este motivo é mostrada apenas a variação para obstáculos com 25mm de profundidade.

A maior depressão que o robô consegue superar é de 37mm, porém, já com alguma dificuldade. Com um obstáculo de 38mm de profundidade, o robô perde estabilidade tomba devido ao impacto das rodas dianteiras com o degrau.

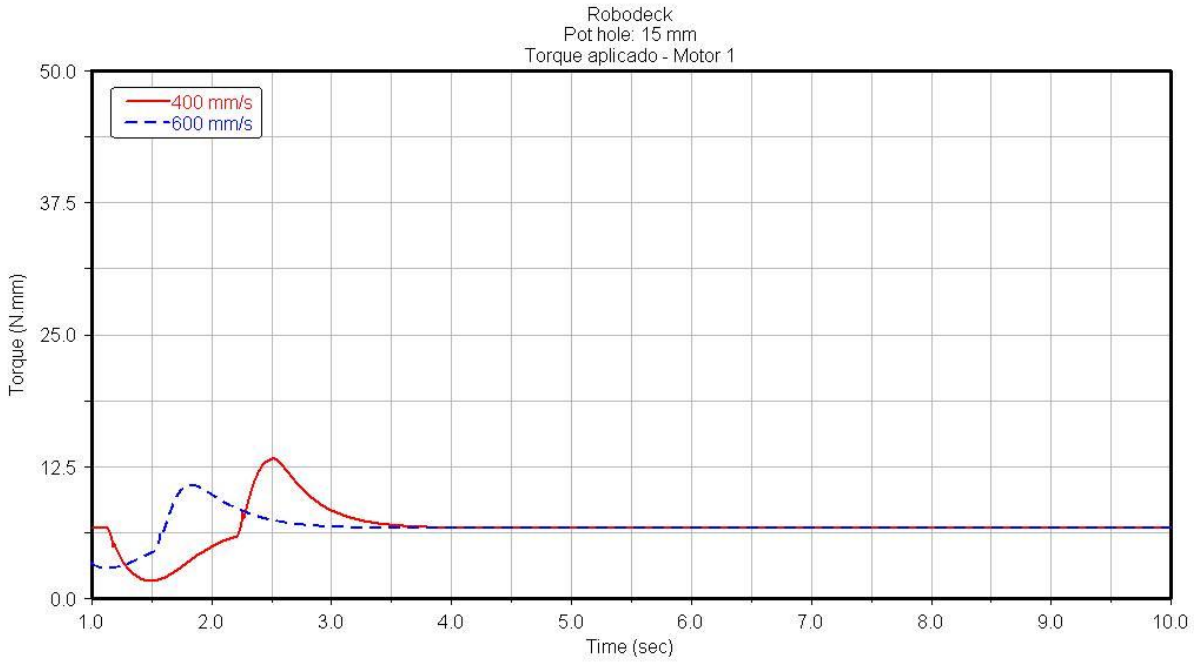

Figura 53: Torque aplicado pelo motor 1 [01]

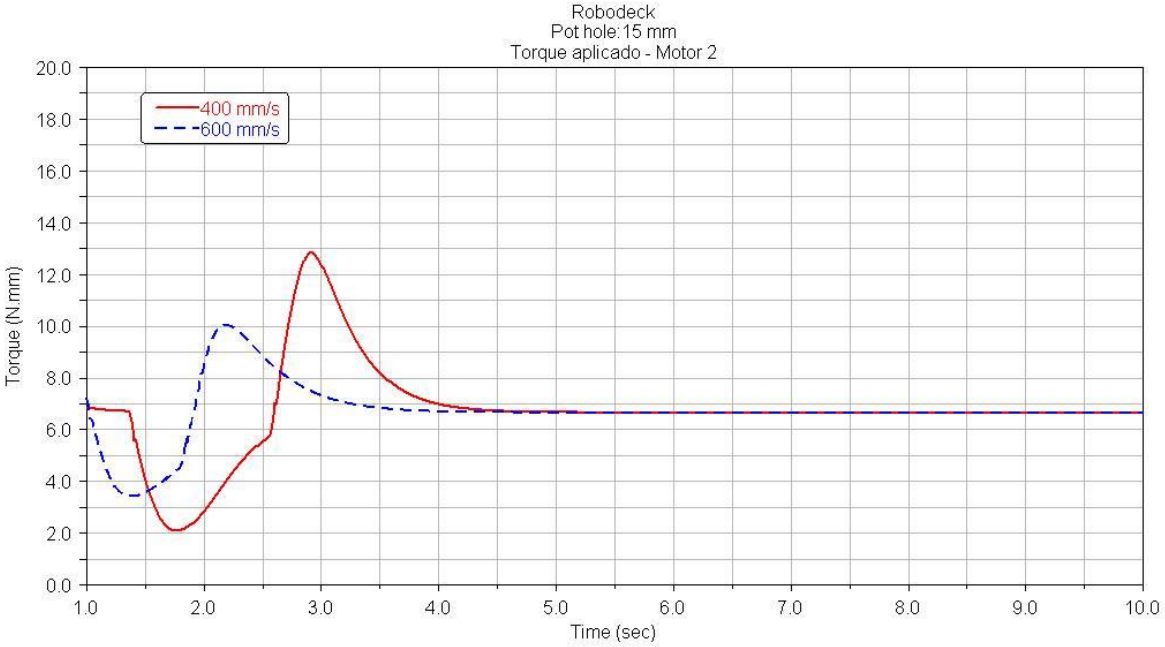

Figura 54: Torque aplicado pelo motor 2 [01]

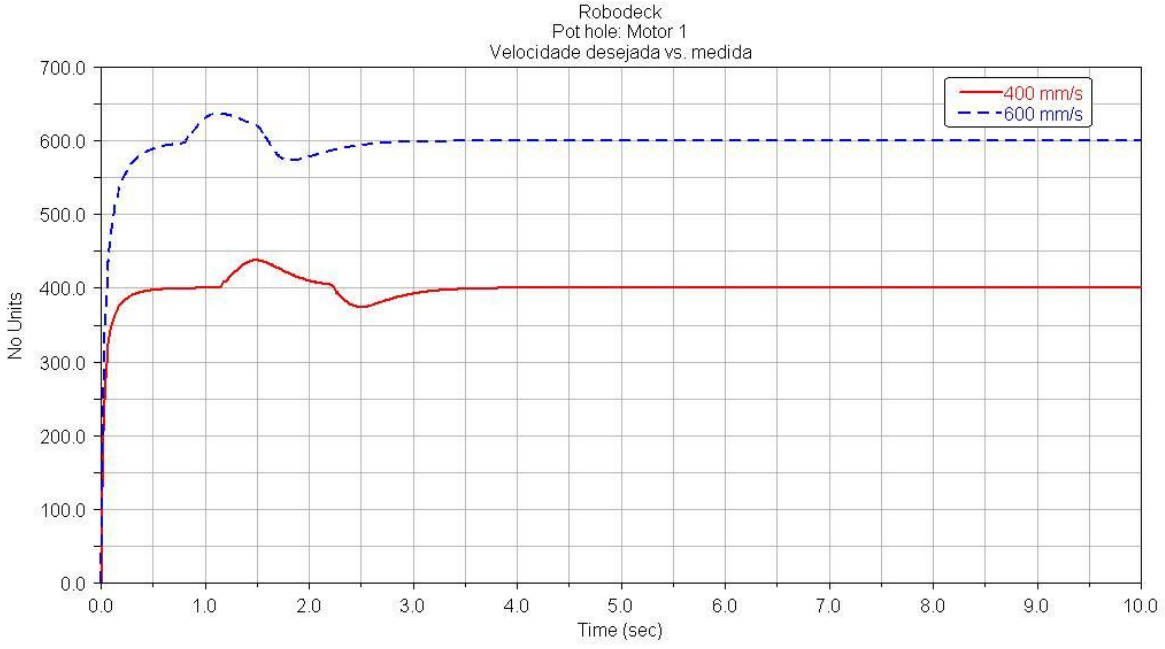

Figura 55: Variação da velocidade durante a passagem pelos obstáculos [01]

6.1.5.3. Lombada

A análise de lombada consiste na passagem em saliências com diferentes alturas, Figura 56. As rodas passam nos obstáculos em momentos diferentes, ou seja, fora de fase. É analisada a maior altura que o robô é capaz de superar. Foram estudadas três alturas: 10mm, 25mm e 35mm, com duas velocidades de passagem: 400mm/s e 600mm/s. Os resultados obtidos para cada análise são mostrados na Tabela 3.

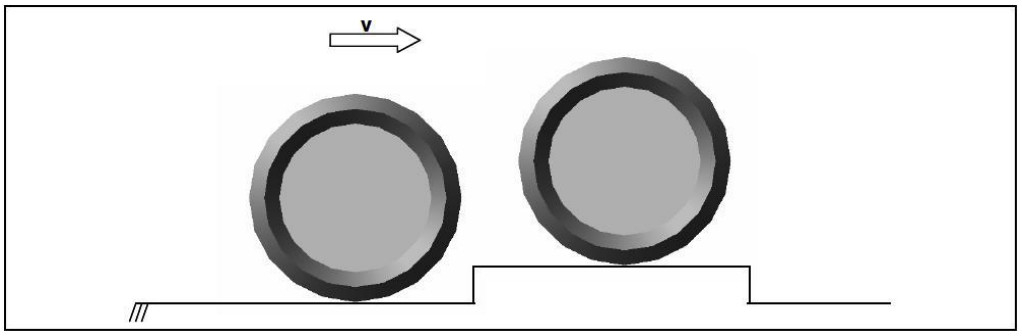

Figura 56: Análise de Passagem por Obstáculos [01].

LOMBADA

Tabela 3: Tabela de resultados das análises de lombada [01].

	Motor 1		Motor 2	
	400 mm/s	600 mm/s	400 mm/s	600 mm/s
Altura: 10 mm Torque máx. (N.mm)	8,92	8,25	8,82	8,19
Altura: 25 mm Torque máx. (N.mm)	12,35	9,47	12,31	9,32

Na Figura 57 e Figura 58, é possível observar a deslocamento dos eixos dianteiro e traseiro do robô na análise com obstáculo de 10 e 25mm de altura, respectivamente, e 400mm/s. Na Figura 59 e Figura 60, é possível observar a deslocamento dos eixos dianteiro e traseiro do robô na aná-

lise com obstáculo de 10 e 25mm de altura, respectivamente, e velocidade longitudinal de 600mm/s.

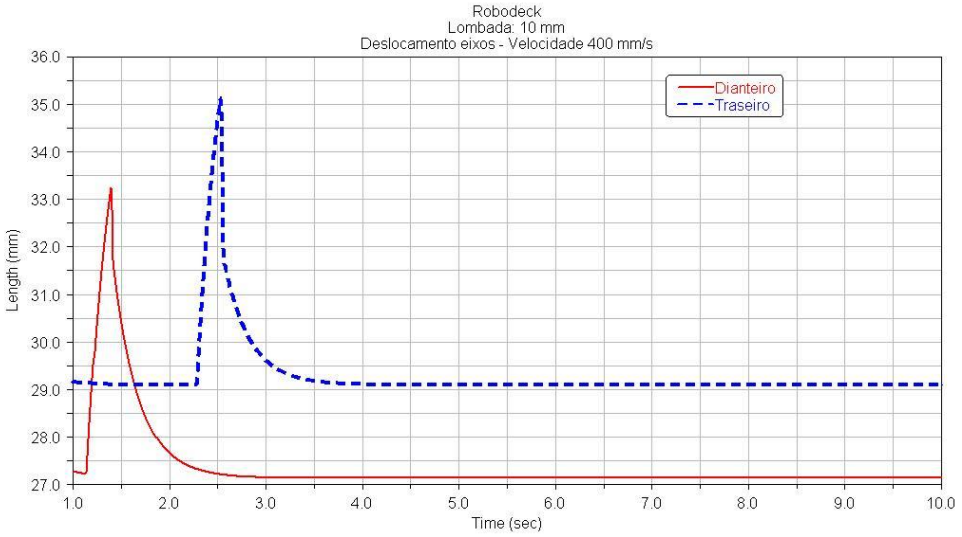

Figura 57: Deslocamento dos eixos durante evento (10 mm) [01].

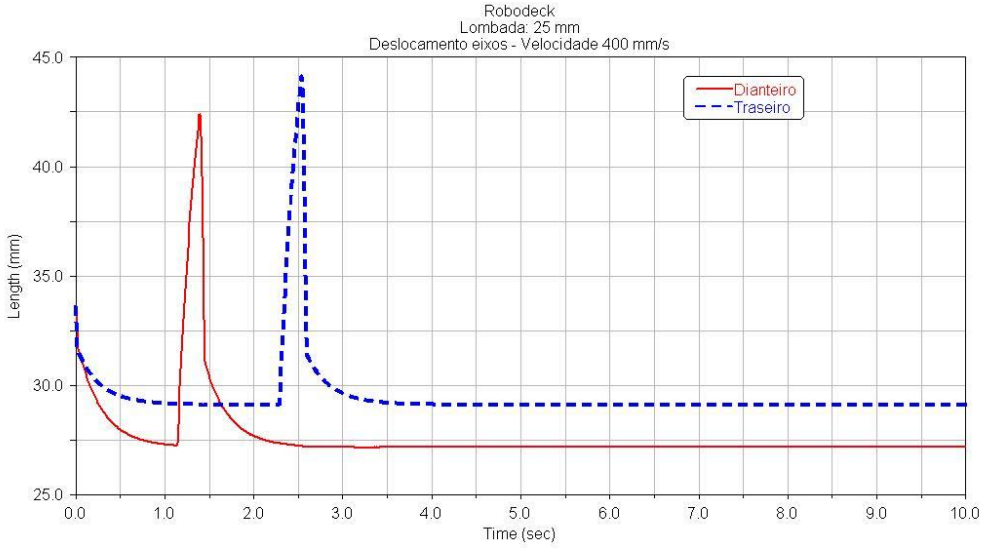

Figura 58: Deslocamento dos eixos durante evento (25 mm) [01].

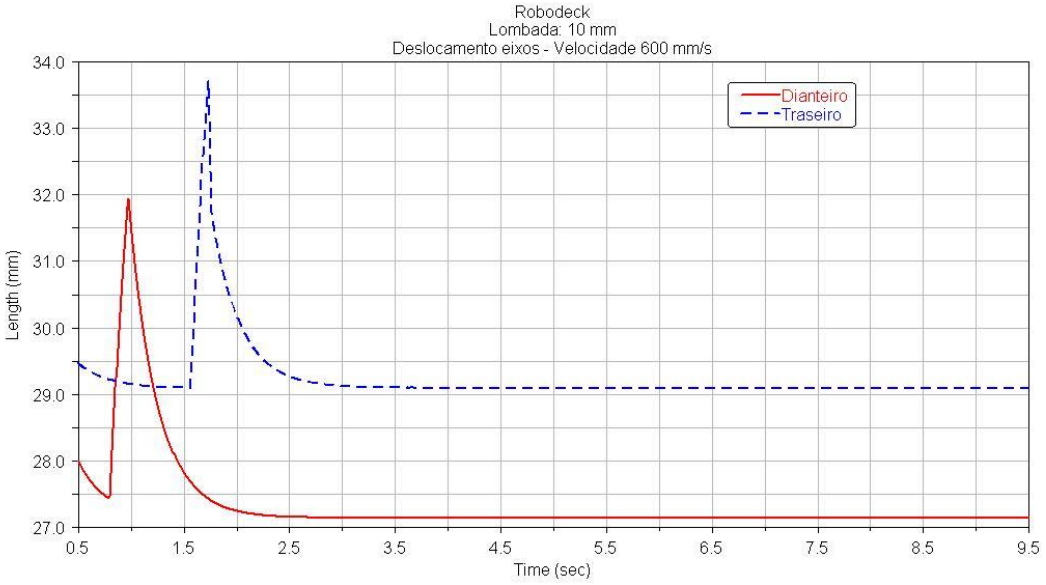

Figura 59: Deslocamento dos eixos durante evento (10 mm) [01].

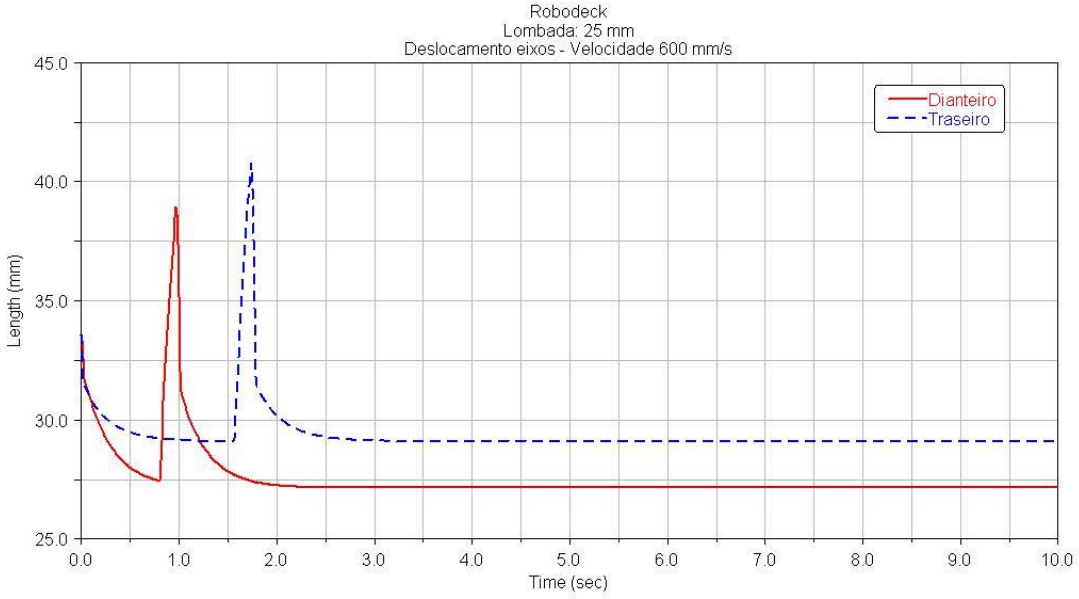

Figura 60: Deslocamento dos eixos durante evento (25 mm) [01].

O gráfico, para uma altura de 10mm, de Torque *versus* tempo pode ser visto na Figura 61, para o Motor 1, e na Figura 62, para o Motor 2. O torque necessário para vencer o obstáculo é inferior ao torque máximo disponível em ambos os motores (Motor 1 e Motor 2).

A velocidade longitudinal do robô ao longo da análise, para as velocidades de 400mm/s e 600mm/s, é mostrada na Figura 63. A variação da velocidade longitudinal é semelhante em todos os casos, e por este motivo é mostrada apenas a variação para obstáculos com 25mm de altura.

A maior altura que o robô consegue superar é de 35mm, porém, já com alguma dificuldade. Com um obstáculo de 36mm de altura, o robô perde estabilidade tomba devido ao impacto das rodas dianteiras com o degrau.

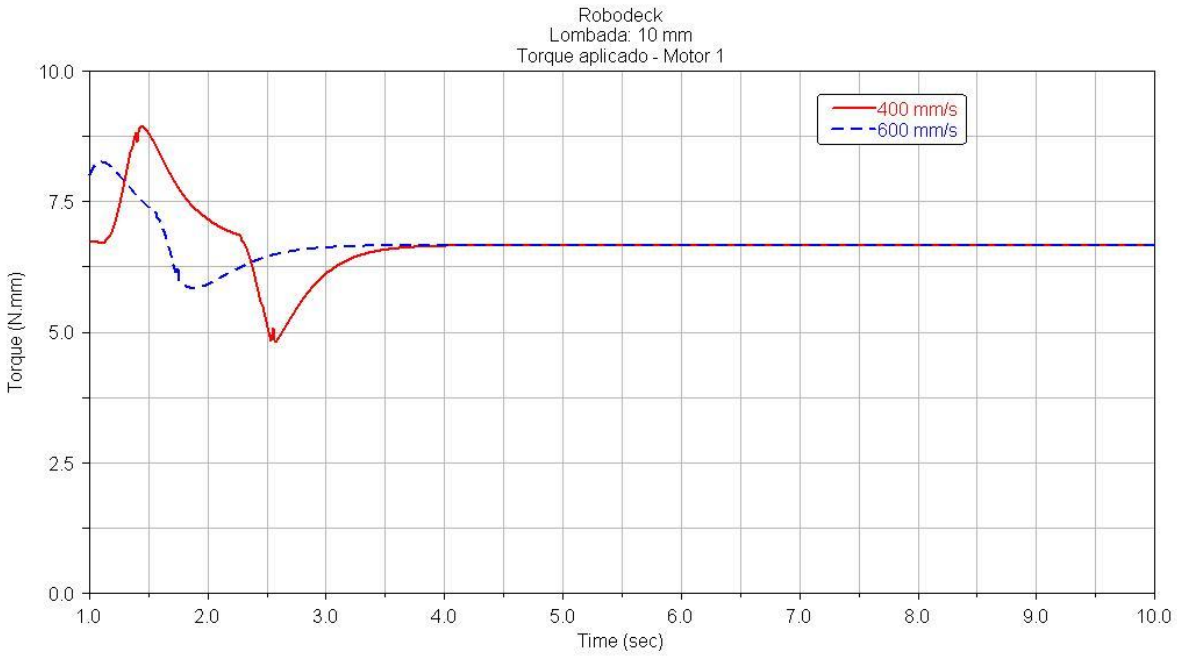

Figura 61: Torque aplicado pelo motor 1 [01].

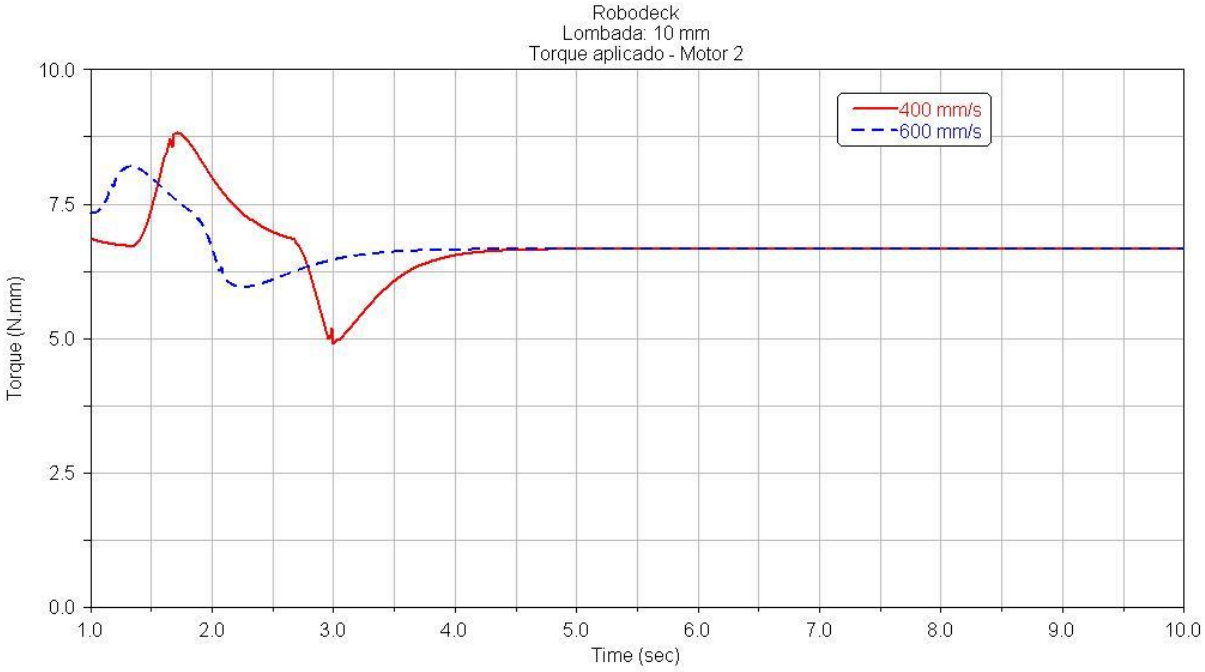

Figura 62: Torque aplicado pelo motor 2 [01].

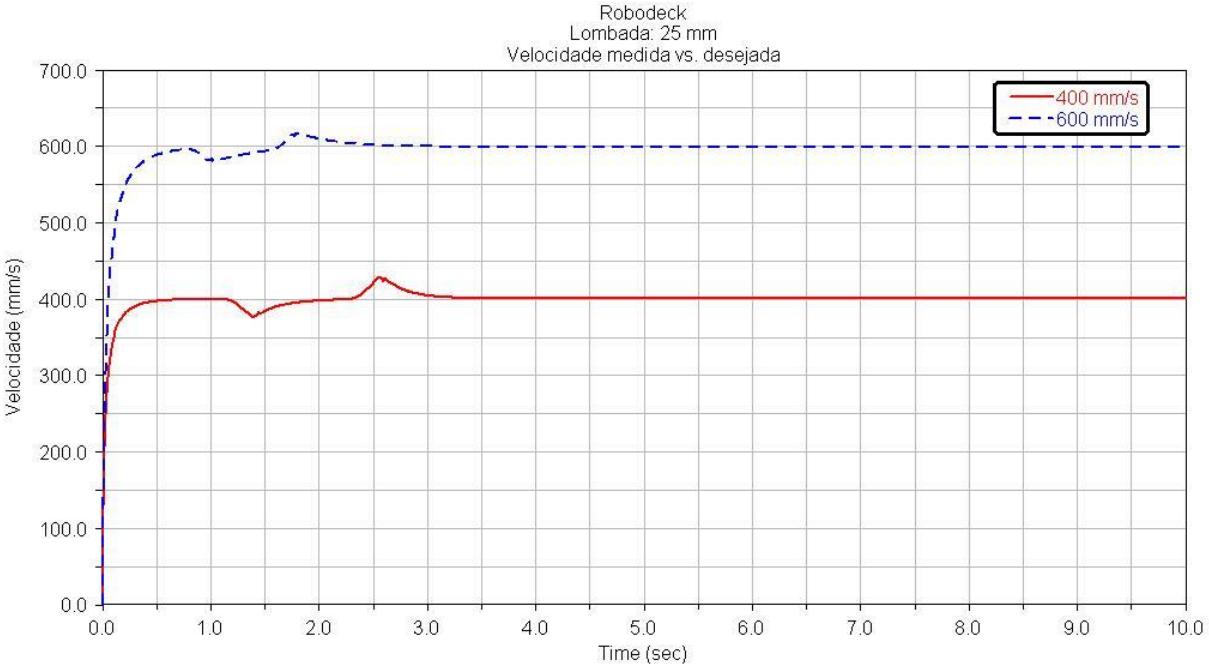

Figura 63: Variação da velocidade durante a passagem pelos obstáculos [01].

6.1.5.4. Análise de Giro

6.1.5.4.1. Análise de Raio Constante

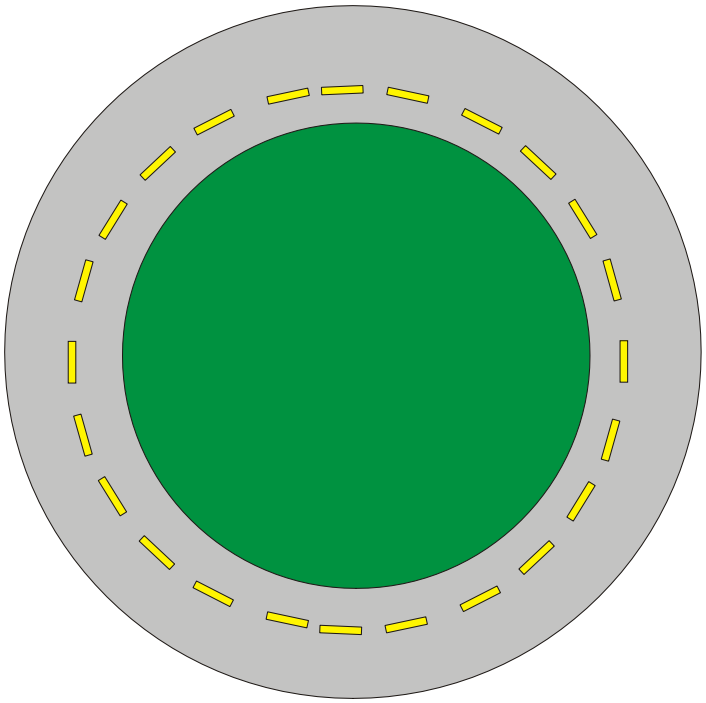

Figura 64: Análise de Raio constante [01].

Nesta análise, é aplicado um ângulo de esterço nas rodas do robô simulando uma pista com raio constante, como mostra a Figura 64. Dessa forma, foi analisado o comportamento lateral do robô.

As análises foram feitas com o robô operando a 500mm/s, e com ângulo de esterço (nas quatro rodas) de 30° - ângulo máximo.

O raio percorrido pelo robô pode ser visto na Figura 65. Os torques executados pelos motores traseiros ao longo do evento podem ser vistos na Figura 66. Na Figura 67, pode-se observar o

torque necessário para proporcionar o esterço nas rodas e, como pode ser constatado neste gráfico, a potência nominal dos servo-motores não é suficiente para viabilizar a manobra. A Tabela 4, mostra uma comparação entre os raios percorridos e os torques aplicados nesta análise.

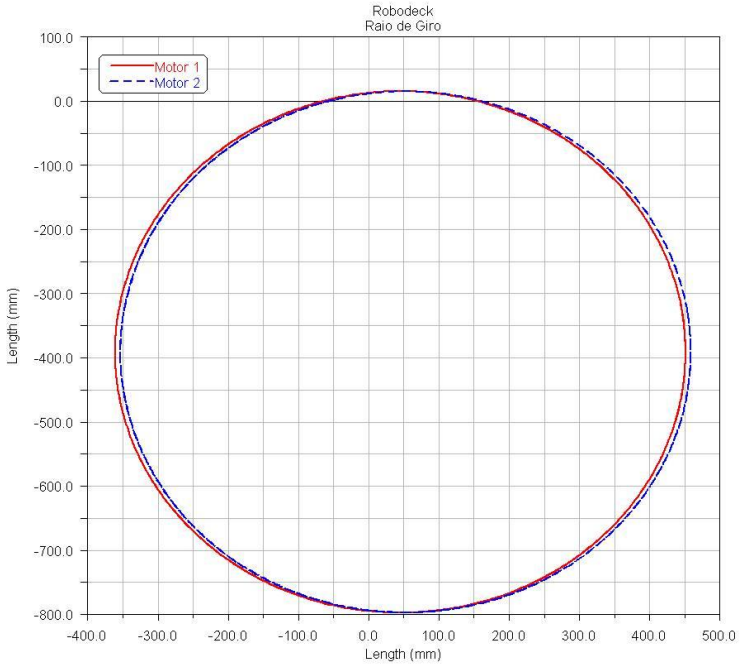

Figura 65: Raio percorrido pelo robô [01].

.

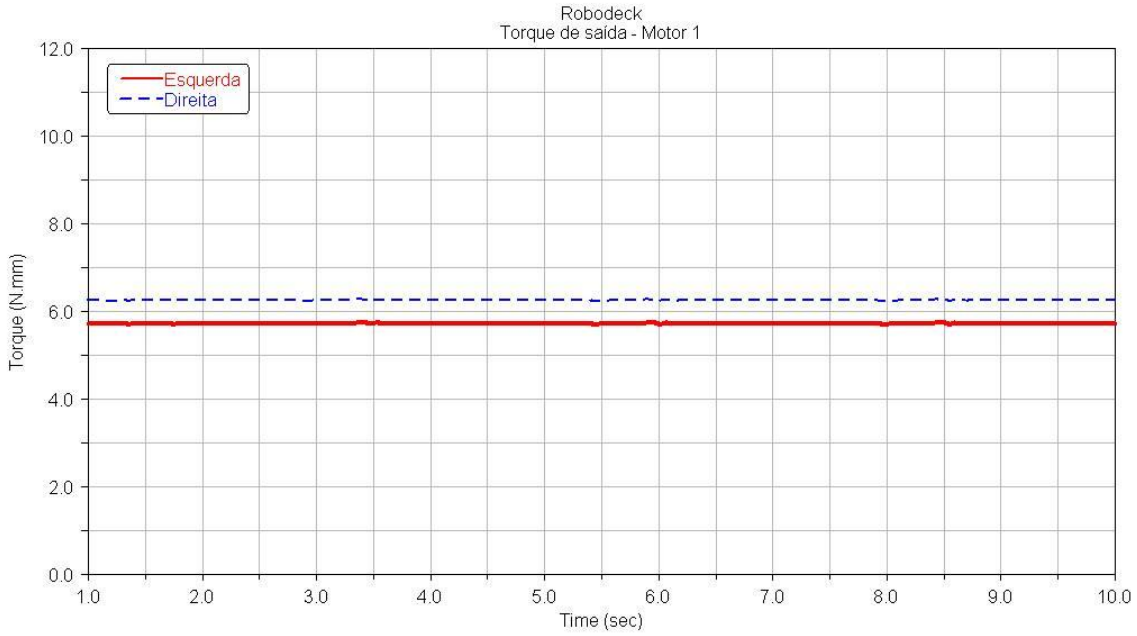

Figura 66: Torque aplicado pelos motores analisados durante evento [01].

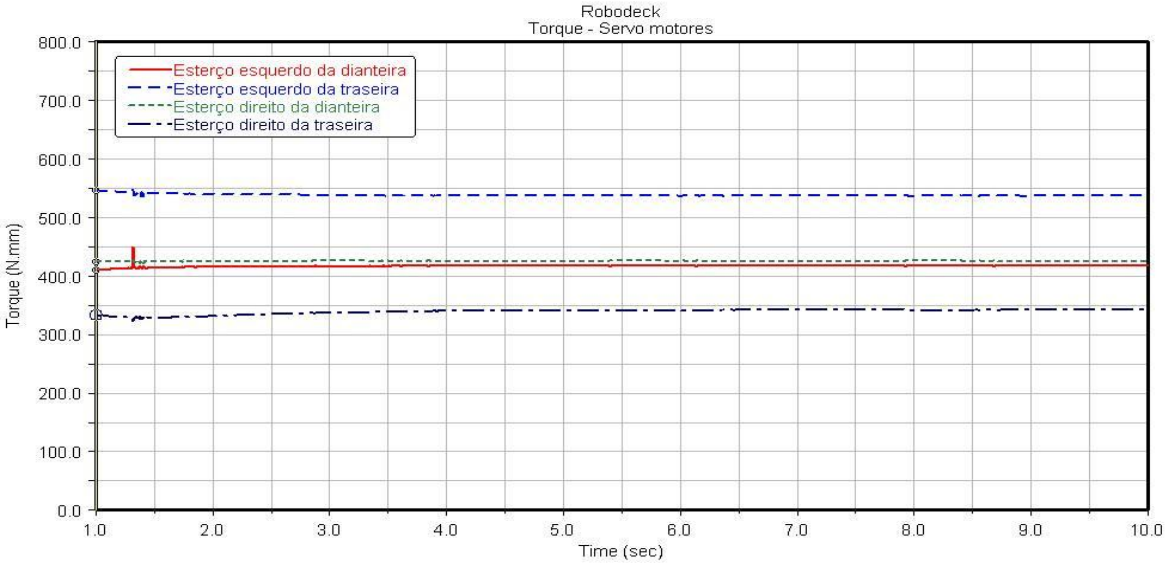

Figura 67: Torque aplicado pelos servos-motores durante a análise.

Tabela 4: Tabela de resultados da análise de giro [01].

		Motor 1	Motor 2
Raio executado (mm)		406	406
Torque aplicado (N.mm)	Direita	6,26	6,19
	Esquerda	5,73	6,82

6.1.5.4.2. Análise de *Turn-in-place*

Na análise de *turn-in-place* o robô executa giros em torno de seu próprio eixo. Para isso, é necessário que os motores direito e esquerdo operem com rotação em sentidos opostos, e as rodas sejam esterçadas conforme ilustra a Figura 68.

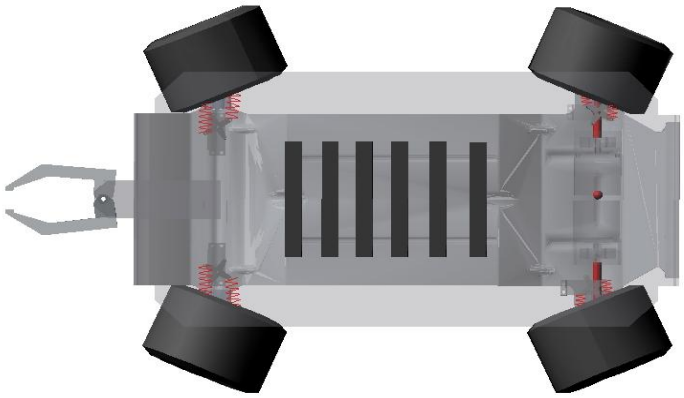

Figura 68: Análise de *Turn-in-place*: disposição das rodas [01].

Para esta análise, foi constatado que a manobra não pode ser realizada com um ângulo de esterço maior que 20º, porque o robô passa a não se comportar como deveria. A trajetória descrita pelo CM (centro de massa) do robô durante a análise pode ser vista na Figura 69. Os resultados obtidos nesta análise podem ser vistos na Tabela 5**.**

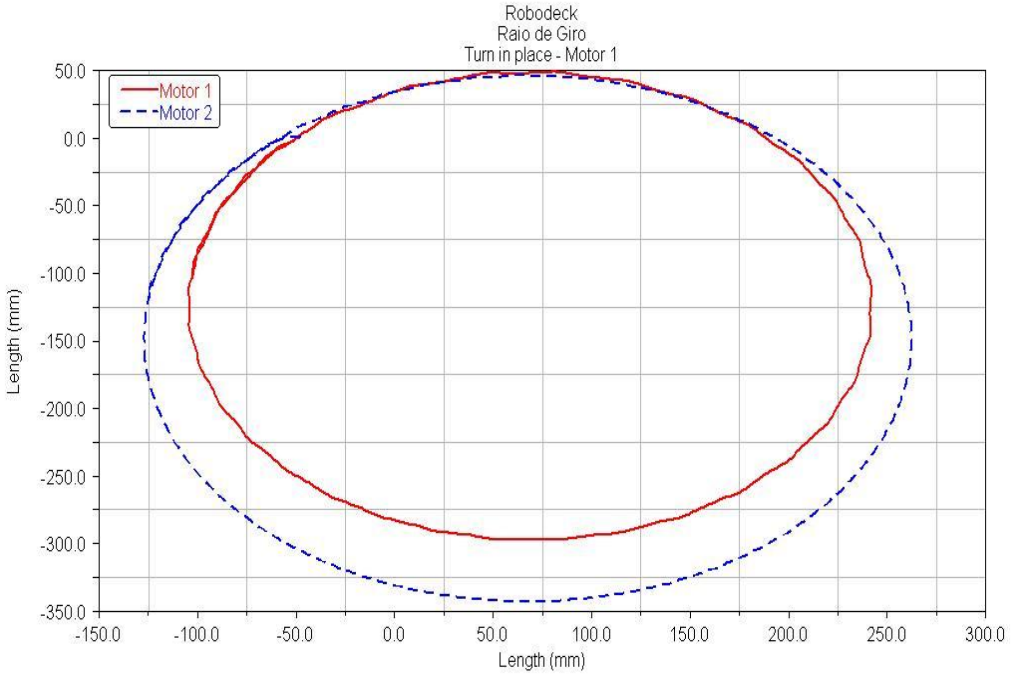

Figura 69: Trajetória descrita pelo robô na análise de *Turn-in-place* [01].

Tabela 5 : Tabela de resultados da análise de Turn in place [01].

		Motor 1	Motor 2
Raio executado (mm)		194	173
Torque aplicado (N.mm)	Direita	203	74
	Esquerda	199	75

6.2. MODELAGEM MATEMÁTICA NÃO LINEAR [01]

Para a sintetização do controle do Robô para várias condições de velocidade longitudinal foi desenvolvido um modelo matemático não linear do Robô. Esse modelo não linear possui dois graus de liberdade (deslocamento lateral e ângulo de guinada) e ele pode ser utilizado para condições variáveis de velocidade longitudinal do robô.

O modelo foi elaborado na forma de equações dinâmicas do movimento e implementado no ambiente do *software* Matlab/Simulink.

O objetivo deste trabalho é o equacionamento e implementação das equações dinâmicas do movimento e da geometria de Ackermannn nas rodas dianteiras e traseiras do robô para ser utilizado na sintetização do controle do robô para uma condição variável de velocidade longitudinal.

6.2.1. Modelo de dois graus de liberdade

A Figura 70**,** ilustra os o modelo de dois graus de liberdade do robô. Os graus de liberdade são deslocamento lateral e ângulo de guinada.

As variáveis de movimento que descrevem a atitude do modelo de dois graus de liberdade do veículo são:

- ângulo de escorregamento lateral (deriva) β ;

- velocidade em guinada *r*;

Para pequenos ângulos (pequeno escorregamento lateral), tem-se que o ângulo de deriva do robô é dado por:

$$\beta = v/V$$

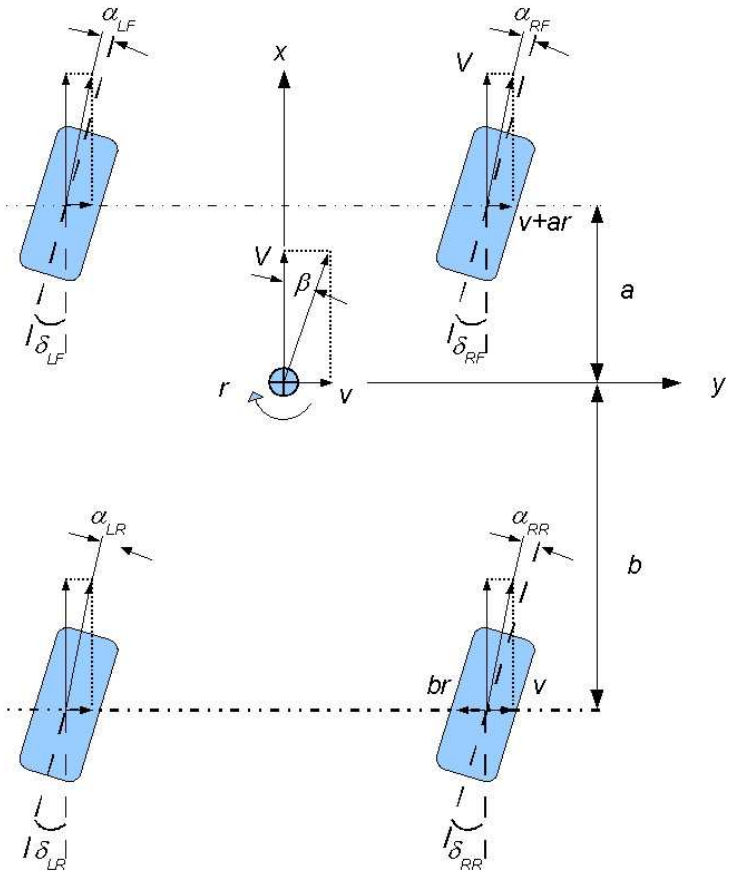

Figura 70: Modelo de dois graus de liberdade do Robô [01].

Os ângulos de deriva dos pneus dianteiros e traseiros são dados por:

$$\beta = v / V$$

$$\alpha_{LF} = \beta + \frac{ar}{V} - \delta_{LF}$$

$$\alpha_{RF} = \beta + \frac{ar}{V} - \delta_{RF}$$

$$\alpha_{LR} = \beta - \frac{br}{V} - \delta_{LR}$$

$$\alpha_{RR} = \beta - \frac{br}{V} - \delta_{RR}$$

Em que:

α_{LF} = ângulo de deriva do pneu dianteiro esquerdo;

α_{RF} = ângulo de deriva do pneu dianteiro direito;

α_{LR} = ângulo de deriva do pneu traseiro esquerdo;

α_{RR} = ângulo de deriva do pneu traseiro direito;

β = ângulo de deriva do robô;

a = distância do centro das rodas dianteiras ao centro de massa do robô;

b = distância do centro das rodas traseira ao centro de massa do robô;

V = velocidade longitudinal do robô;

δ_{LF} = ângulo de esterço do pneu dianteiro esquerdo;

δ_{LR} = ângulo de esterço do pneu dianteiro direito;

δ_{LR} = ângulo de esterço do pneu traseiro esquerdo;

δ_{RR} = ângulo de esterço do pneu traseiro direito.

Fazendo a somatória das forças laterais, temos:

$$\sum Y = \alpha_{LF} C_\alpha + \alpha_{RF} C_\alpha + \alpha_{LR} C_\alpha + \alpha_{RR} C_\alpha = M(\dot{v} + Vr)$$

Em que:

$\sum Y$ = somatória das forças laterais;

C_α = rigidez em curva (*cornering stiffness*) do pneu;

M = Massa do robô.

Fazendo a somatória dos momentos em guinada, temos:

$$\sum N = a(\alpha_{LF} C_\alpha + \alpha_{RF} C_\alpha) - b(\alpha_{LR} C_\alpha + \alpha_{RR} C_\alpha) = I_z \dot{r}$$

Em que:

$\sum N$ = somatória dos momentos em torno do eixo vertical do robô (guinada);

I_z = momento de inércia em torno do eixo vertical do robô (guinada).

Substituindo os valores dos ângulos de deriva nas equações, obtemos as seguintes expressões:

$$\sum Y = 4C_\alpha \beta + 2\frac{C_\alpha}{V}(a - b)r - C_\alpha(\delta_{LF} + \delta_{RF} + \delta_{LR} + \delta_{RR})$$

$$\sum N = 2C_\alpha(a - b)\beta + 2\frac{C_\alpha}{V}(a^2 + b^2)r - aC_\alpha(\delta_{LF} + \delta_{RF} + \delta_{LR} + \delta_{RR})$$

Fazendo a seguinte substituição de variáveis para a somatória de forças:

$$Y_\beta = 4C_\alpha$$

$$Y_r = 2\frac{C_\alpha}{V}(a - b)$$

$$Y_{\delta_{LF}} = -C_\alpha$$

$$Y_{\delta_{RF}} = -C_\alpha$$

$$Y_{\delta_{LR}} = -C_\alpha$$

$$Y_{\delta_{RR}} = -C_\alpha$$

Realizando o mesmo procedimento com relação à somatória de momentos:

$$N_\beta = 2C_\alpha (a - b)$$
$$N_r = 2\frac{C_\alpha}{V}\left(a^2 + b^2\right)$$
$$N_{\delta_{LF}} = -aC_\alpha$$
$$N_{\delta_{RF}} = -aC_\alpha$$
$$N_{\delta_{LR}} = bC_\alpha$$
$$N_{\delta_{RR}} = bC_\alpha$$

As somatórias das forças laterais e dos momentos podem ser expressas como:

$$\sum Y = Y_\beta \beta + Y_r r + Y_{\delta_{LF,RF,LR,RR}} \delta_{LF,RF,LR,RR} = M(\dot{v} + Vr)$$

$$\sum N = N_\beta \beta + N_r r + N_{\delta_{LF,RF,LR,RR}} \delta_{LF,RF,LR,RR} = I_z \dot{r}$$

Isolando os termos $\dot{r}$ e $\dot{v}$ nas equações acima, obtemos:

$$\dot{r} = \frac{N_\beta}{VI_z} v + \frac{N_r}{I_z} r + \frac{N_{\delta_{LF,RF,LR,RR}}}{I_z} \delta_{LF,RF,LR,RR}$$

$$\dot{v} = \frac{Y_\beta}{VM} v + \frac{Y_r}{M} + \frac{Y_{\delta_{LF,RF,LR,RR}}}{M} \delta_{LF,RF,LR,RR}$$

A determinação do deslocamento lateral e longitudinal do robô no sistema de coordenadas inercial pode ser feita através de uma matriz de transformação do referencial local do robô para o referencial inercial:

$$T^{R,I} = \begin{bmatrix} \cos r & \sin r \\ -\sin r & \cos r \end{bmatrix}$$

$$\gamma = \int r dt$$

Em que (ângulo de guinada)

A velocidade lateral e longitudinal do robô no referencial inercial são dadas pelas seguintes equações:

$$V^I = -v\sin\gamma + V\cos\gamma$$
$$v^I = v\cos\gamma + V\sin\gamma$$

O deslocamento lateral e longitudinal no referencial inercial podem ser calculados através da integração das equações acima:

$$y^I = \int v^I dt$$
$$x^I = \int V^I dt$$

6.2.2. Esterço – Geometria de Ackermann

Para o esterço deve-se aplicar uma rotação em cada servo com o ângulo desejado para cada roda. Para a modelagem do esterçamento das rodas, foi adotada a Geometria de Ackermann, que através de análises cinemáticas, proporciona as relações entre os ângulos de esterço das rodas internas com os ângulos das rodas externas, assumindo a hipótese de não escorregamento, o que é válido para baixas velocidades e, portanto, para o robô analisado.

Na Figura 71**,** a seguir, pode-se observar a configuração do esterço das rodas de um veículo segundo a Geometria de Ackermann.

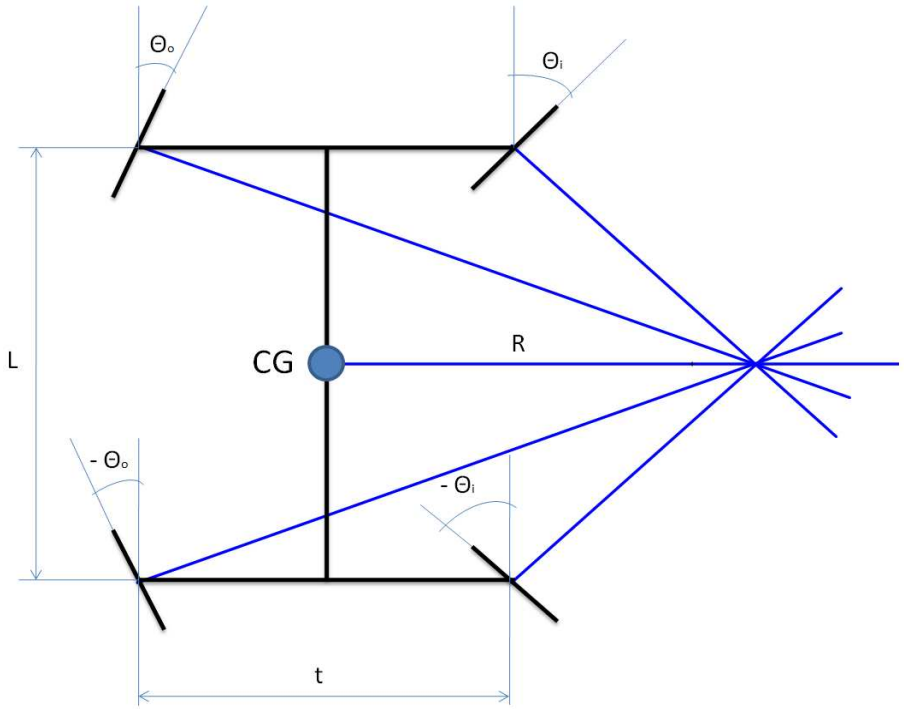

Figura 71: Geometria de Ackermann [01].

A partir da configuração acima e, considerando que as rodas externas ao movimento têm o mesmo esterço (Θ_o), assim como as internas (Θ_i), para que não haja escorregamento lateral, as rodas internas ao movimento devem apresentar um esterço maior que o das rodas externas. A partir da configuração acima, deduz-se que o esterço que cada roda deve apresentar é dado pelas seguintes equações:

$$\theta_i = \arctan\left(\frac{L}{2R - t}\right)$$

$$\theta_o = \arctan\left(\frac{L}{2R + t}\right)$$

Onde:

R = Raio da manobra;

L = Distância entre-eixos do robô;

t = Bitola do Robô.

A partir das duas equações acima, pode-se obter uma relação entre o esterço externo e o interno das rodas, de tal modo a garantir que não ocorra escorregamento lateral, dado por:

$$\Theta_i = \arctan\left(\frac{L * \tan(\Theta_o)}{L - 2 * t * \tan(\Theta_o)}\right)$$

Para que não ocorra escorregamento na direção longitudinal, os motores devem aplicar torques de tal forma que as rotações das rodas obedeçam as seguintes relações:

$$\omega_i = \frac{V_i}{r}$$

$$\omega_o = \frac{V_o}{r}$$

Onde:

ω_i = Rotação da roda interna ao movimento;

ω_o = Rotação da roda externa ao movimento;

V_i = Velocidade longitudinal do CG (centro de gravidade) da roda interna ao movimento;

V_o = Velocidade longitudinal do CG da roda externa ao movimento;

r = raio da roda.

Através de relações cinemáticas, chega-se às seguintes expressões para V_i e V_o:

$$V_i = V_{CG} \cdot \left[1 - \frac{t}{2 \cdot R}\right]$$

$$V_o = V_{CG} \cdot \left[1 + \frac{t}{2 \cdot R}\right]$$

Onde V_{CG} é a velocidade longitudinal do robô e R pode ser determinado em função do esterço aplicado às rodas externas ao movimento:

$$R = \frac{L - t \cdot \tan(\theta_o)}{2 \cdot \tan(\theta_o)}$$

Através da implementação das correlações acima na modelagem, garantiu-se o não escorregamento nas direções lateral e longitudinal.

Portanto, dado um determinado raio de curvatura nas rodas dianteira e traseira o modelo no Matlab/Simulink irá calcular o ângulo de esterço equivalente nas rodas internas e externas à trajetória em curva.

A parametrização do modelo foi realizada através de um programa script no Matlab. Esse script é utilizado para configurar os parâmetros do robô. A listagem do programa script pode ser visto abaixo [01]:

```
Script Matlab model_2dof_vars.m
%------------------Parâmetros do veículo
%Cornering stiffness dos pneus dianteiros e traseiros
Calpha=-1000;   %[N/rad]
%----------------Cornering stiffness com atraso
%Distancia do eixo dianteiro ate o cm do veículo
a=116/1000;     %[m]
%Distancia da eixo traseiro ate o cm do veículo
b=335/1000;     %[m]
%Massa do veículo
M=16.9;     %[kg]
% Distancia entre eixos
L=0.42;  %[m]
% Bitola
T=0.364;  %[m]
%Velocidade longitudinal do veículo
V=0.4;    %[m/s]
cor='g';
%Momento de inércia em z
Izz=0.65;   %[kg.m^2]
```

```
%------------------Fatores de estabilidade derivativas
%Força lateral devido ao angulo de escorregamento lateral do veículo
%Constante de amortecimento
Yb=4*Calpha;
%Força lateral devido a velocidade em guinada
Yr=(2/V)*Calpha*(a-b);
%Força lateral devido ao ângulo de esterço
Ydlf=-Calpha;
Ydrf=-Calpha;
Ydlr=-Calpha;
Ydrr=-Calpha;
%Momento em guinada devido ao angulo de escorregamento do veículo
%Constante torcional de mola do veículo
Nb=2*Calpha*(a-b);
%Momento em guinada devido a velocidade em guinada
%Amortecimento em guinada
Nr=(2/V)*Calpha*((a^2)+(b^2));
%Momento em guinada devido ao angulo de esterço
%sensibilidade de controle
Ndlf=-a*Calpha;
Ndrf=-a*Calpha;
Ndlr=b*Calpha;
Ndrr=b*Calpha;
```

Através do modelo no ambiente do Matlab/Simulink é possível implementar um controle de trajetória do robô através do cálculo do raio de curvatura em cada instante da trajetória e , com isso, manter o robô na trajetória desejada através da implementação de um controle do tipo PID (Proporcional, Integral e Derivativo).

6.2.3. Modelagem das Equações Dinâmicas e Cinemáticas do Robô no Matlab/Simulink

A Figura 72, mostra o modelo completo que simula o comportamento do Robô, no ambiente do *software* Matlab/Simulink. O modelo está totalmente parametrizado com relação às dimensões do robô, valores de rigidez longitudinal e em curva do pneu, raio das rodas, massa e momento de inércia, distância entre eixos e bitola.

O modelo implementa os controles de trajetória e velocidade do robô através dos cálculos do raio de curvatura e velocidade longitudinal em cada instante da trajetória e, com isso, o mantém na trajetória desejada e com a velocidade desejada, por meio de controladores PID (Proporcional, Integral e Derivativo).

A Figura 73 ,mostra o interior do bloco *Modelo Robô*, que implementa as equações dinâmicas de acelerações longitudinal, lateral e em guinada, e por meio de integrações, calcula as velocidades longitudinal, lateral e de guinada. O bloco também calcula o ângulo de guinada e os deslocamentos lateral e longitudinal do robô no referencial inercial.

As entradas do bloco *Modelo Robô* são os ângulos de esterçamento e as velocidades angulares das 4 rodas, e as saídas são deslocamento lateral, deslocamento longitudinal, ângulo de guinada e a velocidade longitudinal do robô.

No modelo da Figura 72 , os ângulos de esterçamento e as rotações das 4 rodas são calculados de acordo com a geometria de Ackermann.

Para o cálculo do raio de curvatura descrito pelo robô a partir dos seus deslocamentos lateral e longitudinal, é utilizada a expressão abaixo, originada a partir do equacionamento do triedro de *Frenet* (referência), que calcula a curvatura de uma curva a partir das posições *x* e *y* em cada instante de tempo:

$$\kappa = \frac{\left\| \dot{\gamma} \wedge \ddot{\gamma} \right\|}{\left\| \dot{\gamma} \right\|^3}$$

Em que:

κ = curvatura da trajetória

$\gamma(t) = (x(t), y(t))$ = vetor posição no instante t

$\| \ \|$ = operador norma

$\wedge$ = operador produto vetorial

O raio de curvatura é calculado a partir de:

$$R = \frac{1}{\kappa}$$

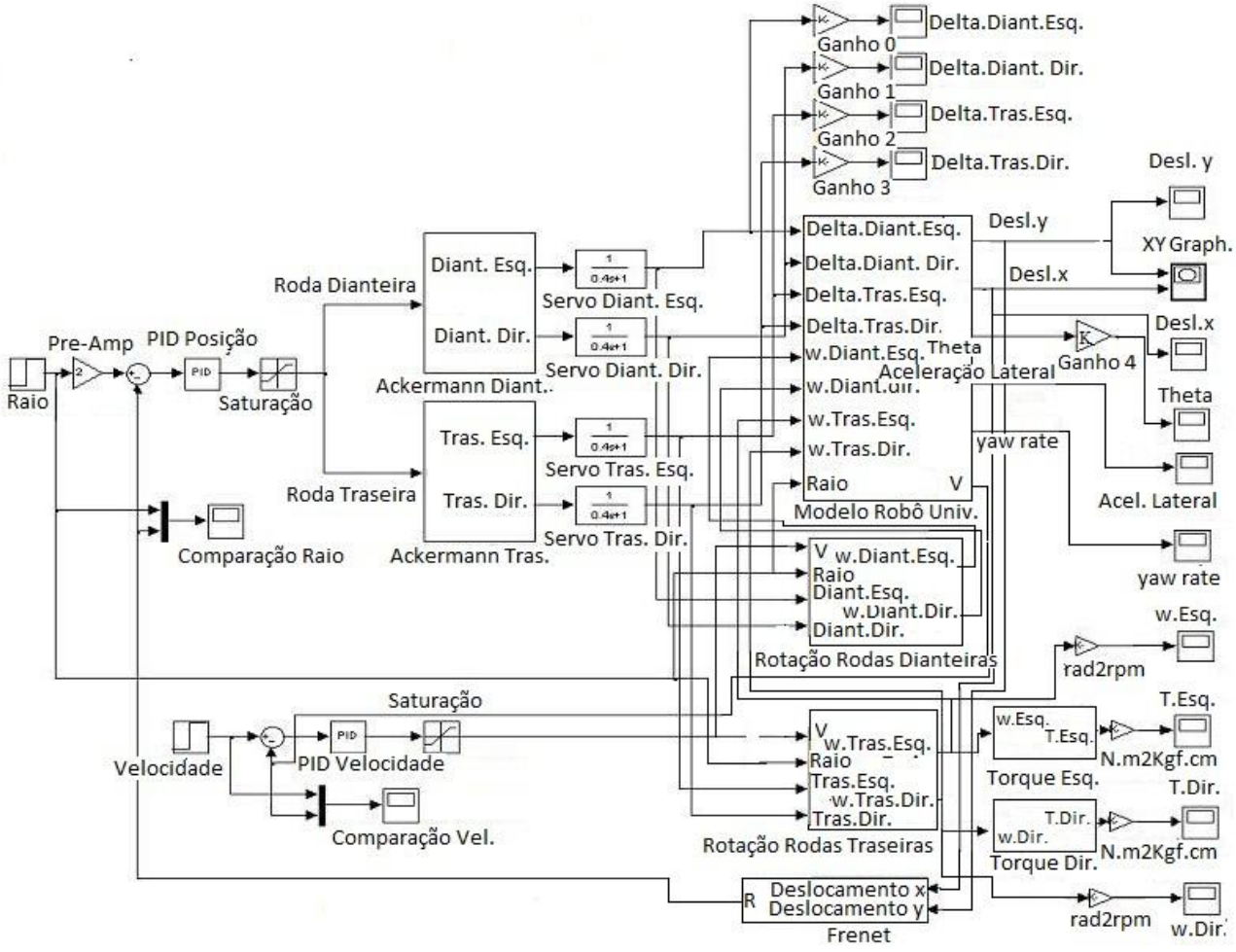

Figura 72:Modelo em forma de equações dinâmicas no Matlab/Simulink [01].

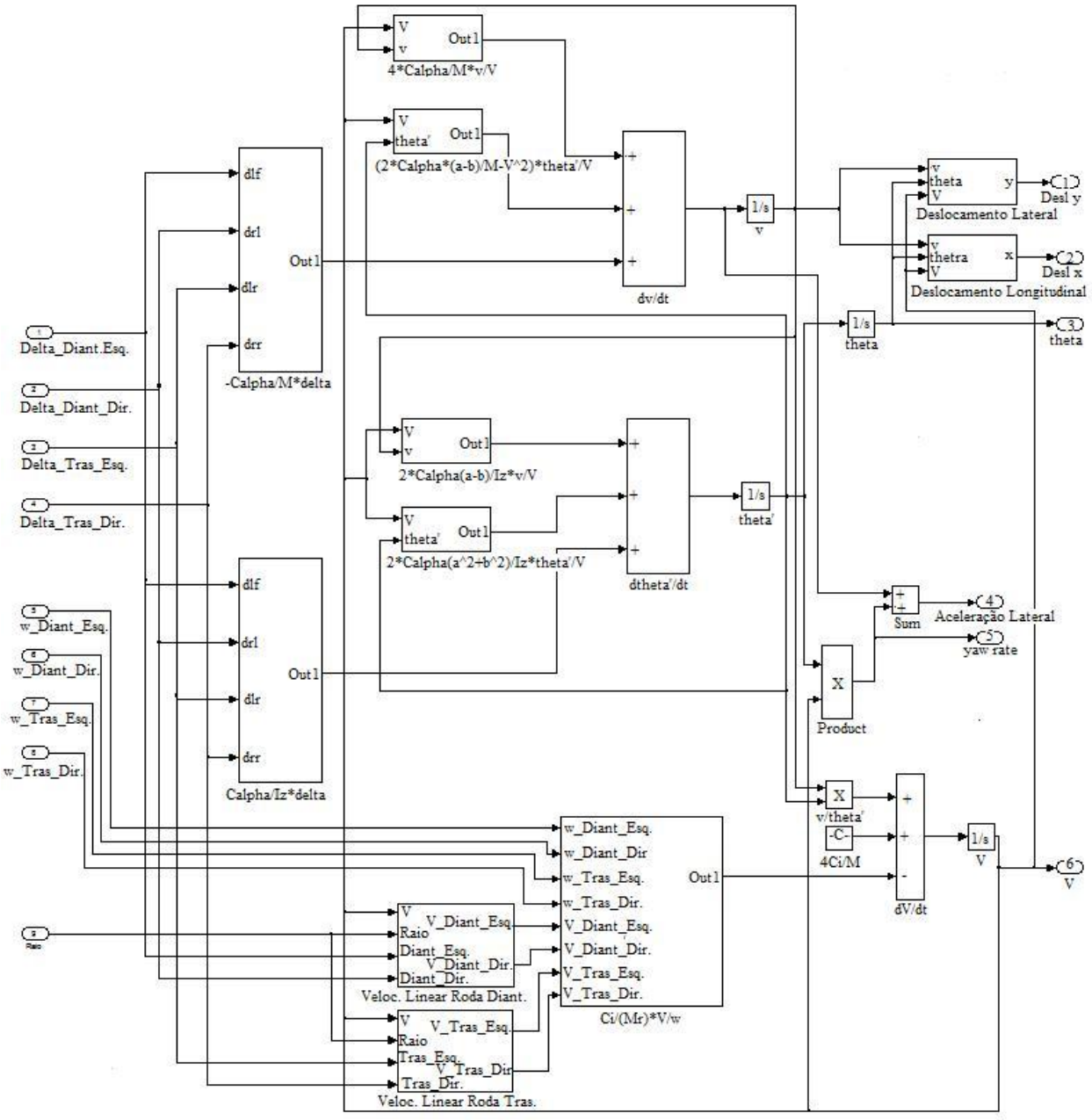

Figura 73:Subsistema no Matlab/Simulink com as equações dinâmicas do Robô [01].

6.2.4. Resultados de Simulações

A Tabela 6**,** mostra os valores dos parâmetros mecânicos do Robô utilizados na simulação no Matlab/Simulink.

Tabela 6: Parâmetros mecânicos [01].

Parâmetros	**Valor**	**Unidade**
C_i = Rigidez longitudinal dos pneus	1	**N**
$C\alpha$ = Rigidez em curva dos pneus	-1000	N/rad
r = Raio (rodas+pneus)	7,5	cm
a = Distância do eixo dianteiro ao centro de massa do veículo	11,6	cm
b = Distância do eixo traseiro ao centro de massa do veículo	33,5	cm
M = Massa do veículo	16,9	Kg
L = Distância entre eixos do veículo	42	cm
t = Bitola do veículo	36,4	cm
Iz = Momento de inércia em torno do eixo z	0,65	Kg.m2

Para esta simulação, o raio de curvatura desejado foi uma função degrau de amplitude 2m, conforme pode ser observado na Figura 74. Na Figura 75, pode ser observada a função de entrada de velocidade desejada.

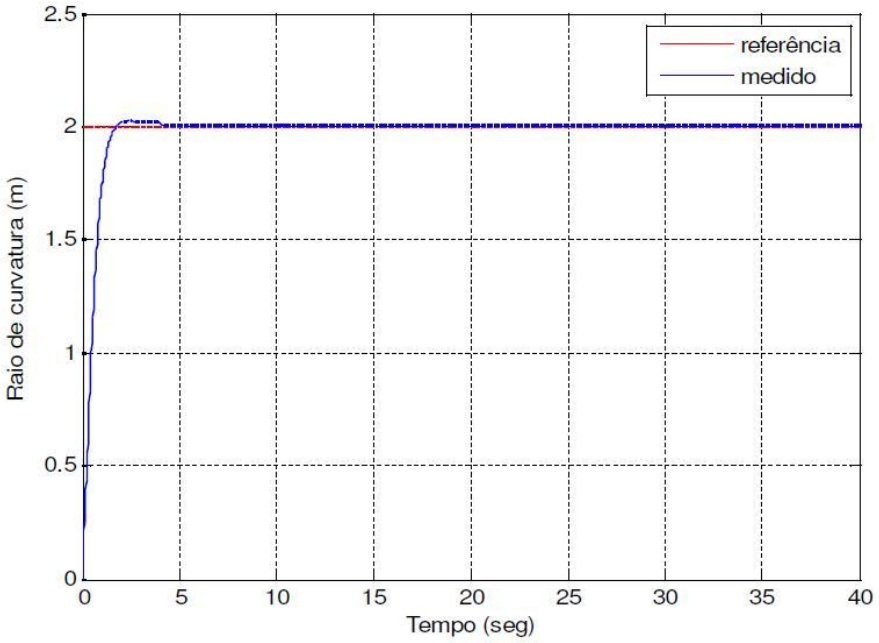

Figura 74: Resposta ao degrau do sistema de controle de posição [01].

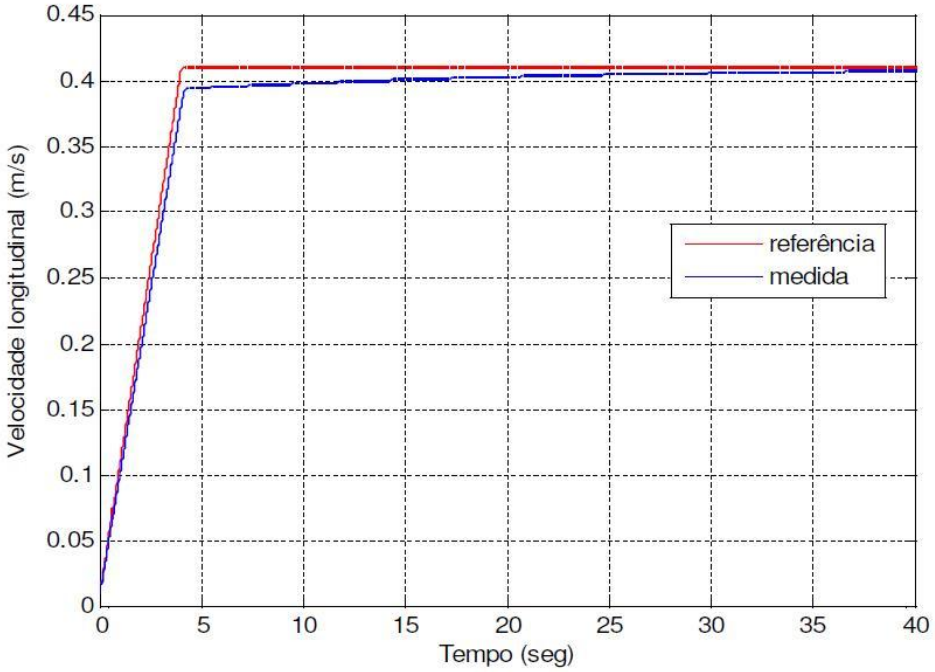

Figura 75: Resposta do sistema de controle de velocidade [01].

Como pode ser visto nas Figura 75, com o devido ajuste dos controladores PID de posição e velocidade, o modelo do Robô respondeu de forma rápida, sem oscilações transitórias e com pequeno erro de estado permanente às funções de entrada de raio de curvatura e velocidade.

A Figura 76, mostra a trajetória percorrida pelo Robô para o raio de curvatura e velocidade desejado. Como se pode ver no gráfico, a trajetória é uma circunferência de raio 2m. As Figura 77 e Figura 78, mostram, respectivamente, os deslocamentos longitudinal e lateral em função do tempo. Com o raio de curvatura positivo, o robô parte da posição (0,0) e percorre o sentido horário da circunferência, ou seja, o ângulo de guinada aumenta de forma linear, como pode ser observado na Figura 79.

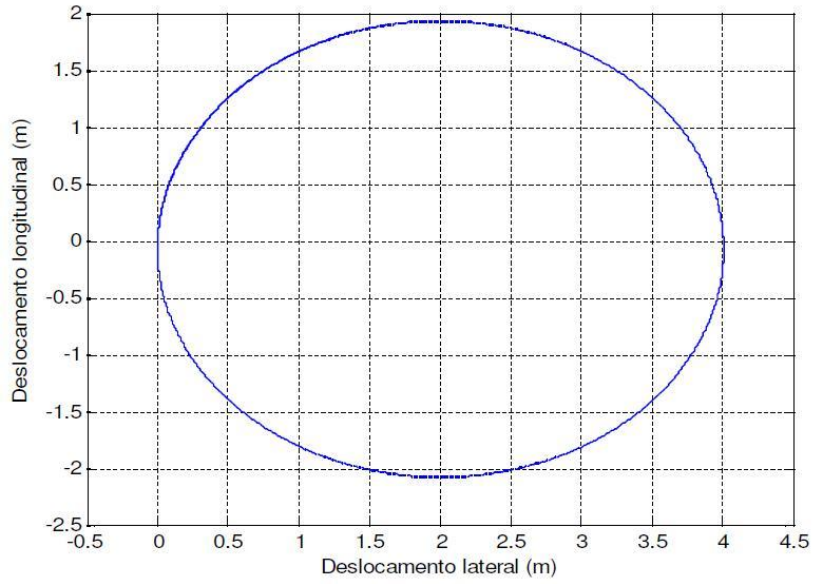

Figura 76: Trajetória percorrida pelo Robô para o raio de curvatura e velocidade desejados [01].

A Figura 80 mostra os ângulos de esterçamento das 4 rodas. Como pode ser observado, os ângulos de esterçamento das rodas internas à curva (rodas direitas) são maiores do que das rodas externas à curva (rodas esquerdas), estando assim o modelo de acordo com a Geometria de Ackermann, de forma que não ocorra escorregamento lateral do veículo.

A Figura 81, mostra as velocidades angulares das rodas de tração do Robô (rodas traseiras). Como pode ser observado, a rotação da roda externa à curva (roda esquerda) é maior do que a

rotação da roda interna à curva (roda direita), evitando assim o escorregamento longitudinal do veículo.

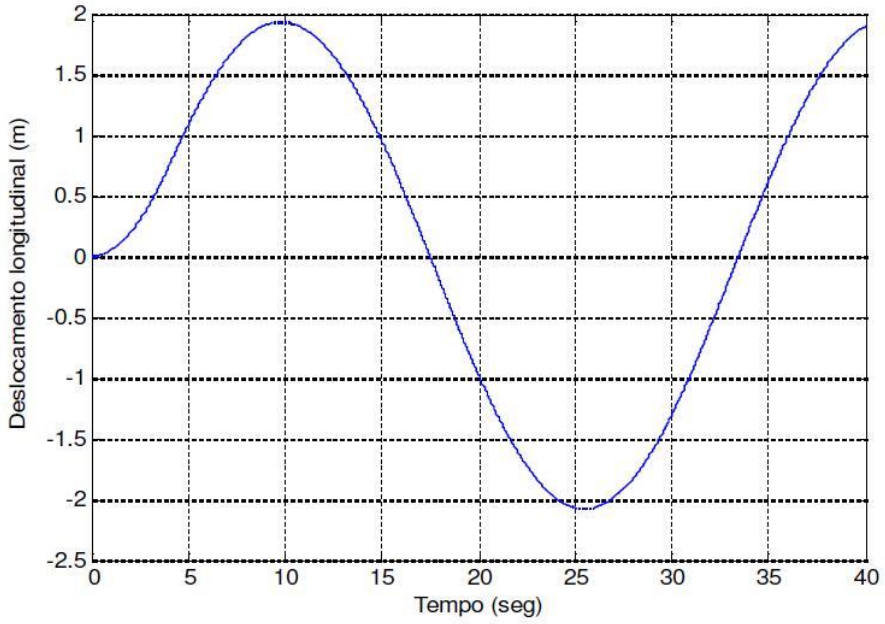

Figura 77: Deslocamento longitudinal do Robô [01].

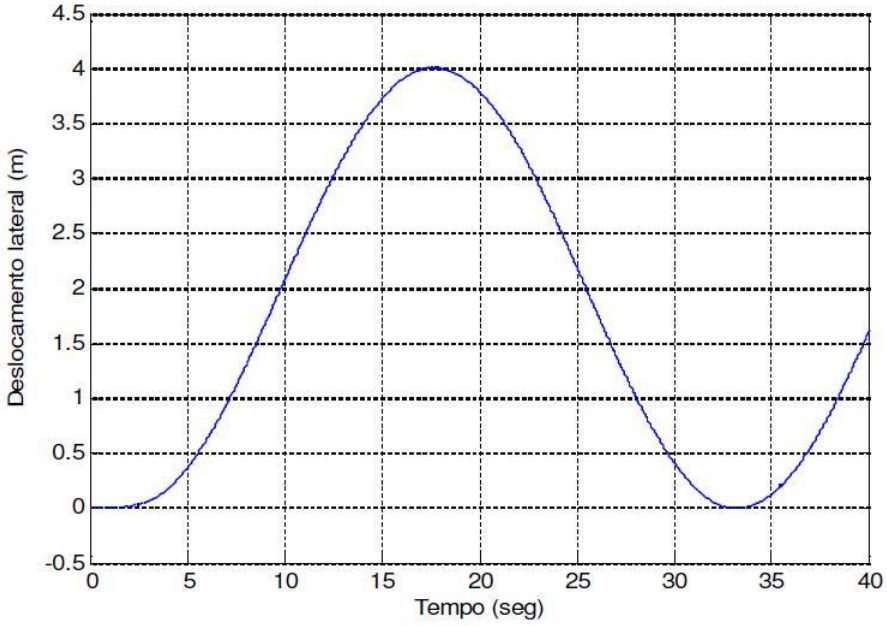

Figura 78: Deslocamento lateral do Robô [01].

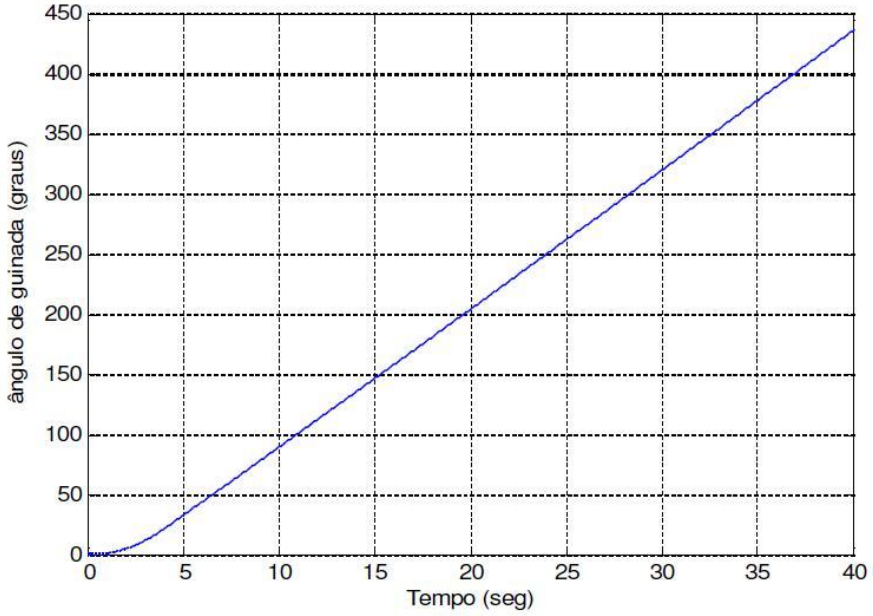

Figura 79: Ângulo de guinada do Robô [01].

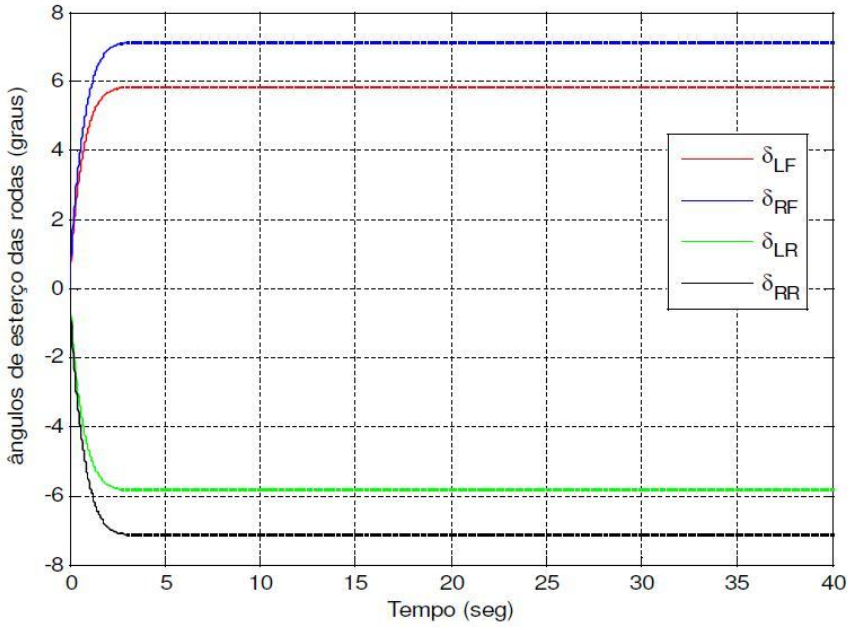

Figura 80: Ângulos de esterçamento das 4 rodas [01].

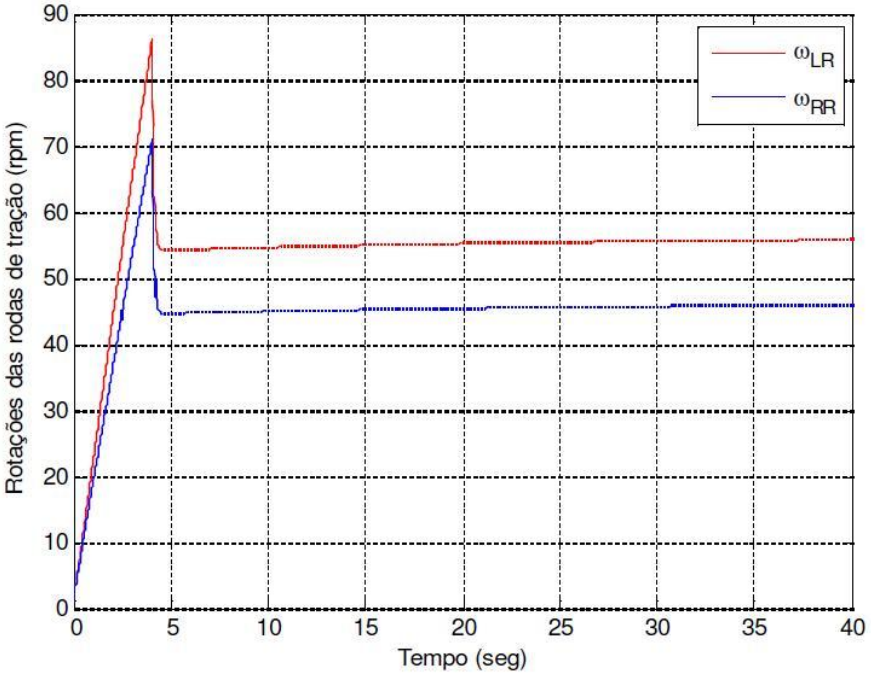

Figura 81: Rotações das rodas de tração (traseiras) [01].

CAPÍTULO 7

VALIDAÇÃO DO PROJETO

7.1. VALIDAÇÃO DO *SOFTWARE* EMBARCADO

Para a validação do *software* embarcado foi desenvolvido um *software* para Windows XP chamado de Giga de Testes [01].

Giga de Testes é um *software* que tem como propósito validar cada componente do robô. Nisto incluem-se todos os motores e todos os sensores. Nestes testes incluem-se também os comandos de alto nível (responsáveis pelo acionamento coordenado dos motores), os comandos de controle de sessão e os comandos associados aos módulos de expansão. Este Giga de Testes, não inclui componentes que dependam do Módulo de Alta Performance, como a câmera e os meios de comunicação por Bluetooth e WiFi. Portanto este *software* trabalha somente com a rede ZigBee e com os componentes que possam ser acionados através desta rede. O Giga de Testes, adicionalmente, testa a própria comunicação através da rede ZigBee.

Em suma, este *software* testa:

• A comunicação através da rede ZigBee.

• Os comandos de controle de sessão: abrir e fechar sessão.

• Os comandos de baixo nível associados às rodas do robô: ligar/desligar motores de tração e de direção; acionar motores de tração e de direção; e adquiri valores relacionados aos *encoders* dos motores de tração.

• Os comandos de alto nível associados à movimentação do robô: mover (*move*), fazer curva (*turn*), andar lateralmente (*strafe*) e girar em torno do próprio eixo (*roll*).

• Todos os dispositivos de sensoriamento do robô: sensores infravermelho, sensores ultrasom, sensores de colisão, bússola, acelerômetro, medidor de temperatura e umidade, e GPS.

• Os comandos associados aos módulos de expansão.

Ou seja, o Giga de Testes engloba todos os comandos existentes no Protocolo Externo de Comunicação pertencente aos grupos “Comandos de Sessão” (iniciados por *01*), “Comandos de Baixo Nível “ (iniciados por *02*), “Comandos de Alto Nível” (iniciados por *03*) e “Comandos dos Slots” (módulos de expansão, iniciados por *05*). Uma exceção poderá ser aberta ao comando “*Session Ended”*, que é enviado pelo robô ao controlador.

O *Software*

O Giga de Testes é desenvolvido para Windows XP (e compatíveis) e faz intenso uso dos *Windows Forms*. Foi adotada a linguagem C# para sua implementação, visto que esta linguagem é nativamente direcionada ao *.NET Framework*, o qual dá amplo suporte aos *Windows Forms*. Foi também considerado o emprego da linguagem C++, mas esta não é suportada pelo *.NET Framework*, sendo necessário o emprego desta linguagem em sua forma estendida (C++/CLI). Uma vez que tal extensão foi concebida apenas para adequar o C++ ao *.NET Framework*, optei por utilizar a linguagem que já foi elaborada com este propósito: C#. O Giga de Testes foi construído sobre o RoboDeckSDK. O propósito deste SDK é oferecer toda a infra-estrutura para o desenvolvimento de aplicações, dentro do ambiente Windows, para o controle do robô. Portanto torna-se adequado seu emprego para a criação do Giga de Testes. O RoboDeckSDK foi implementado em C++/CLI, tornando-se compatível com aplicações a serem elaboradas usando qualquer linguagem direcionada ao *.NET Framework*, o que inclui C#. Entretanto, algumas adequações foi realizadas no RoboDeckSDK, pois ele não oferece um meio de se interceptar as mensagens trocadas com o robô. O Giga de Testes tem acesso a tal troca de mensagens. Então foi adicionado ao SDK recursos que o possibilitem prover acesso a estas trocas de mensagens, não só conforme o Protocolo de Comunicação Externa do Robô, como conforme o protocolo da rede utilizada (atualmente, ZigBee).

7.1.1. Módulo de Controle de Sessão

Utilizando o Giga de Testes, se procedeu a abrir a sessão e fechar a sessão inúmeras vezes durante todos os testes, verificando seu perfeito funcionamento.

7.1.2. Módulo de Controle Robótico

A tabela mostra os comandos aplicados.

Tabela 7: Resultado dos testes [01].

Comandos	Número de comandos	Resultado	Observações
Ligar / desligar motores de tração.	25	100%	
Acionar motores de tração.	25	100%	
Ligar / desligar motores de direção.	25	100%	
Acionar motores de direção.	25	100%	
Obter a quantidade de giros de cada roda.	25	100%	
Mover o robô. Muda a velocidade do robô sem alterar sua "curvatura". Portanto atua somente nos motores de tração, não nos motores de direção.	25	100%	
Fazer curva. Muda a "curvatura" do robô em alterar sua velocidade. Deve atuar tanto nos motores de direção quanto nos de tração.	25	100%	
Mover diagonalmente o robô. Muda a direção do movimento do robô sem alterar sua velocidade nem sua orientação. Atua somente nos motores de direção.	25	100%	

Girar o robô. Gira o robô em torno de seu eixo. Atua nos motores de tração e direção.	25	100%	
Frear o robô. Trava todos os motores de tração e de direção do robô. Este travamento é obtido ligando-se os motores para que eles ofereçam resistência ao movimento.	25	100%	
Ler sensores infra-vermelho.	50	100%	
Ler distância com sensores ultrasom.	50	100%	
Ler luminosidade com sensores ultra-som.	25	100%	
Verificar sensores de colisão (ópticos).	25	100%	
Ler a bússola.	25	100%	
Ler o acelerômetro.	25	100%	
Ler sensor de temperatura e umidade.	25	100%	
Testar a validade de dados provenientes do GPS. Com base no número de satélites detectados, o GPS oferece uma forma de verificar se os dados a serem lidos são válidos ou não.	25	100%	
Obter localização com o GPS.	25	100%	
Obter data e horário com o GPS.	25	100%	
Obter a velocidade (e sua direção) do robô através do GPS. O GPS oferece meios de obter a velocidade linear do robô. Oferece também meios para obter a direção (em relação ao norte magnético) do deslocamento.	25	100%	
Verificar estado das baterias.	25	100%	

7.1.3. Módulo de Controle de Comunicação

Os módulos de comunicação que foram testados são:

- Comunicação via ZigBee testado no item anterior utilizando o giga de Testes
- Comunicação via Bluetooth enviando comandos diretamente de um terminal de comunicação utilizando um módulo de comunicação Bluetooth na USB do computador.
- Comunicação via WiFi enviando comandos diretamente de um terminal de comunicação utilizando um módulo de comunicação WiFi na USB do computador.

Tabela 8: Comandos do Módulo de Alta Performance [01].

Nome do Comandos	Comando	Resposta	Observações
SessionOpen	01 00 01 44 32 ae 00 00 01 05 01 01 45 75 00 e8	04 01 32 ae 01 44 e3 18 03 04 81 01 e3 18 a9	Aprovado
SessionClose	03 00 01 44 32 ae e3 18 01 02 01 02 29	06 03 32 ae 01 44 e3 18 03 03 81 02 d3	Aprovado
SessionEnded		06 03 32 ae 01 44 e3 18 03 03 81 03 b3	Aprovado
MotorDirOnOff Servo motores frente-direita	02 00 01 44 32 ae e3 18 01 04 02 01 00 ff 29	08 02 32 ae 01 44 e3 18 03 03 82 01 ff b2	Aprovado
MotorDirOnOff Servo motores frente-esquerda	05 00 01 44 32 ae e3 18 01 04 02 01 01 00 2d	0a 05 32 ae 01 44 e3 18 03 03 82 01 00 b8	Aprovado
MotorDirOnOff Servo motores trás-direita	07 00 01 44 32 ae e3 18 01 04 02 01 03 00 32	0c 07 32 ae 01 44 e3 18 03 03 82 01 00 bc	Aprovado
MotorDirOnOff Servo motores trás-	09 00 01 44 32 ae e3 18 01 04 02 01 04 ff 34	0e 09 32 ae 01 44 e3 18 03 03 82 01 ff bf	Aprovado

esquerda			
MotorTurn	0b 00 01 44 32 ae e3 18 01 04 02 02 05 ff 38	10 0b 32 ae 01 44 e3 18 03 03 82 01 ff c3	Aprovado
MotorMove	11 00 01 44 32 ae e3 18 01 04 02 03 05 06 46	12 11 32 ae 01 44 e3 18 03 03 82 03 ff cd	Aprovado
MotorGetPeriod	13 00 01 44 32 ae e3 18 01 04 02 04 00 3e	14 13 32 ae 01 44 e3 18 03 04 82 04 04 0a e2	Aprovado
MotorGetEncoder	15 00 01 44 32 ae e3 18 01 04 02 05 ff 40 80	16 15 32 ae 01 44 e3 18 03 03 82 05 04 03 e0	Aprovado
MotorOnOff lado esquerdo	17 00 01 44 32 ae e3 18 01 04 02 06 00 44 88	18 17 32 ae 01 44 e3 18 03 03 82 06 ff dc	Aprovado
MotorOnOff lado direito	19 00 01 44 32 ae e3 18 01 04 02 06 ff 45 8a	1a 19 32 ae 01 44 e3 18 03 03 82 06 ff e0	Aprovado
RobotMove	1b 00 01 44 32 ae e3 18 01 05 03 01 00 44 22 ab	1c 1b 32 ae 01 44 e3 18 03 03 83 01 ff e0	Aprovado
RobotTurn	1d 00 01 44 32 ae e3 18 01 05 03 02 ff b4 21 1c	1e 1d 32 ae 01 44 e3 18 03 03 83 02 ff e5	Aprovado
ConfigGetVersion	1f 00 01 44 32 ae e3 18 01 02 00 00 42	20 1f 32 ae 01 44 e3 18 03 04 80 00 00 01 e7	Aprovado
ConfigHasMAP	21 00 01 44 32 ae e3 18 01 02 00 01 45	22 21 32 ae 01 44 e3 18 03 03 80 01 ff e9	Aprovado
ConfigSetIdentity	23 00 01 44 32 ae e3 18 01 08 00 02 12 aa ca Fe d2	24 23 32 ae 01 44 e3 18 03 03 80 02 ff ee	Aprovado

7.1.4. Módulo Robótico de Alta Performance

Este módulo foi testado no item anterior, pois a comunicação via Bluetooth e WiFi foram feitas via módulo de alta performance.

7.2. VALIDAÇÃO DO *SOFTWARE* DESKTOP/PDA

A validação do módulo Desktop/PDA foi feita no item 7.1 desde que foi utilizado este software para os testes correspondentes ao item, incluindo o SDK e o módulo de controle externo. Também foi validado pela equipe de software do grupo de São Carlos utilizando a norma internacional de IEEE P1061 "*Software Quality Metrics Methodology*" o que deu uma garantia da qualidade do software desenvolvido [028].

7.3. VALIDAÇÃO DO SISTEMA DE ALIMENTAÇÃO

7.3.1. Sistema de alimentação

Foi testada a autonomia utilizando seis baterias fazendo rodar o robô em círculo ate parar, dando uma autonomia de 5 horas e 15 minutos. Mínimo desejado era de 4 horas. Tempo de carregamento da batería é de duas horas cada uma.

7.4. VALIDAÇÃO DAS PLACAS ELETRÔNICAS

7.4.1. Esquemáticos dos Circuitos da Placa do Microcontrolador.

Realizaram-se todas as medições elétricas da placa do microcomtrolador, esta placa possui quatro camadas, sendo que a empresa que elaborou a placa fez todos os testes elétricos verificando a qualidade da placa. Depois de montada se realizaram os testes de funcionamento, medindo todas as tensões e sinais na placa. O resultado dos testes foi ótimo, somente foi encontrado um erro no layout, mas, foi possível solucionar com um simples *jumper*, sem causar deterioro na sua performance.

7.4.2. Esquemáticos dos Circuitos da Placa Mãe

Realizaram-se todas as medições elétricas da placa mãe, esta placa possui duas camadas, sendo que a empresa que elaborou a placa fez todos os testes elétricos verificando a qualidade da placa. Depois de montada se realizaram os testes de funcionamento, medindo todas as tensões e sinais na placa em todos seus circuitos. O resultado dos testes foi ótimo, não se encontrando nenhuma deficiência que reduza a performance do robô.

7.5. VALIDAÇÃO DO MODELO MECÂNICO

7.5.1. Simulação e Comportamento Dinâmico

7.5.1.1. Análise de Subida de Rampa

Foi colocada uma rampa de 30° e se acelerou o robô ate chegar a velocidade de 400 mm/s no plano percorrendo 500 mm e logo subiu a rampa ate chegar a 1000 mm de altura reduzindo sua velocidade ate aproximadamente 250 mm/s. Este valor comparado com a simulação feita teoricamente é um valor bem próximo do calculado (275 mm/s).

7.5.1.2. Análise de Passagem por Obstáculos

Teste de passagem por depressão conforme simulado teoricamente o robô deve superar estes obstáculos sem problemas.

Tabela 9: Testes de passagem por depressão [01].

	Robô com motor 1 (simulado)		Resultado	
	400 mm/s	600 mm/s	400 mm/s	600 mm/s
Profundidade: 15 mm Torque máx. (N.mm)	13,24	10,54	passou	passou
Profundidade: 25 mm Torque máx. (N.mm)	18,09	13,2	passou	passou

O teste mostrou que o robô tem torque suficiente para superar estes obstáculos.

7.5.1.3. Lombada

Teste de passagem por lombada conforme simulado teoricamente o robô deve superar estes obstáculos sem problemas.

Tabela 10: Testes de passagem por lombada [01].

	Robô com motor 1 (simulado)		Resultado	
	400 mm/s	600 mm/s	400 mm/s	600 mm/s
Altura: 10 mm Torque máx. (N.mm)	8,92	8,25	passou	passou
Altura: 25 mm Torque máx. (N.mm)	12,35	9,47	passou	passou

O teste mostrou que o robô tem torque suficiente para superar estes obstáculos.

7.5.1.4. Análise de Giro

7.5.1.4.1. Análise de Raio Constante

Teste de resposta a um giro de raio constante conforme cálculo teórico.

Tabela 11: Teste de resposta a um raio constante [01].

		Robô com motor 1 (simulado)	Resultado
Raio executado (mm)		406	400
Torque aplicado (N.mm)	Direita	6,26	passou
	Esquerda	5,73	passou

7.5.1.4.2. Análise de Turn-in-place

Teste de resposta a um giro sobre seu próprio eixo conforme cálculo teórico.

Tabela 12: Teste de resposta ao giro no próprio eixo [01].

		Robô com motor 1 (simulado)	Resultado
Raio executado (mm)		194	208
Torque aplicado (N.mm)	Direita	203	passou
	Esquerda	199	passou

Os resultados dos testes de validação mostraram que os cálculos teóricos foram cumpridos pelo protótipo montado validando o projeto, Alem destes testes foram feitos testes de durabilidade mostrando que o projeto mecânico tem robustez suficiente para aplicações de laboratório.

CAPÍTULO 8
CONCLUSÕES

8.1. CONCLUSÕES E TRABALHOS FUTUROS

Os resultados de validação comparados aos teóricos simulados no Adans/View e MatLab mostraram excelente acurácia e precisão que nos dá confiança na hora de utilizar o projeto desta plataforma para outras aplicações como: Utilização em veículos não tripulados terrestres, aquáticos ou aéreos, com fines de vigilância, segurança, mapeamentos de áreas, utilização em áreas perigosas e qualquer aplicação onde seja necessária a utilização de robótica móvel, modificando somente a camada HAL e implementando a biblioteca de comandos específicos para a nova aplicação.

A idéia de uma plataforma universal com código fonte aberto permitirá o desenvolvimento de uma quantidade muito grande de aplicações e o aperfeiçoamento constante como acontece com o sistema operacional Linux onde os profissionais desenvolvem aplicações e compartilham com outros profissionais no resto do mundo, criando uma sinergia que trás excelentes e impredizíveis resultados.

Como trabalho futuro agora se abre uma ampla possibilidade de aplicações na área robótica móvel utilizando esta plataforma, onde os pesquisadores poderão desenvolver uma série de aplicativos que poderão se testados nesta plataforma e uma série de outros robôs que utilizem como base o Robô Universal.

REFERÊNCIAS BIBLIOGRÁFICAS

[01] - Relatório interno do projeto RobotDeck. Protocolo de Comunicação Externa. Cientistas Associados (2009).

[02] - ZigBee ZigBee Specication. <site do ZigBee> (2009)

[03] - Norma brasileira NBR 11786 – Segurança do Brinquedo, publicada pela Associação Brasileira de Normas Técnicas (ABNT) e regulamentada pela Portaria Inmetro n.º 177, de 30 de novembro 1998.

[04] - K-Team . http://www.k-team.com/mobile-*Robotics*-products/khepera-iii (ano 2012). ECA..http://www.eca-*Robotics*.com/en/robotic-vehicle/*Robotics*-terrestrial-ugvs-cameleon-lab-develop-your-own-application/511.htm (ano 2012) International standard (2007) ISO 11898.

[05] - ZigBeeStack ZigBeeStack Documentation. <ZigBeeStack URL> (2009)

[06] - Jennic Jennic. <Jennic URL> (2009)

[07] - JN5139 JN5139 Datasheet. <JN5139 URL> (2009)

[08] - ZigBee Alliance, *http://www.caba.org/standard/ZigBee.html*.

[09] - LAN-MAN Standards Committee of the IEEE Computer Society, *Wireless LAN medium Access control(MAC) and physical layer(PHY) specification*, IEEE, New York, NY, USA, IEEE Std 802.11-1997 edition, 1997

[010] - LAN-MAN Standards Committee of the IEEE Computer Society, *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE, 2003

[011] - C. E. Perkins and E. M. Royer, *Ad Hoc On Demand Distance Vector Routing*

[012] - IEEE P802.15 Working Group for WPANs, *Cluster Tree Network*, April 2001 35

[013] - Java 2 Micro Edition (J2ME) (2012) - http://www.javaworld.com/channel_content/jw-j2me-index.shtml

[014] - Microsoft Robotics Developer *Studio* (2012) - http://www.microsoft.com/robotics/.

[015] - Visual Programming Lenguage - http://msdn.microsoft.com/en-us/library/bb483088.aspx

[016] - Electronic stability control coalition fact sheet". ESC Coalition and DEKRA Automotive Research 2003.

[017] - Fukada, Y., 1998. \Estimation of vehicle slip-angle with combination method of model observer and direct integration". *Proceedings of the International Symposium on Advanced Vehicle Control (AVEC), Nagoya, Japan*.

[018] - Van Zanten, A. T., 2002. \Evolution of electronic control systems for improving the vehicle dynamic behavior". *Proceedings of the International Symposium on Advanced Vehicle Control (AVEC), Hiroshima, Japan*.

[019] - Ono, E., Asano, K., and Koibuchi, K., 2003. Estimation of tire grip margin using electric power steering system". *Proceedings of the 18th International Association for Vehicle System Dynamics (IAVSD) Symposium, Kana gawa, Japan.*

[020] - Yasui, Y., Tanaka, W., Muragishi, Y., Ono, E., Momiyama, M., Katoh, H., Aizawa, H., and Imoto, Y., 2004. \Estimation of lateral grip margin based on self-aligning torque for vehicle dynamics enhancement". SAE Paper No.2004-01-1070.

[021] - Endo, M., Ogawa, K., and Kurishige, M., 2006. Cooperative control of active front steering and electric power steering based on self-aligning torque". *Proceedings of the International Symposium on Advanced Vehicle Control (AVEC),Taipei, Taiwan*.

[022] - Hsu, Y. H. J., and Gerdes, J. C., 2006. \A feel for the road: A method to estimate tire parameters using steering torque". *Proceedings of the International Symposium on Advanced Vehicle Control (AVEC), Taipei, Taiwan*.

[023] - Pasterkamp, W. R., and Pacejka, H. B., 1997. Application of neural networks in the estimation of tire/road friction using the tire as sensor". *SAE Paper No. 971122*.

[024] - Hsu, Y. H. J., Laws, S., Gadda, C. D., and Gerdes, J. C., 2006. \A method to estimate the friction coe±cient and tire slip angle using steering torque". *Proceedings of ASME International Mechanical Engineering Congress and Exposition (IMECE)*.

[025] - Pacejka, H. B., 2002. *Tire and Vehicle Dynamics*. Society of Automotive Engineers, Inc, 400 Commonwealth Dr. Warrendale, PA 15096-0001.

[026] - Laws, S., Gadda, C. D., Kohn, S., Yih, P.,Gerdes, J. C., and Milroy, J. C., 2005. \Steer-by-wire suspension and steering design for controllability and observability". *Proceedings of IFAC World Congress, Prague*.

[027] - Rock, K. L., Beiker, S. A., Laws, S., and Gerdes, J. C., 2005. \Validating gps based measurements for vehicle control". *Proceedings of ASME International Mechanical Engineering Congress and Exposition (IMECE)*.

[028] - IEEE P1061, Standard for Software Quality Metrics Methodology (produto de software)

APÊNDICES

APÊNDICE A.
PROTOCOLO DE COMUNICAÇÃO EXTERNA [01]

AP A.1 DEFINIÇÃO DO PROTOCOLO

Este documento tem por objetivo definir o protocolo de comunicação a ser utilizado por um agente externo para enviar comandos ao Robô Universal. Os comandos são enviados ao Robô Universal em pacotes de mensagens, tendo como camada de rede subjacente um dos seguintes protocolos de rede sem fio: *WiFi*, Bluetooth, ZigBee.

Quando um comando é recebido pelo Robô Universal, antes que ele possa ser executado é necessária a comunicação entre os módulos computacionais que compõem o robô e que executam em processadores distintos. Esta comunicação interna não será tratada neste documento.

Inicialmente será descrito o formato dos pacotes utilizados para o envio dos comandos. Em seguida serão descritos os formatos dos comandos e respostas aceitos pelo robô juntamente com a descrição de seus respectivos parâmetros. Por último serão descritas as formas estabelecimento de conexão com o robô utilizando-se cada um dos três protocolos subjacentes.

AP A.1.1 Pacotes

Os pacotes são envelopes para o envio das mensagens entre o controlador e o robô. Cada pacote armazena as informações necessárias as à troca de mensagens. Nas mensagens é que são identificados os comandos, respostas e parâmetros. As mensagens serão tratadas em AP A.1.1.3.

Os protocolos das redes subjacentes utilizados (*WiFi*, Bluetooth e ZigBee) já garantem a entrega do pacote, portanto os pacotes não serão providos de mecanismos para retransmissão.

AP A.1.1.1 Formato dos pacotes

O pacote pode ser interpretado como uma cadeia de bytes resultante da serialização dos dados a serem transmitidos ou recebidos. No pacote utilizado, todos os números com mais de um byte devem ser serializados no pacote com os bytes mais significativos à frente (*network order*).

Todos os textos do pacote devem ser terminados pelo caractere ASCII '\0' (*null terminated string*).

Cada pacote é formado pelos seguintes campos:

Pid	Ref	Src	Dst	Sid	Act	Len	D1	...	Dn	Sum

Onde:

Pid: Identificador do pacote (1 byte). Número sequencial cíclico de 0x00 a 0xFF;

Ref: Referência a outro pacote (1 byte). Este pacote refere-se à mensagem de algum outro pacote anterior enviado pelo destinatário.

Ex: se Ref = 0x09 e Act = 0x02 (resposta), então o conteúdo da mensagem é uma resposta à mensagem enviada pelo destinatário no pacote com Pid = 0x09;

Src: Identificador do remetente da mensagem (2 bytes);

Dst: Identificador do destinatário da mensagem (2 bytes);

Sid: Identificador da sessão corrente (2 bytes).

Act: Ação à qual o pacote se refere (1 byte). O tipo da ação indica como a mensagem deve ser interpretada. A ação pode assumir os seguintes valores:

0x01: Comando. O remetente está solicitando a execução de um comando ao destinatário.

0x02: Resposta parcial. O remetente está enviando uma parte da resposta ao destinatário. Ou seja, para o mesmo comando ainda seguirão outras respostas. Esta resposta refere se ao comando enviado no pacote com o campo Pid igual ao campo Ref deste pacote.

0x03: Resposta final. O remetente está enviando a última parte da resposta ao destinatário. Esta resposta refere-se ao comando enviado no pacote com o campo Pid igual ao campo Ref deste pacote.

0x04: Comando com erro. O remetente está avisando que recebeu a mensagem enviada no pacote indicado por Ref, mas que não conseguiu interpretá-la corretamente. A mensagem pode conter um texto (string ISO-8859-1) descrevendo o erro ou pode ser nula. Por exemplo: o comando não existe, algum parâmetro continha um valor inválido ou a sessão indicada por Sid não está aberta.

Len: Tamanho da mensagem, em bytes (2 bytes);

Di: I-ésimo byte da mensagem (1 byte);

Sum: Byte menos significativo da soma de todos os outros bytes do pacote (1 byte).

AP A.1.1.2 Exemplo de pacotes

Como exemplo da utilização dos pacotes será mostrado um caso de uso.

O Quadro 1 ilustra o caso de uso, onde o controlador abre uma sessão, envia um comando para a leitura da versão do protocolo utilizado e em seguida fecha a sessão.

Quadro 1: Caso de uso [01].

SessionOpen:		01 00 01 44 32 ae 00 00 01 05 01 01 45 75 00 e8	⇨
Rsp:	⇦	04 01 32 ae 01 44 e3 18 03 04 81 01 e3 18 31	
CongGetVersion:		02 00 01 44 32 ae e3 18 01 02 00 00 94	⇨
Rsp:	⇦	05 02 32 ae 01 44 e3 18 03 04 80 00 01 00 1e	
SessionClose		03 00 01 44 32 ae e3 18 01 02 01 02 98	⇨
Rsp:	⇦	06 03 32 ae 01 44 e3 18 03 03 81 02 21	

AP A.1.1.3 Mensagens

Nesta seção são descritas as mensagens que podem ser enviadas como conteúdo dos pacotes utilizados para a o envio de comandos (Act=0x01) e respostas (Act=0x02).

Tanto os comandos quanto as respostas são compostos por um identificador seguido por seus parâmetros. Os identificadores são formados por dois bytes e o número de tipos dos parâmetros depende do comando ou resposta em questão. Assim, comandos e respostas são serializados seguindo o padrão:

IdH	IdL	P1	...	PN

Onde:

IdH: Byte mais significativo do identificador;

IdL: Byte menos significativo do identificador;

Pi: I-ésimo byte dos parâmetros.

Os comandos e suas respectivas respostas estão divididos em quatro categorias, a saber: comandos de sessão, comandos de baixo nível, comandos de alto nível e comandos de configuração. A seguir são descritos os comandos para cada uma das categorias.

AP A.1.1.4 Comandos de sessão

Os comandos desta categoria são utilizados para o estabelecimento, manutenção e fechamento de sessões de comunicação com o robô.

Antes de enviar qualquer outro comando ao robô é necessário primeiro estabelecer uma sessão com o mesmo. Ou seja, o robô só responde a comandos que estejam dentro de uma sessão previamente estabelecida.

A sessão tem o objetivo de garantir que uma sequencia de comandos enviada por um controlador seja executada sem a interferência de comandos provenientes de outro controlador.

Toda a comunicação com o robô deve seguir os seguintes passos:

1. Abrir uma sessão de comunicação com o robô.

2. Enviar comandos ao robô.
3. Fechar a sessão.

É importante notar que o robô pode rejeitar os pedidos de abertura de sessão. Também, o robô pode tomar a iniciativa de fechar uma sessão a qualquer momento. A seguir são descritos os comandos e respostas a serem utilizados para o controle de sessão.

AP A.1.1.4.1 *SessionOpen*

Pedido para a abertura de uma sessão de comunicação com o robô. Se a sessão for aberta, o robô retorna o identificador da sessão aberta (Sid). Caso contrário, retorna 0x0000.

Enquanto durar a sessão o robô só aceitará comandos da sessão, com exceção do próprio comando SessionOpen. Caso outro controlador peça para abrir uma sessão e ele tiver maior prioridade sobre o controlador da sessão corrente, o comando SessionEnded é enviado ao controlador atual e sua sessão é tida como finalizada.

Cmd:	0x01	0x01	name

Rsp:	0x81	0x01	sid-H	sid-L

Onde:
name: nome do controlador (string de até 16 bytes);
sid: identificador da sessão ou 0x0000 caso não possa abrir a sessão (2 bytes).

AP A.1.1.4.2 *SessionClose*

Pedido de fechamento da sessão atual com o robô. Fecha a sessão identificada com o Sid do pacote recebido. Caso o identificador de sessão não coincida com a sessão corrente o pedido é ignorado.

Cmd:

0x01	0x02

Rsp:

0x81	0x02	closed

Onde:

closed: 0xff (feito) ou 0x00 (ignorado).

AP A.1.1.4.3 *SessionEnded*

Comunica ao controlador que a sessão corrente foi finalizada. O controlador não precisa responder.

Rsp:

0x81	0x03	reason

Onde:

reason: motivo pelo qual a sessão foi fechada (string de até 16 bytes).

AP A.1.1.5 Comandos de baixo nível

Os comandos de baixo nível possibilitam o acesso direto ao *hardware* do robô pelo controlador, permitindo o controle total do robô. Resumidamente, os comandos de baixo nível permitem a leitura dos estados do *hardware* do robô assim como o acionamento direto e individual de seus atuadores (ex: motores).

O robô possui quatro rodas que são direcionadas de forma independente. A direção de cada roda é realizada pelo controle de um servo-motor. Assim, cada servo-motor é identificado por uma das quatro rodas do robô.

O robô também possui dois motores responsáveis por sua tração. Um motor de tração é responsável por girar a roda motora do lado direito do robô e o outro a do lado esquerdo. Cada motor de tração possui um *encoder* em seu eixo que pode ser lido para saber a posição angular do eixo do motor. Assim, cada motor de tração e cada **encoder** são identificados pelo lado do robô.

AP A.1.1.5.1 *MotorDirOnOff*

Energiza ou desenergiza o servo-motor de uma das quatro rodas do robô. Cada roda do robô possui um servo-motor que é responsável pela direção da roda. Quando energizado, a roda associada ao servo-motor é posicionada (trava) na direção do comando recebido. Quando desenergizada, a direção da roda fica solta.

Servo motores	
0x00	frente-direita
0x01	frente-esquerda
0x02	trás-direita
0x03	trás-esquerda

Cmd:	0x02	0x01	wheel	on-off

Rsp:	0x82	0x01	locked

Onde:

wheel: roda a ser aplicado o comando (1 byte);

on-off: 0xff (energiza) ou 0x00 (desenergiza);

locked: 0xff (energizado) ou 0x00 (desenergizado).

AP A.1.1.5.2 *Motorturn*

Posiciona uma roda conforme o ângulo dado. O ângulo passado será o ângulo final da roda, no sentido horário, relativo à posição de alinhamento da roda com o chassi do robô (reto para frente).

Caso o servo motor não tenha torque para virar a roda na situação corrente o comando é ignorado. Por exemplo, quando o robô está parado o torque necessário é muito maior do que quando a roda está em movimento. Neste caso, pode ser que o comando seja ignorado.

Cmd:	0x02	0x02	wheel	angle-H	angle-L

Rsp:	0x82	0x01	turned

Onde:

wheel: roda a ser aplicado o comando (1 byte);

angle: ângulo final da roda, em 1/64 de grau (2 bytes);

turned: 0xff (feito) ou 0x00 (ignorado).

AP A.1.1.5.3 *MotorMove*

Aciona um dos dois motores de tração, responsáveis por fazer o robô se movimentar. O robô possui um motor de tração que aciona as duas rodas do lado direito e outro que aciona as duas rodas do lado esquerdo.

A intensidade dos motores é dada por um valor inteiro sinalizado no intervalo [−32768, +32767]. Se a intensidade for negativa o motor faz as rodas girarem para trás e se for positiva, para frente. Este valor é normalizado de forma que −32768 representa a maior potência para trás e +32767 representa a maior potência para frente.

Cmd:	0x02	0x03	side	power-H	power-L

Rsp:	0x82	0x03	moved

Onde:

side: lado do motor a ser acionado. Esquerdo (0x00) ou direito (0xff);

power: potência a ser aplicada. Inteiro sinalizado (2 bytes);
moved: 0xff (feito) ou 0x00 (ignorado).

AP A.1.1.5.4 *MotorGetPeriod*

Lê o tempo gasto, em milissegundos, para o eixo de um motor de tração realizar uma revolução. O valor é um inteiro não sinalizado no intervalo [0x0000, 0xffff] e não depende do sentido no qual o eixo está girando.

Cmd:

0x02	0x04	side

Rsp:

0x82	0x04	period-H	period-L

Onde:
side: lado do motor a ser acionado. Esquerdo (0x00) ou direito (0xff);
period: tempo gasto por uma revolução do eixo do motor, em milissegundos. Inteiro não sinalizado (2 bytes).

AP A.1.1.5.5 *MotorGetEncoder*

Lê o número de revoluções realizadas pelo eixo de um dos motores de tração.

Retorna um inteiro não sinalizado no intervalo [0x0000, 0xffff]. Note que o *encoder* é cíclico, ou seja, após atingir o valor 0xffff e ser incrementado, seu valor volta a 0x0000.

Para a navegação do robô seria desejável saber o número de revoluções da roda do robô e não do eixo do motor. Porém, esta relação depende de muitos fatores práticos, não se mantendo constante para todas as rodas. Para saber esta relação é preciso medi-la na prática, o que foge ao escopo dos comandos de baixo nível.

Note, também, que o contador é incrementado tanto no caso em que o eixo do motor gira no sentido horário, quando ele gira no sentido anti-horário. Por exemplo, se o motor realizou quatro revoluções com o robô andando para frente seguidas de quatro revoluções para trás, o comando retornará oito revoluções apesar de o robô ter voltado à posição inicial.

Cmd:	0x02	0x05	side	
Rsp:	0x82	0x05	revol-H	revol-L

Onde:
side: lado do motor a ser acionado. Esquerdo (0x00) ou direito (0xff);
revol: o dobro do número de revoluções do motor. Inteiro não sinalizado (2 bytes).

AP A.1.1.5.6 *MotorOnOff*

Energiza ou desenergiza um dos motores de tração do robô. Quando energizado, o motor de tração apresenta certa resistência à livre movimentação das rodas a ele associadas. Quando desenergizado, o motor não apresenta resistência à movimentação das rodas a ele associadas.

Motor de tração e *encoders*	
0x00	lado esquerdo
0xff	lado direito

Cmd:	0x02	0x06	side

Rsp:	0x82	0x06	locked

Onde:
side: lado do motor a ser acionado. Esquerdo (0x00) ou direito (0xff);
locked: 0xff (energizado) ou 0x00 (desenergizado).

AP A.1.1.6 Comandos de alto nível

Os comandos de alto nível são implementados utilizando-se exclusivamente dos comandos de baixo nível. Assim, estes comandos poderiam ser totalmente implementados nos controladores externos do robô.

As tarefas cotidianas de controle do robô, tais como andar em linha reta ou realizar uma curva com curvatura constante, na prática acaba sendo se mostrando complexas e dependentes do *hardware*. Como estas tarefas muitas vezes fogem aos objetivos dos desenvolvedores dos controladores do robô, o protocolo prevê a disponibilização de comandos que realizem tais tarefas.

AP A.1.1.6.1 *RobotMove*

Movimenta o robô para frente ou para trás respeitando sua curvatura atual. O comando recebe um valor de intensidade a ser aplicado nos motores de tração do robô. Devido à inércia, o robô não modificará imediatamente sua velocidade. Este comando é responsável por balancear a intensidade aplicada a cada um dos motores de forma que não haja mudança da curvatura atual. Por exemplo, se o robô está andando em linha reta e este comando for executado com um valor de intensidade maior do que o atual, o robô continuará se locomovendo em linha reta, porém a uma velocidade maior.

Cmd:	0x03	0x01	side	intensity-H	intensity-L

Rsp:	0x83	0x01	done

Onde:

side: lado do motor a ser acionado. Esquerdo (0x00) ou direito (0xff);

intensity: nova intensidade de locomoção. Inteiro sinalizado (2bytes).

done: 0xff (feito) ou 0x00 (ignorado).

AP A.1.1.6.2 *RobotTurn*

Vira as quatro rodas do robô até que a trajetória do robô ajuste-se à curvatura fornecida. A curvatura é um valor inteiro sinalizado no intervalo [-32768,+32767] e segue as seguintes convenções:

Se curvatura < 0:	vira à esquerda;
Se curvatura > 0:	vira à direita;
Se curvatura = 0:	reto para frente;
Se curvatura = −32768:	gira no próprio eixo no sentido anti-horário;
Se curvatura = +32768:	gira no próprio eixo no sentido horário;

Cmd:	0x03	0x02	side	curvature-H	curvature-L

Rsp:	0x83	0x02	done

Onde:

side: lado do motor a ser acionado. Esquerdo (0x00) ou direito (0x);

curvature: curvatura para a trajetória do robô. Inteiro sinalizado (2 bytes).

done: 0xff (feito) ou 0x00 (ignorado).

AP A.1.1.7 Comandos de configuração

Os comandos de configuração são utilizados, basicamente, para dois propósitos: ler a configuração básica do robô e configurar o comportamento básico do robô.

AP A.1.1.7.1 *ConfigGetVersion*

Retorna a versão do protocolo utilizado pelo robô. Este deve ser utilizado pelo controlador para saber a versão do protocolo utilizado pelo robô em questão. Saber a versão do protocolo é importante para permitir uma adaptação automática do controlador em futuras atualizações do protocolo.

Cmd:

0x00	0x00

Rsp:

0x80	0x00	version-H	version-L

Onde:
version: versão do protocolo de comunicação (2 bytes).

AP A.1.1.7.2 *ConfigHasMAP*

Este comando informa ao controlador se o robô possui ou não o Módulo de Alta Performance (MAP).

O MAP do robô é implementado utilizando-se uma placa à parte da infra-estrutura básica do robô e, portanto, é opcional. Este módulo é responsável por adicionar comunicação banda larga (*WiFi* e Blue-Tooth) ao robô, assim como áudio, vídeo, gerenciamento de aplicativos robóticos ou outras funcionalidades extras.

Cmd:

0x00	0x01

Rsp:

0x80	0x01	hasMAP

Onde:
hasMAP: 0xff (com MAP) ou 0x00 (sem MAP).

AP A.1.1.7.3 *ConfigSetIdentity*

Este comando modifica a identificação (ID) e o nome do robô. A identificação do robô é utilizada nos pacotes no campo Src ou Dst para garantir que o pacote está realmente endereçado ao robô. Nesta versão qualquer controlador pode executar este comando. Em versões futuras pode-se adicionar um mecanismo de segurança (ex: senha), para que apenas os controladores autenticados possam executar este comando.

Cmd:

0x00	0x02	Id-H	Id-L	name

Rsp:

0x80	0x02	done

Onde:
Id: novo identificador (2 bytes);
name: novo nome do robô (string de até 16 bytes).
done: 0xff (sucesso) ou 0x00 (ignorado);

APÊNDICE B.

PROTOCOLO DE COMUNICAÇÃO INTERNA [01]

AP B.1 BIBLIOTECA PARA COMUNICAÇÃO INTERNA

A biblioteca IComm permite a comunicação de pacotes de dados entre os microcontroladores do sistema através de interfaces seriais do tipo UART. Mais especificamente, o IComm e utilizado na comunicação entre o MCS e o MAP e também na comunicação entre o MCS e o MCR.

Um ponto a ser ressaltado e que no MCS são utilizadas duas instancias do IComm, uma para cada UART da placa. Para isso, a arquitetura do código da IComm foi projetada de forma a permitir a coexistência de mais de uma instancia no mesmo código executável.

Outro ponto a ser ressaltado e que os microcontroladores do MCS, do MCR e do MAP são diferentes e possuem diferentes APIs (*application programming interface*) de acesso ao *hardware*. Em particular a forma de ler e escrever nas UARTs são diferentes, incluindo os eventos gerados pelo *software* base de cada processador. Portanto, não poderá ser utilizado exatamente o mesmo código para os dois módulos, porem a arquitetura utilizada permite aproveitar grande parte do código.

AP B.1.1 Funções exportadas pelo IComm

As seguintes funções são exportadas pelo IComm [01]:

```
/*
* Retorna verdadeiro se um novo pacote pode ser enviado pela referida
* IComm.
*/
Bool icomm_canSend( IComm *ic );
/*
* Retorna verdadeiro a referida IComm tiver recebido um pacote que está
* pronto para ser lido.
*/
Bool icomm_canRecv( IComm *ic );
/*
* Inicia a instância 'ic' da IComm.
*
* 'maxResend' é o número de vezes que a referida IComm deve tentar
* retransmitir um pacote.
* 'timeout' é número de vezes que a função 'icomm_recv()' pode ser
* chamada enquanto um pacote estiver no meio da recepção sem que
* nenhum caractere seja recebido. Ultrapassado este número o
* pacote é descartado. Neste caso imagina-se que a conexão foi
* interrumpida.
*/
void icomm_init( IComm *ic, uint8 maxResend, uint32 timeout );
/*
* Envia a mensagem contida em 'data' com 'size' bytes pela referida IComm.
* Retorna falso se o pacote não puder ser enviado.
*/
Bool icomm_send( IComm *ic, uint8 *data, uint8 size );
/*
* Lê a mensagem contida em um pacote recebido pela IComm e armazena-o em
* 'data'. Retorna o número de bytes lidos, ou zero caso não haja pacote
* pendente.
*/
uint8 icomm_recv( IComm *ic, uint8 *data );
27
/*
* Configura a função 'callback' para ser chamada sempre que a referida
* IComm estiver pronta para enviar um pacote.
*/
void icomm_setSendCallback( IComm *ic, SendCallback callback );
/*
* Configura a função 'callback' para ser chamada sempre que a referida
* IComm tiver recebido um pacote e este estiver pronto para ser lido.
*/
void icomm_setRecvCallback( IComm *ic, RecvCallback callback );
/*
* Retorna um byte da referida IComm que deve ser escrito na UART
* a ela associada.
*/
```

```
int16 icomm_getTxByte( IComm *ic );
/*
* Todos os bytes lidos da UART associada à referida IComm devem ser
* inseridos na IComm usando esta função.
*/
Bool icomm_putRxByte( IComm *ic, uint8 byte );
```

Basicamente, a biblioteca IComm cuida da formação dos pacotes, verificação de erros, reenvio e transmissão de *acknowledge*s. Mas não escreve nem diretamente na UART utilizada. Isto porque a IComm foi projetada para poder ser utilizada por sistemas que lêem e escrevem de formas diferentes nas UARTs. Quando uma mensagem e inserida na IComm, um pacote e montado e armazenado em um buffer. Os dados deste buffer são lidos sequencialmente por chamadas a função icomm_getTxByte(). Quando todos os bytes do pacote forem lidos a IComm espera por um pacote de *acknowledge* confirmando o correto recebimento do pacote ou informando que o pacote chegou corrompido. No caso do correto recebimento, o buffer de saída da IComm e liberado e a IComm esta pronta para enviar outra mensagem. Caso contrario a mensagem e reinserida no buffer de saída e o processo e repetido por ate maxResend' vezes, quando então a mensagem e ignorada.

Os bytes que chegam pela UART são inseridos na IComm pela função icomm_putRxByte(). Quando um pacote bem formado e detectado a IComm espera por uma chamada a função icomm_recv(). Quando esta e chamada, a mensagem do pacote e retornada e um pacote de *acknowledge* e inserido no buffer de saída da IComm. Caso um pacote mal formado seja detectado, um pacote de *acknowledge* e inserido, indicando que houve erro na recepção.

Note que pela arquitetura da IComm, uma segunda mensagem só pode ser enviada depois que o cliente da IComm destino leu a mensagem enviada. Pois só apos este evento o *acknowledge* e retornado.

Este mecanismo impede que se percam mensagens por *overrun*, ou seja, que uma nova mensagem sobrescreva a anterior porque o receptor não teve tempo de lê-la.

AP B.1.2 Configuração da UART

Para que os dados enviados via *WiFi*, Bluetooth ou ZigBee cheguem à placa de controle, através da conexão UART, esta deve ser configurada corretamente, e da seguinte forma:

Baud rate: 19200

bits: 8

Parity check: None

Stop: 1

Flow control None

Para que a comunicação fosse possível, e alem disso, garantir que o dado realmente seja entregue a seu destino, foi acrescentado à camada de aplicação a lógica descrita no diagrama de fluxo mostrado na Figura 82.

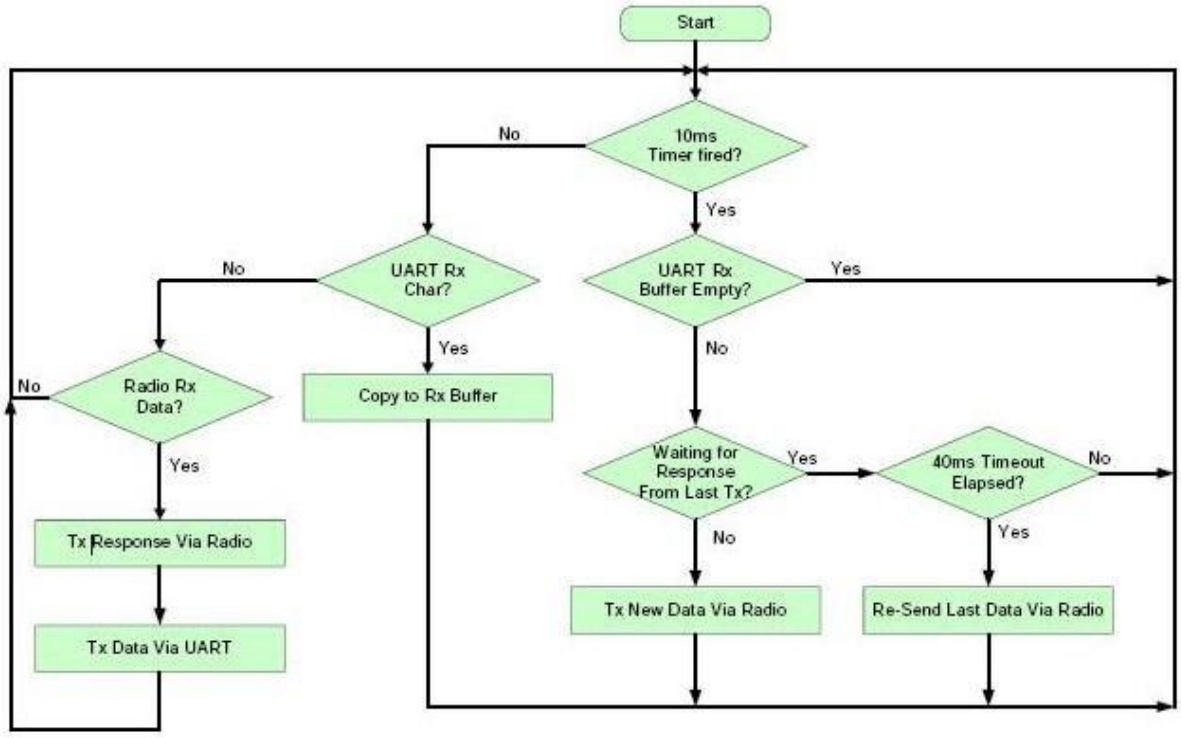

Figura 82: Lógica de comunicação através da UART [01].

Alem da lógica de comunicação foi necessária a criação de um protocolo para que a comunicação fosse possível. Esse protocolo foi desenvolvido pensando-se na reutilização na placa de alta-performance, onde será utilizado o mesmo método de comunicação através da UART, comunizando assim a forma de comunicação entre as placas. O protocolo foi desenvolvido de uma forma que possa ser expansível, isto e, possa aceitar novos comandos sem que o protocolo tenha que ser alterado.

Quando o protocolo foi criado, percebeu-se que pelo motivo da rede ZigBee criar seus endereços dinamicamente era necessário que outra forma de endereçamento fixo fosse utilizada, para que pudéssemos garantir que sempre estaríamos nos referindo ao mesmo robô, caso ele se desconecta e reconecta à rede. Dessa forma foi criada o seguinte método de endereçamento:

Cada robô devera ter um número único de identificação. Este número independe da forma de comunicação utilizada e devera ser definido pela placa de controle. Quando o robô for ligado, a placa de controle devera enviar o seu numero de identificação para o *hardware* de ZigBee, *WiFi* e também para o Bluetooth. Se os *hardware* existirem (são opcionais) eles deverão enviar um sinal de confirmação em no Maximo 100ms. Esses números de identificação serão associados aos respectivos endereços dinâmicos na rede e serão utilizados para identificar cada robô durante a comunicação.

Depois de especificada a forma de endereçamento e que o protocolo seria comum para todas as placas de comunicação (ZigBee e alta-performance), foi determinado o protocolo com os seguintes pacotes de mensagens:

Para se enviar ou receber uma mensagem através do *WiFi*, Bluetooth ou ZigBee e necessário que seja seguido um formato para as mensagens. Esse formato deve ser seguido para a comunicação UART como descrito a seguir:

N	X	SRC	DEST	D1	D2	...	Dn	SUM

Onde:

N = Numero de bytes no pacote (8 bits)

X = Meio utilizado para a comunicação (*WiFi*, Bluetooth ou ZigBee)

00 = Usado para comandos especiais

01 = ZigBee

02 = *WiFi*

03 = Bluetooth

SRC = Numero de identificação da fonte da comunicação (16 bits)

DEST = Numero de identificação do destino da comunicação (16 bits)

D1..Dn = Dados que podem ser comandos ou respostas (8 bits)

SUM = *Check Sum* (8 bits)

São somados todos os dados do pacote e são somente utilizados os 8 bits menos significativos do resultado.

Exemplo: Vamos supor que um computador conectado a um *hardware* ZigBee através da UART, possui o código de identificação 0x01 e ira enviar os dados 0x0a3c para o robô cujo código de identificação e 0x20. Então o pacote a ser enviado será:

0x09	0x01	0x00	0x01	0x00	0x20	0x0a	0x3c	0x71

Quando um pacote for enviado através da UART, o receptor do pacote devera enviar ao transmissor do pacote um sinal de que recebeu os dados corretamente. Isto e feito enviando um sinal de *acknowledge* como descrito abaixo:

N	SUM

Onde:

N = Numero de bytes no pacote (8 bits)

SUM = ao valor do *checksum* calculado pelo receptor do pacote (8 bits)

Continuando com o exemplo anterior, a placa do ZigBee quando receber este pacote, devera responder com um sinal de *acknowledge*. Recebeu-se corretamente o pacote, o *acknowledge* devera ser:

0x02	0x71

Quando a placa controladora do robô 0x20 receber este pacote, devera responder da mesma forma.

E a forma para associação de endereços que a placa controladora devera enviar para a placa ZigBee ficara da seguinte forma:

0x6	0x00	0x01	IDH	IDL	SUM

ID-H = Identificação mais significativa (8 bits)

ID-L = Identificação menos significativa (8 bits)

SUM = *Check Sum* (8 bits)

APÊNDICE C.

INTERFACE DO SUB-MÓDULO “COMMANDS” COM A CAMADA “HAL” [01]

AP C.1 SUB-MÓDULOS [01]

AP C.1.2 Command

```
#ifndef _COMMANDH_
#define _COMMANDH_
typedef unsigned char uchar;
typedef unsigned long ulong;
#include "motor.h"
// concluído ??%
// commandos para controle da direção e velocidade
u8 move(s16 intensity); // 100%
u8 turn(s8 curvature); // 100%
u8 strafe(s8 angle, s16 intensity); // 100%
u8 roll(s8 side, s16 intensity); // 100%
u8 read_velocity(void); // 100%
u32 read_distance(void); // 0%
u8 reset_distance(void); // 0%
//*** motor.h comandos INICIO ***
u8 wheel_onOff( Wheel wheel, bool onOff ); //u8 MOTOR_DirectionInit(void);
u8 wheel_turn ( Wheel wheel, s16 angle ); //u8 MOTOR_DirectionMove(Wheel nWheel, s8
Angulo);
u8 motor_onOff( Motor motor, bool onOff ); //u8 MOTOR_Lock(Motor nMotor); //u8
MOTOR_UnLock(Motor nMotor);
u8 motor_move ( Motor motor, s16 power ); //u8 MOTOR_Move(Motor nMotor, s16
```

```
intensity);
u16 motor_getPeriod ( Motor motor ); //u16 MOTOR_GetPeriod(Motor nMotor);
u16 motor_getEncoder( Motor motor ); //u16 MOTOR_GetEncoder(Motor nMotor);
u8 motor_resetEncoder( void ); //u8 MOTOR_ResetEncoder( void );
//*** motor.h comandos FIM ***
//*** sensor.h comandos INICIO ***
u16 sensor_infraredRead(u8 nSensor); //u16 SENSOR_InfraredRead(u8 nSensor);
u16 sensor_infraredCalc( void ); //u16 SENSOR_InfraredCalc(void);
u16 sensor_ultrassonic(u8 I2CSensAddr, bool distLuz); //u16 SENSOR_Ultrasonic(u8
I2CSensAddr, bool distLuz);
u16 sensor_compass(u8 I2CSensAddr); //u16 SENSOR_Compassing(u8 I2CSensAddr);
u8 sensor_accelerometerRead(s16 *x, s16 *y, s16 *z, s16 *ref); //s16 SENSOR_Get_Accel(u8
eixo); ler 4 vezes
u8 sensor_accelerometer(s16 *x, s16 *y, s16 *z); //u8 SENSOR_Accelerometer(s16 *x, s16 *y,
s16 *z);
u8 sensor_tempHumiRead(s16 *temp, s16 *humi); //s16 SENSOR_TempHumiRead(bool
Temp0_Humi1); ler 2 vezes
u8 sensor_tempHumi(s16 *temp, s16 *humi); //u8 SENSOR_TempHumi(s16 *temp, s16
*humi);
u8 sensor_gps(u8 GPS_read[], u8 command); //u8 SENSOR_Gps(u8 GPS_read[], u8
command);
u8 sensor_optical(void); //u8 SENSOR_Optical(void);
//*** sensor.h comandos FIM ***
#endif //_COMMANDH_
```

AP C.1.2 *Hardware Abstract Layer*

AP C.1.2.1 Motor

```
#ifndef _MOTORH_
#define _MOTORH_
// Correção do ângulo de direção do robô
#define K_FD -0x01FF
#define K_FE 0
#define K_TD 0
#define K_TE 0
typedef enum {
MotorEsq = 0,
MotorDir = 1
} Motor;
typedef enum {
DiantEsq = 0,
DiantDir = 1,
TrasEsq = 2,
TrasDir = 3
} Wheel;
// Concluído - ??%
// controle dos motores de locomoção
u8 MOTOR_Init(void); // 0%
u8 MOTOR_Move(Motor nMotor, s16 intensity); //100%
u8 MOTOR_OnOff(Motor nMotor, bool onOff); //100%
u8 MOTOR_EncoderInit(void); //100%
u16 MOTOR_GetPeriod(Motor nMotor); //100%
u16 MOTOR_GetEncoder(Motor nMotor); // 0%
u8 MOTOR_ResetEncoder( void ); // 0%
// controle dos servos para a direção
u8 MOTOR_DirectionInit(void); // Inicializando //100%
u8 MOTOR_DirectionMove(Wheel nWheel, s8 Angulo); // Controle das rodas //100%
#endif //_MOTORH_
```

AP C.1.2.2 Sensor

```
#ifndef _SENSORH_
#define _SENSORH_
#define I2COwnAddr 0xA0 // Endereco da I2C do ARM
#define I2CCmdReg 0x00 // SRF08 Ultrasonic command register address
#define I2CSenReg 0x01 // SRF08 Ultrasonic light sensor register address
#define I2CUltrRegH 0x02 // SRF08 Ultrasonic echo 1 high byte
#define I2CUltrRegL 0x03 // SRF08 Ultrasonic echo 1 low byte
#define I2CUltrFront 0xEE // Endereco do Ultrassom frontal
#define I2CUltrRight 0xEC // Endereco do Ultrassom da direita
#define I2CUltrLeft 0xEA // Endereco do Ultrassom da esquerda
#define I2CUltrRear 0xE8 // Endereco do Ultrassom traseiro
#define I2CCompAddr 0xC0 // Endereco da bussola
// Constantes para o acelerômetro
#define porta_H48C GPIO2
#define H48C_SCK GPIO_Pin_4
#define H48C_D_OUT GPIO_Pin_5
#define H48C_D_IN GPIO_Pin_6
#define H48C_CS GPIO_Pin_7
#define porta_SEL_SSP_IO GPIO7
#define SEL_SSP_IO GPIO_Pin_5
#define A_XAxis 0
#define A_YAxis 1
#define A_ZAxis 2
#define A_VRef 3
// defines do sensor de temperatura e umidade
#define SHT11_PORT_CLK GPIO6
#define SHT11_CLK GPIO_Pin_5
#define SHT11_PORT_DATA GPIO0
#define SHT11_DATA GPIO_Pin_5
```

```
// constantes GPS
#define GetInfo 0x00 //versão do módulo receptor GPS - hardware(1B), firmware(1B)
#define GetValid 0x01 //checa validade da string de dados(1B) - (0)not valid, (1)valid
#define GetSats 0x02 //número de satélites adquiridos(1B)
#define GetTime 0x03 //hora meridiano de Greewich(horas(1B), minutos(1B),
segundos(1B))
#define GetDate 0x04 //data meridiano de Greewich(mês(1B), dia(1B), ano(1B))
#define GetLat 0x05 //latitude (graus(1B), minutos(1B), minutos fracionais (2B), direção(1B)
(0-N,1-S))
#define GetLong 0x06 //longitude (graus(1B), minutos(1B), minutos fracionais (2B),
direção(1B) (0-L,1-O))
#define GetAlt 0x07 //altitude acima do nível do mar(2B)(em dezenas de metros, 65535
max)
#define GetSpeed 0x08 //velocidade(2B)
#define GetHead 0x09 //direção(2B)(em dezenas de graus)
// Concluído - ??%
u8 SENSOR_Init(void); // 0%
u8 SENSOR_IsMoving(void); // 0%
u16 SENSOR_Hall(u8 nSensor); //100%
u16 SENSOR_Press(u8 nSensor); //100%
u8 SENSOR_InfraredInit(void); //100%
u16 SENSOR_InfraredRead(u8 nSensor); //100%
u16 SENSOR_InfraredCalc(void); //100%
u8 SENSOR_I2C_Init(void); //100%
u16 SENSOR_Ultrasonic(u8 I2CSensAddr, bool distLuz); //100%
u8 SENSOR_I2CUltrassonicChangeAddr(u8 I2COldAddr, u8 I2CNewAddr); //100%
u16 SENSOR_Compassing(u8 I2CSensAddr); //100%
u8 SENSOR_Accelerometer_Init(void); //100%
s16 SENSOR_Get_Accel(u8 eixo); //100%
u8 SENSOR_Accelerometer(s16 *x, s16 *y, s16 *z); //100%
u8 SENSOR_TempHumi_Init(bool in1_Out0); //100%
s16 SENSOR_TempHumiRead(bool Temp0_Humi1); //100%
```

```
u8 SENSOR_TempHumi(s16 *temp, s16 *humi); //100%
u8 SENSOR_Gps_Init(void); //100%
u8 SENSOR_Gps(u8 GPS_read[], u8 command); //100%
u8 SENSOR_Optical_Init(void); //100%
u8 SENSOR_Optical(void); //100%
#endif //_SENSORH_
```

AP C.1.2.3 *Slot*

```
#ifndef _SLOTH_
#define _SLOTH_
// Concluído - ??%
u8 SLOT_Init(void); // 0%
u8 SLOT_OutDigital(u8 nSlot, u8 Value); //100%
u8 SLOT_InDigital(u8 nSlot); //100%
u8 SLOT_OutPwm(u8 nSlot, u16 width_high); //100%
u16 SLOT_Adc(u8 nSlot, u8 nPin); //100%
u8 SLOT_SerialConfig(void); //100%
u8 SLOT_UARTRead(vu8 nSlot); //100%
u8 SLOT_UARTWrite(vu8 nSlot, u8 dado[], u8 size); //100%
#endif //_SLOTH_
```

APÊNDICE D.

KIT DE DESENVOLVIMENTO DE *SOFTWARE* [01]

AP D.1 O ROBÔ-UNIVERSAL SDK

Este anexo traz a documentação referente à elaboração do Robô Universal SDK, uma biblioteca de classes em C++ a ser empregada como interface entre uma aplicação para o robô e o robô propriamente dito. Está organizado nas seguintes seções:

- **Requisitos para Elaboração do Robô Universal SDK.** Levanta as necessidades e os itens que o SDK deverá conter.
- **A Forma de Comunicação (Síncrona ou Assíncrona).** Trata das características consideradas na escolha da forma de comunicação.
- **A Escolha da Linguagem.** Relata as questões envolvidas na escolha da linguagem de programação empregada no SDK.
- **O Modelo de Comunicação.** Exibe as estruturas e os processos envolvidos na comunicação com o robô.
- **As Classes do Robô Universal SDK.** Exibe a estrutura de classes que modela o SDK.
- **Diagramas de Sequência num Exemplo de Uso.** Mostra uma série de diagramas de sequência, usados para explicar os processos envolvidos na transmissão de um comando para o robô.

AP D.1.1 Requisitos para Elaboração do Robô Universal SDK

O Kit de Desenvolvimento de *Software* (*Software Development Kit* – SDK) do Robô Universal (aqui chamado Robô Universal SDK) tem dois objetivos: primeiramente, deve dar suporte aos serviços para o controle do Robô Universal através do *Microsoft Robotics Developer Studio* (MRDS – ou simplesmente *Robotics Studio*); depois, também tem por objetivo oferecer uma biblioteca independente do *Robotics Studio*, contendo todos os mecanismos necessários para controlar o robô. O presente Documento de Requisitos é baseado nos recursos que o robô oferece (seus componentes e possibilidades de controle) e nas necessidades provenientes do emprego deste SDK dentro ou fora do ambiente do *Robotics Studio*.

AP D.1.1.1 Canais de comunicação

O Robô Universal é equipado com um dispositivo de rede ZigBee. Poderá também estar equipado com uma *placa de alta performance* que permitirá a comunicação através das redes Bluetooth e WiFi.

Cada uma destas três redes oferece um meio para troca de mensagens (comandos e respostas) entre o controlador e o robô. A rede ZigBee, entretanto, não permite o controle da câmera do robô. Já a rede Bluetooth, por sua vez, permite o controle da câmera, mas devido a sua baixa velocidade, não poderá transmitir vídeo, podendo apenas obter fotografias. A rede WiFi vem a ser, então, a mais completa, pois suporta o controle total do robô, incluindo a transmissão de vídeo.

Os robôs equipados com as redes Bluetooth e WiFi poderão emitir constantemente sinais informando sua disponibilidade. Controladores poderão, portanto, detectar tais sinais e "saber" da existência dos robôs nas proximidades. O Quadro 2 exibe o comando a ser usado pelo controlador para detectar os robôs nas proximidades.

Quadro 2:Comando para detectar robôs nas proximidades [01].

Operação / descrição	Parâmetros	Resposta
Obter robôs. Detecta, nas proximidades, os robôs equipados com as redes Bluetooth ou WiFi.		Os endereços dos robôs e suas respectivas redes (Bluetooth ou WiFi).

AP D.1.1.2 Controle de sessão

Para um robô executar ordens de um controlador, há necessidade de que uma sessão de comunicação entre este robô e este controlador esteja aberta. Cada robô poderá ter no máximo uma sessão aberta, ou seja, poderá receber ordens de apenas um controlador. Para que outro controlador possa acionar o mesmo robô, é necessário que a sessão mais antiga seja fechada, para que seja aberta uma nova sessão com este outro controlador. O Quadro 3 mostra as operações necessárias para efetuar o controle de sessão.

Quadro 3: operações para o controle de sessão [01].

Operação / descrição	Parâmetros	Resposta
Abrir sessão. Tenta adquirir uma sessão para comandar um robô.	O robô a comandar e o canal de comunicação (ZigBee, Bluetooth ou WiFi).	Referência à sessão aberta ou notificação de fracasso.
Fechar sessão. Encerra uma sessão previamente aberta	Referência à sessão.	Notificação de sucesso ou de fracasso.
Verificar se a sessão está aberta. Testa se uma referência a uma sessão é ainda válida.	Referência à sessão.	Está aberta ou não está aberta.

AP D.1.1.3 Movimentação (maior controle / maior complexidade)

O Robô Universal é equipado com quatro motores de direção e dois motores de tração, usados para a movimentação do robô. O SDK deverá oferecer comandos elementares para o controle independente sobre cada roda do robô. O Quadro 4 exibe tais comandos.

Quadro 4: comandos para controle sobre cada roda [01].

Operação / descrição	Parâmetros	Resposta
Ligar / desligar motores de tração.	Quais motores serão ligados e quais serão desligados.	
Acionar motores de tração.	A velocidade que cada correspondente roda deverá tomar.	
Ligar / desligar motores de direção.	Quais motores serão ligados e quais serão desligados.	
Acionar motores de direção.	O ângulo que cada correspondente roda deverá tomar.	
Obter a quantidade de giros de cada roda.		A quantidade de giros de cada roda.

AP D.1.1.4 Movimentação (menor controle / menor complexidade)

O Robô Universal SDK deverá oferecer também comandos de movimentação comumente usados. O desenvolvedor não precisa controlar o movimento de cada roda. Em vez disso, informa apenas o movimento que o robô deverá descrever. Estes comandos, por sua vez, coordenam o acionamento de cada roda a fim de realizar a movimentação desejada. O Quadro 5 lista estes comandos.

Quadro 5: comandos de movimentação comumente usados [01].

Operação / descrição	Parâmetros	Resposta
Mover o robô. Muda a velocidade do robô sem alterar sua "curvatura". Portanto atua somente nos motores de tração, não nos motores de direção.	A nova velocidade que o robô deverá desempenhar.	
Fazer curva. Muda a "curvatura" do robô em alterar sua velocidade. Deve atuar tanto nos motores de direção quanto nos de tração.	O raio que descreverá a trajetória do robô.	
Mover diagonalmente o robô. Muda a direção do movimento do robô sem alterar sua velocidade nem sua orientação. Atua somente nos motores de direção.	O ângulo que as rodas deverão tomar.	
Girar o robô. Gira o robô em torno de seu eixo. Atua nos motores de tração e direção.	A velocidade de rotação (angular) que o robô deverá desempenhar	
Frear o robô. Trava todos os motores de tração e de direção do robô. Este travamento é obtido ligando-se os motores para que eles ofereçam resistência ao movimento.		

AP D.1.1.5 Sensoriamento

O Robô Universal é equipado com diversos tipos de sensores. Sensores infravermelhos e ultra-som são empregados para a detecção de obstáculos relativamente distantes. Sensores

ópticos de colisão, como o próprio nome sugere, são empregados para informar sobre o contato do robô com obstáculos. O robô possui também uma bússola, um acelerômetro, um sensor de temperatura e umidade, e um dispositivo de GPS. O Quadro 6 mostra os comandos que atuarão nestes sensores.

Quadro 6: comandos de sensoriamento [01].

Operação / descrição	**Parâmetros**	**Resposta**
Ler sensores infra-vermelho.		A distância dos obstáculos detectados por cada um dos sensores.
Ler distância com sensores ultra-som.		A distância dos obstáculos detectados por cada um dos sensores.
Ler luminosidade com sensores ultra-som.		A luminosidade em cada um dos sensores.
Verificar sensores de colisão (ópticos).		A detecção ou não de colisão em cada um dos sensores.
Ler a bússola.		A orientação do robô segundo a bússola
Ler o acelerômetro.		A força "g" que atua sobre o robô
Ler sensor de temperatura e umidade.		Os valores de temperatura e umidade.
Testar a validade de dados provenientes do GPS. Com base no número de satélites detectados, o GPS oferece uma forma de verificar se os dados a serem lidos são válidos ou não.		Validade ou invalidade dos dados.
Obter localização com o GPS.		A latitude, longitude e altitude do robô.

Obter data e horário com o GPS.		A data e o horário correntes.
Obter a velocidade (e sua direção) do robô através do GPS. O GPS oferece meios de obter a velocidade linear do robô. Oferece também meios para obter a direção (em relação ao norte magnético) do deslocamento.		A velocidade e a direção de deslocamento do robô.
Verificar estado das baterias.		O estado de cada bateria.

AP D.1.2 A Forma de Comunicação (Síncrona ou Assíncrona)

Com relação ao tempo de reposta, o Robô Universal possui três classes de comandos:

1. **Comando imediato.** Ex.: leitura de sensor infravermelho. O controlador envia o comando e o robô responde quase imediatamente. A transação então acaba.
2. **Comando longo.** Ex.: andar 1 metro. O controlador envia o comando, e o robô responde imediatamente informando que recebeu o comando e que irá executá-lo. Então o robô começa a executar o comando. Finalizada a execução, o robô responde indicando se o comando foi ou não corretamente executado.
3. **Comando continuado.** Ex.: receber vídeo. O controlador envia o comando, e o robô responde imediatamente informando que recebeu o comando e que irá, ou não, executá-lo. O robô passa a enviar respostas para o controlador de tempos em tempos. O controlador envia um comando para desativar o comando anterior. O robô responde que recebeu este último comando e pára de enviar as respostas referentes ao primeiro comando.

Os comandos das classes 2 e 3 deverão ser assíncronos para as respostas prolongadas. Ou seja, a função que envia o comando deverá retornar imediatamente, impedindo que o programa fique bloqueado esperando a devida resposta.

Já os comandos da classe 1, assim como os das classes 2 e 3 para as respostas imediatas, poderiam ser síncronos ou assíncronos. Síncrono significa que o programa que enviou o comando ficará bloqueado até a chegada da resposta. Aqui, a escolha entre a forma síncrona ou assíncrona deve levar em conta dois aspectos: por um lado, o desempenho do programa e a liberdade para explorar as potencialidades do robô; por outro, a simplicidade na elaboração dos programas.

Optou-se por adotar a forma ASSÍNCRONA em todo o Robo-Universal SDK. Assim, apesar dos programas se tornarem um pouco mais complexos, haverá maior liberdade para a elaboração destes programas e haverá uniformidade, uma vez que somente um padrão é adotado.

AP D.1.3 A Escolha da Linguagem

O *Robotics Studio* é baseado em linguagens direcionadas à *Common Language Infrastructure* (CLI), como C#, VB.NET e C++/CLI. Entretanto, a maior parte da documentação do *Robotics Studio* está em C#. Fica claro que a Microsoft vem dando mais atenção a esta linguagem. Portanto, optou-se por empregar C# como linguagem para elaboração dos serviços do Robo-Universal para o *Robotics Studio*.

O objetivo primário do Robo-Universal SDK é dar suporte aos serviços para o *Robotics Studio*. No entanto, objetiva-se também ter este SDK como um produto independente. Assim, este SDK deve empregar uma linguagem bem consolidada, mas que possa ser integrada aos serviços em C#. A linguagem C++ vem então a ser a opção mais conveniente. Contudo, como veremos a seguir, não se empregou a linguagem C++ pura, mas sim uma extensão desta linguagem definida pela Microsoft e chamada C++/CLI.

AP D.1.3.1 Escolha da biblioteca

É impossível desacoplar completamente do sistema operacional um *software* como o Robo-Universal SDK escrito em C++. Características como o controle de *threads* e o acesso à porta serial são muito dependentes do sistema operacional. Então haverá necessariamente certa dependência, que no mínimo aparecerá na biblioteca empregada para o controle de *threads*, por exemplo.

No Windows, há pelo menos três formas de se trabalhar com estes *threads*: através da *C Run-time Library* (CRT); através da *Microsoft Foundation Classes* (MFC); e através do *Microsoft .NET Framework*. Observando que a Microsoft vem incentivando o emprego desta última tecnologia, optou-se por adotá-la no Robô Universal SDK. Mais ainda, o emprego do NET Framework torna mais simples o desenvolvimento do SDK.

O .NET Framework não opera com a linguagem C++ pura, mas sim com a C++/CLI, uma extensão da linguagem C++ definida pela Microsoft. Esta linguagem estendida surge exatamente para dar suporte ao .NET Framework. Consequentemente o Robô Universal SDK emprega a linguagem C++/CLI.

C++ pura com C++/CLI, ou C++/CLI somente?

Visando a portabilidade do Robo-Universal SDK para outros sistemas operacionais além do Windows, torna-se atrativo separar as partes dependentes das independentes do sistema operacional, e então implementar as partes independentes através da linguagem C++ pura.

C++/CLI define elementos denominados *gerenciados*, que são elementos manuseados pelo ambiente de execução do .NET Framework (chamado *Common Language Runtime*, ou CLR). Já a linguagem C++ pura só define elementos *nãogerenciados*. Acontece que surgem alguns problemas de compatibilidade quando se misturam elementos gerenciados (provenientes do uso da C++/CLI) com elementos não-gerenciados (provenientes do uso da C++ pura). Como exemplo, uma classe gerenciada não permite que se agregue diretamente um objeto não-gerenciado. Isto não impede, mas atrapalha muito o desenvolvimento dos programas. Deve-se

lembrar ainda que os serviços para o *Robotics Studio* sejam escritos em C#, uma linguagem voltada ao .NET Framework e consequentemente baseada em objetos gerenciados.

A mistura das duas linguagens também quebra a uniformidade do código. Estruturas diferentes são utilizadas com o mesmo propósito ao longo do programa. Como exemplo, a classe *std::vector* (C++ pura) que pode ser empregada com o mesmo objetivo que a classe *cli::array* (C++/CLI).

Avaliando todos estes aspectos, adotou-se somente a linguagem C++/CLI ao longo de todo o Robo-Universal SDK.

AP D.1.4 O Modelo de Comunicação

A essência do Robo-Universal SDK está na troca de mensagens com o robô. Assumindo uma das redes de comunicação (ZigBee, Bluetooth ou WiFi), duas são as principais classes envolvidas no modelo de comunicação: *SerialCommunication* e uma classe específica à rede adotada. Aqui se usa a classe *ZigBee* como exemplo. A seção *As Classes do Robo-Universal SDK* exibe a estrutura das classes *SerialCommunication* e *ZigBee*.

As mensagens manipuladas pelo SDK seguem dois protocolos distintos: o Protocolo do Robo-Universal e o Protocolo da Rede ZigBee. As mensagens da rede ZigBee caracterizam-se por “empacotar” as mensagens do Robo-Universal.

O modelo assíncrono de comunicação entre o controlador e o robô exige que o programa do usuário fique pouco tempo bloqueado à espera do trabalho do SDK. Portanto a arquitetura do Robo-Universal SDK está baseada em diversos *threads* trabalhando concorrentemente. A Figura 83 ilustra as estruturas envolvidas no processo de comunicação. Na sequência, descrevem-se cada um dos componentes deste modelo.

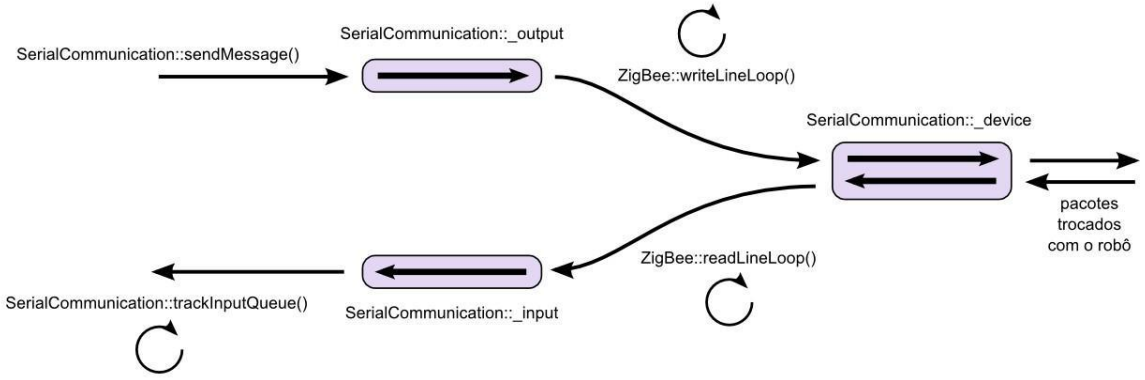

Figura 83: As estruturas envolvidas no modelo de comunicação [01].

SerialCommunication::_output* e *SerialCommunication::_input

Representam respectivamente as filas de saída e de entrada de mensagens. São objetos da classe *System::Collections::Queue* e armazenam mensagens segundo o protocolo do Robo-Universal. A existência destas estruturas torna mais organizada o modelo de troca de mensagens. A fila de saída, em particular, impede também que o programa do usuário fique bloqueado por muito tempo, como visto adiante no método *sendMessage*.

SerialCommunication::_device

Representa o dispositivo de comunicação serial. Trata-se de um objeto da classe *System::IO::Ports::SerialPort* e recebe as mensagens segundo o protocolo ZigBee.

SerialCommunication::sendMessage()

Método responsável por inserir as mensagens na fila de saída. Sua execução roda na mesma *thread* do programa do usuário. Portanto, uma vez invocado, o programa do usuário estará esperando o término da execução deste método. Isto justifica o emprego da fila de saída. Desta forma, o método simplesmente insere a mensagem na fila de saída e já retorna a execução ao programa do usuário.

SerialCommunication::trackInputQueue()

Método responsável por receber as mensagens da fila de entrada e invocar os correspondentes métodos que tratarão cada uma destas mensagens (*callbacks*). A execução de *trackInputQueue* se dá numa *thread* própria e permanece em um laço, monitorando constantemente a fila de entrada. Um novo *thread* é aberto para cada *callback* invocado.

ZigBee::writeLineLoop()

Método responsável por coletar as mensagens da fila de saída, empacotá-las segundo o protocolo ZigBee, e então encaminhá-las para o dispositivo de comunicação serial. Sua execução também se dá num *thread* própria e permanece em um laço, mas aqui monitorando a fila de saída.

ZigBee::readLineLoop()

Método responsável por receber as mensagens provenientes do dispositivo de comunicação serial, extrair as mensagens segundo o protocolo do Robo-Universal, e então inserir estas últimas na fila de entrada. Assim como o método anterior, sua execução também se dá num *thread* própria e permanece em um laço, aqui monitorando o dispositivo de comunicação serial.

AP D.1.5 As Classes do Robo-Universal SDK

O Robo-Universal SDK é orientado a objetos, portanto é modelado com um conjunto de classes e seus relacionamentos. Os tópicos a seguir tratam destas classes, destacando seus objetivos e principais aspectos.

AP D.1.5.1 *Robo-Universal*, a classe principal

Esta classe funciona como ponto de acesso ao SDK a partir do programa do usuário. Agrega todos os componentes do robô, como os motores de tração, os motores de direção e os diversos sensores. Agrega também o objeto responsável pela comunicação serial e mantém referência a características como endereço do controlador, endereço do robô e sessão aberta. A Figura 84 mostra a estrutura desta classe e como ela se integra com as outras.

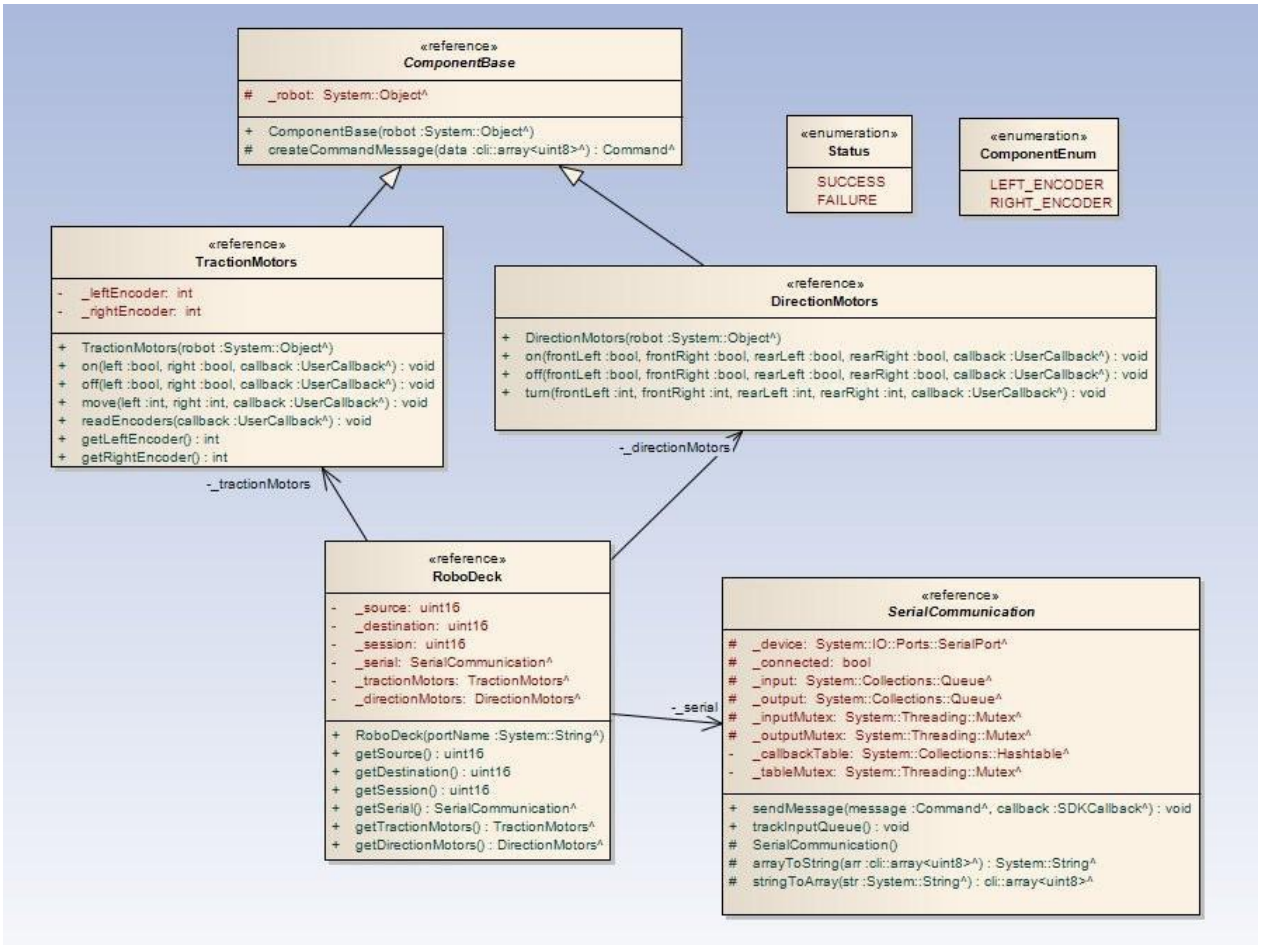

Figura 84: A classe Robo-Universal e as classes dos componentes [01].

AP D.1.5.2 Os componentes do robô

As classes *TractionMotors* e *DirectionMotors* (e outras que existirão) representam os grupos de componentes do robô. Elas definem as operações cabíveis sobre seus componentes. A classe *ComponentBase* representa a superclasse dos componentes, concentrando características comuns a eles. A Figura 85 mostra a estrutura destas classes.

AP D.1.5.3 As mensagens do Robo-Universal

O Robo-Universal SDK é provido de uma hierarquia de classes capaz de organizar as mensagens de acordo com o protocolo do robô. A exibe esta hierarquia. *Message* é a classe-base das mensagens do Robo-Universal e mantém as características comuns a qualquer mensagem.

Command representa as mensagens de comando. Cada comando deverá ter um identificador único. Para tanto, esta classe mantém um atributo estático (*classId*) usado como referência para suas instâncias obterem os identificadores. O acesso a este atributo deve ser sincronizado.

Dividida em três subclasses – *PartialResponse*, *FinalResponse* e *Error* – a classe *Response* representa as mensagens de resposta provenientes do robô.

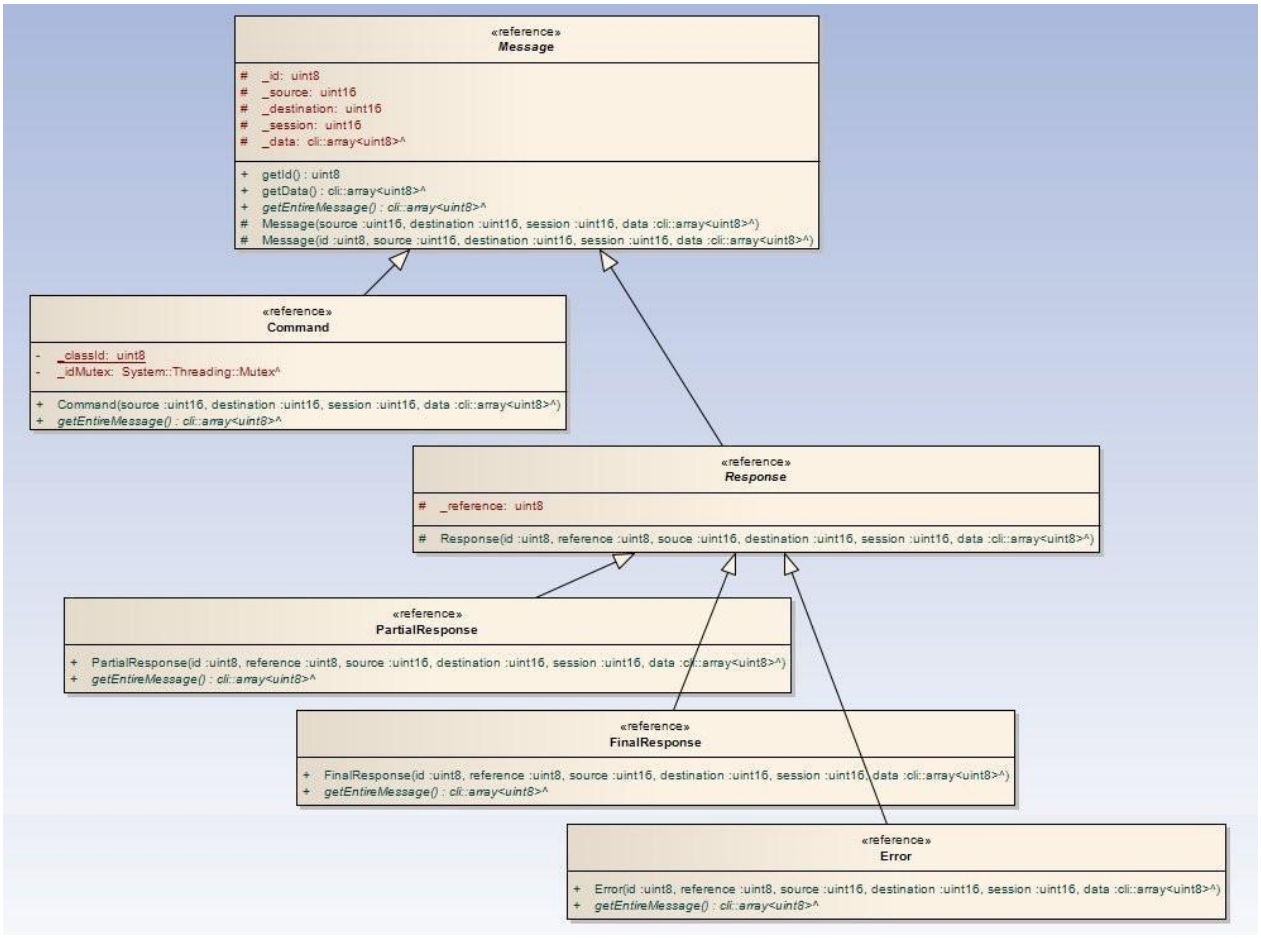

Figura 85: A hierarquia de mensagens do Robo-Universal [01].

AP D.1.5.4 A comunicação serial

O Robo-Universal SDK define classes responsáveis pela comunicação serial entre o controlador e o robô. A cada uma das redes de comunicação (ZigBee, Bluetooth e WiFi) corresponde uma classe dentro do SDK. Esta classe contém características particulares a cada uma das redes. A Figura 86 mostra as classes envolvidas na comunicação serial.

Já a classe *SerialCommunication*, por sua vez, define características que não dependem da rede utilizada. *_device* representa o dispositivo de comunicação. *_input* e *_output* representam respectivamente as filas de entrada e de saída. *_callbackTable* é uma tabela que associa os i-

dentificadores das mensagens de comando (enviadas pelo controlador) aos respectivos métodos (*callbacks*) que tratarão as respostas referentes a estes comandos.

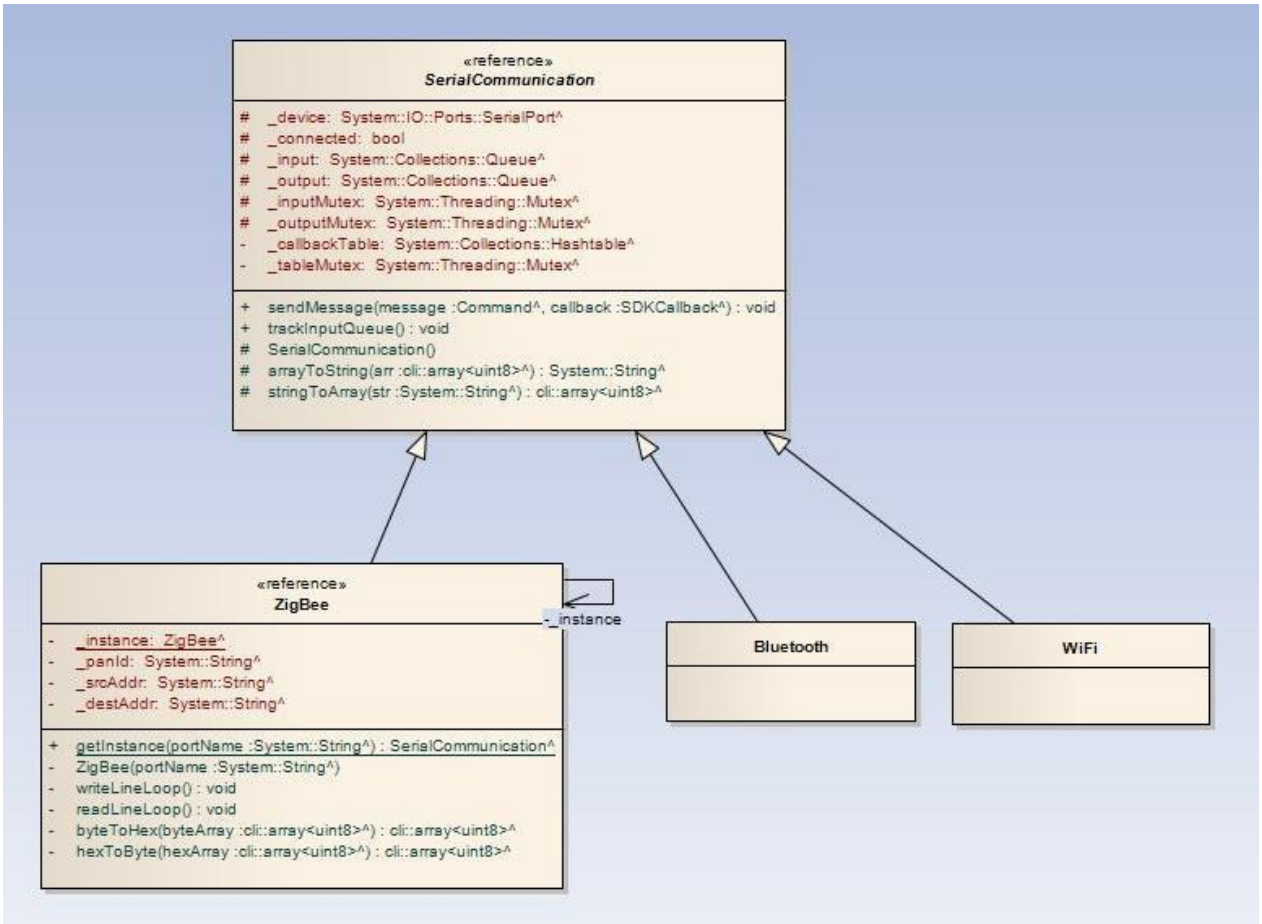

Figura 86: As classes envolvidas na comunicação serial [01].

AP D.1.5.5 A estrutura de *callbacks*

O Robo-Universal SDK também define uma estrutura de *callbacks*. Estes *callbacks* são responsáveis por tratar assincronamente às diversas mensagens de resposta que trafegam pelo

SDK. Todas definem um método chamado *doCallback*, o qual contém o programa a ser executado. A Figura 87 exibe as classes envolvidas neste processo.

A cada comando que o usuário invoca sobre o robô, corresponderá uma resposta proveniente do SDK. Tal troca de mensagens é assíncrona, e o programa do usuário não deve ficar bloqueado esperando estas respostas. Para tanto, o usuário deve definir as ações a serem executadas a cada resposta por ele recebida. Então ele deve criar classes que estendam *UserCallback* e implementam o método *doCallback*. Este será o método invocado quando a referente resposta chegar.

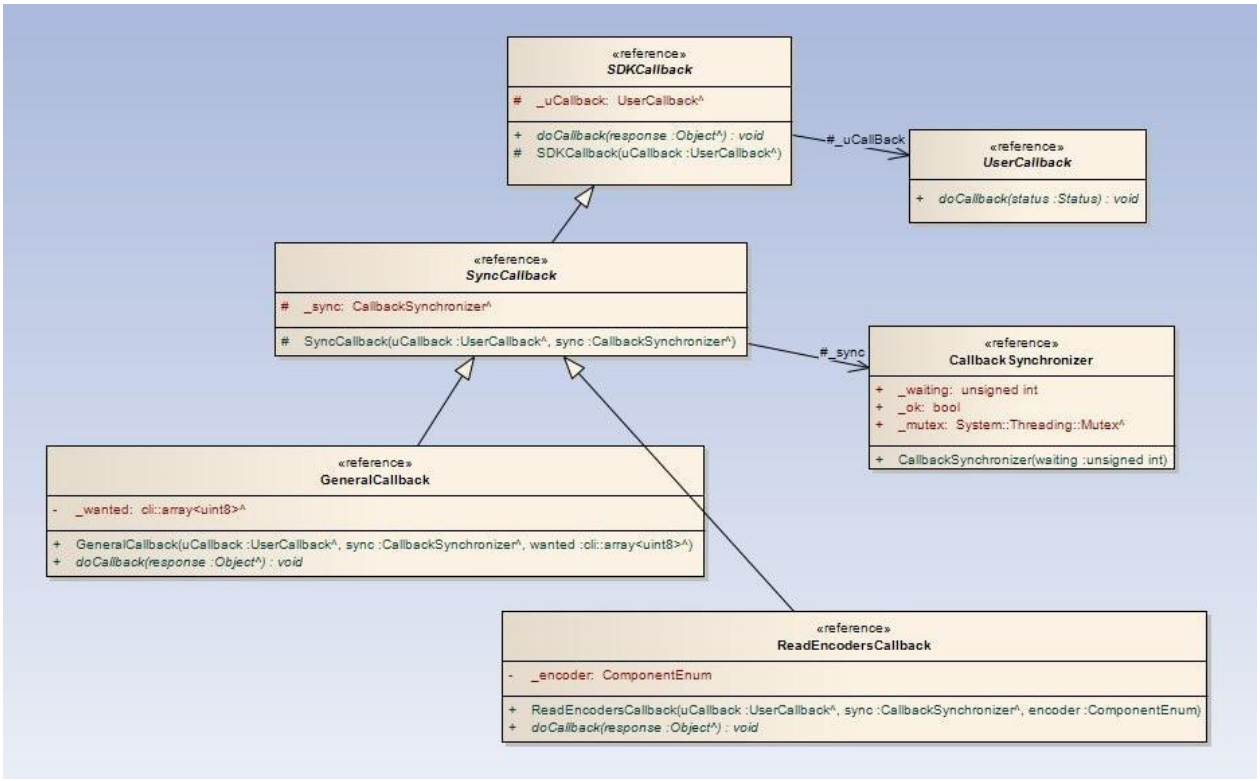

Figura 87: As classes de callback [01].

SDKCallback, por sua vez, é a superclasse de todos os *callbacks* empregados internamente pelo SDK. Funcionam como *callbacks* intermediários, por isso mantêm referência a *callbacks* do usuário. Então o sistema chama estes *callbacks* do SDK, os quais realizarão as devidas tarefas e depois chamarão os *callbacks* do usuário.

Alguns comandos solicitados pelo usuário podem ser quebrados em múltiplos comandos pelo SDK. A seção *Diagramas de Sequência em um Exemplo de Uso* ilustra esta situação. Para estes casos o SDK emprega classes derivadas de *SyncCallback* (*Synchronized Callback*), que agregam objetos de sincronização (*CallbackSynchronizer*) e coordenam as múltiplas respostas que virão para o SDK. Tal sincronização faz com que o programa do usuário receba apenas uma (e não múltiplas) resposta para cada comando por ele solicitado.

GeneralCallback compartilha ações comuns a grande parte dos comandos empregados pelo SDK. Já *ReadEncodersCallback* é específico para o método *readEncoders* de *TractionMotors*. Outros *callbacks* serão definidos para os demais comandos (dos demais componentes) do Robo-Universal.

AP D.1.6 Diagramas de Sequência em um Exemplo de Uso

Mostram-se agora alguns diagramas de sequência envolvidos num exemplo de uso. Aqui, ilustra-se a situação onde o usuário deseja ligar (energizar) os dois motores de tração do robô. Para tanto, ele utiliza o método *on* de *TractionMotors*. O usuário define também um programa a ser executado no momento da chegada da resposta referente ao comando *on*.

AP D.1.6.1 Esquema geral

Aqui, mostra-se resumidamente todo o processo envolvido na chamada e tratamento do comando *on*, a partir da aplicação do usuário. A Figura 88 contém o correspondente diagrama de sequência.

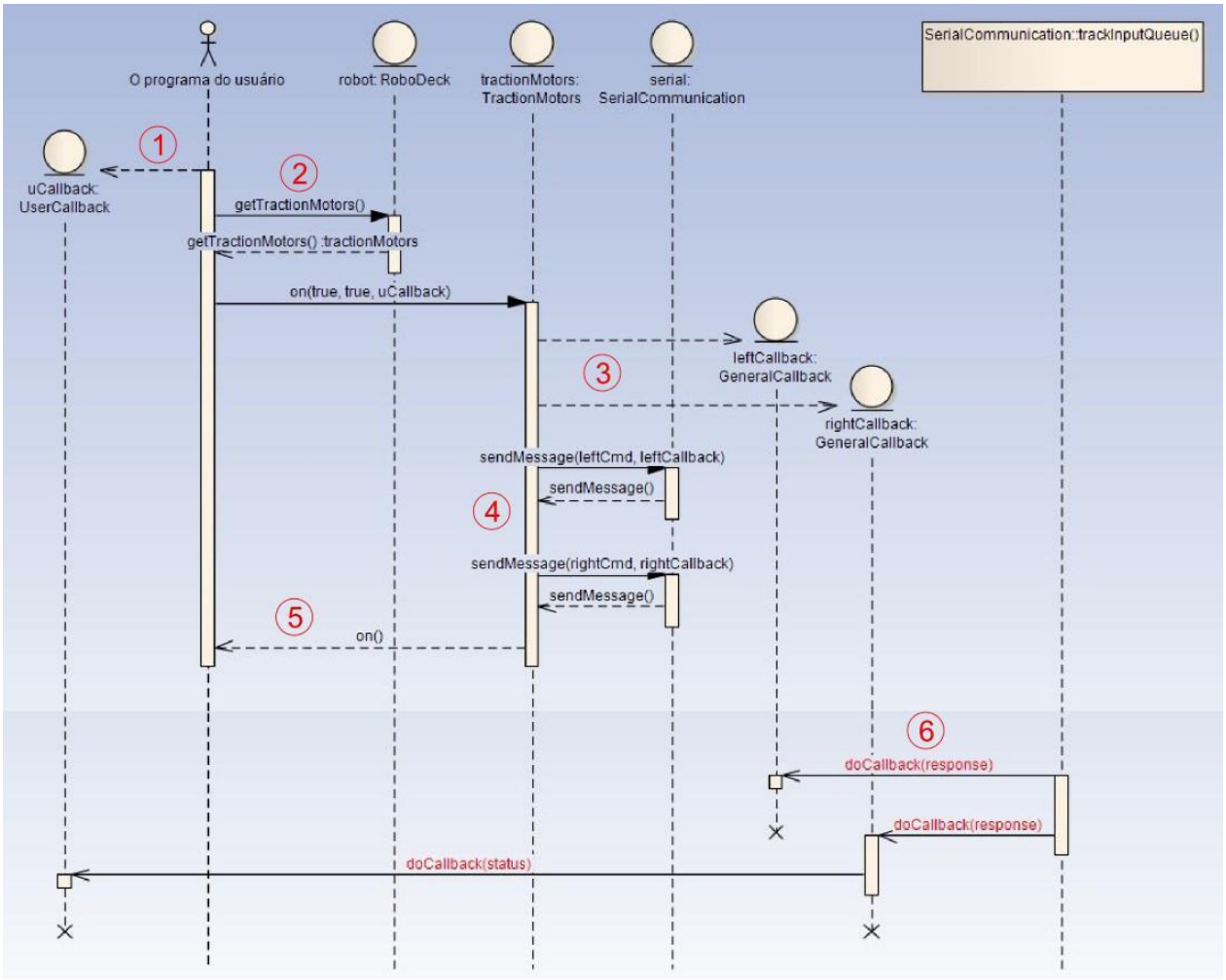

Figura 88: O processo envolvido na chamada e tratamento do comand on [01].

1. O usuário cria (e instancia) uma classe que estende *UserCallback.* Esta classe conterá o programa, embutido no método *doCallback*, que será executado quando chegar a resposta referente ao comando *on*.
2. O usuário adquire a instância de *TractionMotors* e invoca o comando *on*, passando o objeto (aqui chamado *uCallback*) que tratará a futura resposta.
3. O comando *on* deve ser decomposto pelo SDK em dois comandos distintos: *ligar motor esquerdo* e *ligar motor direito*. Estes serão os dois comandos a serem enviados ao robô, e a cada um deles corresponderá uma resposta distinta. O SDK só re-

portará *sucesso* ao programa do usuário caso as duas respostas provenientes do robô também reportarem *sucesso*. Então o SDK cria duas instâncias de *GeneralCallback* que tratarão cada uma das respostas do robô.

4. O SDK constrói as duas mensagens de comando e as envia através do método *sendMessage* de *SerialCommunication*.
5. Com o retorno do método *on*, o programa do usuário fica liberado para executar outras tarefas, inclusive enviar novos comandos. Quando chegar a resposta do robô, ela será tratada numa nova *thread* de execução, cujo programa está definido no método *doCallback* de *uCallback* (o *callback* do usuário).
6. *trackInputQueue* representa o programa que monitora a *fila de entrada* a espera de novas respostas. Neste exemplo, supomos que a resposta referente ao comando de ligar o motor esquerdo chegou antes. Quando chegar a resposta referente ao motor direito, o correspondente *callback* identificará que não há mais respostas pendentes e invocará então *doCallback* definido em *uCallback*, reportando o *status* da execução do comando (*sucesso* ou *falha*).

AP D.1.6.2 Os passos de *sendMessage* (Figura 89)

Essencialmente, dois são os principais passos envolvidos no método *sendMessage*. Primeiro, adiciona-se na tabela *callbackTable* uma entrada contendo o identificador do comando enviado (usado como chave) junto com a referência ao *callback* que tratará a futura resposta. Depois, insere-se o comando propriamente dito na fila de saída (*output*). Aqui, o método *sendMessage* é invocado duas vezes: uma para cada motor de tração.

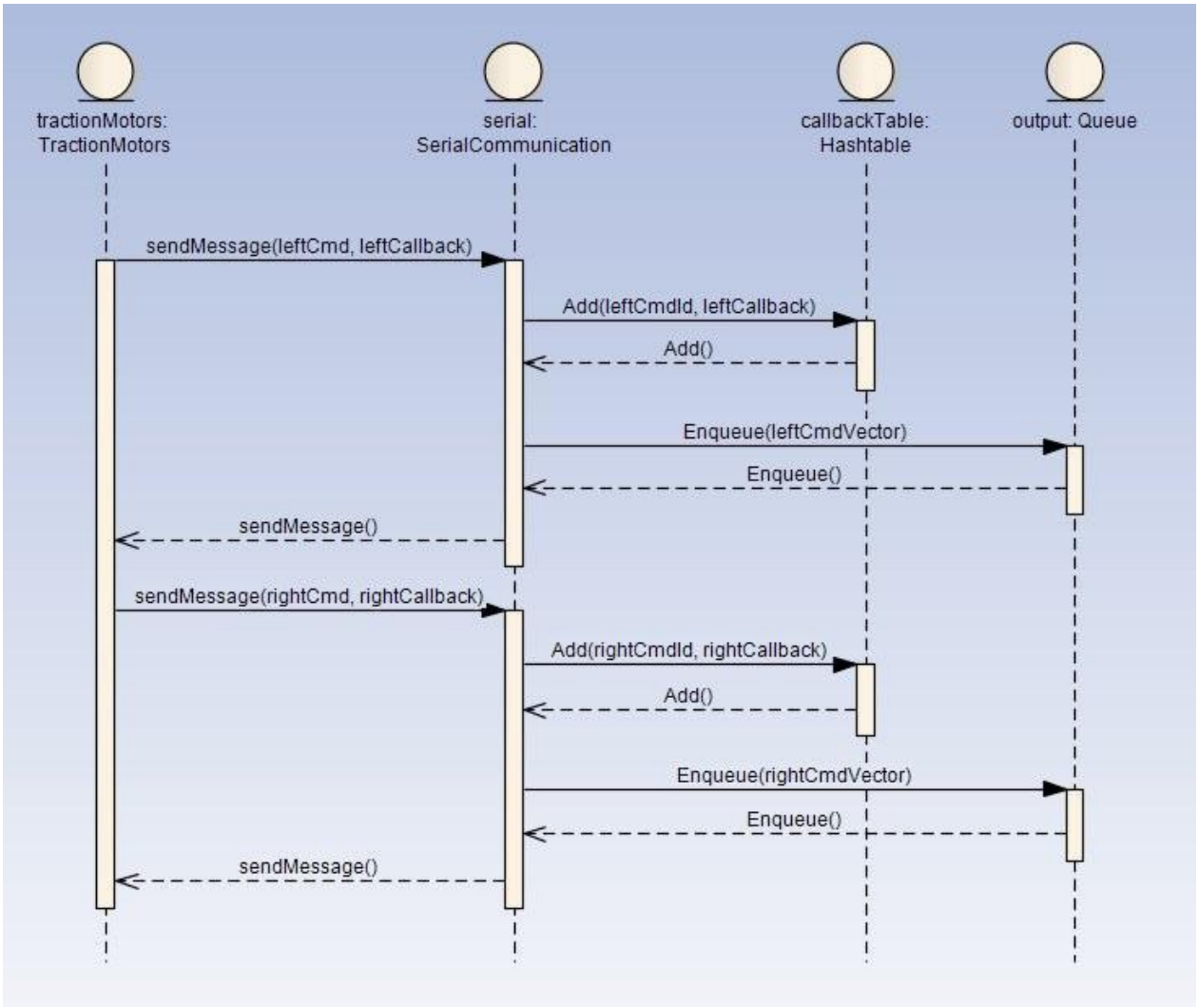

Figura 89: Os passos de sendMessage [01].

AP D.1.6.3 Os acessos ao dispositivo de comunicação serial (Figura 90)

O Robo-Universal SDK mantém duas *threads* especializadas em acessar o dispositivo de comunicação serial (*device*): uma para enviar e outra para receber os pacotes trocados com o robô, através dos métodos *writeLineLoop* e *readLineLoop*, respectivamente.

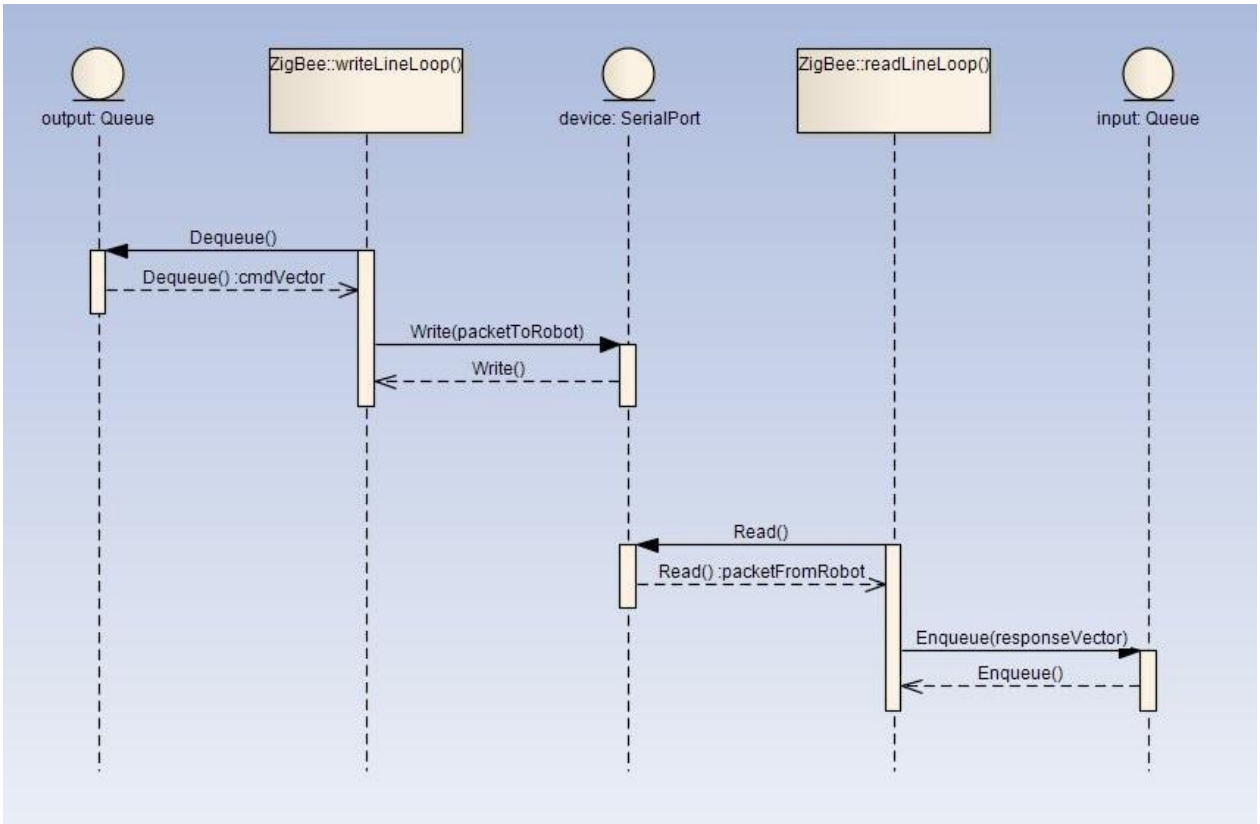

Figura 90: Os acessos ao dispositivo de comunicação serial [01].

AP D.1.6.4 O monitoramento da fila de entrada (Figura 91)

O método *trackInputQueue* é executado numa *thread* exclusiva e é responsável por receber as mensagens que chegam à fila de entrada (*input*). Primeiro, ele adquire a mensagem da fila. Depois, ele acessa a tabela *callbackTable* a procura do *callback* correspondente à resposta que chegou. Finalmente, ele invoca o método *doCallback*, a ser executado numa *thread* independente. Neste exemplo, assumimos que a resposta referente ao motor esquerdo chegou primeiro.

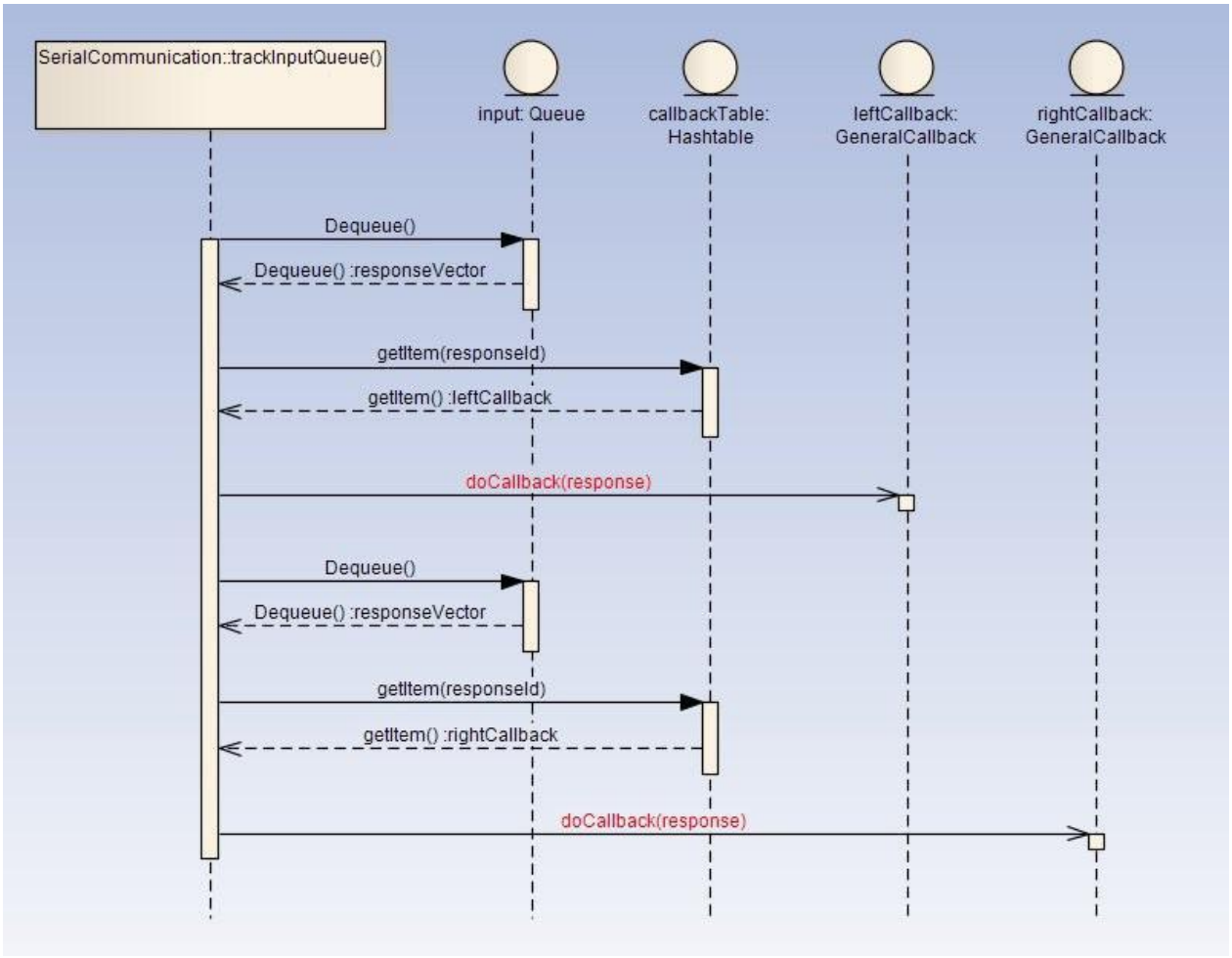

Figura 91: O monitoramento da fila de entrada [01].

AP D.1.6.5 A coordenação das múltiplas respostas (Figura 92)

Aqui se ilustra o caso aonde a resposta referente ao motor de tração esquerdo chega primeiro. Assim sendo, *trackInputQueue* chama *doCallback* de *leftCallback*, que por sua vez acessa o objeto de sincronização (*sync*) e verifica que há ainda mais respostas pendentes. Feito isso, quando for chamado *doCallback* de *rightCallback*, este também acessará o objeto de sincronização, mas desta vez verificará que não há mais respostas a chegar. Então *rightCallback* passa a ser responsável por invocar o *callback* do usuário. Importante notar que *right-*

Callback e *leftCallback* rodam em *threads* distintas, e os acessos ao objeto de sincronização devem ser, evidentemente, sincronizados.

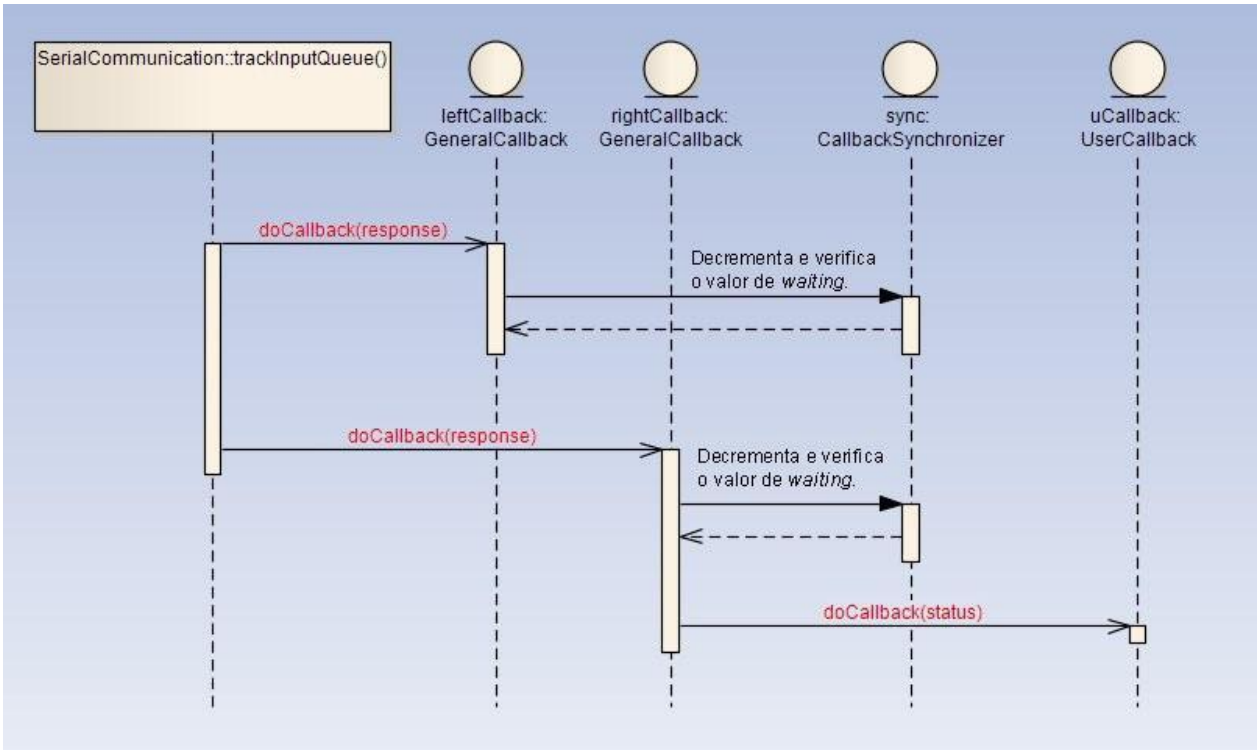

Figura 92: A coordenação das múltiplas respostas [01].

APÊNDICE E.
MÓDULO DE EXPANSÃO [01]

AP E.1 MÓDULOS DE EXPANSÃO

O Robo-Universal também permite o desenvolvimento de módulos que podem ser acoplados ao robô via *slots* de expansão. Um exemplo de módulo de expansão é a garra que foi desenvolvida, acoplada ao robô via *slot* e se comunica serialmente através de uma interface UART (*Universal Asynchronous Receiver/Transmitter*).

AP E.1.1 GARRA

A garra possui duas liberdades de movimento:

- Abrir e fechar, chamado movimento de pinça: utilizado para pegar objetos localizados no interior de seus "dedos" metálicos
- Frente e trás, chamado movimento longitudinal: utilizado para levar a garra para dentro e fora da carenagem do robô, e para fazê-la alcançar os objetos a serem agarrados

Há um sensor de pressão colocado na ponta do "dedo" metálico mostrado na Figura 93, que permite identificar a pressão aplicada no objeto agarrado e até fazer com que a garra pare de apertar o objeto ao chegar num valor de pressão pré-estipulado.

Os movimentos são gerados por dois motores DC que são controlados por uma placa eletrônica que recebe comandos dos *slots* da placa-mãe do Robo-Universal, os motores estão acoplados a sistemas de rosca sem fim para executarem os movimentos de pinça e longitudinal, como mostram as Figura 94 e Figura 95 respectivamente.

Figura 93: Robô Universal com módulo da garra, detalhe para o sensor de pressão [01].

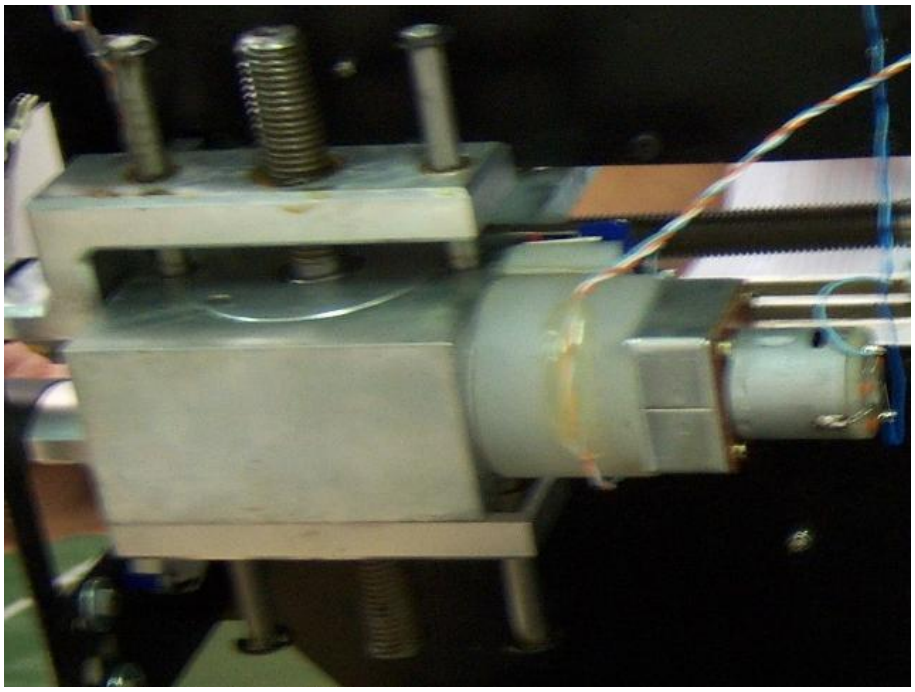

Figura 94: Sistema motor DC acoplado a uma rosca sem fim que executa o movimento de pinça [01].

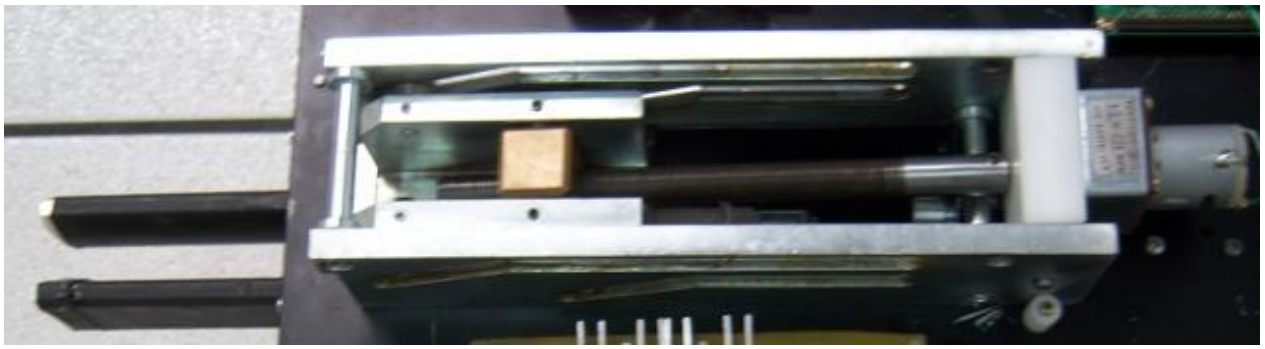

Figura 95: Sistema motor DC acoplado a uma rosca sem fim que executa o movimento longitudinal [01].

A garra consegue executar os seguintes comandos:

1 mover-se para frente ou para trás sem restrição de tempo ou pressão

2 abrir ou fechar seus “dedos” metálicos sem restrição de tempo ou pressão

3 mover-se para frente ou para trás por um tempo especificado

4 abrir ou fechar seus “dedos” metálicos por um tempo especificado

5 fechar seus “dedos” metálicos até medir uma pressão igual ou maior a especificada

6 parar o movimento de pinça, longitudinal ou os dois simultaneamente

7 informar a leitura do sensor de pressão

8 informar as leituras dos sensores de fim de curso das liberdades de movimento

AP E.1.1.1 Eletrônica

A eletrônica da garra é baseada em um microcontrolador que controla os motores através de *drivers*. Tendo em vista um possível aproveitamento da mesma placa controladora para outros dispositivos, a eletrônica é capaz de controlar até quatro motores, inclusive ao mesmo tempo e também se pode ligar até dois sensores de pressão.

O esquemático da placa se divide basicamente em quatro partes:

AP E.1.1.1.1 Alimentação

É responsável pela entrada da energia e regulagem das tensões utilizadas na placa, os circuitos dessa parte são mostrados na Figura 96.

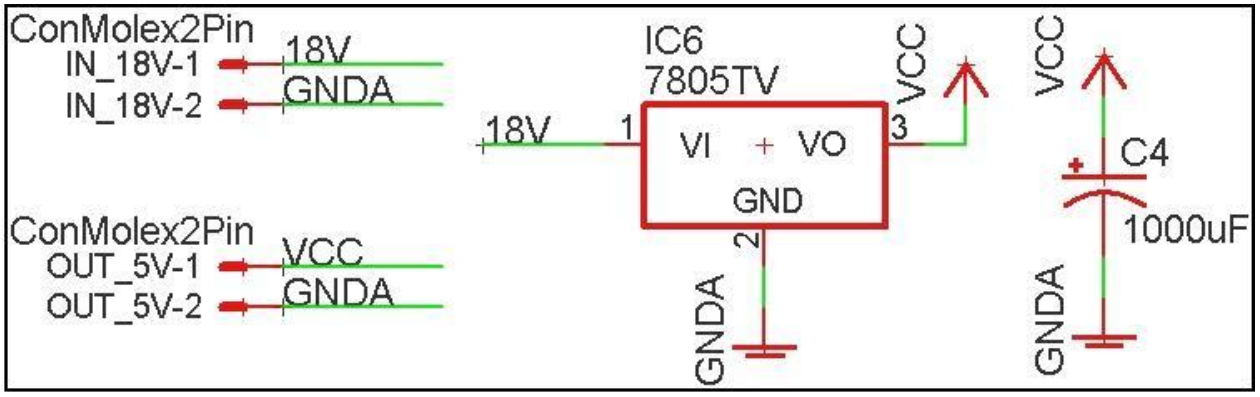

Figura 96: Esquemático da parte de alimentação [01].

O circuito é composto por uma entrada de tensão de 18V que será ligado no barramento de baterias do Robo-Universal, um regulador de tensão para 5V, um capacitor de filtro ligado nos 5V e uma saída de tensão de 5V.

AP E.1.1.1.2 Sensores de pressão

Responsável por linearizar os sinais dos sensores e enviar para a entrada analógica do microcontrolador, esquemático apresentado na Figura 97.

O circuito é composto por duas entradas de três pinos para sensores de pressão, um conversor de tensão de +5V para -5V, usado para linearizar a função dos sensores e dois amplificadores operacionais para converterem o sinal linearizado em tensões analógicas que variam de 0 a +5V, que entrarão no microcontrolador.

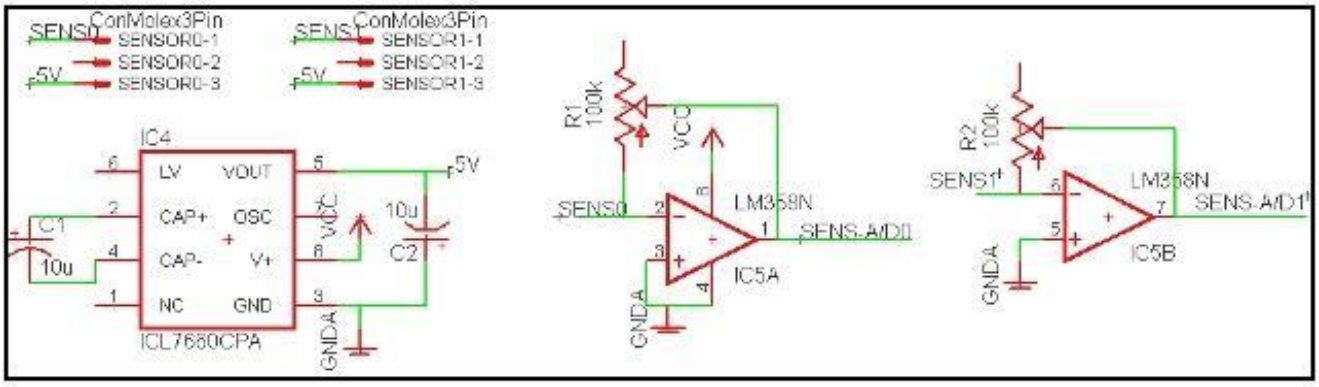

Figura 97: Esquemático da parte de sensores de pressão [01].

AP E.1.1.1.3 Microcontrolador

Conexões com o microcontrolador, seu circuito de *reset e a conexão UART,* mostrado na Figura 98.

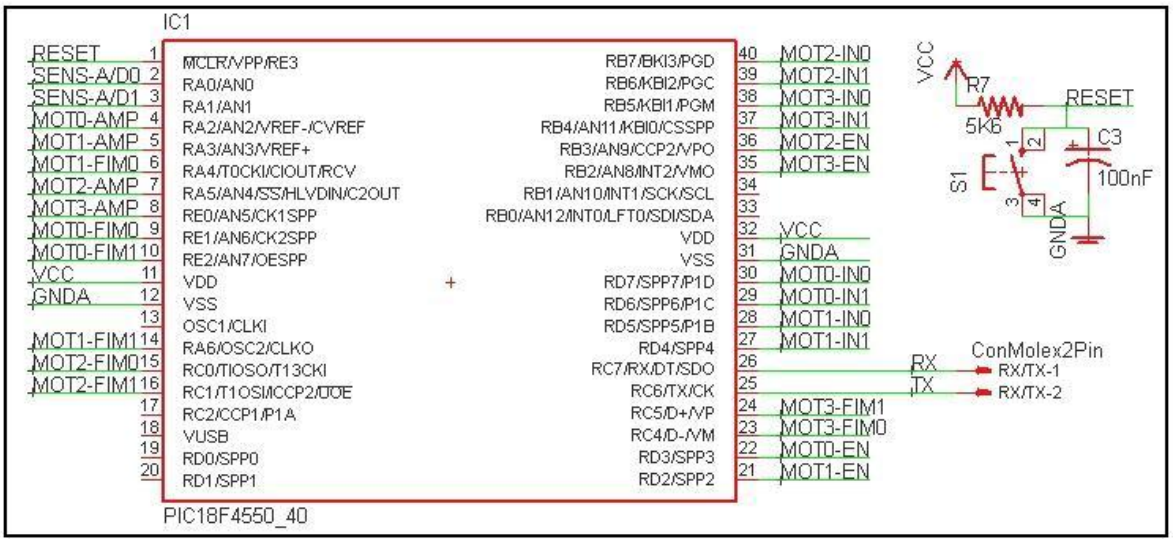

Figura 98: : Esquemático da parte do microcontrolador [01].

AP E.1.1.1.4 Motores

Circuitos que acionam e controlam os motores, mostrados nas Figura 99 e Figura 100.

Circuito composto por quatro entradas de dois pinos cada para leitura dos sensores do fim de curso e dois circuitos integrados que cada um consegue controlar dois motores.

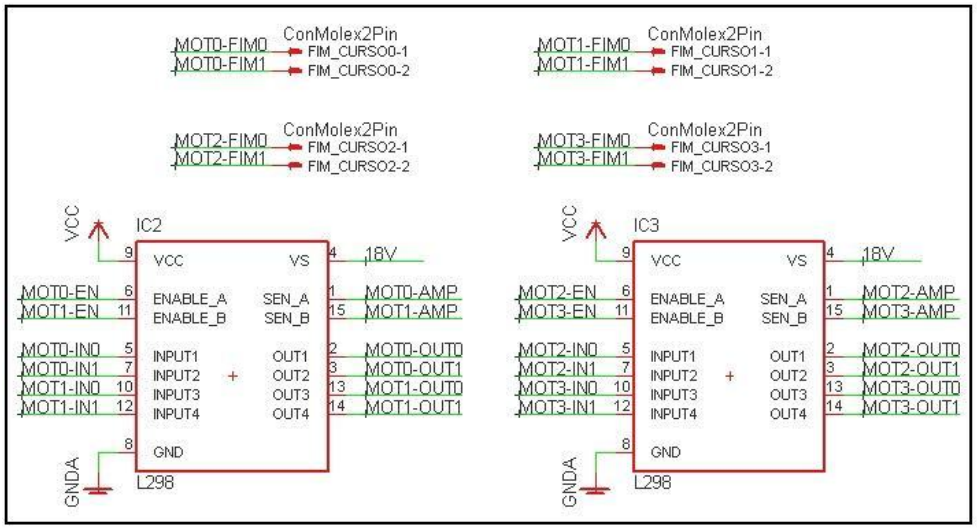

Figura 99: Esquemático dos sensores de fim de curso e dos *drivers* dos motores [01].

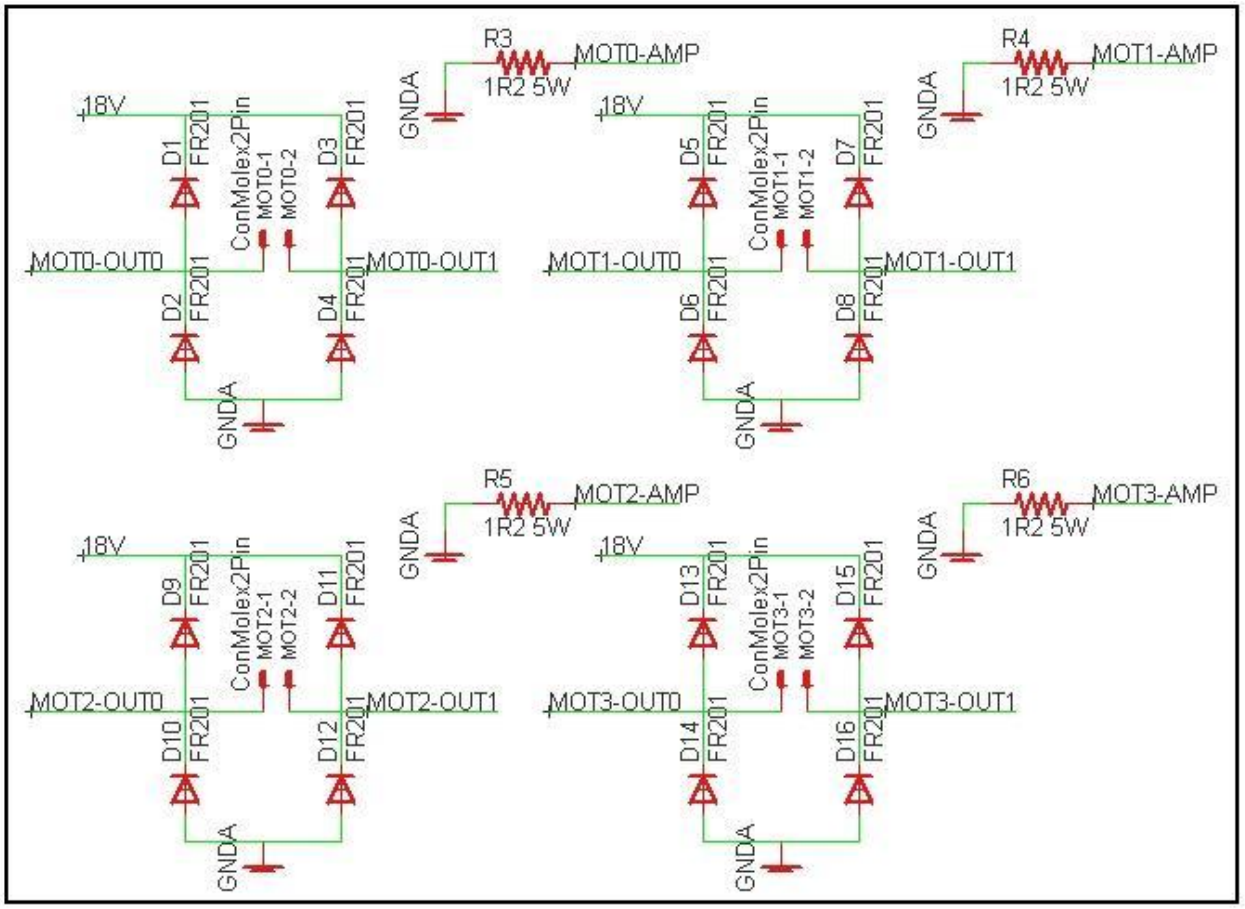

Figura 100: Esquemático da parte de acionamento e potência dos motores [01].

Circuito composto por diodos retificadores para permitirem o acionamento dos motores com a potência necessária e resistores para limitação de corrente.

AP E.1.1.2 *Software* embarcado

O *software* embarcado foi desenvolvido para acompanhar as funcionalidades eletrônicas, portanto ele consegue controlar quatro motores, executa a leitura de dois sensores de pressão, consegue ler a corrente em cada um dos quatro motores e executa a leitura dos oito sensores de fim de curso, fazendo com que os motores parem ao chegarem ao seu limite de movimento.

AP E.1.1.2.1 Funções de Interrupção

O *Software* é composto por funções de interrupção, que fazem o tratamento das interrupções geradas por *timers* e pela comunicação serial, uma interface de comunicação que implementa um protocolo específico que será tratado à frente e pelas seguintes funções:

AP E.1.1.2.1.1 unsigned char motMove(unsigned char mot, unsigned char dir)

Faz algum motor mover-se sem restrição de tempo ou pressão, o movimento cessará somente quando receber um comando para parar ou chegar ao fim de curso.

Parâmetros de entrada:

1 mot: motor que se moverá, (0, 1, 2 ou 3)

2 dir: direção que o motor moverá (0 horário, 1 anti-horário)

Retorna 0xFF caso execute com sucesso ou 0x00 caso falhe.

AP E.1.1.2.1.2 unsigned char motMoveTime(unsigned char mot, unsigned char dir, unsigned char time)

Faz algum motor se mover por um tempo estipulado.

Parâmetros de entrada:

21. mot: motor que se moverá, (0, 1, 2 ou 3)

22. dir: direção que o motor moverá (0 horário, 1 anti-horário)

23. time: tempo que o motor se movimentará em múltiplos de 0.13 seg (ex. se time for igual a 100 o motor se moverá 100 x 0.13 seg = 13 seg).

Retorna 0xFF caso execute com sucesso ou 0x00 caso falhe.

AP E.1.1.2.1.3 unsigned char motMovePress(unsigned char mot, unsigned char dir, unsigned char sens, unsigned short int press)

Faz algum motor se mover até algum sensor encontrar um valor de pressão maior ou igual ao estipulado.

Parâmetros de entrada:

24. mot: motor que se moverá, (0, 1, 2 ou 3)
25. dir: direção que o motor moverá (0 horário, 1 anti-horário)
26. sens: sensor de pressão que será lido (0 ou 1)
27. press: valor de pressão em que o motor deve encerrar seu movimento, varia de 0 a 760

Retorna 0xFF caso execute com sucesso ou 0x00 caso falhe.

AP E.1.1.2.1.4 unsigned char motStop(unsigned char mot)

Cessa o movimento de algum motor ou todos ao mesmo tempo.

Parâmetro de entrada:

28. mot: motor a parar, (0, 1, 2, 3 ou 4 para parar todos simultaneamente)

Retorna 0xFF caso execute com sucesso ou 0x00 caso falhe.

AP E.1.1.2.1.5 unsigned short int readPress(unsigned char sens)

Faz leitura de pressão do sensor estipulado.

Parâmetro de entrada:

9. sens: sensor de pressão que será lido (0 ou 1)

Retorna o valor lido do sensor, varia de 0 a 760.

AP E.1.1.2.1.6 unsigned short int readCurrent(unsigned char mot)

Faz leitura da corrente do motor estipulado.

Parâmetro de entrada:

29. mot: motor que se moverá, (0, 1, 2 ou 3)

Retorna o valor lido pela entrada analógica, pode variar de 0 a 512, é um índice apenas, 0 corresponde a 0 A e 512 corresponde a corrente máxima, que na placa eletrônica está limitada em 2 A, os valores variam linearmente, portanto 256, por exemplo, equivale a 1 A.

AP E.1.1.2.1.7 Protocolo de comunicação

O protocolo de comunicação utiliza a interface de comunicação *IComm*, a mesma utilizada pelo Robo-Universal, o protocolo segue o seguinte padrão:

SYNC	SYNC	STX	id	size	D1	...	Dsize	chksum	EOT

Onde:

1. SYNC é um caractere de sincronismo
2. STX é o indicador de começo de texto (*start of text*)
3. id é o identificador do pacote
4. size é o tamanho da mensagem
5. D1 a Dsize são os dados da mensagem
6. chksum é o *checksum,* para saber se o pacote chegou sem erro
7. EOT é o indicador de fim de texto (*end of text*)

Além desse protocolo, a garra possui um protocolo de comandos, que é colocado dentro dos dados da mensagem (D1 a Dsize), esse protocolo é mais simples e possui o seguinte padrão:

size	cmd1	...	cmd size

Onde:

1. size é o tamanho do comando
2. cmd1 até cmd size são os *bytes* referentes ao comando que se deseja enviar

Os comandos seguem o seguinte protocolo:

AP E.1.1.2.1.7.1 GripperId

Comando para identificação do dispositivo.

Comando:

00	01

Resposta:

C0	01	47	52	49	50	50	45	52

AP E.1.1.2.1.7.2 GripperMove

Faz a garra executar movimento longitudinal.

Comando:

01	01	dir

Resposta:

C1	01	done

Onde:

dir: indica a direção do movimento, 00 – frente ou 01 – trás

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.3 GripperCloseOpen

Faz a garra executar movimento de pinça.

Comando:

01	02	dir

Resposta:

C1	02	done

Onde:

dir: indica a direção do movimento, 00 – abre ou 01 – fecha

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.4 GripperMoveTime

Faz a garra executar movimento longitudinal por tempo determinado.

Comando:

01	03	dir	time

Resposta:

C1	03	done

Onde:

dir: indica a direção do movimento, 00 – frente ou 01 – trás

time: é o tempo de execução do movimento, múltiplos de 0.13 seg (ex. se time for igual a 100 o motor se moverá 100 x 0.13 seg = 13 seg), pode variar de 1(0.13s) a 255(33.15s)

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.5 GripperCloseOpenTime

Faz a garra executar movimento de pinça por tempo determinado.

Comando:

01	04	dir	time

Resposta:

C1	04	done

Onde:

dir: indica a direção do movimento, 00 – abre ou 01 – fecha

time: é o tempo de execução do movimento, múltiplos de 0.13 seg (ex. se time for igual a 100 o motor se moverá 100 x 0.13 seg = 13 seg), pode variar de 1(0.13s) a 255(33.15s)

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.6 GripperCloseUntilPress

Faz a garra fechar a pinça até a pressão no sensor do "dedo" metálico ser maior ou igual a estipulada.

Comando:

01	05	press-H	press-L

Resposta:

C1	05	done

Onde:

press-H: *byte* mais significativo do valor da pressão

press-L: *byte* menos significativo do valor da pressão

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.7 GripperMoveStop

Faz a garra parar o movimento longitudinal.

Comando:

01	06

Resposta:

C1	06	done

Onde:

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.8 GripperCloseOpenStop

Faz a garra parar o movimento de pinça.

Comando:

01	07

Resposta:

C1	07	done

Onde:

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.9 GripperStop

Faz a garra parar todos os movimentos (de pinça e longitudinal).

Comando:

01	08

Resposta:

C1	08	done

Onde:

done: é a indicação da execução, FF – sucesso ou 00 – falha

AP E.1.1.2.1.7.10 GripperGetPress

Faz a leitura da pressão do sensor localizado na ponta do "dedo" metálico.

Comando:

01	09

Resposta:

C1	09	press-H	press-L

Onde:

press-H: *byte* mais significativo do valor da pressão

press-L: *byte* menos significativo do valor da pressão

AP E.1.1.2.1.7.11 GripperGetFimCurso

Faz a leitura dos sensores de fim de curso.

Comando:

01	0A

Resposta:

C1	0A	value

Onde:

value: é uma composição de *bits* que indicam a situação de cada sensor de fim de curso, 0 caso esteja ativado impedindo o movimento e 1 caso esteja desativado permitindo o movimento.

value é composto assim:

bit7: movimento de pinça fechando

bit6: movimento de pinça abrindo

bit5: movimento longitudinal para trás

bit4: movimento longitudinal para frente

bit3-bit0: valem 0

APÊNDICE F.
PROJETO MECÂNICO

AP F.1 PROJETO MECÂNICO

AP F.1.1 Sistema de locomoção

AP F.1.1.1 Mecanismo moto-redutor

O mecanismo propulsor do robô é constituído basicamente pelos motores e seus respectivos redutores. As Figura 101 e Figura 102 mostram a parte de trás onde são localizados os motores e redutores.

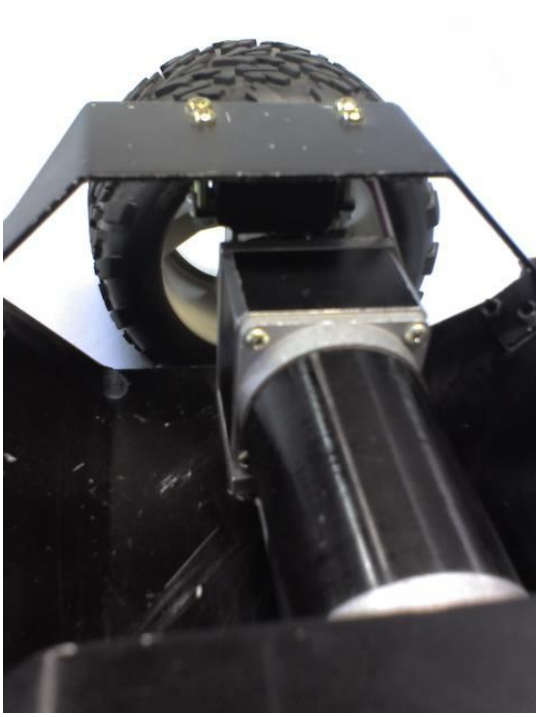

Figura 101: Motor e Redutor [01].

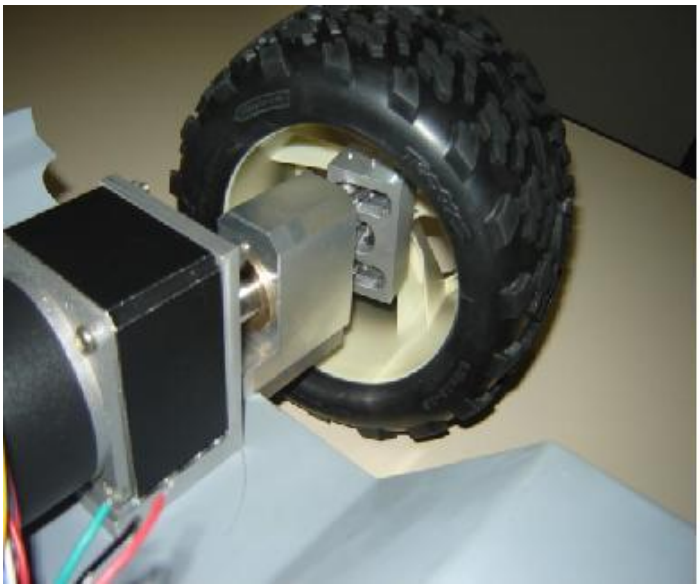

Figura 102: Mecanismo de locomoção [01].

Os motores utilizados para a tração do Robô são do tipo *Brushless*, Esses motores possuem acoplados redutores 1:30, conforme ilustra Figura 103 com as dimensões indicadas na Figura 104. A tensão nominal é de 12V com capacidade de corrente máxima de 2,1A a 70 rpm. Na Tabela 13 temos as características do conjunto motor/redutor sem carga, na Tabela 14 as especificações para o máximo rendimento e finalmente a Tabela 15 a máxima potência que podemos alcançar com ele.

Tabela 13: Características do motor brushless sem carga

Rotação	**Corrente**
109 rpm	**1,36 A**

Tabela 14: Máximo rendimento do conjunto com 51,1% da potência máxima

Rotação	Corrente	Torque	Potência
86 rpm	1,36 A	9,3 Kgf.cm	8,4 W

Tabela 15: Máxima potência alcançada pelo conjunto motor/redutor

70 RPM	2,1 A	16,50	12,1 W
Rotação	Corrente	Torque	Potência

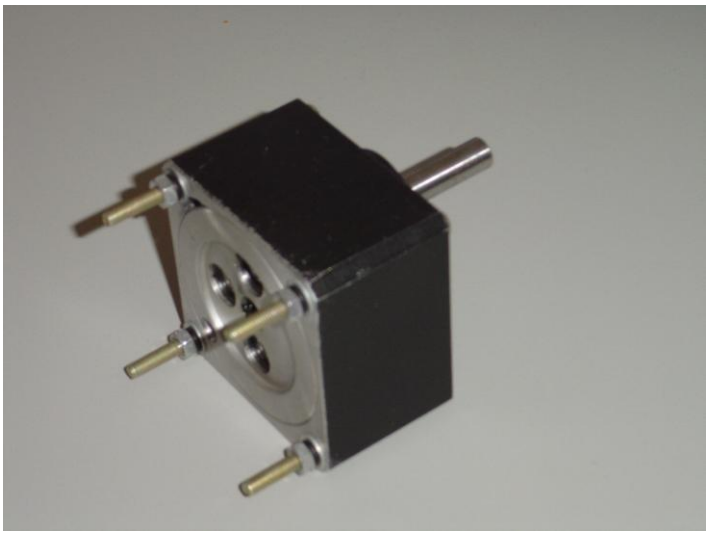

Figura 103: Redutor que será acoplado ao motor de *Brushless*.

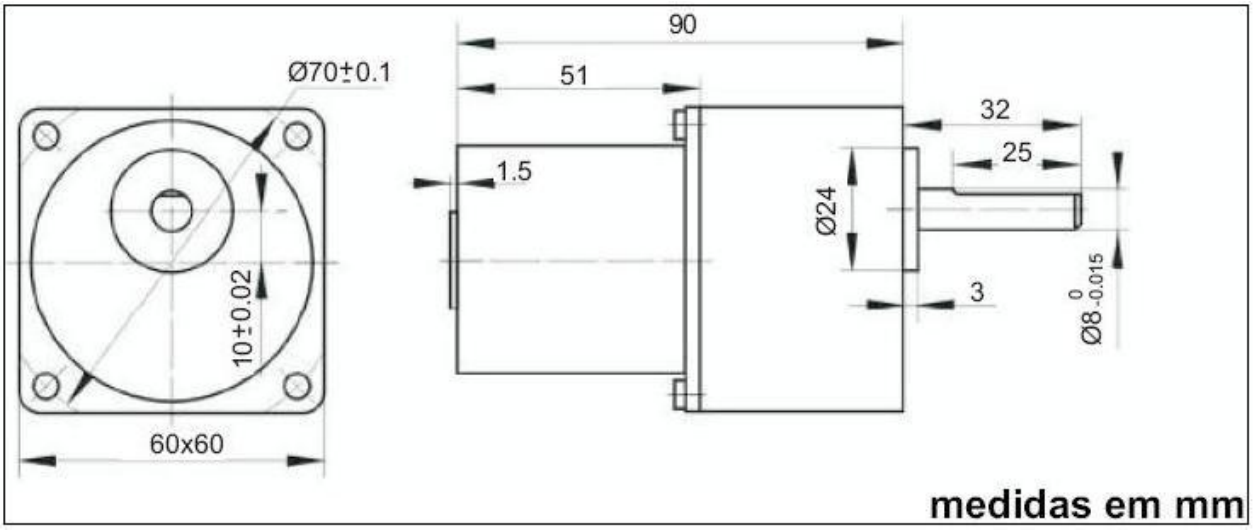

Figura 104: Motor de *Brushless*, que vai acoplado ao redutor 1:30.

AP F.1.1.2 Mecanismo de esterçamento

O esterçamento das rodas do robô é feito por meio de quatro servos-motores. O modelo de servo utilizado é o S3003 da marca Futaba mostrado nas Figura 105 e Figura 106. Suas características, de acordo com o fabricante, são mostradas na Tabela 16.

Tabela 16: Servo-motor: especificações

	4,8 V	6,0 V
Torque Nominal	**320**	**410**
Velocidade	**0,23**	**0,19**
Massa (g)	**370**	

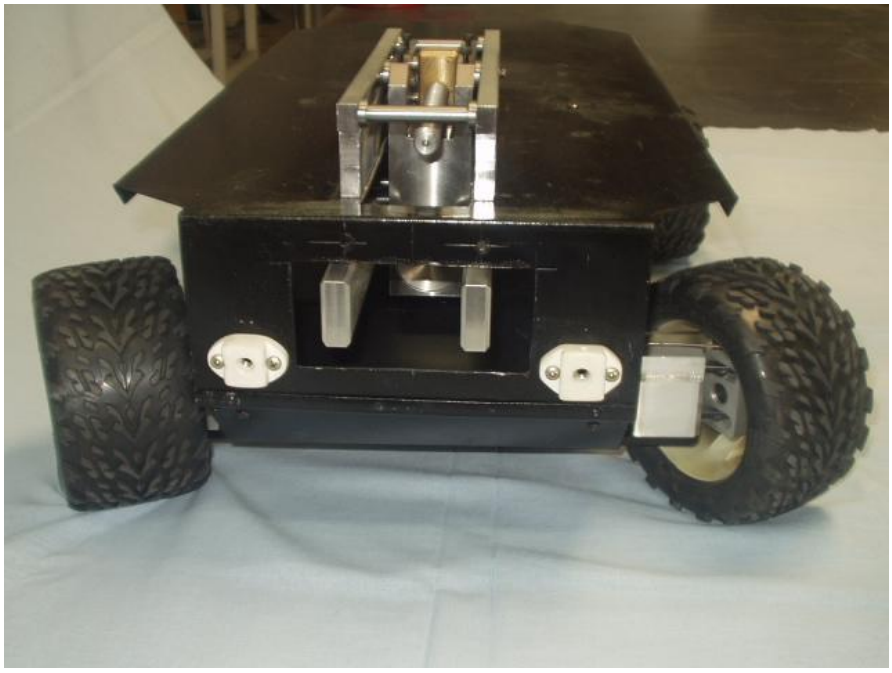

Figura 105: Mecanismo de Esterço [01].

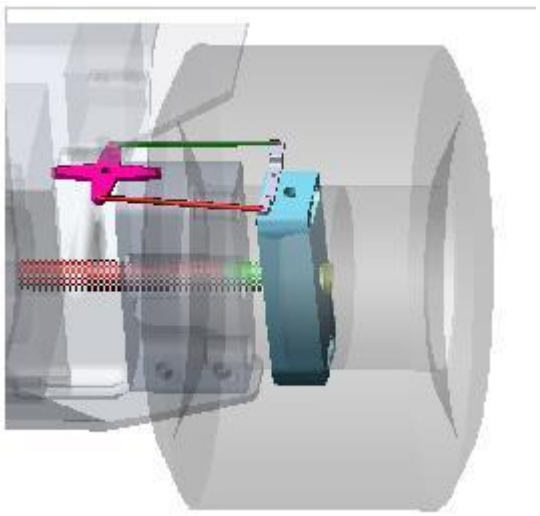

Figura 106: Modelagem do Mecanismo de Esterço [01].

AP F.1.2 Pinça mecânica

O Módulo Pinça Mecânica é configurado como um *feature* – um adicional para mostrar a condição de expansão da plataforma robótica.

Considerando as condições de projeto:

1. Peso a ser movimentado = 400gr máximo
2. Produto sem movimentação durante a operação da Pinça 'acoplar – desacoplar' - redução de energia
3. Sistema construtivo tipo rampa para levantar / abaixar o peso
4. Pinça robusta
5. Pontas (mandíbulas) tipo substituíveis. Poderão ser desenhados diferentes tipos de pontas em função da aplicação.

Defini uma Pinça Frontal Tipo Mandíbula. Nas fotos das Figura 107, Figura 108 e Figura 109 pode-se ver o mecanismo da pinça sem as pontas que poderão ser trocadas conforme a necessidade. A pinça tem três posições uma escondida dentro da carenagem, outra fora num ponto intermediário e ainda elevada e uma terceira abaixada e totalmente fora. Ela abre e fe-

cha para pegar o objeto, e logo se recolhe no ponto intermediário elevando-se para transportar o objeto.

Figura 107: Robô mostrando a posição da pinça mecânica [01].

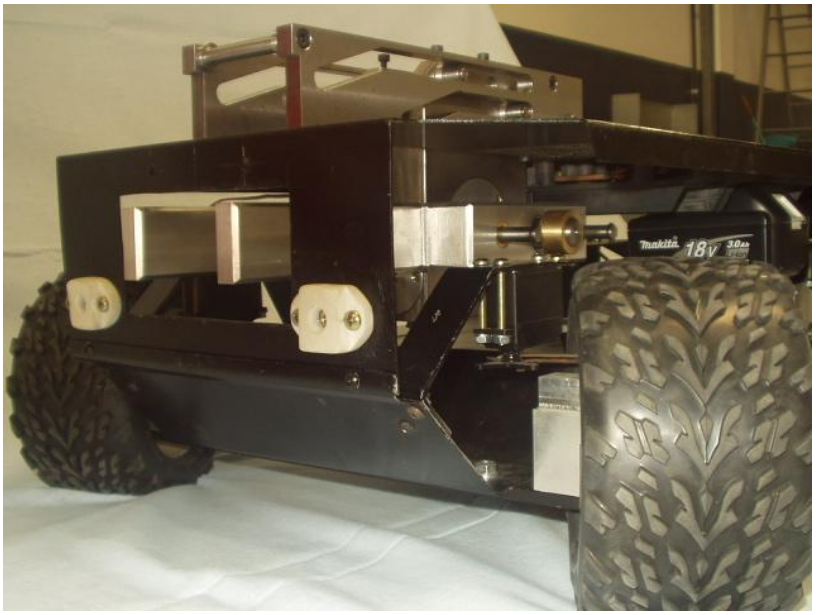

Figura 108: Mecanismo da pinça parte inferior [01].

Figura 109: Foto mostrando o mecanismo da pinça parte superior [01].

AP F.1.3 Projeto do Chassi

Considerando que a plataforma robótica irá atingir um público estudantil jovem – Escolas Técnicas Nível Médio, que irão manuseá-lo, desmontá-lo, agregar modificações, participação em competições - o conceito robustez é tratado como elemento importante.

Busquei dar ao produto aspecto de resistência física, formas encorpadas, volume, não ser um produto 'leve', que ao ser utilizado ou manuseado tenha a sensação de peso, materiais com espessuras adicionais, aspecto reforçado, etc.

Na mesma linha de robustez, o Chassi também é desenvolvido como elemento estrutural importante no projeto.

Considerando os vários elementos de locomoção, desenhei um Chassi em chapa de aço com vincos diversos, garantindo maior resistência mecânica ao conjunto.

A opção por *pack* de baterias recarregáveis contribuiu para consolidar este conceito de produto pesado, aliado ao design final, rodas grandes, chassi metálico e carenagem com sobre-espessura, garantiram o aspecto de robustez objetivado.

AP F.1.3.1 Funções do produto

Para o projeto dividi em três níveis as funções do chassi, distribuindo as partes mais pesadas no nível inferior e as mais leves no nível superior, conforme a seguir:

Nível I - Fontes de Energia = Baterias = maior peso = maior estabilidade

Nível I - Mecanismos de movimentação = sistema mecânico + moto-redutores

Nível I - Módulo Pinça – Garra Frontal Tipo Mandíbula

Nível II - Inteligência e Elementos de Potência

Nível III - Topo = Plataforma Universal para agregar os Módulos Visão + Opcional usuário.

Resultado geral

Chassi em Aço espessura 2,0mm, com desenho (vincos) que aumentem sua resistência mecânica

Pinça frontal em Aço Tipo Mandíbula

Divisão intermediária em Aço espessura 1,2mm, com vincos de reforço

Carenagem em poliestireno de Alto Impacto (PSAI) moldado em vácuo-form, com espessura de 4mm

As Figura 110, Figura 111e Figura 112 mostram as partes do robô em varias posições.

Figura 110: Robô mostrando a carenagem [01].

AP F.1.4 Protótipo

Fotos do protótipo são mostradas nas figuras Figura 110 ate Figura 113:

Figura 111: Foto mostrando o protótipo sem carenagem [01].

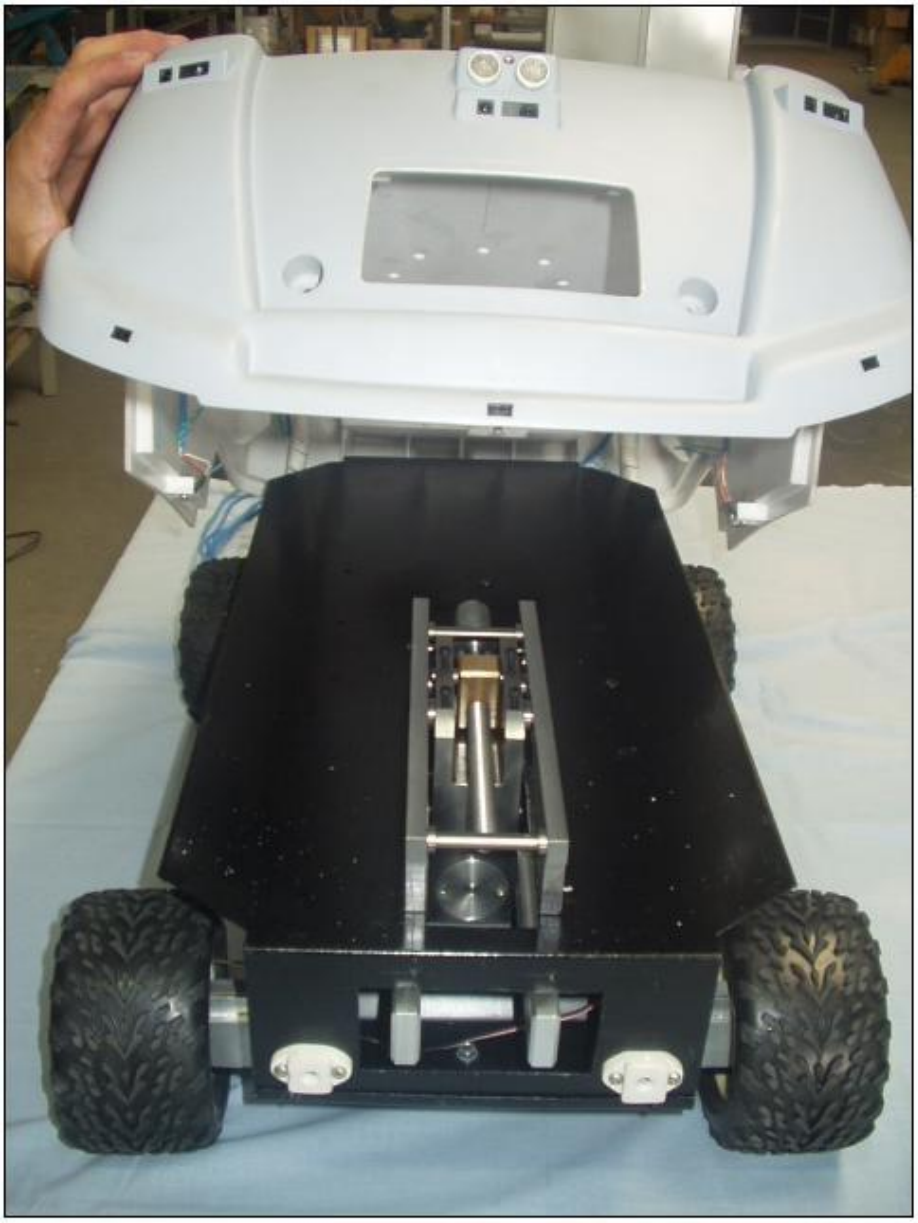

Figura 112: Foto mostrando o robô com a carenagem levantada [01].

Figura 113: Foto mostrando o robô de cima [01].

APÊNDICE G.

MODELAGEM MATEMÁTICA [01]

AP G.1 MODELAGEM MATEMÁTICA – LINEARIZAÇÃO

AP G.1.1 Linearização do Modelo para Sintetização de Controlador

Para o desenvolvimento do controlador de trajetória do robô foram fornecidas as matrizes de estado *A*, a matriz de controle *B* e a matriz de saída *C*. A matriz *D* é a matriz de realimentação.

$$\dot{x} = [A]x + [B]u$$
$$y = [C]x + [D]u$$

A linearização do modelo no *software* Adams foi realizada para quatro pontos de operação, relativo a quatro atitudes do robô (0°, 90°, 180° e 270°), como pode ser visto na Figura 114.

Esse modelo linearizado poderá ser importado para dentro do ambiente do *software* Matlab/Simulink, Octave ou SCiLab para a sintetização do controlador a ser desenvolvido. (Figura 115).

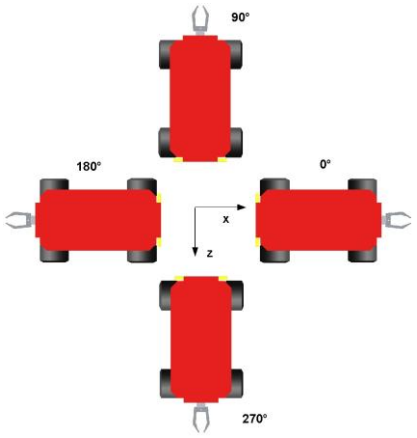

Figura 114: Posições do robô para a linearização e geração das matrizes [01].

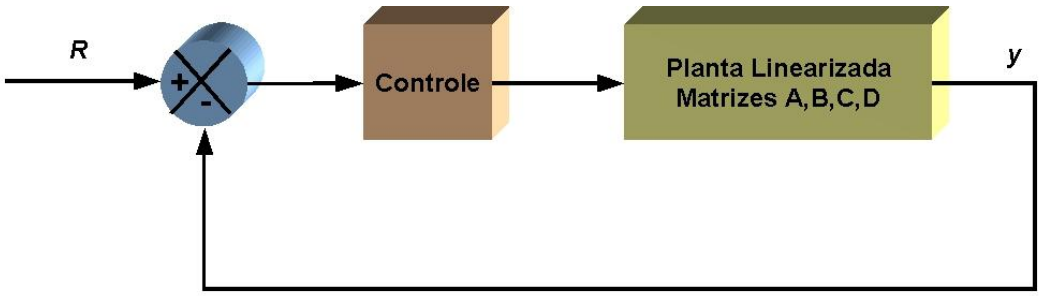

Figura 115:Modelo linearizado importado para dentro do ambiente do Matlab/Simulink [01].

O modelo do robô possui 42 estados, portanto, a matriz característica A gerada pela linearização do modelo é uma matriz de 42 x 42 elementos.

Foram também geradas as matrizes B, C e D do modelo linearizado nas quatro posições.

As entradas e saídas definidas no modelo podem ser vistas abaixo.

Saídas da planta do modelo linearizado Adams:

1) Velocidade da Roda direita
2) Velocidade da Roda esquerda
3) Posição roda FD (frente-direita)
4) Posição roda FE
5) Posição roda AD
6) Posição roda AE

As entradas da planta do modelo linearizado Adams:

1) Velocidade do motor direito
2) Velocidade do motor esquerdo
3) Torque do motor direito
4) Torque do motor esquerdo
5) Posição roda FD (frente-direita)
6) Posição roda FE

7) Posição roda AD

8) Posição roda AE

AP G.1.2 Formulação das Forças no Pneu Fiala

Ref.: [6] até [18]

Estudo das formulações dos seguintes tipos de forças:

Força Normal da pista no pneu

Força Longitudinal

Força Lateral

Momento de Resistência ao Rolamento

Momento de Alinhamento

Suavização das Forças Transientes Iniciais

AP F.1.3 Força Normal da Pista no Pneu

A força normal que uma pista exerce em um pneu na região de contato, utilizando o sistema de coordenadas SAE (+Z para baixo) é sempre negativa (apontada para cima). A força normal é:

$$F_z = \min (0.0, \{F_{zk} + F_{zc}\})$$

onde:

F_{zk} é a força normal devido à rigidez vertical do pneu

F_{zc} é a força normal devido ao amortecimento vertical do pneu

F_{zk} = - rigidez vertical × Spen

F_{zc} = - amortecimento vertical × Vpen

Em vez da rigidez vertical linear do pneu, existe também uma deflexão arbitrária do pneu – a curva de carga pode ser definida no arquivo de propriedade do pneu, na secção *[DEFLECTION_LOAD_CURVE]*. Se uma secção chamada [DEFLECTION_LOAD_CURVE]

existir, os dados da deflexão da carga apontam para uma spline cúbica - e é feita uma extrapolação para o cálculo da força vertical do pneu. Note que deve ser especificada a rigidez vertical [*VERTICAL_STIFFNESS*] no arquivo de propriedade do pneu, mas ela não tem nenhuma função.

AP G.1.4 Força Longitudinal

A força longitudinal depende da força vertical (F_z), do coeficiente de atrito (U), o *slip ratio* longitudinal (S_s), e o *slip angle* (Alpha). O coeficiente de atrito depende dos coeficientes estático (UMAX) e dinâmico (UMIN) e o *slip ratio* detalhado (S_sAlpha).

UMAX especifica o coeficiente de atrito pneu/pista sem escorregamento e representa o coeficiente de atrito estático. Este é a intercessão em Y da curva do coeficiente de atrito versus escorregamento. Note que este valor máximo não é possível de ser obtido, uma vez que sempre há escorregamento na *footprint* (área de contato). Esse valor é utilizado, juntamente com o UMIN, para definir uma relação linear entre atrito versus escorregamento. UMAX será, geralmente, maior que UMIN.

UMIN especifica o coeficiente de atrito pneu/pista para o caso de escorregamento total e representa o coeficiente de atrito de escorregamento. Este é o coeficiente de atrito para 100% de escorregamento, ou escorregamento puro. Esse parâmetro é utilizado juntamente com o UMAX para definir uma relação linear entre o atrito e o escorregamento.

O atrito detalhado (S_sAlpha):

$$S_s\text{Alpha} = \left(S_s^2 + \tan^2(\text{Alpha}) \right)^{1/2}$$

O valor do coeficiente de atrito (U):

$$U = U\_max - (U\_max - U\_min) \times S_s\text{Alpha}$$

O Fiala define um escorregamento longitudinal crítico (S_critical):

$$S\_critical = \left| \frac{U \times F_z}{(2 \times CSLIP)} \right|$$

Este é o valor do escorregamento longitudinal no qual o pneu está escorregando.

Caso 1. Condição de Deformação Elástica: $|S_s| < S\_critical$

$F_x = -CSLIP \times S_s$

Caso 2. Condição de Escorregamento Completo: $|S_s| > S\_critical$

$F_x = -sign(S_s)(F_{x1} - F_{x2})$

onde:

$$F_{x1} = U \times F_z$$

$$F_{x2} = \left| \frac{(U \times F_z)^2}{(4 \times |S_s| \times CSLIP)} \right|$$

AP G.1.5 Força Lateral

Assim como a força longitudinal, a força lateral depende da força vertical (F_z) e do coeficiente de atrito que está atuando no momento (U). Também similar ao cálculo da força longitudinal, o Fiala define um escorregamento lateral crítico (Alpha_critical):

$$\text{Alpha\_critical} = \arctan\left(\frac{3 \times U \times |F_z|}{CALPHA}\right)$$

O pico da força lateral é igual a $U \times |F_z|$ quando o *slip angle* (Alpha) se iguala ao *slip angle* crítico (Alpha_critical).

Caso 1. Condição de Deformação Elástica: $|Alpha| \leq Alpha\_critical$

$F_y = - U^* |F_z| * (1-H^3) * sign(Alpha)$

onde:

$$H = 1 - \frac{CALPHA \times |\tan(Alpha)|}{3 \times U \times |F_z|}$$

Caso 2. Condição de escorregamento: $|Alpha| > Alpha\_critical$

$F_y = -U * |F_z| * sign(Alpha)$

AP G.1.6 Momento de Resistência ao Rolamento

Quando o pneu rola para frente (sentido horário): T_y = -*rolling_resistance* * F_z

Quando o pneu rola para trás (sentido anti-horário): T_y = *rolling_resistance* * F_z

AP G.1.6.1 Momento de Alinhamento

Caso 1. Condição de Deformação Elástica: |Alpha| ≤ Alpha_critical

Mz = U * |Fz| * WIDTH * (1-H) * H3 * sign(Alpha)

Caso 2. Condição de Escorregamento Completo: |Alpha| > Alpha_critical

M_z= 0.0

AP G.1.6.2 Suavização das Forças Transientes Iniciais

ADAMS/Tire pode minimizar as forças iniciais transientes do pneu nos primeiros 0.1 segundos de simulação. As forças longitudinal e lateral, e o torque alinhante são multiplicados por uma função degrau do tempo de terceira ordem.

Força Longitudinal: $F_{Lon} = S*F_{Lon}$

Força Lateral: $F_{Lat} = S*F_{Lat}$

Torque Alinhante: $M_z = S*M_z$

O parâmetro [*USE_MODE*], no arquivo de propriedade do pneu, permite que a suavização seja habilitada ou desabilitada:

USE_MODE = 1, suavização desabilitada;

USE_MODE = 2, suavização habilitada.

ANEXOS

ANEXO I – SENSORES E ATUADORES

ANEXO I

SISTEMAS DE SENSORES E ATUADORES

AI.1 SENSORES

A I.1.1 Acelerômetro

O acelerômetro Hitachi H48C 3-Axis é um módulo integrado que pode sensoriar a força gravitacional (g) de +/- 3g nos três eixos (X, Y e Z). O módulo possui *onboard* um regulador de 3.3 volts para alimentar o condicionador de sinal analógico H48C, e o conversor analógico/digital MCP3204 (quatro canais, 12 bits) usado para ler a saída de tensão do H48C. O módulo possui uma interface serial sincrônica para a aquisição dos dados.

Características:

Faixa de medição de ±3 g nos três eixos

Usa tecnologia MEMS (Micro Electro-*Mechanical System*), com compensação para operação em *calibration-free*.

Regulador *Onboard* e *high-resolution* ADC para conexão direta ao microcontrolador *host,* utilizando o protocolo de comunicação SSP *(Serial Syncronous Protocol).*

Indicação de saída *Free-fall* simultaneamente 0g nos três eixos

Pequena placa de circuito impresso de dimensões 0.7" x 0.8" (17.8 mm x 20.3 mm)

Faixa de operação: -25° to 75° C

A Figura 116, ilustra a placas do módulo acelerômetro de 3 eixos da marca Hitachi H48C.

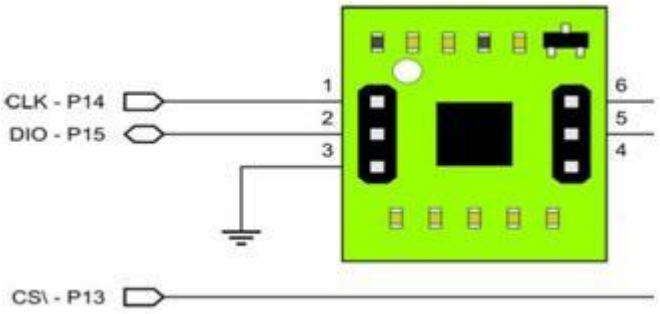

Figura 116: Placa do circuito do módulo Hitachi H48C

No presente projeto o acelerômetro poderá ser usado para detectar se os comandos enviados aos atuadores, como os motores de tração, por exemplo, estão sendo executados corretamente sem nenhum distúrbio aparente. Ele fará parte dos ferramentais disposto ao usuário para montar suas aplicações em programação do sistema.

A I.1.2 *Encoders* duas rodas

Os motores brushless adquiridos estão acoplados a um sensor Hall de três fases. Estas 3 saídas são ligadas ao *driver*, que converte a rotação do motor em pulsos a uma taxa de 2 pulsos por revolução e é usada pelo *driver* para realizar o acionamento sincronizado das bobinas do estator. A Figura 117 mostra a topologia de ligação do sensor Hall cujos terminais são ligados no *driver* .

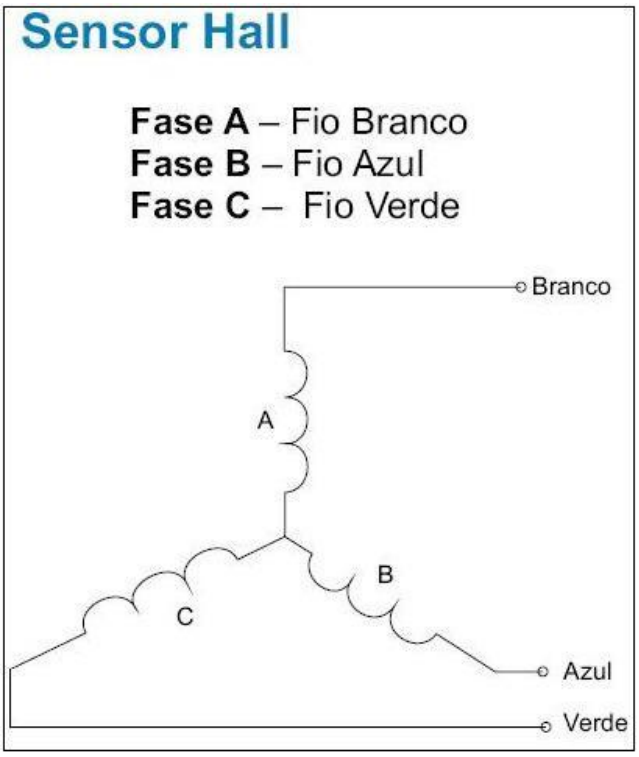

Figura 117: Topologia das Ligações das 3 fases do sensor Hall

A I.1.3 Sensor de proximidade por reflexão

O sensor em questão (mostrado na Figura 118) mede a distância por infravermelho que indica, mediante uma saída analógica, a distancia medida. A tensão de saída varia de forma não linear quando se detecta um objeto numa distância próxima ao robô. A saída está disponível de forma contínua e o seu valor é atualizado em um período de tempo fixo, na ordem dos milissegundos.

Normalmente a saída é ligada à entrada de um conversor analógico digital o qual converte a distância num número que pode ser usado pelo microprocessador. A saída também poderia ser usada diretamente num circuito analógico caso não seja necessário tomadas de decisões críticas.

Deve-se estar atento ao fato de que a saída não é linear. Cada um dos sensores utiliza só uma linha de saída para comunicar-se com o processador principal. Esses sensores distribuídos ao redor do robô servirão como auxiliares na percepção do meio, para obstáculos próximos.

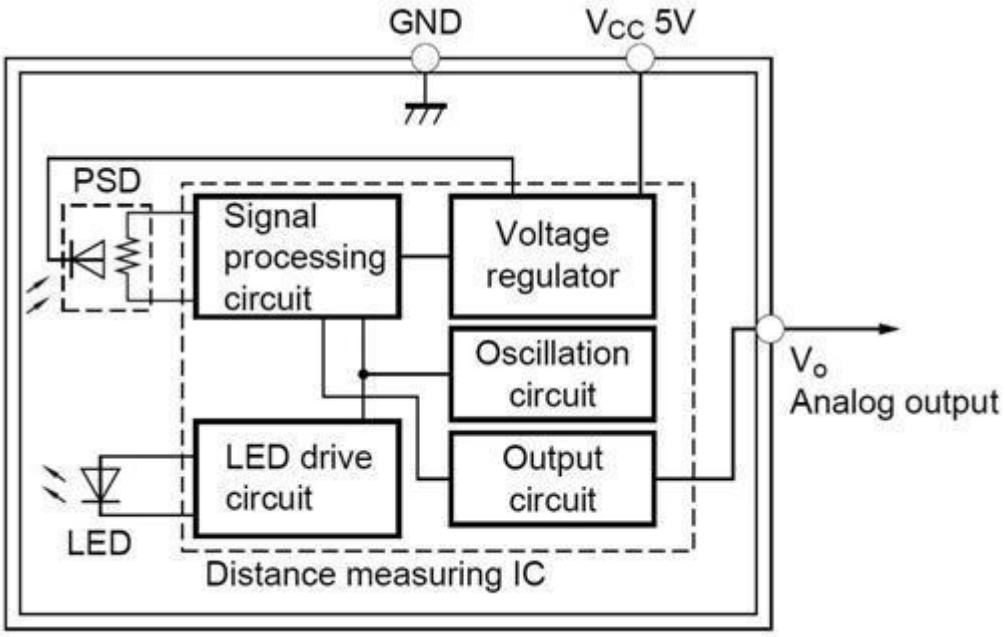

Figura 118:Diagrama de blocos do sensor de proximidade comercial GP2D12.

A I.1.4 Sensor de Ultra-som

Sensores para distância são um dos mais importantes dispositivos de sensoriamento para uma plataforma robótica móvel, pois dá a capacidade de se fazer um mapeamento robusto do ambiente que ele vai explorar capacitando o mesmo a determinar as manobras possíveis de serem executadas traçando o caminho possível. O funcionamento desses sensores, como descrito na Figura 119, se baseia em curtos períodos na emissão do ultra-som, na faixa de freqüência entre 50kHz e 250kHz, e o tempo de resposta, no caso o eco, é medido desde o momento em que é emitido até o seu retorno. O tempo de vôo do eco é proporcional a duas vezes a distância do obstáculo mais próximo. Se nenhum sinal de eco é recebido dentro de certo limite de tempo, então não existe nenhuma barreira obstruindo a passagem. É recomendável usar esses sensores com sensores de proximidade, pois um complementa o outro, caso exista uma barreira com ângulo oblíquo, pode causar distorções na recepção com informações equivocadas a cerca do obstáculo.

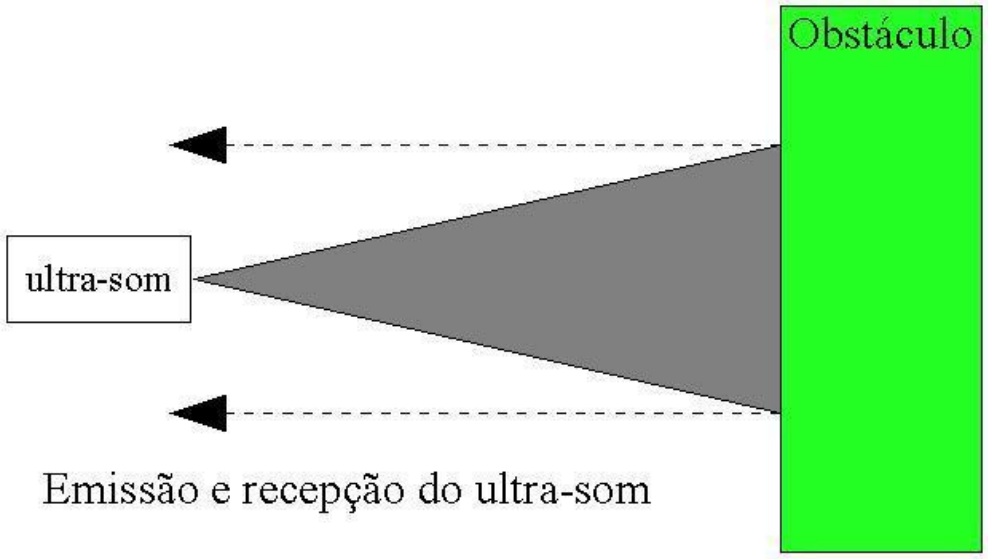

Figura 119: Comportamento do ultra-som desde o emissor até ser refletido pelo obstáculo.

A I.1.5 Sensor Tátil

Um sensor tátil pode funcionar segundo diferentes princípios. Sensores típicos possuem a seguinte estrutura:

a) Uma superfície tátil que consiste de uma matriz de pinos de pressão (piezoelétricos), resistores, capacitores variáveis ou dispositivos óticos.

b) Um transdutor, que converte forças ou momentos locais em impulsos elétricos.

c) Um multiplexador que conecta os elementos da matriz com um circuito de medição e amplificação.

d) Uma interface de controle para trazer os dados medidos para o microprocessador para avaliação.

Existem essencialmente dois tipos de sensores tátis: sensores de toque e de força. Os sensores de toque são usados geralmente para detectar a presença ou ausência de um objeto sem levar em conta a força de contato. Os sensores de força medem forças localizadas. No caso deste trabalho adotei como sensor tátil sistema binário, ou seja, a resposta pode ser apenas 0 ou 1 como mostrado na Figura 120. Eles serão distribuídos ao redor do robô como último re-

curso de detecção de obstáculos, quando todos os anteriores falharem, através da eminência de colisão haverá reflexão do emissor de infra-vermelho por meio do obstáculo no receptor acusando que há uma barreira.

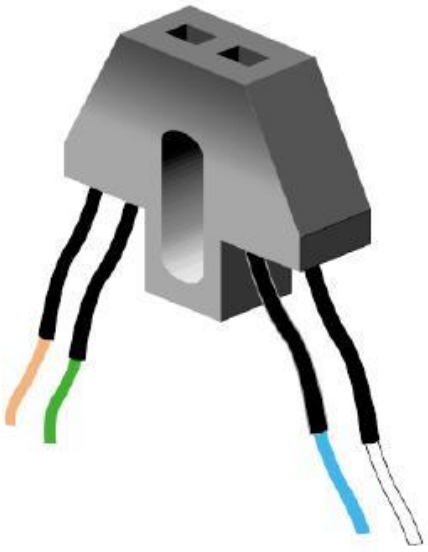

Figura 120: Sensor tátil refletivo da Fairchild modelo QRB1134

A I.1.6 Sensor de Campo Magnético Terrestre

Estes sensores possuem internamente bobinas de compensação, tendo assim boa sensibilidade e imunidade contra possíveis distúrbios de campos magnéticos externos. Os componentes modernos já dispensam o uso de bobinas externas, simplificando então o projeto para aplicações de medição de campos magnéticos mais fracos, como o campo magnético terrestre, por exemplo. Um sensor de campo magnético sensível integra a detecção de duas coordenadas em um só encapsulamento, ideal para aplicação em bússolas. Um sensor desses pode dar condições de o robô andar em linhas retas diminuindo a dependência por sensores menos confiáveis. É sabido que *encoders* nas rodas possuem imprecisões devido a escorregamento nas partidas e frenagem além de erros intrínsecos na leitura dos mesmos. Para se adotar um

sistema mais complexo de referenciamento, envolve algoritmos complexos com sensores de precisão apurada aumentando os custos. O uso de uma bússola digital estaria eliminando todos esses elementos de dificuldade simplificando o projeto. A Figura 121 ilustra as placas do R117-*Compass*.

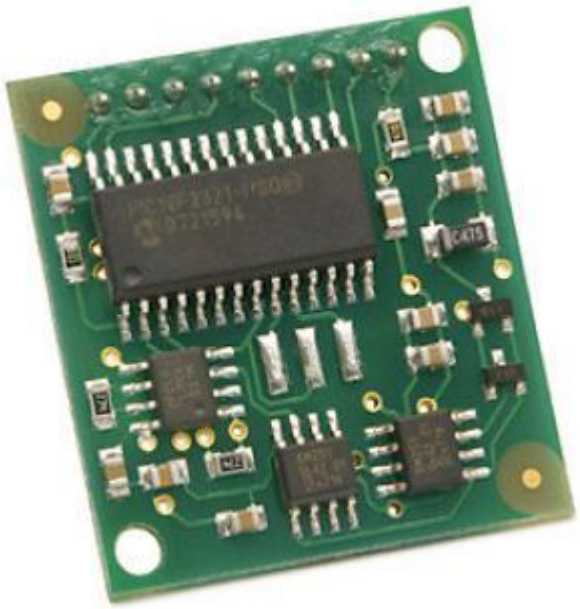

Figura 121: Placas de sensor de campo magnético terrestre modelo R117- Compass

A I.1.7 Módulo GPS da Hobby Engineering

O GPS *Receiver Module* proporciona uma informação standard no formato NMEA0183 (*National Marine Electronics Association*) ou dado especificados pelo usuário via interface de comando serial, detecta e segue até 12 satélites, e WAAS/EGNOS (*Wide Area Augmentation System/European Geostationary Navigation Overlay Service*) para um resultado com maior precisão de posicionamento.

O Módulo proporciona a hora atual, data, latitude, longitude, altitude, velocidade, direção e orientação do movimento, alem de outros dados, e pode ser usado em uma larga variedade de

aplicações hobbistas e comerciais, incluindo navegação, sistemas de rastreamento, mapeamento, gerenciamento de frota, piloto automático e robótica.

Características principais do módulo:

a. Módulo receptor de GPS de baixo custo totalmente integrado com antena passiva;

b. Interface serial TTL de 4800 baud e *Single-wire* para BASIC *Stamp*, SX, *Propeller* e outros processadores;

c. Proporciona informação no formato NMEAO183 ou num formato específico de dados requisitados via interface de comandos;

d. Requer somente alimentação de +5VDC @ 115mA (típico);

e Espaçamento de pinos de “0.100” para fácil prototipagem e integração.

O Módulo receptor GPS é integrado no circuito utilizando três linhas de conexão como mostra na Figura 122.

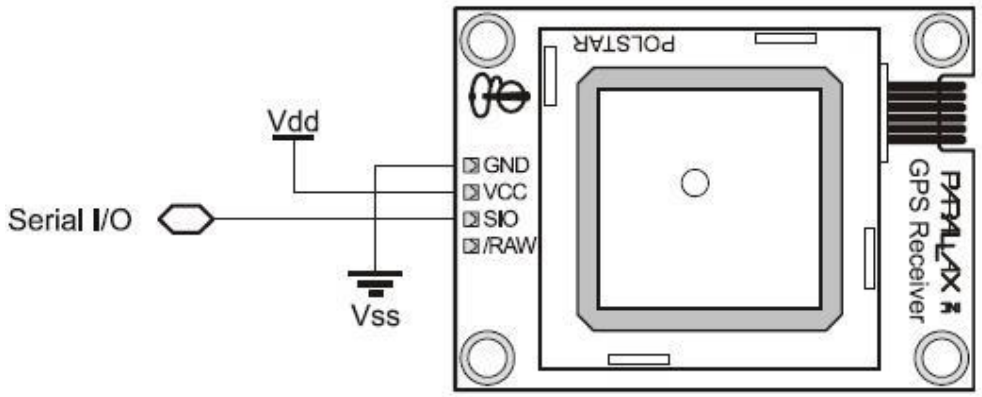

Figura 122: Conexão do GPS

O Módulo é montado em forma horizontal com a face da antena direcionada para o céu, este módulo deve ser usado em áreas não cobertas para que possa detectar os satélites, como todo módulo de GPS, não sendo uma limitação exclusiva deste módulo.

A I.1.8 Sensores de Temperatura e Umidade

A umidade é notoriamente difícil de ser medida. Alguns dos cientistas e eletrônicos consideram que a medição de umidade inclui: uma interface Analógico/Digital e um circuito externo o qual pode requerer *op-amps* ou circuito oscilador; ajuste de compensação de temperatura para calcular o *dew point* (temperatura para o qual o ar se torna saturado); calibração contra fontes de umidade conhecidas; montagem, proteção e tempo de resposta na situação real de uso. Detalhes podem ser vistos no *web site da EME Systems.*

O *Sensirion SHT1x* escolhido para este trabalho cobre estes requerimentos citados no parágrafo anterior num produto compacto de dimensões pequenas. Trata-se de um sensor inteligente para a umidade e a temperatura, que incorpora a interface analógica/digital. Tudo o que o microcontrolador deve fazer é ler a través de uma interface serial digital de duas linhas a umidade e a temperatura. O único requerimento matemático é uma calibração de escala e um offset. O SHT1x vem calibrado de fábrica e retorna os valores de temperatura com uma resolução de 0.01 graus Celsius e a umidade relativa com uma resolução de 0.03%. A precisão no pior caso chega a +/- 2 graus Celsius, mas para temperatura ambiente a precisão é menor que +/- 1 grau Celsius. Para a medição da umidade a precisão é de +/- 3.5% dentro da faixa de 20% a 80%.

As Figura 123 mostra uma foto do sensor SHT1x.

Figura 123: Sensor SHT1x

AI.2 ATUADORES

A I.2.1 *Driver*s de Motores

No presente trabalho, para a tração do robô foi adotado dois motores *brushless*. Esses motores demandam *drivers* especiais, pois o acionamento de suas bobinas depende de uma série de fatores para que o motor funcione, como a velocidade da rotação e o torque.

Foi escolhido dois *drivers* da Akiyama AKDBL12-30W, com alimentação de 12V e potência 30W para acionamento dos motores. Conforme indicado na Figura 125 as entradas de controle desse *driver* consiste em pulsos PWM ou nível de tensão para controlar a velocidade, além disso possui entrada para indicar o sentido da rotação, freio e uma entrada de *enable*. Existem as entradas do *driver* referente ao sensor Hall de três fases acoplado diretamente no eixo do motor, eles

são usados pelo *driver* para poder executar adequadamente o controle de acionamento das bobinas do motor.

A Figura 124 mostra uma foto do *driver* com seus conectores, a pinagem dos conectores e as características do drive.

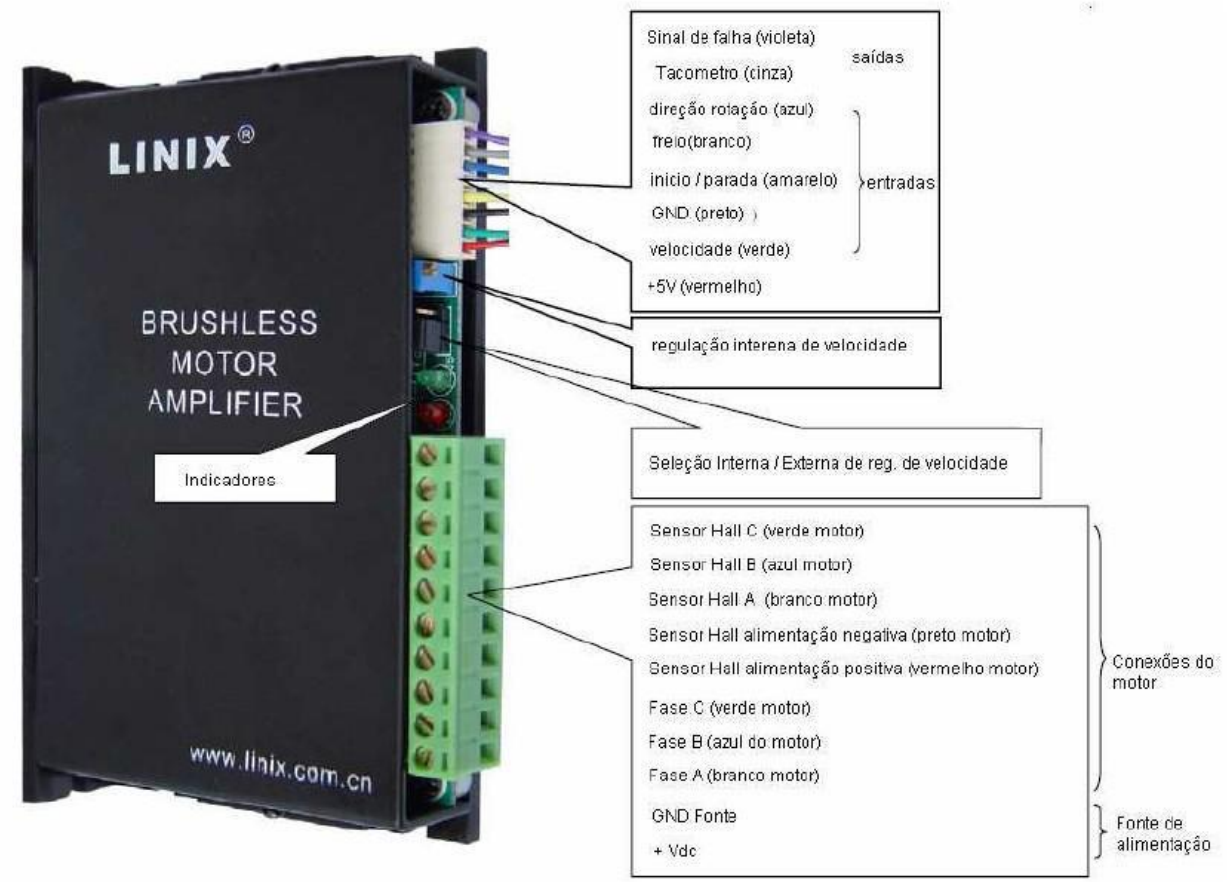

Figura 124: Foto do *Driver* indicando suas entradas

A I.2.2 Servos-motores

Para direcionar as rodas foi usado quatro servos-motores da fabricante Futaba modelo S3003. Esses servos possuem sistema de controle interno onde as entradas são a alimentação de 5V (positivo e negativo) e o sinal de posicionamento angular do eixo. O controle de posicionamento é feito através de comprimento de pulso, espaçados em 20ms. O comprimento do sinal, com nível lógico 1, está compreendido entre 1ms e 2 ms, conforme ilustra Figura 125.

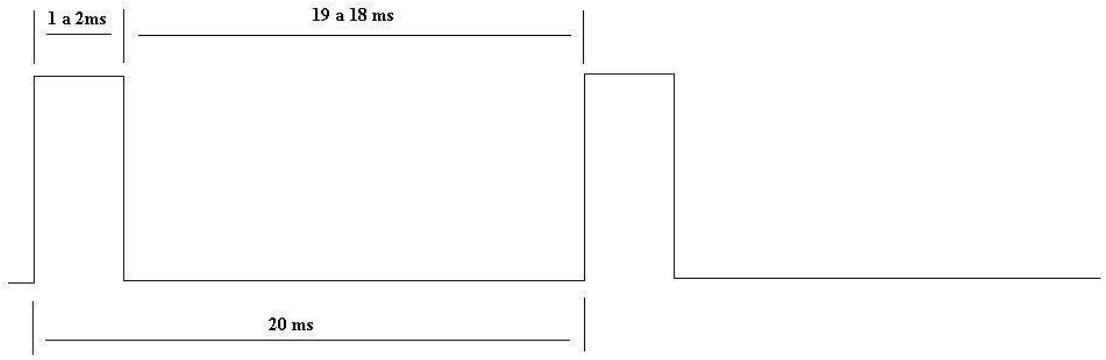

Figura 125: Sinal de entrada ao servo-motor para posicionamento do eixo

Cada roda terá um sistema de direcionamento independente usando um servo-motor cada. A Figura 126, ilustra um dos servos-motores adquiridos para o projeto.

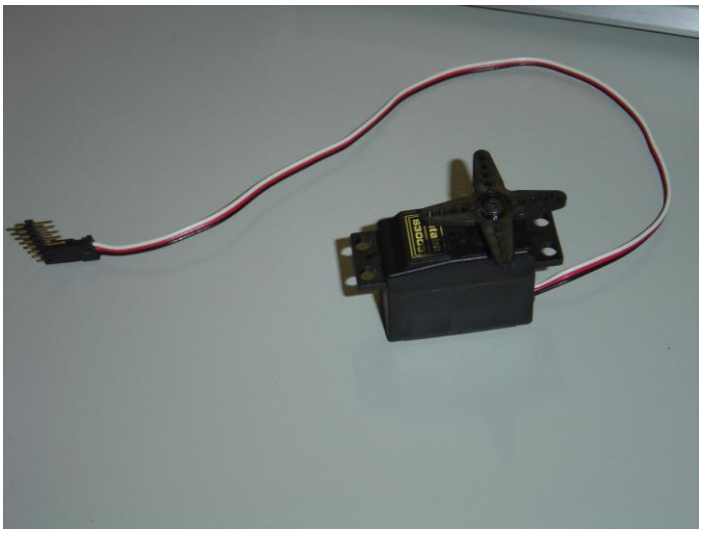

Figura 126: Servo-motor para direcionamento de cada uma das quatro rodas do Robô