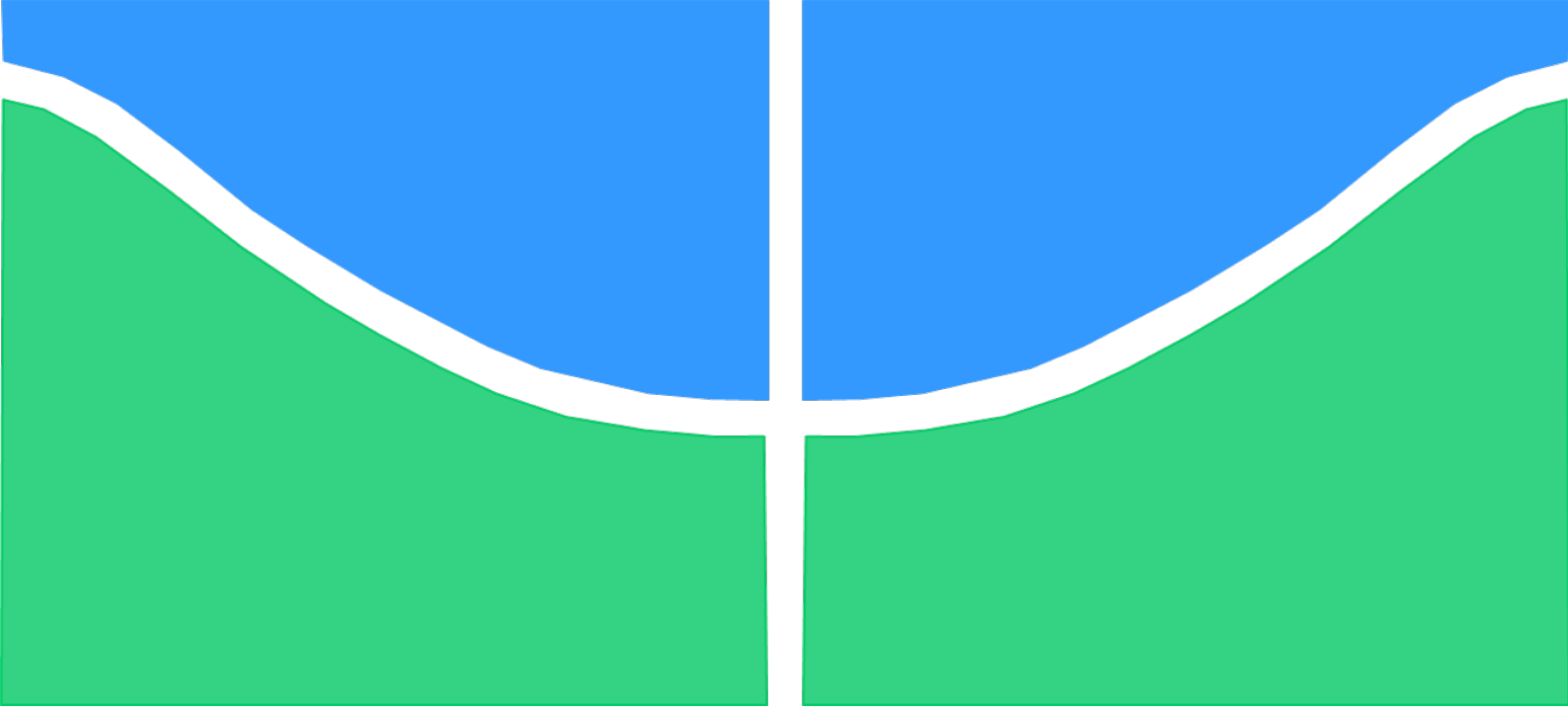


TRABALHO DE GRADUAÇÃO

COMUNICAÇÃO DE DADOS E PROTOCOLOS EM ROBÔS MODULARES AUTO-RECONFIGURÁVEIS

Por,

Ana Carolina Cardoso de Sousa

Brasília, Outubro de 2012

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

COMUNICAÇÃO DE DADOS E PROTOCOLOS EM ROBÔS MODULARES AUTO-RECONFIGURÁVEIS

Por,
Ana Carolina Cardoso de Sousa

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Carla Maria Chagas e Cavalcante Koike, CIC/UnB _____
Orientador

Prof. Marcus Vinicius Lamar, CIC/UnB _____
Examinador interno

Prof. João José Costa Gondim, CIC/UnB _____
Examinador interno

Brasília, Outubro de 2012

FICHA CATALOGRÁFICA

SOUSA, ANA CAROLINA CARDOSO DE
Comunicação de dados e protocolos em robôs modulares auto-reconfiguráveis.

[Distrito Federal] 2012.

vii, 67p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2012). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Robótica Modular 2.Transmissão de Dados
3. Protocolos de Comunicação

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

SOUSA, A. C. C., (2012). Comunicação de dados e protocolos em robôs modulares auto-reconfiguráveis. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$06/2012, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 67p.

CESSÃO DE DIREITOS

AUTOR: Ana Carolina Cardoso de Sousa

TÍTULO DO TRABALHO DE GRADUAÇÃO: Comunicação de dados e protocolos em robôs modulares auto-reconfiguráveis.

GRAU: Engenheiro ANO: 2012

Ana Carolina Cardoso de Sousa

SMT Conjunto 10 Lote 12 Casa 1 - Taguatinga Sul.

72023-450 Brasília – DF – Brasil.

Dedicatória

Aos meus pais.

Ana Carolina Cardoso de Sousa

Agradecimentos

Quero agradecer a todos que de alguma forma estiveram presentes ao longo dos meus cinco anos de graduação. Peço desculpas desde já aos muitos nomes não citados que também foram muito importantes durante esse processo.

Primeiramente, aos meus pais agradeço o carinho, o apoio, a orientação e a confiança que me deram para chegar até aqui. Especialmente por me ensinarem o que é responsabilidade e determinação.

De forma muito especial, agradeço à Pati, não só por todo o carinho, o amor, a amizade, a atenção (que já é tanto!), mas também pelas várias horas corrigindo os intermináveis erros de ortografia, concordância, regência e semântica do meu relatório.

Claro que não deixarei de citar meus amigos que sempre partilharam ideias, se preocuparam, me apoiaram e torceram por mim desde o colégio, especialmente o Dan e a Lores. À Ju agradeço por tornar muito menos chatas minhas longas horas dirigindo para a UnB. Ao Murilo, agradeço por todos os dias tentar transformar nossa graduação em algo interessante, menos difícil e, principalmente, divertido.

Não posso esquecer de citar meus colegas do Ereko que se esforçaram para auxiliar o desenvolvimento do grupo, principalmente o Nathan e o David que quase me pegaram pela mão no início. À professora Dianne agradeço por estar sempre tão prestativa. E, por fim, mas extremamente importante, agradeço à professora Carla pela orientação, apoio, encorajamento e dedicação, almejo um dia ser capaz de mostrar aos outros uma competência semelhante.

Ana Carolina Cardoso de Sousa

RESUMO

Robôs modulares são máquinas autônomas com morfologia variável. Além das características convencionais de outros robôs, (como atuadores, sensores e sistemas de controle), os robôs auto-reconfiguráveis são também capazes de mudar sua própria forma, reorganizando as conectividades de suas partes. O componente básico do robô modular é chamado de módulo, que consiste em um gabinete com processadores, baterias, atuadores e mecanismos especiais como ganchos ou câmeras. O projeto das partes mecânica, elétrica, de acionamento e computacional embarcada, assim como dos programas que controlam o robô e seu funcionamento, são fundamentais para que o robô se locomova e execute eficazmente suas atividades. A estrutura dos programas e suas inter-relações tem grande influência sobre o desempenho do robô modular, e a forma de comunicação entre os módulos e entre os módulos e o computador central (ou host) é um dos principais itens dessa estrutura. Esse trabalho consiste no projeto e desenvolvimento de toda a estrutura relacionada à comunicação do robô modular: hardware, programas, protocolos e relacionamentos entre os diversos softwares envolvidos.

Palavras Chave: Robótica Modular, Transmissão de Dados, Protocolos de Comunicação

ABSTRACT

Modular robots are autonomous machines with variable morphology. In addition to the conventional features of other robots, (such as actuators, sensors and control systems), self-reconfigurable robots are also capable of changing its own shape by rearranging the connectivity of its parts. The basic component of the modular robot is called a module, and it consists of a chassis with controllers, batteries, actuators and special mechanisms such as hooks or cameras. The design of mechanical and electrical parts, motor drivers and embedded computing, as well as the programs that control the robot and its operation are essential for the robot locomotion and execution of its their activities effectively. The structure of the programs and their interrelationships have great influence on the performance of the modular robot, and communication between the modules and between modules and the central computer (or host) is one of the main items of this project. This work is about the design and development of all communications related structure of the modular robot, including hardware, software, protocols, and relationships between the various software involved.

Keywords: Modular Robotics, Data Transmission, Communications Protocol

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

t	tempo	s
T	período	s
v	velocidade	m/s
c	número de ciclos	ciclos
f	frequência	Hz

Símbolos Gregos

θ	ângulo	$\underline{\text{o}}$

Grupos Adimensionais

n	número de módulos

Subscritos

t	total
i	inicial
f	final

Siglas

MRS	*Modular Robot System*
IR	*Infrared*
UnB	Universidade de Brasília
A/D	*Analog-to-digital converter*
PWM	*Pulse-Width Modulation*
RX	Receptor
TX	Transmissor
CANbus	*Controller Area Network*
ISO	*International Standards Organizations*
RM-OSI	*Reference Model of Open Systems Interconnection*
CI	Circuito Integrado
bps	Bits por segundo
DOF	*Degrees of Freedom*
ISR	*Interrupt Service Routine*
API	*Application programming interface*
UART	*Universal Asynchronous Receiver/Transmitter*
USART	*Universal Synchronous-Asynchronous Receiver/Transmitter*

Capítulo 1

Introdução

Esse Capítulo mostra a relevância da robótica modular no campo dos mecanismos de resgate.

1.1 Contextualização

Situações que envolvem desmoronamentos, desabamentos ou terremotos provocam frequentemente vítimas fatais em todo o mundo. Só do início do ano de 2012 até o mês de setembro foram registrados 15 terremotos, totalizando mais de 600 mortes[1].

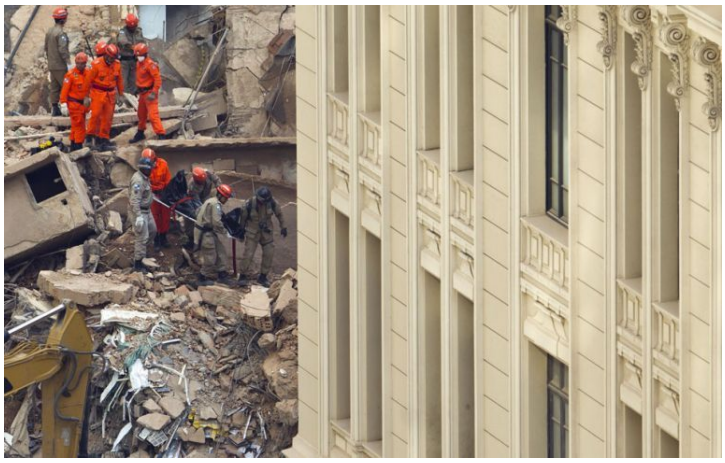
Apesar de terremotos não serem comuns no Brasil, a situação não é diferente: desabamentos e desmoronamentos causam grandes prejuízos em todo território. Em janeiro de 2012, três prédios desabaram na Avenida Treze de Maio no centro do Rio de Janeiro, causando a morte de 22 pessoas[2].

Em situações como de uma construção que tenha sido danificada por um desmoronamento, desabamento ou terremoto, há dificuldade de acesso das equipes de resgate, como pode ser visto na Figura 1.1(a). Vários robôs móveis já foram projetados e vem sendo utilizados para realizar operações urbanas, como, por exemplo, o Urbie (Figura 1.1(b)), desenvolvido no Japão em 1997 pela iRobot®.

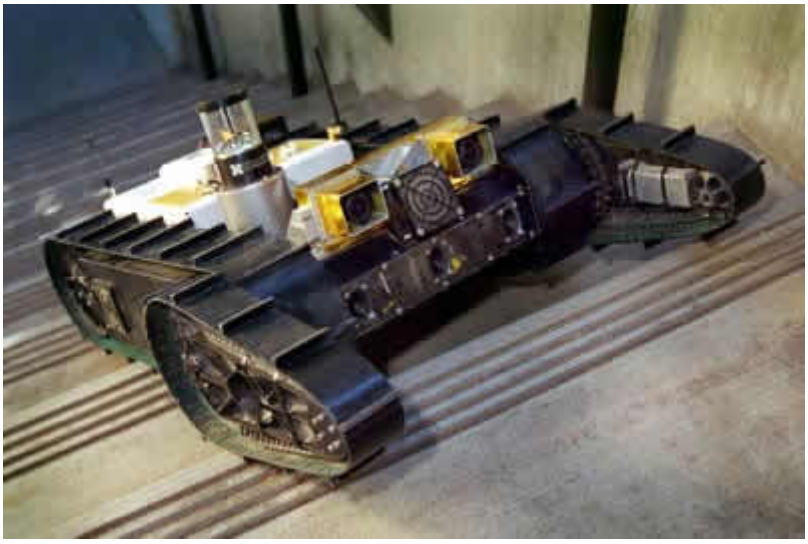
Mas diversos obstáculos podem ainda impedir a entrada de equipamentos ou de robôs de resgate de configuração fixa.

A adaptabilidade do robô modular reconfigurável permitiria atravessar passagens estreitas configurado como uma cobra e, em seguida, alterar sua forma e passar a se locomover como um hexápode, andando sobre os escombros ou mesmo subindo escadas. Ou seja, um robô modular teria a capacidade de se locomover sobre uma maior variedade de terrenos, adaptando-se bem a uma gama de tarefas.

[1]Fonte: http://earthquake.usgs.gov/earthquakes/eqarchives/year/2012/2012_deaths.php
[2]Fonte: http://www.bbc.co.uk/portuguese/noticias/2012/03/120324_desabamentos_rio_dois_meses_jc.shtml



(a)



(b)

Figura 1.1: Resgate (a) Bombeiros buscam sobreviventes entre escombros de prédios no Rio de Janeiro (b) Urbie

1.2 Apresentação do tema

Robôs móveis de estrutura rígida, os chamados robôs convencionais, não alteram sua estrutura física durante a locomoção ou atuação. São extremamente especializados, de forma que para se adaptar a novas tarefas é necessária a realização de um novo projeto.

Tendo isso em mente, Mark Yim, da Universidade de Stanford, concebeu a Robótica Modular em 1994 [1]. Seu trabalho mostra os esforços das pesquisas da época direcionados à versatilidade e à adaptabilidade do sistema.

Os robôs modulares auto-reconfiguráveis podem, de forma autônoma, alterar sua configuração e, desse modo, são capazes de executar uma ampla variedade de tarefas em ambientes desconhecidos.

1.2.1 Objetivos do projeto e Metodologia

A estrutura dos programas e suas inter-relações têm grande influência sobre o desempenho dos programas que controlam o robô modular, e a forma de comunicação entre os módulos e entre os módulos e o computador central (ou *host*) é um dos principais itens dessa estruturação.

O principal objetivo desse projeto é desenvolver toda a estrutura, desde hardware a protocolos, relacionada à comunicação do ErekoBot, o robô modular que está sendo projetado e implementado pelo grupo Ereko.

Como este é o primeiro trabalho do grupo relacionado à área de comunicação, é necessário primeiramente um estudo da bibliografia e uma avaliação dos recursos disponíveis do ErekoBot para conceber um projeto.

Após essa revisão bibliográfica, a plataforma de hardware e eletrônica associada à comunicação será estabelecida, para só então projetar e desenvolver os protocolos específicos para esses robôs.

Por fim, tanto o projeto de hardware quanto os protocolos serão verificados em testes reais em ambientes controlados.

1.3 Apresentação do relatório

No Capítulo 2 é realizada uma introdução aos robôs modulares e ao ErekoBot e no Capítulo 3 é revisada a fundamentação teórica necessária à respeito da transmissão de dados. No Capítulo 4 é apresentado o protocolo de comunicação proposto para o robô modular do Grupo Ereko. Os resultados do Capítulo 5 demonstram o bom funcionamento do sistema e confirmam a sua validade. O Capítulo 6 contém as conclusões mais pertinentes e sugestões para trabalhos futuros. Os anexos contêm material complementar.

Capítulo 2

Revisão Bibliográfica

Introdução aos Robôs Modulares Auto-Reconfiguráveis e apresentação do ErekoBot.

2.1 Robôs Modulares Auto-Reconfiguráveis

Robôs modulares auto-reconfiguráveis são máquinas autônomas com morfologia variável. Além das características convencionais de autômatos com morfologias fixas (como atuadores, sensores e sistemas de controle), os robôs auto-reconfiguráveis são também capazes de mudar sua própria forma, reorganizando as conectividades de suas partes, a fim de adaptar-se a novas circunstâncias, executar novas tarefas ou recuperar danos [2].

O robô modular pode assumir formas simples, como cobras, ou formas complexas, como hexápodes. A ideia principal da robótica modular é a versatilidade, uma vez que ela possui potencial para uma ampla utilização.

O componente básico do robô modular é chamado de módulo, que consiste em um gabinete com processadores, baterias, atuadores e mecanismos especiais como ganchos ou câmeras. Cada módulo funciona transmitindo força, momento, energia e dados através de todo o robô.

2.1.1 Robótica Modular

Esse campo especial da robótica moderna apresenta diversos desafios de mecânica, comunicação, locomoção e processamento da informação. Através de atuadores, sensores e controles convencionais, robôs modulares são capazes de mudar sua forma rearranjando suas partes conforme a necessidade. Tecnologias recentes aplicadas à robótica, como atuadores inteligentes, conexões magnéticas e controle distribuído são muitas vezes associadas à robótica modular.

Pesquisas desse tema começaram com o trabalho desenvolvido por Mark Yim [3], que propôs em sua tese de PhD uma nova solução para o problema de locomoção de robôs. Nas últimas duas décadas, esse campo avançou do uso de sistemas *proof-of-concept* (realização de um método ou idéia para demonstrar a sua viabilidade.) a implementações físicas elaboradas.

No início, as pesquisas estavam concentradas em simuladores, evitando as complexidades de

montagens eletromecânicas, e isso impedia a caracterização desses sistemas como reconfiguráveis. Conforme os projetos se tornaram mais sofisticados, os módulos ganharam, além de implementações fisicas, novos componentes como sensores, sistema de ancoragem automática e comunicação sem fio.

Atualmente existem muitos grupos de pesquisa acadêmicos trabalhando com robótica modular (exemplos: PolyBot, M-Tran II, Atron e SuperBot), mas a maioria das soluções ainda está restrita a protótipos que não possuem a aplicabilidade almejada.

2.1.1.1 Motivação e Inspiração

A utilização da robótica modular no lugar da tradicional possui três principais motivações:

1. Baixo custo: sistemas modulares acarretam uma economia no projeto, pois substituem o custo de fabricação de máquinas complexas pela produção em série de várias unidades mais simples.
2. Versatilidade: sistemas auto-reconfiguráveis podem realizar um número de atividades muito maior do que os tradicionais, adaptando-se mais facilmente a situações diversas.
3. Robustez: como o robô é feito de módulos idênticos é mais fácil realizar a substituição de partes com defeito, o que torna o robô menos dependente de intervenção humana.

Apesar dessas motivações, um robô convencional projetado para uma tarefa específica sempre possui um melhor desempenho que um robô modular. Sistemas modulares são vistos como uma alternativa para ambientes em que as missões são muito diversas e acabam ultrapassando as habilidades do robô convencional.

2.1.1.2 Introdução

Conceitualmente, é possível citar diversos robôs fictícios que possuem a habilidade de mudar seus formatos (Optimus Prime, da série *Transformers*), copiar formas (T-1000, do filme *Terminator 2: Judgment Day*) ou efetuar auto-reparo (R2D2, da série *Star Wars*). Desses robôs, o mais próximo da robótica modular é o T-1000, composto de um metal líquido futurista, que pode tanto mudar seu formato quanto copiar formas e se reconstruir.

Robôs reais que também podem mudar suas formas estão sendo estudados e construídos por diversos grupos ao redor do mundo desde os anos 70, quando surgiram braços robóticos cujas extremidades eram compostas por módulos automaticamente intercambiáveis.

O conceito de aplicar um mecanismo de conexões idênticas a um robô totalmente modular foi introduzido por Fukuda, ao final dos anos 80 [4], o CEBOT tinha 18x9x5 cm, pesava aproximadamente 1.1kg e possuía motores e processadores. Cada módulo podia se comunicar para se conectar ou se separar automaticamente.

No início dos anos 90, os robôs começaram a apresentar capacidade de movimento. Em 1994, Yim explorou formas de locomoção estáveis com o Polypod [1], um módulo bem menor e mais leve que o CEBOT. Um módulo sozinho não podia se movimentar, mas quando conectado corretamente a outros era possível atuar em diferentes marchas de locomoção: rolando ou se movimentando como uma mola ou lagarta.

Por meio desse trabalho foi possível perceber a complexidade de controlar um grande número de módulos. O controle inicial do Polypod utilizava uma tabela para programar movimentos simples utilizando posições já prescritas. Em adição ao problema de controle coordenado, a complexidade das configurações arbitrárias e a sequência de reconfigurações rapidamente se tornou um problema computacional interessante.

Chirikjian e Murata desenvolveram sistemas de configuração *lattice* [5, 6]. Como descrito em 6, um sistema *lattice* possui módulos estilo treliça, de forma que se tornam mais fáceis de serem visualizados computacionalmente. Como resultado, esse estilo de sistema rapidamente se tornou popular na robótica computacional.

Ao final dos anos 90, Rus [7] e Shen [8] também desenvolveram um hardware aberto, com contribuições mais voltadas para aspectos de sistemas distribuídos. Dois ramos de suas pesquisas incluem configurações de auto-reconhecimento e planejamento cinemático dos movimentos para rearranjamento entre configurações.

2.1.1.3 Taxonomia de arquiteturas

De acordo com [2], a classificação dos robôs modulares é baseada em geometrias e modelos de reconfiguração. Abaixo estão descritos os principais critérios.

De acordo com o arranjo geométrico das suas unidades, o robô modular pode ser:

- Arquitetura Lattice: é caracterizado por sistemas que apresentam módulos dispostos em um arranjo tridimensional, como uma rede cúbica ou hexagonal. O controle e a locomoção podem ser executados em paralelo. Arquiteturas desse tipo têm a vantagem de permitir uma reconfiguração mais simples, uma vez que os módulos se movem apenas para posições vizinhas.

- Arquitetura Chain: é caracterizado por sistemas cujas unidades são conectadas em série, como uma corda ou corrente, que pode eventualmente se dobrar para reduzir a área. Essa arquitetura pode alcançar qualquer orientação no espaço através de articulações, mas a complexidade computacional acaba sendo muito maior, dificultando o controle.

- Arquitetura Mobile: as unidades utilizam o ambiente para manobrar e podem se conectar para formar arquiteturas *lattice* e *chain* mais complexas. Podem ainda, formar uma série de robôs menores que executam movimentos coordenados e juntos formam uma rede "virtual" maior.

De acordo com o modo de reconfiguração, o robô modular pode possuir:

- Reconfiguração determinística: presente em sistemas nos quais as posições dos módulos são controladas o tempo todo. A posição exata de todos os módulos é conhecida, inclusive quando o caminho do módulo é alterado. Para esse tipo de reconfiguração, um controle de malha fechada é essencial.

- Reconfiguração estocástica: presente em sistemas nos quais os módulos se movem aleatoriamente. O local exato do módulo é conhecido somente quando ele é conectado à estrutura principal da nova morfologia, sendo que, durante o processo de reconfiguração, as trajetórias são randômicas. O tempo de reconfiguração só pode ser determinado por meios estatísticos.

De acordo com o tipo de módulo:

- Módulos homogêneos: os módulos são idênticos.

- Módulos heterogêneos: existem módulos que podem possuir características/funções diferentes que permitem favorecer algumas atividades, por exemplo, uma câmera pode ser anexada a um dos módulos, fazendo com que seja possível visualizar o caminho percorrido pelo robô.

2.1.1.4 Aplicações

A seguir são citados alguns exemplos de aplicações que poderiam ser beneficiadas com o desenvolvimento de um sistema de robótica modular funcional.

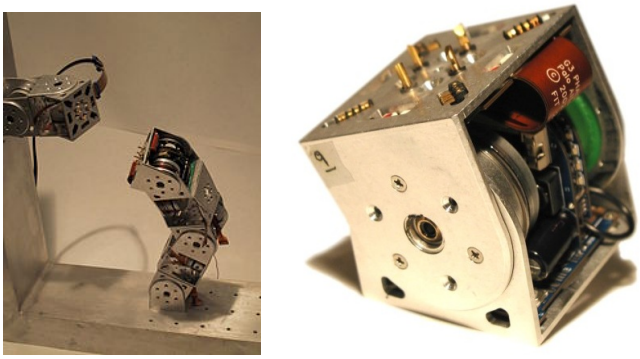
- Espaço: A exploração no espaço apresenta diversos desafios, incluindo um ambiente imprevisível e uma significativa limitação da massa e do volume dos equipamentos a serem utilizados. A robótica modular resolveria ambos os problemas em função da sua versatilidade e intercambiabilidade.

- Resgate: Áreas de desastre apresentam outro tipo de ambiente imprevisível no qual utilizar a robótica modular seria benéfico. O sistema poderia, por exemplo, assumir a forma de uma cobra para atravessar lugares estreitos à procura de vítimas. Depois de encontrá-las, poderia ainda emitir um sinal e tomar a forma de um abrigo para protegê-las até a chegada do resgate.

- Tarefas domésticas: O sistema seria um produto de consumo com vários módulos que poderiam ser reconfigurados de maneiras arbitrárias para a realização de tarefas domésticas. Essa aplicação pode ser vista como o objetivo prático da robótica modular: um sistema que pode se adaptar a qualquer tarefa em tempo real.

2.1.1.5 Sistemas de Robótica Modular em desenvolvimento

Exemplos de trabalhos similares de robótica modular que podem ser citados:



(a) (b)

Figura 2.1: PolyBot G3 (a) conjunto de módulos (b) detalhado

- PolyBot (Figura 2.1): proposto no Centro de Pesquisa de Palo Alto (PARC) em 2001 [9]. O PolyBot é um robô de módulos heterogêneos, auto-reconfiguráveis e de arquitetura *chain*. Cada módulo da geração 3 do PolyBot (PolyBot G3) possui um grau de liberdade, duas superfícies conectáveis e dimensões de um cubo de $50x50x50mm$.

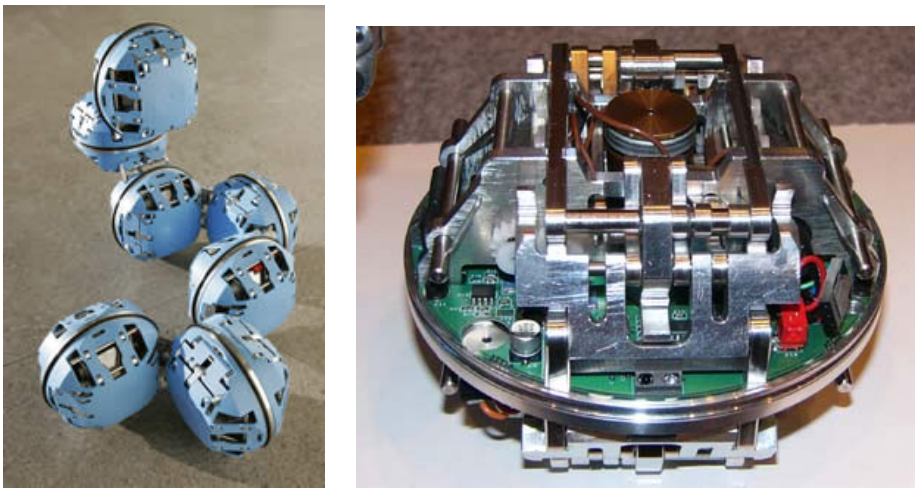
- M-Tran II (Figura 2.2): proposto no Instituto Nacional de Ciência Industrial Avançada e Tecnologia do Japão (AIST) e no Instituto de Tecnologia de Tokyo (TiTech) em 2002 [10]. O M-Tran II é um robô cujos módulos são homogêneos, auto-reconfiguráveis e de arquitetura *mobile*. Os módulos são compostos de duas partes, que ligadas possuem dimensões de $60x120x60mm$. As duas partes conectadas possuem dois graus de liberdade e seis superfícies conectáveis.

- Atron (Figura 2.3): proposto na Universidade da Dinamarca do Sul (USD) em 2005 [11]. O Atron é um robô de módulos homogêneos, auto-reconfiguráveis e de arquitetura *lattice*. Os módulos são compostos de dois hemisférios conectados, que ligados formam uma esfera de diâmetro de $114mm$. A esfera possui um grau de liberdade e quatro conectores (dois machos e dois fêmeas).

- SuperBot (Figura 2.4): proposto na Universidade do Sul da California (USC) em 2006 [12]. O SuperBot é um robô de módulos homogêneos, auto-reconfiguráveis e de arquitetura *mobile*. Os módulos são compostos de duas partes, cada uma com $84x84x84mm$ (comprimento total de $168mm$), e possuem juntas três graus de liberdade e seis superfícies conectáveis.

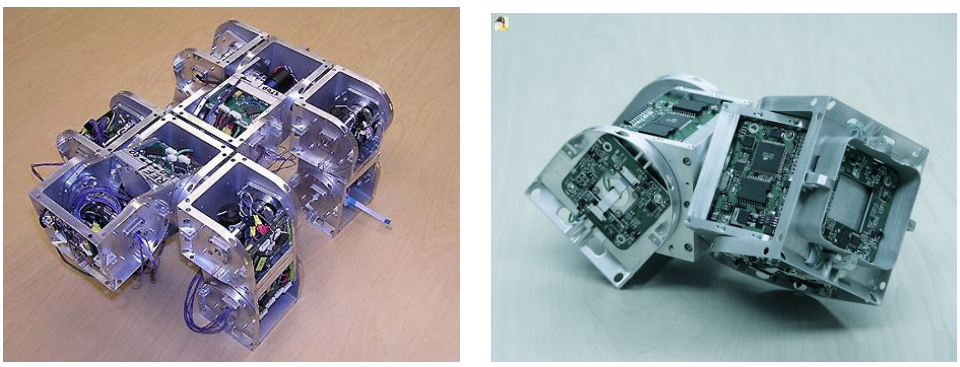
(a) (b)

Figura 2.2: M-Tran II (a) conjunto de módulos (b) detalhado



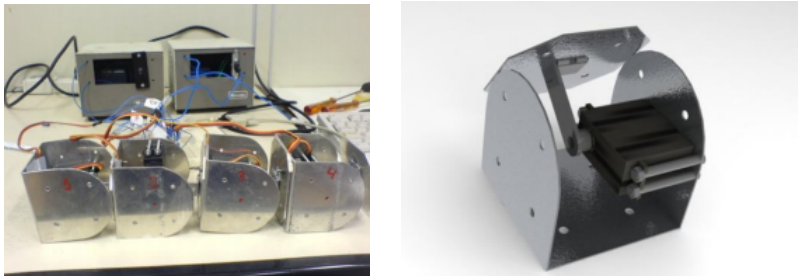
(a) (b)

Figura 2.3: Atron (a) conjunto de módulos (b) detalhado



(a) (b)

Figura 2.4: SuperBot (a) conjunto de módulos (b) detalhado



(a) (b)

Figura 2.5: ErekoBot $\beta - 5$ (a) conectado em série (b) detalhado

2.1.1.6 Direções Futuras

O propósito dos sistemas de robótica modular é um dia utilizá-los em um vasto número de aplicações práticas nas quais, sem supervisão, os módulos poderiam se reorganizar quando necessário. Se alguns desafios (como a construção de grandes sistemas e a possibilidade de auto-reparo) forem superados, as próximas gerações de robôs modulares possuirão capacidades muito superiores às atuais.

2.2 Ereko

O Ereko é um grupo de pesquisa que surgiu em 2009 desenvolvendo projetos em robótica modular. A equipe é composta por professores e alunos do Departamento de Engenharia Mecânica e do Departamento de Ciência da Computação da UnB.

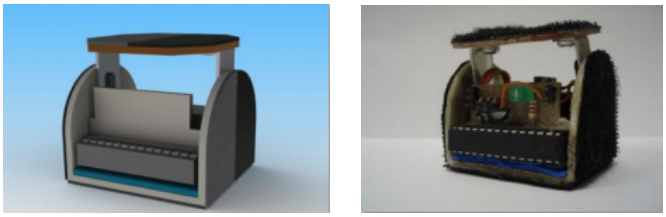
2.2.1 Projetos do ErekoBot

O primeiro protótipo funcional desenvolvido no Grupo Ereko foi o *ErekoBot* $\beta - 5$ [13], ilustrado na Figura 2.5. O objetivo desse protótipo era estudar aspectos dos sistemas modulares, como as diferentes montagens de configurações, algoritmos de locomoção e reconfiguração.

O projeto do ErekoBot $\beta - 5$ foi inspirado no M-Tran [10] e no PolyBot [3] construído em alumínio, com dimensões $70x70x70mm$, os módulos conectavam-se manualmente por parafusos. Cada unidade tratava-se de um sistema embarcado composto por um servomotor (parte do motor, controlada por posição), uma bateria e um microcontrolador.

Ao serem executados os algoritmos de locomoção apodal de Gomez [14] descobriu-se que o ErekoBot $\beta - 5$ era muito pesado e seus encaixes eram ineficientes, dificultando a obtenção de resultados razoáveis.

Para resolver esses problemas, algumas modificações foram feitas, projetando o ErekoBotα, descrito em[15].



(a) (b)

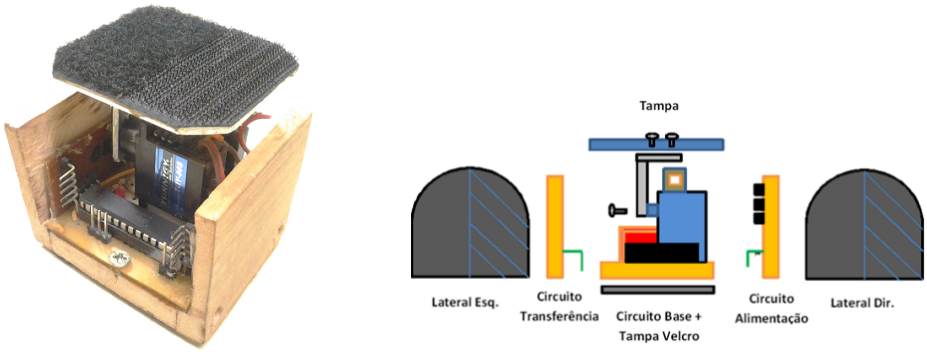
Figura 2.6: ErekoBot $\alpha\, v.1$(a) design Prévio (b) implementação real

As primeiras modificações tiveram o objetivo de diminuir o peso dos módulos, então o ErekoBot $\alpha\, v.1$ foi projetado para ser significamente menor que o ErekoBot $\beta - 5$, com dimensões de $40x40x40mm$ (Figura *2.6*). O módulo foi construído com fibra de carbono, um material mais leve, mas ainda suficientemente forte para suportar os esforços necessários.

Após a realização de testes nos quais foi utilizado o protótipo ErekoBot $\alpha\, v.1$, percebeu-se a necessidade de melhoria no projeto, desenvolvendo-se, então, o ErekoBot $\alpha\, v.2$ (Figura 2.7), descrito em [16].

Tais melhorias foram:

- Maior número de portas I/O: o ErekoBot $\alpha\, v.1$ só possuía uma porta de comunicação, limitando a capacidade de reconfiguração à formação em linha. O projeto foi então modificado para permitir quatro barramentos de comunicação intermodular.
- Maior número de entradas para sensores : o ErekoBot $\alpha\, v.1$ só possuía espaço para um sensor, já no ErekoBot $\alpha\, v.2$ permitiu-se a abertura de mais portas para possibilitar o uso de outros.
- Abertura de portas padrão Inter-Integrated Circuit (I2C): o protocolo I2C é muito utilizado para interfacear acelerômetros, compassos magnéticos, giroscópios eletrônicos, entre outros. Para possibilitar testes com esses sensores embarcados, foram feitas, no ErekoBot $\alpha\, v.2$, aberturas para as portas I2C.
- Gravação no próprio módulo: no módulo ErekoBot $\alpha\, v.1$ era necessário retirar o microcontrolador do soquete e gravá-lo na *protoboard*, o que causava fadiga de flexão nas pernas do chip e sua eventual ruptura. Como medida de prevenção, a nova versão foi projetada para permitir a gravação e a depuração no próprio módulo.
- Medidor de tensão da bateria: no ErekoBot $\alpha\, v.1$ não era possível medir a carga da bateria, o que causava muitos problemas durante a depuração do sistema. Portanto, um medidor de tensão foi adicionado ao no ErekoBot $\alpha\, v.2$ para estimar a carga da bateria e para não comprometer a eficiência do módulo.
- Montagem mecânica facilitada: foram feitos ajustes e modificações no projeto mecânico do ErekoBot $\alpha\, v.1$ para facilitar a construção mecânica e evitar que os módulos fossem



(a) (b)

Figura 2.7: ErekoBot $\alpha\, v.2$(a) implementação real (b) esquemático construtivo

Especificações	ErekoBot $\alpha\, v.1$	ErekoBot $\alpha\, v.2$	ErekoBot $\beta - 5$
Tipo	Homogêneo	Homogêneo	Homogêneo
Dimensões	$40x40x40\, mm^3$	$50x50x50\, mm^3$	$70x70x70\, mm^3$
Material	Alumínio	Madeira	Alumínio
Arquitetura	Mobile	Mobile	Mobile
Reconfiguração	Determinístico	Determinístico	Determinístico
Auto-Reconfiguração	Sim / Manual	Sim / Manual	Sim / Manual
Lados Conectáveis	4	4	4
Graus de Liberdade (DOF)	1 (180° rotacional)	1 (180° rotacional)	1 (180° rotacional)
Mecanismo de encaixe	Velcro	Velcro	Parafusos
Auto-Locomoção	Sim (2D)	Sim (2D)	Sim (1D)

Tabela 2.1: Especificações dos módulos ErekoBot.

danificados durante o processo de manufatura. Com o objetivo de proteger o circuito eletrônico e as trilhas expostas, armaduras de fibras de carbono foram adicionadas ao módulo. Posteriormente, passaram a ser construídas em madeira, para facilitar a construção mecânica.

Na Tabela 2.1, é possível encontrar as classificações do ErekoBot$\alpha\, v.1$, ErekoBot $\alpha\, v.2$ e ErekoBot $\beta - 5$.

2.2.1.1 Eletrônica

O circuito elétrico embarcado do módulo encontra-se no Anexo I. O microcontrolador utilizado para o controle do circuito é o Atmel ATmega 8® (Figura I.1). A fonte de alimentação é composta por duas baterias recarregáveis de polímero de lítio (LiPo), com 7.4V e 138mA, e para proteger o circuito da polaridade reversa e reduzir a tensão de entrada, de 7.4V para 6.35V, são utilizados três diodos 1N4007 na entrada VCC dos servos. O regulador de tensão 78L05 reduz a tensão aos 5V

necessários para o microcontrolador (Figura I.3). O circuito controlado dissipa aproximadamente 0.84W, enquanto o consumo do servo chega a 25W, o que demonstra que o consumo da bateria está associado principalmente à atuação do servo [15].

O pino de saída do microcontrolador OC1A é conectado ao servo, cuja posição é indicada pelo sinal de controle PWM. Para a comunicação intermodular, existe uma conexão em barramento que liga os pinos TX e GND, do módulo transmissor, aos pinos RX e GND, respectivamente, do módulo receptor.

2.3 Comunicação em Robôs Modulares

2.3.1 Comunicação e Controle em Robôs Modulares

Idealmente, a robótica modular prevê a existência de muitos módulos e, consequentemente, um grande número de graus de liberdade. O controle dos graus de liberdade é um problema complexo, e muitas soluções, envolvendo diferentes modos de comunicação, podem ser adotadas[1].

Em [17], Yim comenta também sobre a complexidade da organização do controle, em função das numerosas configurações necessárias para proporcionar versatilidade ao sistema.

Fatores como a organização do processador (centralizado ou descentralizado), esquemas de comunicação entre módulos (barramento global, vizinho a vizinho local, tanto global quanto local) e endereçamento de módulos (com identidade ou sem identidade) determinam a complexidade e a coordenação do controle.

2.3.1.1 Organização do Controlador

O controle pode ser centralizado ou descentralizado.

Controle Centralizado Um sistema centralizado possui um único controlador que comunica ao restante como deve ser seu comportamento, por exemplo, enviando os ângulos em que o servo deve se posicionar.

Quando um robô modular possui um controlador central, é natural atribuir identidades aos módulos para que comandos explícitos sejam enviados individualmente a cada um. Já quando os módulos acessam igualmente o barramento e não existe indicação da posição relativa dos módulos dentro da configuração, alguns mecanismos devem ser utilizados para localizar cada módulo em uma estrutura e, assim, o controlador central pode mapear o controle das unidades.

Controle Descentralizado O estudo sobre sistemas descentralizados é um dos mais importantes tópicos de pesquisa da atualidade, pois permite a divisão do processamento das informações entre mais de uma unidade computacional[1]. Muitas pesquisas com complexidade semelhante à da Robótica Modular utilizam esse tipo de controle, por exemplo, em estruturas flexíveis de grande

porte, como estações de energia e controle de tráfego.

É possível, então, utilizar os conceitos de controle de sistemas descentralizados em robôs redundantes (tais como os modulares, formados por módulos semelhantes), nos quais não é nada prático existir um único controlador processando uma linha de controle para cada grau de liberdade, uma vez que a quantidade de informação seria muito confusa.

Um exemplo de um sistema totalmente descentralizado está sendo desenvolvido por Rus [18]. O controle dos autômatos celulares (*cellular automata* - CA) utiliza as mesmas regras para todos os módulos, sendo que uma regra pode ser vista como um conjunto de condições prévias. Se todas essas condições são satisfeitas, uma certa ação é aplicada e, dessa forma, os módulos ganham autonomia.

Controle Híbrido Um sistema híbrido é uma combinação dos controles centralizado e descentralizado. Por exemplo, a movimentação de um conjunto de módulos pode ser controlada de modo centralizado, mas um módulo desconectado do sistema central, de modo descentralizado.

Na maioria dos casos, é mais simples desenvolver e analisar um sistema centralizado. Contudo, um sistema descentralizado tem a vantagem da computação ser compartilhada entre os módulos, ou seja, nenhuma unidade precisaria realizar toda a computação pesada sozinho.

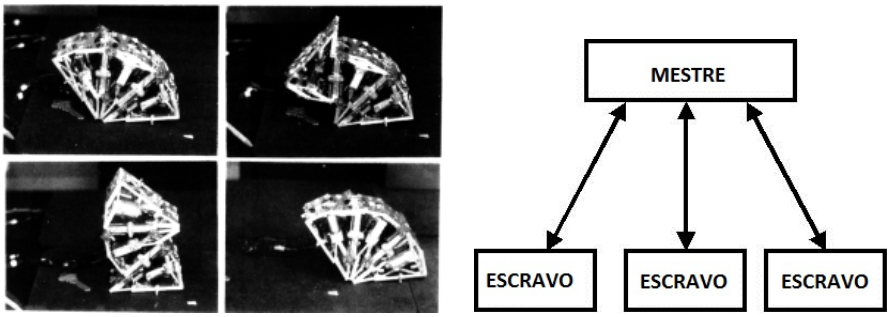
A implementação do controle depende de como está estruturada a comunicação. Essas estruturas podem ser globais, como a CANbus (um protocolo automotivo que recentemente tem sido utilizado também em robótica), e/ou locais, utilizando comunicação ponto a ponto, como o infravermelho. Muitos sistemas utilizam a combinação dessas estruturas (Polybot, CKBot, M-TRAN, CONRO e Superbot). Outro tipo de implementação é a comunicação sem fio, que, em nível de arquitetura, é similar à estrutura global.

Na prática, é mais fácil implementar o controle centralizado utilizando a comunicação global e o descentralizado utilizando a local.

2.3.1.2 Organização do Controlador

O primeiro robô modular, o Polypod [1], possuía uma comunicação serial por barramento em cada módulo, e cada unidade era identificada por um endereço único (Figura 2.8). Dessa forma, o número de módulos no sistema passou a ser limitado pela largura de banda do barramento serial. Além desses endereços únicos, existe ainda um global, que permite que uma mesma mensagem pode ser enviada para todos os receptores ao mesmo tempo, ou seja, uma comunicação *broadcast* (transmitir ou difundir uma informação para todos receptores ao mesmo tempo).

O Polypod tem um modelo de comunicação no qual um dos dispositivos ou processadores possui um controle unidirecional sobre os outros, chamado modelo "mestre-escravo". Nesse modelo, um nó mestre geralmente é o processador mais potente, como um computador, mas no caso da robótica modular, é comum que um módulo também exerça esse papel.



(a) (b)

Figura 2.8: Polypot (a) design (b) comunicação mestre-escravo

Em [3], Yim prova que o aumento do número de módulos causa uma complexidade exponencial ao planejamento e à programação do sistema para, por exemplo, desvios de auto-colisão. A solução do problema de cinemática inversa de uma arquitetura *chain* com um número grande de módulos também é não-trivial, e ao se adicionar parâmetros como limites de torques e juntas ou estabilidade gravitacional, torna-se impossível encontrar uma solução ótima (solução factível que fornece o melhor valor para a função objetivo) em tempo real.

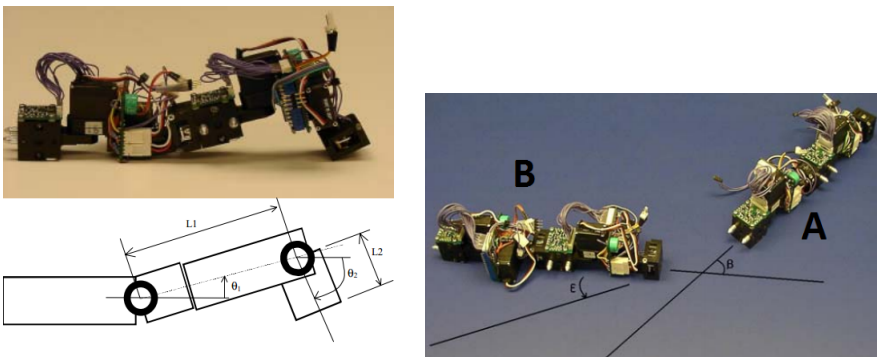
Em uma demonstração do PolyBot G2, Yim testa o robô sob controle semi-teleoperado, no qual um dos módulos possuía uma Tabela de Controle de *Gaits*. Essa tabela era, então, transmitida dinamicamente aos módulos vizinhos, em tempo real, para que eles também se movimentassem. Esse método pode ser aplicado em outras reconfigurações, ao decompor-se o problema em subestruturas que possuam movimentos previamente computados.

Como em muitas aplicações um conjunto fixo de configurações é suficiente, elas podem ser pré-planejadas *offline* para cada membro do conjunto de módulos e depois guardadas na Tabela de Controle de *Gaits* (ciclo de um movimento repetitivo utilizado para locomoção).

Quando se trata de reconfiguração, é lógico que os robôs modulares precisam ser capazes de transferir mensagens de uns aos outros sem um meio físico cabeado. O CONRO [19] (Figura 2.9(a)), por exemplo, possui um transmissor e um receptor IR em cada módulo, utilizado para comunicação local e remota. Esse par de dispositivos também é utilizado como *beacon* (dispositivo utilizado para atrair atenção para um local específico) durante o processo de reconfiguração para indicar a localização das partes do robô.

Cada módulo do CONRO pode enviar mensagens de acordo com uma tabela que descreve a topologia da rede. Contudo, em casos mais complexos, nos quais a comunicação é feita sem uma topologia específica, a mensagem precisa ser repassada em *broadcast* dentro de toda a configuração para garantir sua recepção. Para ambos os casos, cada módulo só se comunica diretamente com os adjacentes e necessita de intermediários para se comunicar com não-adjacentes.

Quando um módulo se desconecta da base na reconfiguração, o sistema é tratado como se um único robô se dividisse em dois independentes. Então, a comunicação entre eles passa a ser remota e o canal físico deve permitir que a difusão da mensagem seja corretamente executada em



(a) (b)

Figura 2.9: CONRO (a) design (b) ancoragem

grandes distâncias. No momento que Shen [8] chama de ancoragem, quando os robôs tentam se reconectar, os módulos devem ser capazes de enviar suas posições e guiar uns aos outros durante todo o processo.

A ancoragem é semelhante a um jogo de encaixe de blocos: o módulo B deve agir como um *beacon* e enviar sua posição para o módulo A, que por sua vez deve ser capaz de perceber essa posição e se movimentar em direção a B (Figura 2.9(b)).

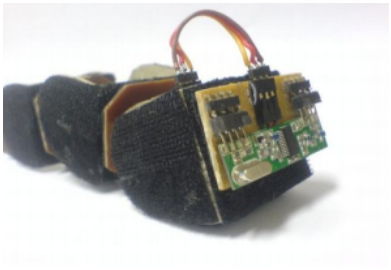
A comunicação é sempre um desafio quando se trata de robótica modular. Estruturas de robôs modulares mais complexos são feitas com uma grande quantidade de unidades individuais, então, é crucial que a comunicação depois da reconfiguração seja robusta. Independente da conexão ser por contato de pinos elétricos ou por comunicação sem fio, existem imperfeições, como a perda de dados e atrasos [20].

2.3.2 Primeiro Modo de Comunicação no ErekoBot

O primeiro modo de comunicação no ErekoBot [15] é baseado em uma arquitetura de controle centralizada, na qual o computador, ou *host* (nó central da rede que fornece informação aos demais.), computa em tempo real os resultados do cálculo dos ângulos que serão realizados pelos servos e envia-os aos módulos.

O *host* se comunica diretamente apenas com um módulo, designado *mochileiro*, que é equipado com uma "mochila RF" (Figura 2.10(a)). Essa mochila possui um módulo de rádio frequência de recepção, Módulo RF RX (Figura 2.10(b)), que recebe os dados transmitidos pelo módulo de rádio frequência de transmissão, Módulo RF TX (Figura 2.10(c)), conectado ao *host*.

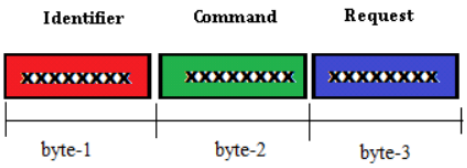
Os transmissores de rádio frequência não são seletivos, portanto todos os módulos recebem a mesma informação. Para diferenciá-las, no começo de cada mensagem existe um identificador (*identifier*) que indica para qual módulo o pacote (sequência de bytes, com determinada estrutura que passa informações pela rede) é endereçado. Os comandos (*commands*) enviados são sempre os valores dos ângulos que devem ser aplicados pelo servo do módulo. E, ao final do pacote de



(a)

(b)

(c)



(d)

Figura 2.10: Comunicação de Teste (a) Módulo com o módulo RF RX (b) Módulo RF RX (receptor) (c) Módulo RF TX (transmissor) (d) Protocolo de Comunicação

transmissão, existe o comando indicando o início da locomoção (*start* ou *request*).

A Figura 2.10(d) ilustra o pacote de comunicação, composto por um primeiro byte de endereçamento, seguido por um byte de comando e finalizado por um byte de requerimento de início de locomoção.

Como interface entre os módulos e o computador existe um hardware de conexão com o emissor e um driver de comunicação serial, que possibilita a fácil utilização do protocolo.

Capítulo 3

Fundamentação Teórica

Revisão dos conceitos fundamentais de Transmissão de Dados.

Embora muitos componentes que compreendem sistemas de comunicação possam parecer complexos, seu objetivo é muito simples: transmitir dados entre máquinas [21]. Do ponto de vista desses sistemas, não importa se as máquinas são diferentes, uma vez que é possível simplificá-los a três componentes básicos: o dispositivo de transmissão, o dispositivo de recepção e um meio para carregar os dados entre eles[1].

Os dispositivos de transmissão e de recepção podem tomar várias formas, desde um modem até um simples multiplexador. E o mesmo vale para o meio, que pode ser um fio de cobre ou até uma fibra óptica.

3.1 Estruturas Básicas

Uma rede é composta por terminais, ou nós, interligados fisicamente entre si por vias de comunicação denominadas enlaces físicos[21].

O conjunto de regras de procedimentos que determinam como dois ou mais elementos devem interagir é chamado Protocolo de Comunicação, ou simplesmente Protocolo. Se dois elementos são interligados por um enlace físico e possuem um mesmo protocolo, que permite a comunicação entre ambos, diz-se que existe um enlace lógico entre eles.

3.1.1 Modelo Básico de uma Rede de Computadores

A forma pela qual os nós são interligados pelos enlaces lógicos permite a classificação da rede segundo dois critérios: quanto à topologia de interligação e quanto ao comprimento dos enlaces.

[1] O meio de transmissão é um conjunto de recursos físicos e de regras lógicas que permite a transmissão bit a bit entre os pontos fim-a-fim que compõem a rede[22].

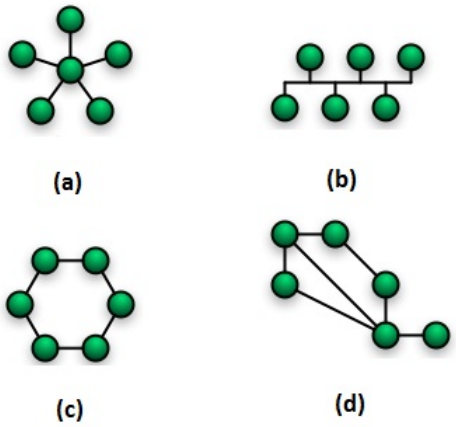


Figura 3.1: Topologias de rede (a) Estrela (b) Barramento (c) Anel (d) Malha

3.1.1.1 Quanto à topologia de interligação

As quatro topologias mais utilizadas são: estrela, barramento, anel e malha (Figura 3.1). Cada uma possui características específicas de desempenho e de confiabilidade.

É importante observar que o arranjo físico, isto é, a distribuição geográfica dos nós, não é o único elemento que determina a topologia de rede, pois o protocolo interno também pode especificar características topológicas diversas. Assim, por exemplo, uma rede em anel pode comportar-se como uma rede em estrela se todas as mensagens trocadas tiverem que passar por um nó central. É o caso de certos protocolos que utilizam uma estação mestre.

Em sistemas centralizados, existe um nó que possui a responsabilidade de enviar aos outros as informações mais importantes da rede, esse nó específico é chamado de hospedeiro (*host*).

3.1.1.2 Quanto ao comprimento dos enlaces físicos

Redes com enlaces físicos de comprimentos diferentes possuem comportamentos diversos. Conforme a distância entre nós vizinhos aumenta, o nível de inter-dependência entre eles diminui, isto é, a informação sobre o estado dos nós vizinhos torna-se cada vez mais insegura e a comunicação passa a exigir protocolos que dependem cada vez menos de informações de estado global.

Costuma-se classificar as redes da seguinte forma:

- Redes Confinadas: distâncias entre os nós de até $2\,m$.
- Redes Locais: distâncias entre os nós de até $25\,km$.
- Redes de Grande Porte: distâncias entre os nós de até $1000\,km$.
- Redes Intercontinentais: distâncias de até $10000\,km$.

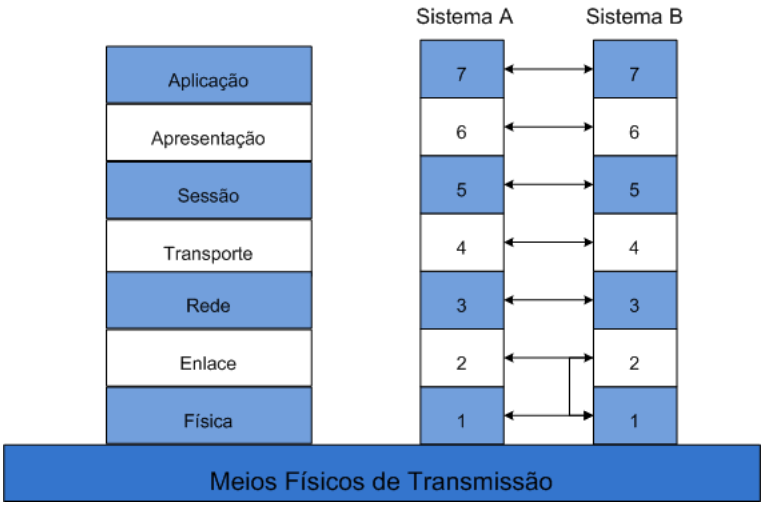


Figura 3.2: Arquitetura de rede de acordo com o modelo RM-OSI

3.1.2 Arquitetura de redes de computadores

A arquitetura de uma rede de computadores é a definição precisa das funções que a rede e seus componentes devem realizar.

Essas funções são divididas em camadas, as quais são tão importantes que são adotadas por todas as redes de computadores. Até a discussão e aprovação de um padrão para a arquitetura de computadores, a RM-OSI[2], cada rede possuía uma divisão própria, diferente das demais, o que gerava problemas de compatibilidade para interconexões.

Esse padrão divide a arquitetura da rede em sete camadas, que são estruturadas como ilustrado na Figura 3.1.2.

As principais funções de cada camada dentro do modelo OSI são:

- Camada Física:
  - Transmissão de bits sobre o canal de comunicação;
  - Estabelecimento e/ou encerramento de conexões físicas;
  - Gestão de conexões, por exemplo, para a monitoração da taxa de transmissão.
- Camada de Enlace:
  - Sincronização lógica entre as entidades;

[2]O padrão completo pode ser encontrado em:
http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip

  - Montagem e delimitação dos quadros[3];
  - Controle de erros de transmissão.

- Camada de Rede:
  - Roteamento (processo de reencaminhamento de quadros);
  - Mapeamento de endereços lógicos físicos;
  - Determinação de rotas apropriadas para a transmissão dos quadros;
  - Verificação de erro de quadro.

- Camada de Transporte:
  - Numeração e reconhecimento explícito dos segmentos[4] a fim de evitar perdas, duplicação ou entrega fora de ordem;
  - Mensagem de confirmação para entrega confiável de fim a fim;
  - Mensagem de controle de tráfego.

- Camada de Sessão:
  - Gestão do controle de diálogo, para controlar o nó que deve transmitir em um determinado momento.

- Camada de Apresentação:
  - Sintaxe dos dados recebidos, ou seja, a forma como os tipos e os valores dos dados são definidos.

- Camada de Aplicação:
  - Janela para usuários e processos de aplicativos que podem acessar os serviços de rede.

A seguir são apresentados detalhes das camadas implementadas no protocolo.

3.1.3 Camada Física

O projeto da camada física diz respeito à consideração dos componentes de hardware envolvidos e ao meio de transmissão através dos quais irão trafegar os dados. Na robótica modular, essa camada é responsável pelo envio dos dados, no formato de sinais elétricos, entre o *host* e os módulos e entre os próprios módulos.

[3]Sequência delimitada de bits transmitida na rede.
[4]"Quadro" da camada de transporte.

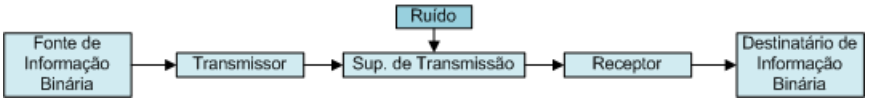


Figura 3.3: Sistema de transmissão binária ponto-a-ponto (modelo unidirecional)

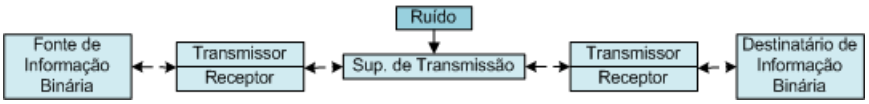


Figura 3.4: Sistema de transmissão binária ponto-a-ponto (modelo bidirecional)

3.1.3.1 Transmissão bit a bit

O processo de transmissão de bits pode ser estudado a partir do modelo mais simples de transmissão digital: o sistema ponto a ponto. À exceção da topologia em barramento, à qual devem ser atribuídas algumas parcularidades, todos os outros sistemas podem ser descritos dessa maneira.

De uma maneira geral, o sistema de transmissão binário ponto a ponto, ilustrado na Figura 3.3, é um modelo típico em comunicações de dados. A fonte de alimentação fornece dígitos binários, isto é, bits que são convertidos num sinal elétrico ou eletromagnético pelo transmissor, e enviados ao receptor através do suporte de transmissão. O suporte de transmissão é o elemento que faz a ligação física, permitindo o transporte de informação binária (bits) entre o transmissor e o receptor. O receptor, por sua vez, tem a função de recuperar a informação transmitida a partir do sinal recebido e fornecê-la ao destinatário, completando o processo de comunicação no nível de bits.

Em geral, o processo é bidirecional, cada nó de comunicação necessita de um transmissor e de um receptor (Figura 3.4). A transmissão de bits pode ser feita simultaneamente nos dois sentidos (transmissão *full-duplex*) ou alternada, ora num sentido, ora no outro (transmissão *half-duplex*).

Ao longo da transmissão dos bits, podem ocorrer efeitos indesejáveis que têm a propriedade de alterar a forma do sinal transmitido. Esses efeitos são provocados por imperfeições do suporte de transmissão e por perturbações externas ao sinal, conhecidas como ruído. O ruído e as imperfeições ao longo do suporte são responsáveis pela ocorrência dos erros de transmissão que determinam o mau desempenho do sistema de transmissão de bits.

3.1.3.2 Modos de Transmissão

Para que os bits transmitidos possam ser recebidos adequadamente pelo destinatário da mensagem, deve ser mantida uma separação entre cada bit transmitido. A forma de se efetuar essa separação dá origem aos chamados modos de transmissão.

A utilização de variáveis espaço e tempo para realizar a separação entre os bits permite identificar dois modos elementares de transmissão:

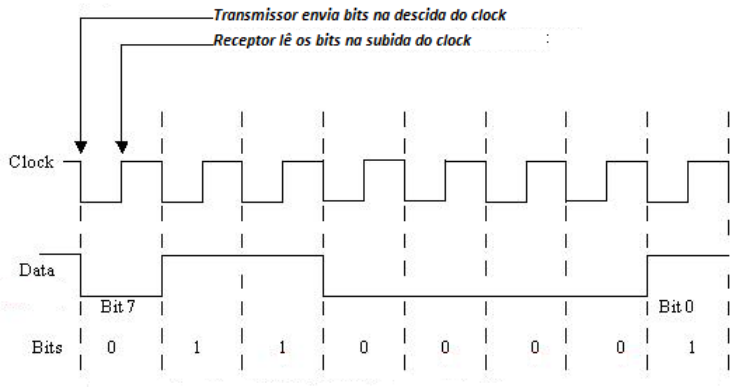


Figura 3.5: Comunicação Síncrona

(a) Série: no qual os bits são transmitidos um a um de cada vez (variável tempo), utilizando um único meio físico.

(b) Paralelo: no qual os bits são transmitidos simultaneamente através de vários suportes físicos em paralelo (variável espaço).

A forma de se utilizar a variável tempo para separar os bits em uma transmissão em série permite distinguir dois outros modos de transmissão:

(a) Síncrona: baseia-se no estabelecimento de uma cadência fixa para transmissão sequenciada dos bits, conforme ilustra a Figura 3.5. A cada T segundos o transmissor emite, por exemplo, um pulso de tensão elétrica (ou corrente), que significa o envio de um símbolo binário 1, ou nenhum pulso, que significa o envio de um bit 0. O receptor, que conhece os intervalos de tempo representativos dos bits, identifica a sequência de bits transmitida, fazendo uma amostragem do sinal recebido a intervalos regulares de T segundos. A amostragem deve, portanto, ser feita em instantes apropriados na mesma cadência utilizada na emissão. Para isso, o receptor deve conhecer a priori a temporização obedecida pelo transmissor. Essa temporização básica corresponde à onda de relógio de período T segundos que estabelece a taxa ou velocidade de transmissão $1/T$, expressa em bits por segundo.

(b) Assíncrona: caracteriza-se por não exigir a fixação prévia de um padrão de tempo entre o transmissor e o receptor. A separação entre dois bits de informação é feita através de um símbolo ou sinal especial de duração variável.

Na prática, é comum utilizar uma transmissão por caracteres, tipo *start-stop* (Figura 3.6). Nesse tipo de transmissão, a cada conjunto de bits que forma um caractere, é adicionado um elemento de sinalização que marca seu início (*start*) e outro que marca seu fim (*stop*). Essa estratégia de transmissão permite que os caracteres possam ser transmitidos e recebidos sem levar em conta os caracteres precedentes ou seguintes. O termo assíncrono refere-se à aleatoriedade dos instantes de emissão e, consequentemente, dos instantes de recepção. Em outras palavras, o intervalo de tempo entre dois caracteres transmitidos pode ser variável sem que o receptor tome conhecimento.

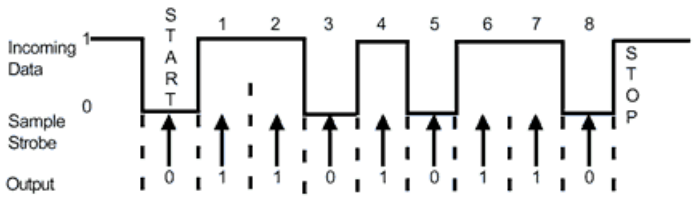


Figura 3.6: Comunicação Assíncrona.

3.1.3.3 Erros de transmissão

O sinal transmitido através do suporte de transmissão e dos estágios de entrada do receptor pode sofrer alterações da sua forma original. Essas alterações resultam de imperfeições na propagação do sinal e de perturbações conhecidas como ruído, que atuam não só no suporte de transmissão, como também nos estágios de processamento do sinal que compõem o receptor.

Um dos critérios utilizados para avaliar o desempenho de um sistema de transmissão de dados é o seu grau de confiabilidade na transmissão dos bits. Essa medida é geralmente expressa pela taxa de erros que representa a probabilidade de ocorrência de erros de transmissão. Assim, por exemplo, um sistema de transmissão com taxa de erros de 10^{-9} corresponde a um sistema onde em média ocorre um erro de transmissão a cada um bilhão de bits transmitidos.

Os valores aceitáveis para a taxa de erros dependem das aplicações envolvidas. A existência de funções de controle e recuperação de erros associadas aos protocolos de comunicação na camada de enlace introduz a noção de taxa de erros residual ou taxa de erros não detectados.

3.1.3.4 Comunicação por cabos

As razões da larga utilização de cabos são fáceis de ser explicadas: podem ser facilmente instalados, são baratos e relativamente imunes a ruídos. Todavia, os sistemas de transmissão por cabos possuem diversas limitações que devem ser consideradas, como limitação da banda passante, distorções de fase e distorções não lineares.

3.1.3.5 Sistemas banda-base

Uma sequência de pulsos deve ser considerada, do ponto de vista de transmissão, como um sinal analógico, da forma:

$$s(t) = \sum_{n=-\infty}^{\infty} a_n g(t - nT) \tag{3.1}$$

Onde:

a_n: amplitude do sinal fornecido pela fonte;
$g(t)$: forma dos pulsos;
T: duração dos pulsos

O sistema no qual o sinal analógico transmitido possui a forma acima é chamado sistema banda-base e constitui a maneira mais simples de transmitir informação digital em forma analógica.

Sistema Banda-base com Pulsos Retangulares Um caso particular importante de sistemas banda-base é aquele no qual os pulsos utilizados possuem forma retangular com amplitude binária, isto é, $a_n = \pm 1$ (valores normalizados). A importância desse sistema decorre do fato de ele ser utilizado nas formas do tipo RS-232, que são largamente empregadas nas comunicações entre computadores e equipamentos periféricos.

De fato esse sistema é bastante simples e pode ser utilizado desde que as seguintes condições sejam satisfeitas:

- Baixas taxas de transmissão (até $9600\,bps$);
- Pequena distância entre o transmissor e o receptor (até $15\,m$, segundo RS-232);
- Grande disponibilidade de banda de transmissão;
- Distorções de fase do meio irrelevantes.

3.1.3.6 Interface física RS-232

Existe uma longa lista de normas disponíveis relativas à interface física, a mais conhecida e utilizada é a EIA RS-232C[5].

Características Mecânicas Essas características definem o tipo de conector a ser utilizado, a atribuição dos circuitos dos pinos no conector e outras características de montagem. Os conectores mais utilizados são o DB9 e o DB25 (Figura 3.7).

Características Elétricas As características elétricas da interface RS-232C correspondem a um circuito não balanceado com tensão de saída bipolar. Antigamente a eletrônica exigia que as tensões fossem mais altas e estivessem em um intervalo que abrangessem valores positivos e negativos. Por isso os intervalos de tensão RS-232 são de $-12V$ a $12V$.

Características Funcionais A interface padrão RS-232 é uma interface para transferência serial de dados a uma taxa de transmissão de até $20\,kbps$, com uma distância entre os nós de até $15\,m$.

Cada comunicação através da interface RS-232 envolve uma troca inicial de sinais de controle (RTS, CTS); uma fase de transferência de dados propriamente dita, onde atuam essencialmente os sinais de transmissão (TD) e recepção (RD), os sinais de relógio (RC, TC ou ETC) e os sinais de monitoração (por exemplo, DCD); e finalmente, uma fase de finalização com a desativação dos sinais de controle inicializados.

[5]Sobre: http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html

Figura 3.7: Conectores DB25 e DB9

3.1.3.7 Interrupção de hardware

Interrupção é um sinal de um dispositivo que indica a necessidade de atenção, mudando a forma da execução do software. A interrupção de hardware força o processador a salvar seu estado de execução e começar a executar uma nova subrotina (ISR).

Existem vários tipos de interrupção de hardware:

- Nível Desencadeado: a presença da interrupção é indicada por um estado particular, nível alto ou baixo (0 ou 1).
- Borda Desencadeada: a interrupção é sinalizada por uma borda de descida (1 a 0) ou por uma borda de subida (0 a 1).
- Híbrido: alguns sistemas utilizam um híbrido do nível e da borda desencadeada, o hardware não procura só pela borda, mas também verifica o estado do sinal por um certo período de tempo.
- Mensagem Sinalizada: esse tipo de interrupção não utiliza uma linha física, e sim uma pequena mensagem pelo canal de comunicação, tipicamente, o barramento.
- Campainha: mecanismo pelo qual um sistema de software pode sinalizar ou verificar um dispositivo de hardware que tenha algum trabalho para ser feito.

As linhas de interrupção devem possuir resistores *pull-down* ou *pull-up*, para não serem ativarem acidentalmente.

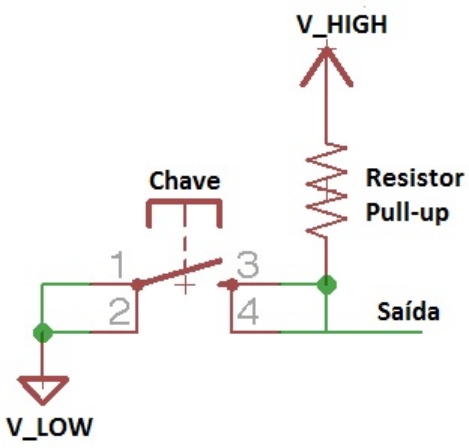


Figura 3.8: Resistor Pull-up

Resistores *Pull-up* e *Pull-down* Resistores *pull-up* e *pull-down* são utilizados em projetos de circuitos lógicos eletrônicos para garantir que as entradas se ajustem em níveis lógicos esperados se algum dispositivo externo estiver desconectado.

O objetivo desses resistores é "puxar" (*pull*) sutilmente a tensão do condutor para nível alto ou baixo, contudo o resistor é intensionalmente fraco (alta resistência) para que qualquer outro sinal possa puxar bruscamente a tensão para o nível oposto.

3.1.4 Camada de Enlace

A principal função dessa camada é enquadrar os dados a serem enviados e detectar possíveis erros de transmissão. Ela é responsável pela transição dos bits, de forma que possam ser entendido por níveis superiores (módulos e *host*).

3.1.4.1 Delimitação de Quadros e *Checksum*

Quando o fluxo de bits chega na camada de enlace, ele é divido em quadros que são submetidos a uma soma de verificação chamada *checksum*. Ao chegarem no receptor, esse, por sua vez, faz uma verificação de erro para descartar quadros defeituosos.

A soma de verificação (ou *checksum*) é um código utilizado para verificar a integridade dos dados transmitidos através de um canal com ruído. O funcionamento é simples: na transmissão, somam-se alguns bytes específicos do pacote e esse resultado é enviado junto com o quadro. Na recepção, somam-se novamente os bytes específicos e compara-se o resultado com o byte de soma recebido. Caso sejam diferentes, o quadro é considerado corrompido e é descartado.

3.1.5 Camada de Rede

A Camada de Rede é responsável pelo endereçamento de cada módulo no sistema, de forma que os pacotes sejam entregues no endereço correto. Essa camada também é responsável pela rota que os pacotes devem seguir, ou seja, como os módulos devem estar conectados entre si.

3.1.6 Camada de Apresentação

Essa camada é responsável por transformar os dados recebidos dos pacotes em valores funcionais, e também por converter o formato do dado recebido em um formato comum que será utilizado pelos módulos ou pelo *host*.

3.1.7 Camada de Aplicação

A Camada de Aplicação é a camada mais próxima do usuário, na qual são implementadas rotinas de interface entre o sistema de comunicação e o programa desenvolvido. É nessa camada que o usuário define qual a aplicação do robô modular e, dentro dela, como será a sua movimentação.

Capítulo 4

Desenvolvimento do Protocolo

Apresentação do Protocolo de Comunicação proposto para o ErekoBot.

4.1 Arquitetura

Existem duas metodologias que podem ser utilizadas para desenvolver um projeto que envolva transmissão de dados: *top-down* ("de cima para baixo") e *bottom-up* ("de baixo para cima"). O primeiro parte de uma visão ampla do sistema proposto e, gradualmente, define componentes menores. Já o segundo, parte da definição dos componentes mais básicos e utiliza-os para, gradualmente, incrementar a complexidade do sistema.

No caso do ErekoBot, uma vez que o hardware é conhecido em detalhes, a melhor abordagem para o desenvolvimento do protocolo é a *bottom-up*, ou seja, as características e funções das camadas (Seção 3.1.2) do Protocolo são definidas de baixo para cima: Camada Física, Camada de Enlace, Camada de Rede, Camada de Transporte, Camada de Sessão, Camada de Apresentação e Camada de Aplicação.

4.1.1 Camada Física

As características da camada física do Protocolo foram baseadas nos modelos de Sistema Banda-Base com Pulsos Regulares, utilizando comunicação por cabos (Seção 3.1.3).

Essas características são descritas e justificadas na Tabela 4.1.

Para a comunicação com o computador, utilizou-se a interface física RS-232 (Seção 3.1.3.6). Como o Atmel® ATmega 8® opera com tensões diferentes do RS-232, foi utilizado um CI conhecido como MAX232[1]. Esse CI, desenvolvido pela empresa Maxim IC®, converte os sinais $\pm 12V$ do RS-232 para os $0V$ a $5V$ utilizados por grande parte dos microcontroladores do mercado.

Para o envio dos dados do computador para o Atmel® ATmega 8® e vice-versa, deve-se passar o sinal pelo MAX232®. A imagem da Figura 4.1 mostra o circuito da interface RS-232.

[1] O *datasheet* pode ser encontrado em: http://www.ti.com/lit/ds/symlink/max232.pdf

Característica	Descrição/Justificativa
Half-Duplex	Tanto os módulos quanto o *host* podem enviar e receber dados, alternadamente, devido às limitações do Atmel® ATmega 8®.
Serial	Os dados são enviados bit a bit, utilizando apenas os cabos como meios físicos. Utiliza-se a porta de comunicação USART (formato padrão para comunicação de dados de forma serial) do Atmel® ATmega 8®, que é serial.
Assíncrona	A qualquer momento o *host* pode enviar dados para o sistema.
Taxa de transmissão de 9600 bps	A cada segundo, 9600 bits são transmitidos pelos cabos, uma das taxas mais utilizadas pelo Atmel® ATmega 8® e pela RS-232.
Interrupção a nível baixo	A interrupção do Atmel® ATmega 8® é do tipo borda de descida desencadeada.

Tabela 4.1: Características do Protocolo

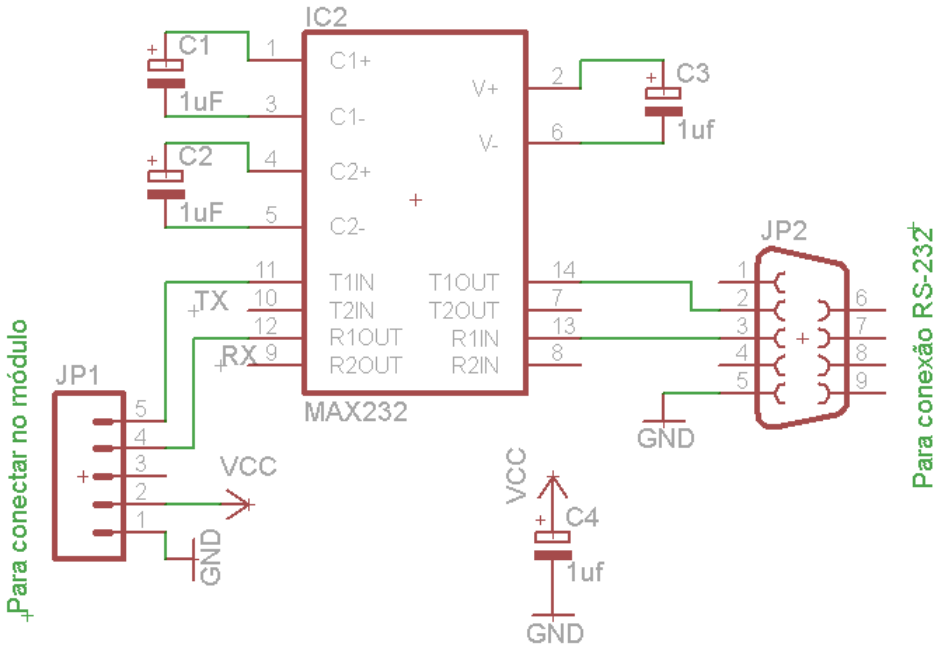


Figura 4.1: Circuito da interface RS-232

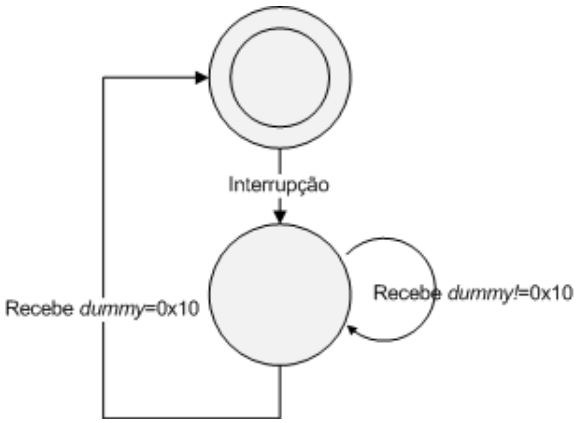


Figura 4.2: Sincronização de pacotes

Para sua operação, o MAX232® utiliza três capacitores (C_1, C_2 e C_3) de $0.1\mu F$ [2]. O quarto capacitor (C_4) é chamado de "capacitor de dissociação". Conforme o MAX232® muda as tensões, também utiliza parte da corrente da fonte de $5V$, que ocorre em rajadas, o que pode prejudicar a comunicação. Esse capacitor de dissociação ajuda a remover esse efeito, diminuindo o ruído de alimentação do sistema.

4.1.2 Camada de Enlace

4.1.2.1 Sincronização lógica entre as entidades

Como descrito na Seção 3.1.3.2, o Atmel® ATmega 8® sincroniza apenas a recepção de bytes, ou seja, não é capaz de sincronizar a recepção de um pacote [3].

O primeiro passo para o enquadramento do pacote é garantir a sincronização lógica, nome dado à sincronização de pacotes. Para que ela ocorra no Protocolo, o pacote começa com um byte de valor único reservado ($10H$). Quando um módulo recebe uma interrupção, que indica o recebimento de um byte, permanece em um *loop*, esperando o recebimento do byte de sincronização, como descrito na Figura 4.2.

4.1.2.2 Montagem e Delimitação dos pacotes

Existem, no total, dois tipos de pacotes no Protocolo [4]:

- Pacote Padrão (Figura 4.3(a)): 5 bytes que podem enviar cinco tipos diferentes de informação.
- Pacote de Configuração (Figura 4.3(b)): 11 bytes que têm a função de configurar os módulos, informando o novo endereço de cada um deles, o modo de operação (ou modo de movimen-

[2] Funcionam como *charge pumb*, ou seja, armazenam elétrons que são "bombeados" sem um transformador.

[3] Existe apenas um momento de enquadramento dentro de todo o Protocolo, então os termos pacote e quadro são equivalentes.

[4] O significado de cada um desses campos do pacote estão descritos na Seção 4.1.6.

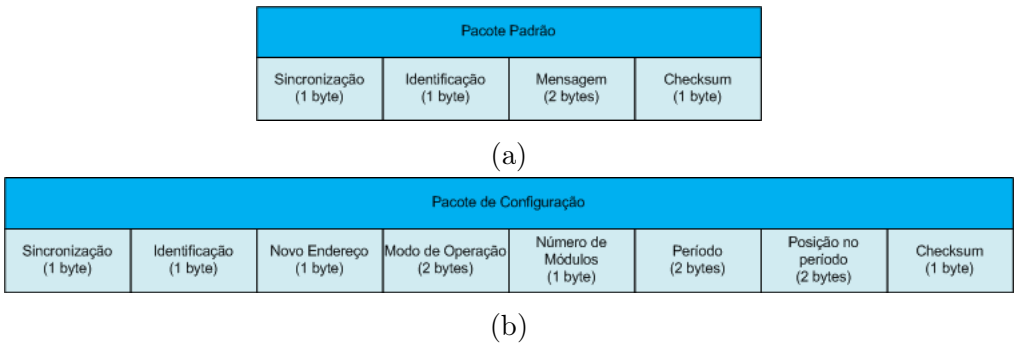


Figura 4.3: Tipos de pacote (a) Pacote Padrão (b) Pacote de Configuração

tação), o número de módulos do sistema, o período e o momento t no qual deve ser iniciada a movimentação (em relação ao período).

4.1.2.3 Controle de erros de transmissão

Todos esses pacotes são iniciados com um byte de sincronização, seguidos por um byte de IDENTIFICAÇÃO (endereço) e finalizados com um pacote de soma de verificação (*checksum*) para detecção de erros.

4.1.3 Camada de Rede

4.1.3.1 Roteamento

Com o Atmel® ATmega 8®, é possível criar apenas um canal de transmissão e um canal de recepção em cada módulo, como descrito na Seção 4.1.1. Por simplicidade, então, o protocolo foi desenvolvido na topologia linha (Figura 4.4(a)), que pode posteriormente ser adaptada para a topologia em barramento (Figura 4.4(b)).

O *host* transmite todos os dados para o primeiro ErekoBot, que, por sua vez, guarda as informações que são endereçadas a ele e repassa o restante para o segundo módulo, que faz o mesmo. Se a comunicação ocorrer com sucesso, quando a mensagem chegar no ErekoBot n, a informação terá sido entregue a todos os módulos do sistema.

É possível aindar "fechar" a linha, transformando a topologia em um anel, dessa forma o ErekoBot n poderia enviar mensagens para o *host*, como, por exemplo, a situação da bateria ou um erro de algum módulo.

4.1.3.2 Mapeamento de endereços lógicos físicos

Exceto pelos pacotes padrão que informam o início ou a parada do movimento, todos os outros pacotes são do tipo *unicast* (endereçamento de um pacote para um único destino), ou seja, o

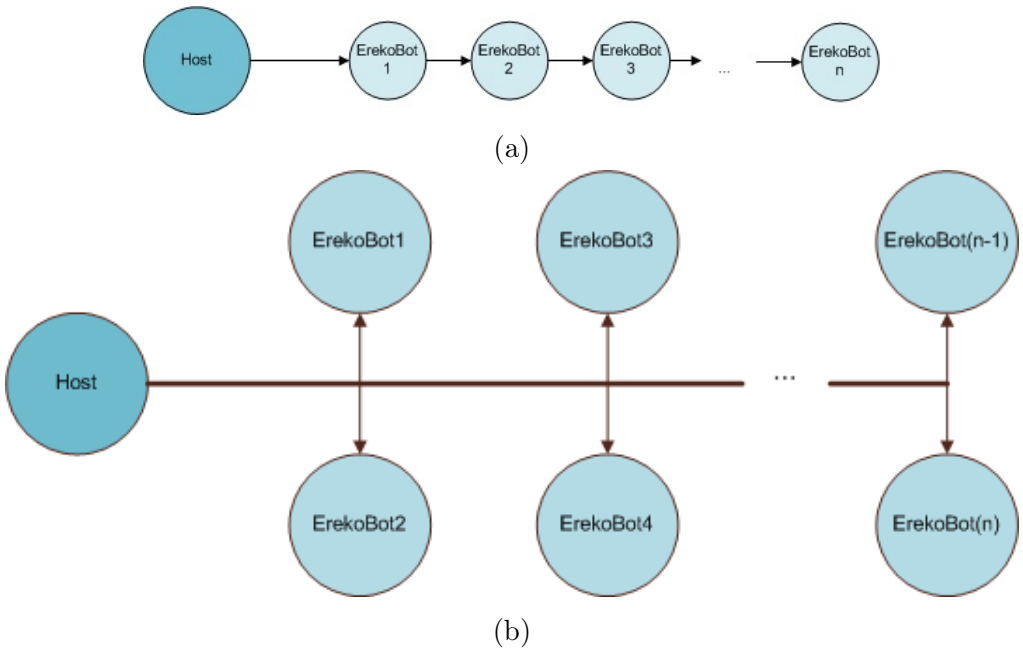


Figura 4.4: Topologia do protocolo para n ErekoBots (a) linha (b) barramento

transmissor preenche cada campo IDENTIFICAÇÃO com o endereço de cada módulo para o qual se deseja enviar um pacote (Figura 4.3).

A partir do algoritmo da Figura 4.2, constrói-se um algoritmo mais complexo, ilustrado na Figura 4.5. Ao receber um pacote, cada módulo verifica se o campo IDENTIFICAÇÃO corresponde ao seu próprio endereço. Se sim, o módulo utiliza a mensagem, se não, transmite esse pacote para o módulo seguinte.

No caso dos pacotes que informam o iníco ou a parada do movimento, o campo IDENTIFICAÇÃO é preenchido com o valor reservado $0CH$, que indica que o pacote é do tipo *broadcast*. Quando um módulo recebe um pacote *broadcast*, não só utiliza a informação como a transmite aos demais.

Os valores dos bytes de sincronização e de *broadcast* devem ser reservados, para que não sejam confundidos com outras informações e, assim, não comprometam a comunicação. Então, os valores dos endereços enviados pelo canal devem ser adaptados:

$$endereço_{envio} = endereço + 17 \tag{4.1}$$

Onde:

endereço:	valor que o usuário deseja enviar - de 0 ($00H$) a 49 ($31H$) [5]
$endereço_{envio}$:	valor que é enviado de fato pelo canal - de 17 ($11H$) a 66 ($42H$)

[5] Todos esses valores podem ser vistos no Anexo III

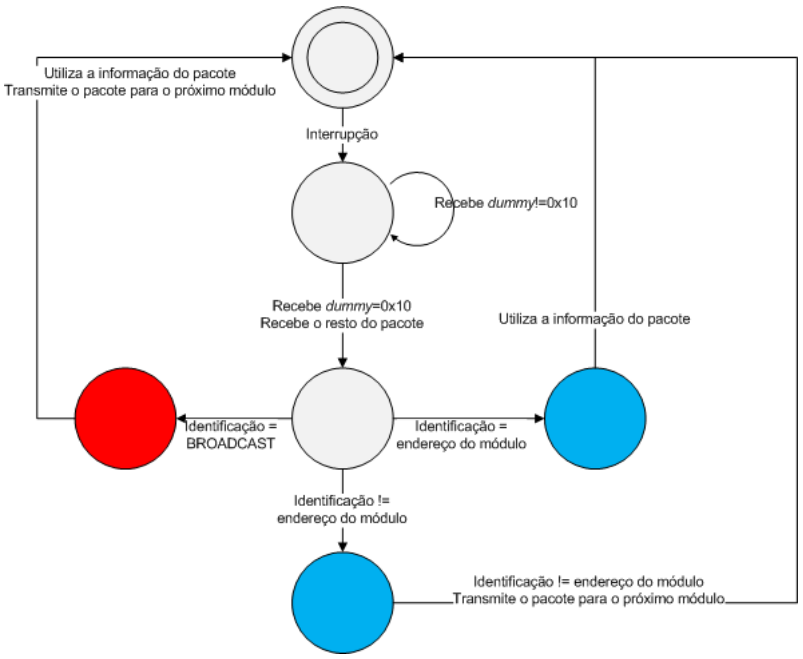


Figura 4.5: Identificação de pacotes*: broadcast* (vermelho) ou *unicast* (azul)

4.1.3.3 Verificação de erro de pacote

O campo CHECKSUM é preenchido, então, com a soma do byte do campo IDENTIFICAÇÃO com os dois bytes do campo MENSAGEM (para o Pacote Padrão) ou com a soma do byte do campo IDENTIFICAÇÃO com os bytes de NOVO ENDEREÇO e de MODO DE OPERAÇÃO (para o Pacote de Configuração). No receptor esses valores são conferidos, se forem diferentes, o pacote é descartado.

4.1.4 Camada de Transporte

Devido à utilização, unicamente, de comunicação por fios, não foram implementadas no Protocolo funções da Camada de Transporte, tais como: gestão de perdas, duplicação de pacotes, entregas fora de ordem, mensagens de confirmação para entrega confiável ou controle de tráfego.

4.1.5 Camada de Sessão

Pelo mesmo motivo que as funções da Camada de Transporte não foram implementadas, o mesmo ocorreu com a Camada de Sessão, e a função de gestão do controle de diálogo não foi elaborada para o Protocolo.

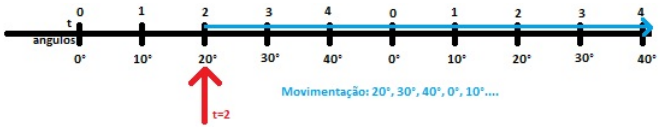


Figura 4.6: Exemplo de um módulo que se movimenta com período $T = 5$ e *gait* 0°, 10°, 20°, 30°, 40°

4.1.6 Camada de Apresentação

4.1.6.1 Sintaxe dos dados recebidos

Como foi apresentado na Seção 4.1.2, existem dois tipos de pacotes dentro do Protocolo, que podem produzir mensagens diferentes.

4.1.6.2 Pacote Padrão

Esse tipo de pacote possui cinco bytes e o campo MENSAGEM pode ser preenchido com cinco informações diferentes: o ângulo no qual o servo deve se posicionar, o momento t em que os módulos devem começar o movimento, o início da movimentação (*start*), a parada da movimentação (*stop*) e o início do recebimento do pacote de configuração (*config*).

MENSAGEM $= \hat{A}ngulo$ A modulação por largura de pulso (PWM) é utilizada para controlar o servo dos módulos e, para facilitar o algoritmo, os valores de ângulos enviados já são os próprios valores PWM. Ou seja, para um transmissor enviar um pacote com uma mensagem do valor de ângulo que deve ser aplicado, deve primeiramente converter esse valor para o valor correspondente em PWM.

MENSAGEM $= t$ Uma mensagem que também pode ser enviada pelo Pacote Padrão é a do tempo t em relação ao período T no qual o movimento deve ser iniciado. Por exemplo, na Figura 4.6, um módulo se movimenta com período $T = 5$ e *gait* 0°, 10°, 20°, 30°, 40°, se o módulo recebe um Pacote Padrão com a mensagem $t = 2$, deve iniciar sua movimentação a partir de 20°.

Para que os valores enviados no pacote estejam dentro do padrão e não sejam confundidos com outros valores fixos, t é convertido em t_{envio}:

$$t_{envio} = t + 439 \tag{4.2}$$

Onde:

t: valor que o usuário deseja enviar, de 0 ($00H$) a 49 ($31H$)
t_{envio}: valor que é enviado de fato pelo canal, de 439 ($1B7H$) a 488 ($1E8H$)

Mensagem = *Start* ($0FH$) Quando deseja-se iniciar o movimento, manda-se uma mensagem de requerimento de início de movimentação, com o valor fixo $0FH$.

Mensagem = *Stop* ($0DH$) Quando deseja-se parar o movimento, manda-se uma mensagem de requerimento de parada de movimentação, com o valor fixo $0DH$.

Mensagem = *Config* ($0EH$) Para mandar o Pacote de Configuração, deve-se mandar antes uma mensagem, que corresponderia a um cabeçalho. Essa mensagem tem o valor fixo $0EH$.

4.1.6.3 Pacote de Configuração

A função do Pacote de Configuração é configurar o ErekoBot, informando o novo endereço, o modo de operação, o número de módulos que existe em todo o sistema, o período e o momento t no qual os módulos devem começar o movimento.

Esse pacote é sempre precedido pelo Pacote Padrão com Mensagem = *Config* ($0EH$).

Campo Novo endereço Caso a aplicação necessite, é possivel, utilizando esse campo do pacote, reidentificar um módulo, dando a ele um novo endereço.

Esses valores são definidos da mesma forma que os endereços da Seção 4.1.3, *endereço_novo* é convertido em $endereço\_novo_{envio}$:

$$endereço\_novo_{envio} = endereço\_novo + 17 \tag{4.3}$$

Onde:

endereço_novo:	valor que o usuário deseja enviar, de 0 ($00H$) a 49 ($31H$)
$endereço\_novo_{envio}$:	valor que é enviado de fato pelo canal, de 17 ($11H$) a 66 ($42H$)

Campo Modo de Operação Esse campo é preenchido com um valor identificando em qual modo de operação o módulo deve se movimentar.

Para que os valores enviados no pacote estejam dentro do padrão e não sejam confundidos com outros valores fixos, *modo_operação* é convertido em $modo\_operação_{envio}$:

$$modo\_operação_{envio} = modo\_operação + 339 \tag{4.4}$$

Onde:

$modo\_operação$:	valor que o usuário deseja enviar, de 0 ($00H$) a 29 ($1DH$)
$modo\_operação_{envio}$:	valor que é enviado de fato pelo canal, de 339 ($153H$) a 368 ($170H$)

Campo Número de Módulos do Sistema Em muitas aplicações na Robótica Modular é necessário que os módulos saibam quantos deles existem no sistema. Esse campo, então, é preenchido com essa informação.

Esses valores são definidos da mesma forma que os endereços da Seção 4.1.3, m é convertido em m_{envio}:

$$m_{envio} = m + 17 \tag{4.5}$$

Onde:

m:	valor que o usuário deseja enviar, de 0 ($00H$) a 49 ($31H$)
m_{envio}:	valor que é enviado de fato pelo canal, de 17 ($11H$) a 66 ($42H$)

Campo Período É necessário que cada módulo saiba quantos dados existem dentro do *gait*. Esse campo, então, é preenchido com essa informação.

Esses valores são definidos da mesma forma que o t da Seção 4.1.6.2, T é convertido em T_{envio}:

$$T_{envio} = T + 439 \tag{4.6}$$

Onde:

T:	valor que o usuário deseja enviar, de 0 ($00H$) a 49 ($31H$)
T_{envio}:	valor que é enviado de fato pelo canal, de 439 ($1B7H$) a 488 ($1E8H$)

Campo t Por fim, também é necessário que cada módulo saiba o momento dentro do período que deve começar a movimentação. Esse campo, então, é preenchido com esse valor.

Esses valores são definidos da mesma forma que t da Seção 4.1.6.2, t é convertido em t_{envio}:

$$t_{envio} = t + 439 \tag{4.7}$$

Onde:

t:	valor que o usuário deseja enviar, de 0 ($00H$) a 49 ($31H$)
t_{envio}:	valor que é enviado de fato pelo canal, de 439 ($1B7H$) a 488 ($1E8H$)

4.1.7 Camada de Aplicação

Existem quatro tipos de aplicações no Protocolo, denominadas Perfis de Comunicação.

4.1.7.1 Primeiro Perfil

É o perfil mais simples, no qual os módulos são totalmente dependentes do *host* para locomoção, já que não existe nenhum tipo de autonomia por parte de cada ErekoBot. O *host* deve calcular todos os valores para cada um dos módulos e transmiti-los para todo o sistema em tempo real.

Nesse perfil, apenas o Pacote Padrão com MENSAGEM = $\hat{A}ngulo$ é enviado pelo canal.

Receptor Ao ser iniciado, o receptor configura a comunicação e configura o sinal PWM do motor, permanecendo em um estado de *loop*, no qual não faz nada até a ocorrência de uma interrupção. Ao ocorrer essa interrupção, o programa espera o recebimento do byte de sincronização e, então, o módulo recebe o pacote. A partir do pacote recebido sem erros, o ErekoBot verifica se o endereço contido no campo IDENTIFICAÇÃO é igual ao seu próprio endereço, caso seja, aplica o valor do PWM no motor, caso não, envia o pacote para o próximo módulo (Figura 4.7). Nesse perfil não existe envio *broadcast*, apenas *unicast*.

Transmissor Foi desenvolvido para o Primeiro Perfil um programa em Java para possibilitar a transmissão de pacotes com diversos valores de ângulos para os módulos, através da interface RS-232 (Figura 4.8).

Inicialmente, dois campos devem ser preenchidos:

- QUANTOS MÓDULOS (DE 1 A 10)? - determina a quantidade de módulos que existirá no sistema (m), entre 1 e 10 módulos.
- TAMANHO DO PERÍODO (DE 1 A 32)? - determina a quantidade de dados que existe nas tabelas (T), entre 1 e 32 valores.

Após o correto preenchimento desses valores, é possível apertar o botão "Inicializar tabelas" para que novas abas sejam habilitadas. Em cada aba, os campo "Endereço" é inicializado com valores de 0 a $m-1$ ($módulo_1$ tem $endereço = 0$, $módulo_2$, $endereço = 1$, assim por diante, até o $módulo_m$). Já os campos t são inicializados com valores de 0 a $T-1$. Por fim, os campos Ângulo e PWM são inicializados com os valores zero.

Em cada uma das abas habilitadas é possível editar o *endereço* dos módulos, de 0 a 50, e os valores dos ângulos, de $-90°$ a $90°$.

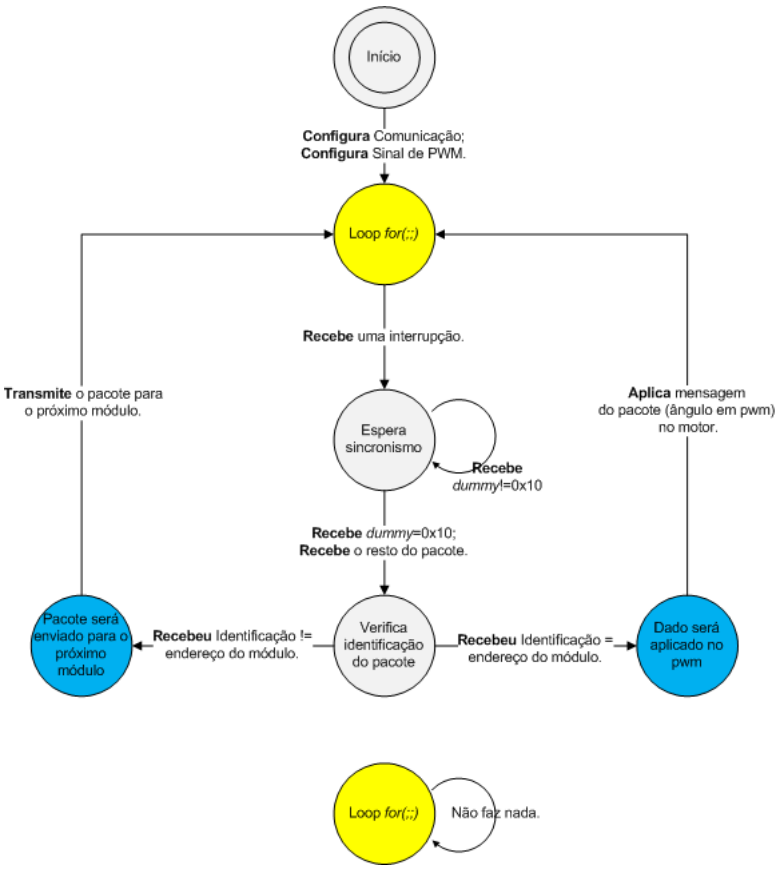


Figura 4.7: Primeiro Perfil: *unicast* (azul) e *loop* (amarelo)

Após preencher os campos e apertar o botão "Converter", é possível converter os valores dos ângulos para o sinal PWM a partir de uma equação de aproximação de sinal, estabelecida por tentativa e erro[6]:

$-90°$corresponde aproximadamente ao sinal 75

$0°$corresponde aproximadamente ao sinal 195

$90°$corresponde aproximadamente ao sinal 315

Então:

$$Sinal_{PWM} = 195 + (4/3) * \theta \tag{4.8}$$

Onde:

$Sinal_{PWM}$:	valor do sinal que será aplicado no servomotor.
θ:	valor de ângulo preenchido pelo usuário

Por fim, o usuário pressiona o botão "Start!" e os valores são enviados através da interface RS-232 para os módulos e a qualquer momento o usuário pode pressionar o botão "Stop!" para interromper essa transmissão.

O botão "Limpar Tabela" serve para retirar todos os valores das tabelas e reiniciar a configuração do sistema. A figura no canto superior direito ilustra os valores dos ângulos que podem ser movimentados pela haste do servo do módulo.

O programa converte os valores de m, T e *endereço*, utilizando as equações 4.5, 4.6 e 4.1.

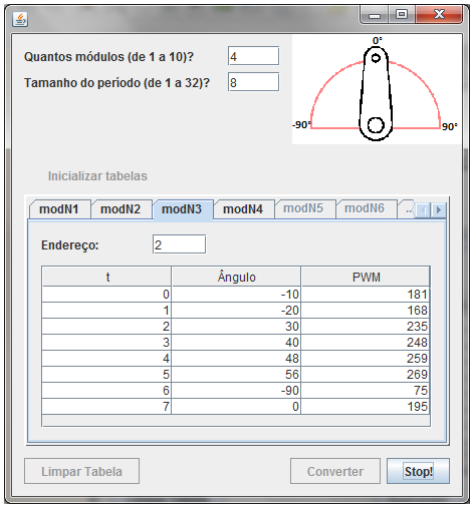


Figura 4.8: Programa do Primeiro Perfil

[6]É aconselhado que esses valores sejam revisados em trabalhos futuros relacionados ao controle de posição.

4.1.7.2 Segundo Perfil

Nesse perfil, os módulos já possuem certa autonomia, pois cada um deles tem uma Tabela de *Gait*s. O papel do *host* passa a ser apenas o de informar em qual momento t os módulos devem iniciar o movimento e enviar sinais de início e parada.

Nesse perfil são enviados os seguintes tipos de pacote:

- Pacote Padrão com mensagem = t;
- Pacote Padrão com mensagem = *start*;
- Pacote Padrão com mensagem = *stop*.

Receptor Ao ser iniciado, o receptor configura a comunicação e o sinal PWM do motor, permanecendo em um estado de *loop*, no qual não faz nada enquanto o sinal de início (*start*) é falso. Caso o sinal seja verdadeiro, procura na Tabela de *Gaits* o valor PWM do servo motor correspondente ao estado atual (t) e aplica-o no servo.

No caso de uma interrupção de hardware, o programa espera o recebimento do byte de sincronização e, então, o módulo recebe o pacote. Se o pacote for recebido corretamente e possuir um endereço *broadcast*, o receptor compara o campo MENSAGEM com os bytes fixos *start* e *stop*. Caso seja igual ao byte de *start*, o módulo muda o valor do sinal de início para verdadeiro, e caso contrário, muda o valor para falso.

Se todas as verificações acima forem falsas, provavelmente é um pacote *unicast*, então o módulo verifica se o endereço do pacote é igual ao seu próprio endereço. Se for igual, o módulo procura os valores de ângulos da Tabela a partir do valor t informado pelo pacote, salvando esse valor na variável *estadoinicial*. Caso contrário, envia a mensagem para o próximo módulo (Figura 4.10).

Transmissor Foi desenvolvido para o Segundo Perfil um programa em Java para possibilitar a transmissão de Pacotes Padrão com mensagens t, *start* ou *stop*, através da interface RS-232 (Figura 4.10).

Inicialmente, três campos devem ser preenchidos:

- QUANTOS MÓDULOS (DE 1 A 10)? - determina a quantidade de módulos que existirá no sistema (m), entre 1 e 10 módulos.
- TAMANHO DO PERÍODO (DE 1 A 32)? - determina a quantidade de dados que existe nas tabelas (T), entre 1 e 32 valores.
- COMEÇAR DE QUAL t? - define o momento (t) no qual o movimento começará, entre 1 e T valores.

Após o correto preenchimendo desses valores, é possível apertar o botão "Inicializar endereços" para que novas abas sejam habilitadas. Em cada aba, o campo "Endereço" é inicializado com valores de 0 a $m-1$ ($módulo_1$ tem $endereço = 0$, $módulo_2$, $endereço = 1$, assim por diante, até o $módulo_m$).

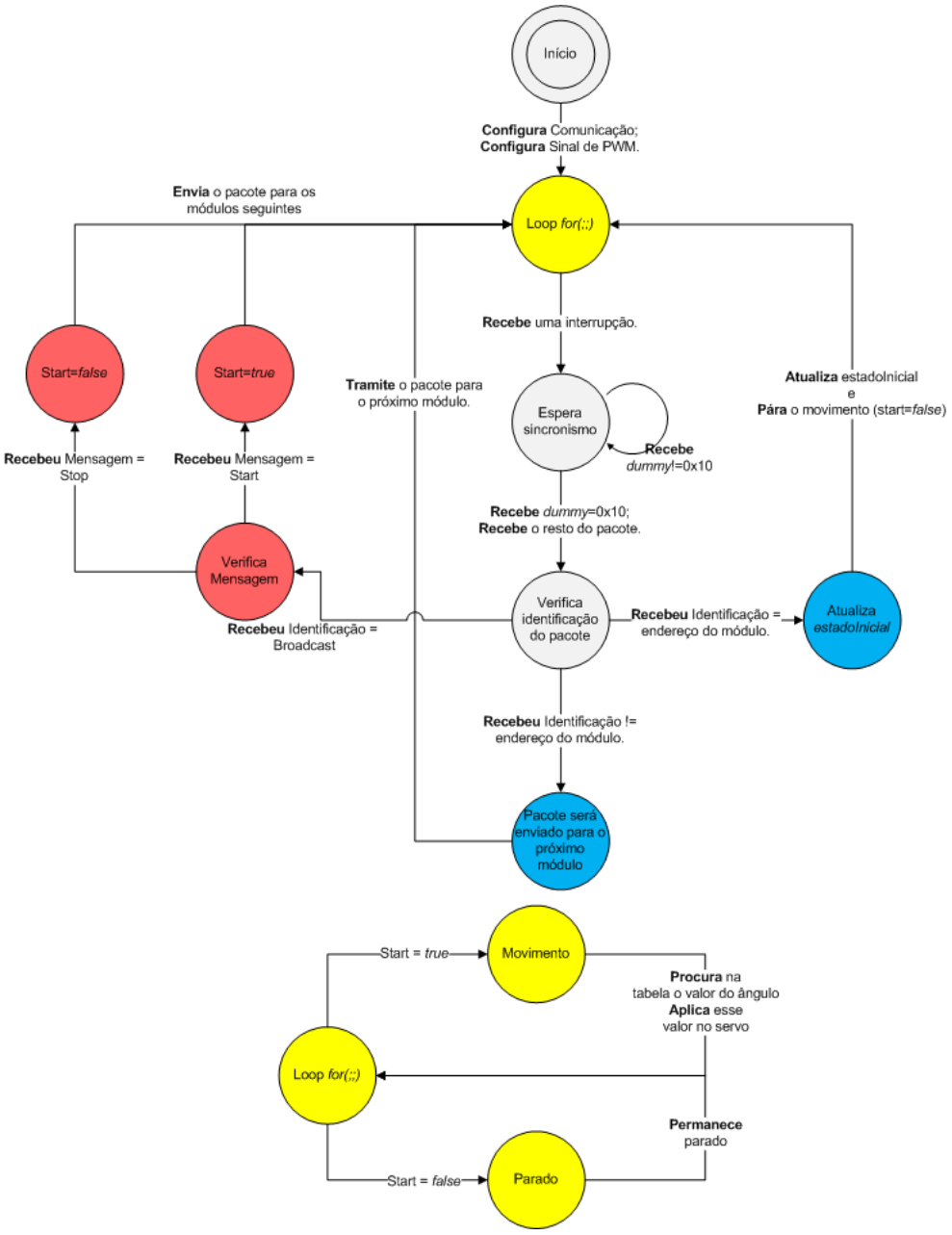


Figura 4.9: Segundo Perfil*: broadcast* (vermelho), *unicast* (azul) e *loop* (amarelo)

Em cada uma das abas habilitadas é possível editar o campo “Endereço”, com valores de 0 a 50.

Após preencher os campos e apertar o botão “Configurar”, é possível enviar aos módulos os Pacotes Padrão com mensagem = t, descritos na Seção 4.1.6. São enviados m pacotes, cada um com o endereço de um módulo e com o t escolhido.

Por fim, o usuário pressiona o botão “Start” e o Pacote Padrão com mensagem = *start* ($0FH$) é enviado através da interface RS-232 para os módulos, que, por sua vez, iniciam o movimento. A qualquer momento o usuário pode pressionar o botão “Stop” para enviar o Pacote Padrão com mensagem = *stop* ($0DH$) para interromper o movimento.

O programa converte os valores de m, T, *endereço* e t utilizando as equações 4.5, 4.6, 4.1 e 4.2.

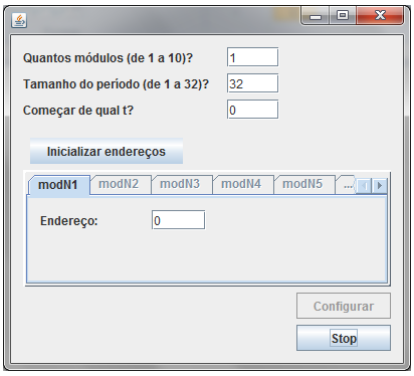


Figura 4.10: Programa do Segundo Perfil

4.1.7.3 Terceiro Perfil

É no Terceiro Perfil que os módulos começam a apresentar a possibilidade de reconfiguração, pois contêm internamente mais de uma Tabela de *Gaits*. O papel do *host* é configurar o sistema e enviar sinais de início e parada.

Nesse perfil são enviados os seguintes tipos de pacote:

- Pacote Padrão com mensagem = *start*;
- Pacote Padrão com mensagem = *stop;*
- Pacote de Configuração.

Receptor Ao ser iniciado, o receptor configura a comunicação e o sinal PWM do motor, permanecendo em um estado de *loop*, no qual não faz nada enquanto o sinal de início (*start*) for falso. Caso o sinal seja verdadeiro, procura a Tabela de *Gaits* na qual está configurado e, dentro dela, o valor correspondente PWM do servo motor para o estado atual (t), o qual aplica no servo (Figura 4.10).

No caso de uma interrupção de hardware, o programa espera o recebimento do byte de sincronização e, então, o módulo recebe o pacote. Se o pacote for recebido corretamente e possuir um endereço *broadcast*, o receptor compara o campo MENSAGEM com os bytes fixos *start* e *stop*. Caso seja igual ao byte de *start*, o módulo muda o valor do sinal de início para verdadeiro e, caso contrário, para falso.

No caso do recebimento de um pacote com endereço diferente de *broadcast*, o módulo verifica se a mensagem corresponde ao byte *config*. Em caso positivo, o programa entende que, em seguida, receberá um Pacote de Configuração, e, então, entra em um estado de *loop* à espera do recebimento de um byte de sincronização. Após o recebimento desse byte, o módulo compara o valor do endereço com o seu próprio e, se for diferente, envia para o próximo módulo o Pacote Padrão com Mensagem = *config* e o Pacote de Configuração.

Se o endereço for igual ao endereço do próprio módulo, todas as informações recebidas do Pacote de Configuração são configuradas:

- NOVO ENDEREÇO: o módulo procura a Tabela correspondente a esse endereço e não ao seu identificador;
- MODO DE OPERAÇÃO: esse valor corresponde à Tabela que o módulo deverá operar;
- NÚMERO DE MÓDULOS: algumas Tabelas dependem do número de módulos existente no sistema;
- PERÍODO: algumas Tabelas possuem quantidades diferentes de valores, para as quais o módulo deve ser configurado.

Transmissor Foi desenvolvido para o Terceiro Perfil um programa em Java para possibilitar a transmissão de Pacotes Padrão com mensagens *start* ou *stop* e de Pacotes de Configuração, através da interface RS-232 (Figura 4.11).

Inicialmente, quatro campos devem ser preenchidos:

- QUAL MODO DE OPERAÇÃO (DE 1 A 8)? - determina em que modo de operação interno os módulos devem se movimentar (*modo_operação*), entre 1 e 8 modos de operação. Esse modo de operação corresponde ao modo de locomoção que os módulos podem apresentar, dependendo de como estão conectados.
- QUANTOS MÓDULOS (DE 1 A 10)? - determina a quantidade de módulos que existirá no sistema (m), entre 1 e 10 módulos.
- TAMANHO DO PERÍODO (DE 1 A 32)? - determina a quantidade de dados que existe nas tabelas (T), entre 1 e 32 valores.
- COMEÇAR DE QUAL t? - define o momento (t) no qual o movimento começará, entre 1 e T valores.

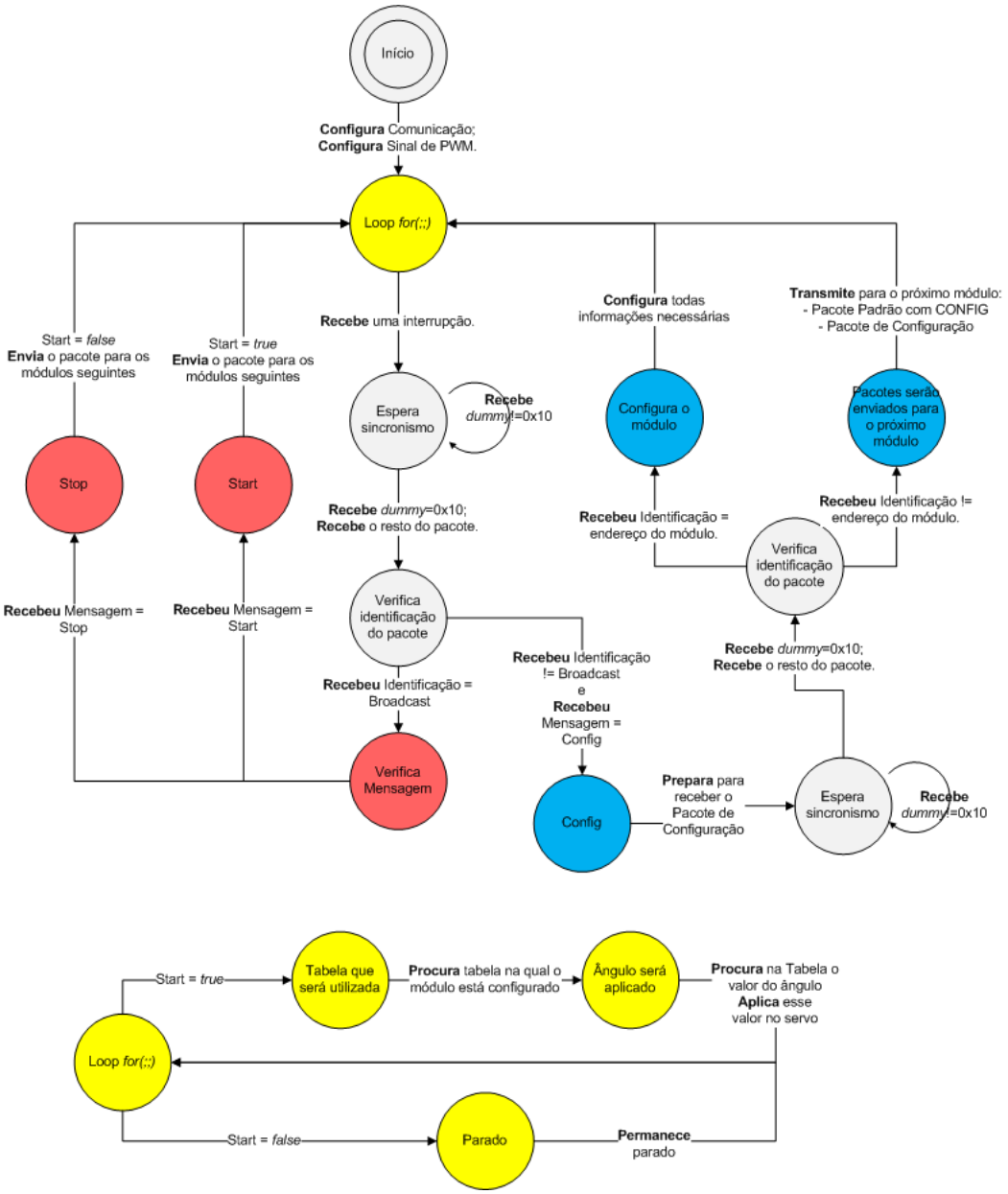


Figura 4.11: Terceiro Perfil*: broadcast* (vermelho), *unicast* (azul) e *loop* (amarelo)

Após o correto preenchimendo desses valores, é possível apertar o botão "Inicializar endereços" para que novas abas sejam habilitadas. Em cada aba, o campos "Endereço" é inicializado com valores de 0 a $m-1$ ($módulo_1$ tem $endereço = 0$, $módulo_2$, $endereço = 1$, assim por diante, até o $módulo_m$). O campo "Novo Endereço" é preenchido da mesma forma.

Em cada uma das abas habilitadas é possível editar os campos "Endereço" e "Novo Endereço", com valores de 0 a 50.

Após preencher os campos e apertar o botão "Configurar", é possível enviar aos módulos os Pacotes de Configuração descritos na Seção 4.1.6. São enviados m pacotes, cada um preenchido com as configurações determinadas acima.

Por fim, o usuário pressiona o botão "Start" e o pacote de mensagem = *start* ($0FH$) é enviado através da interface RS-232 para os módulos, que, por sua vez, iniciam o movimento. A qualquer momento o usuário pode pressionar o botão "Stop" para enviar o Pacote Padrão com mensagem = *stop* ($0DH$) para interromper o movimento.

O programa converte os valores de m, T, $endereço$, $endereço_{novo}$, t e $modo\_operação$ utilizando as equações 4.5, 4.6, 4.1, 4.3, 4.2 e 4.4.

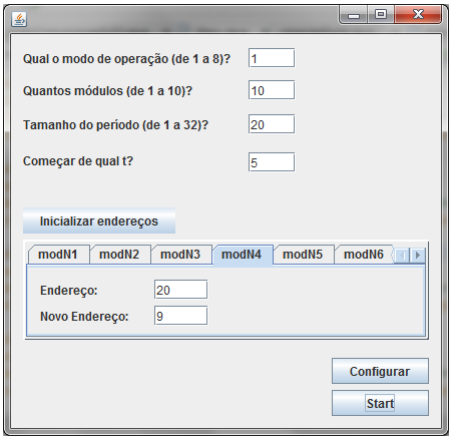


Figura 4.12: Programa do Terceiro Perfil

4.1.7.4 Quarto Perfil

No quarto e último perfil, os circuitos embarcados dos módulos já são reprogramáveis, ou seja, é possivel enviar, em tempo real, algoritmos, fórmulas e/ou Tabelas de *Gait*s. Portanto, o *host* passa a ter o papel de enviar essas informações para a reprogramação.

Esse perfil não foi implementado nem para o receptor nem para o transmissor.

Capítulo 5

Resultados Experimentais

Apresentação dos principais problemas do Protocolo, além das avaliações dos resultados dos testes de confiabilidade e de cálculo do atraso de propagação.

5.1 Introdução

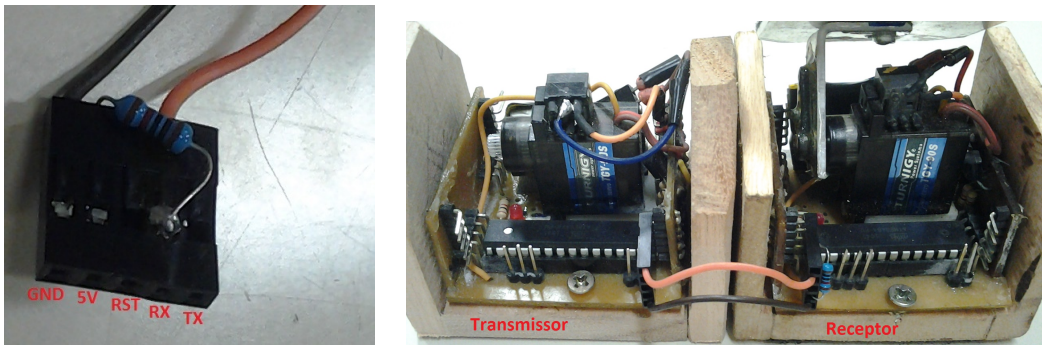
Conforme o protocolo foi utilizado, foram identificados problemas relacionados a ruídos e perda de sincronismo entre os módulos. Após a resolução desses problemas, testes de confiabilidade de transmissão e testes para a estimativa do atraso de transmissão foram realizados para comprovar a usabilidade e eficiência do protocolo.

5.2 Ruídos

Quando o Segundo e Terceiro Perfil foram executados, percebeu-se que o pino de recepção (RX) recebia ruídos e acabava habilitando a porta de interrupção. Dessa forma, o módulo entrava no *loop* de interrupção até receber um sinal de sincronização. Essa situação impede que o módulo continue a execução do programa, caracterizando uma condição de *deadlock*.

Para evitar esse problema, foi adicionado um resistor *pull-up* de $1\,k\Omega$ (entre o pino RX e o pino 5V) em cada módulo (Figura 5.1). Esse resistor impede que pequenos ruídos atrapalhem a comunicação, causando interrupções indesejáveis.

Como garantia de que, mesmo com o resistor *pull-up*, o sistema não entre em *deadlock*, foi realizada uma alteração nos algoritmos dos receptores do Segundo e do Terceiro perfis, descritos nas Seções 4.1.7.2 e 4.1.7.3 (Figuras 5.2 e 5.3). Depois da recepção do sinal de interrupção, o módulo espera o byte de sincronização por um tempo máximo. Caso esse tempo termine e a sincronização não ocorra, o programa sai da *thread* de interrupção e retorna para o programa principal. Essa alteração impede que o sistema entre em *deadlock*.



(a) (b)

Figura 5.1: Resistor *Pull-up* (a) Adicionado ao conector (b) Dois módulos conectados

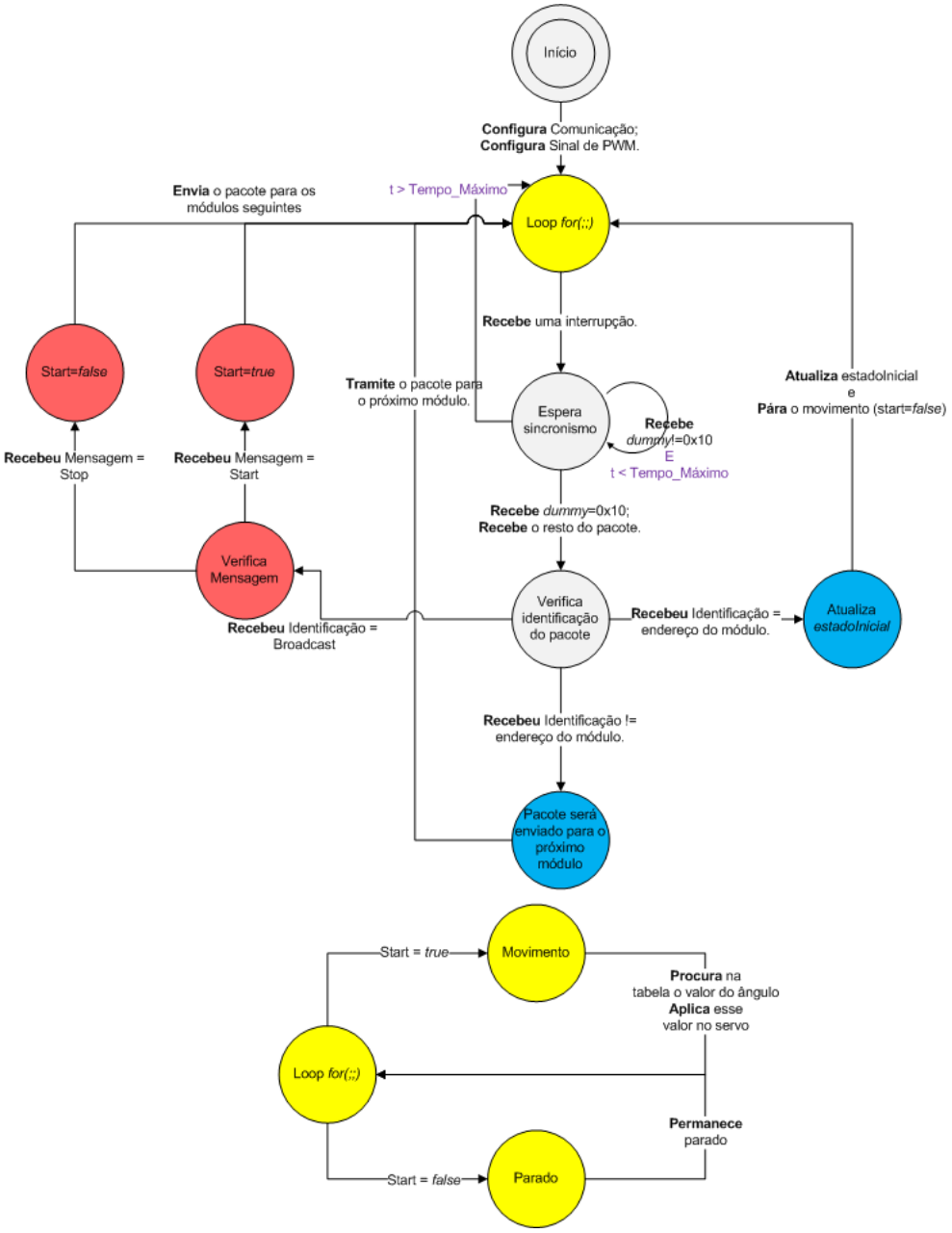


Figura 5.2: Segundo Perfil Alterado (roxo)

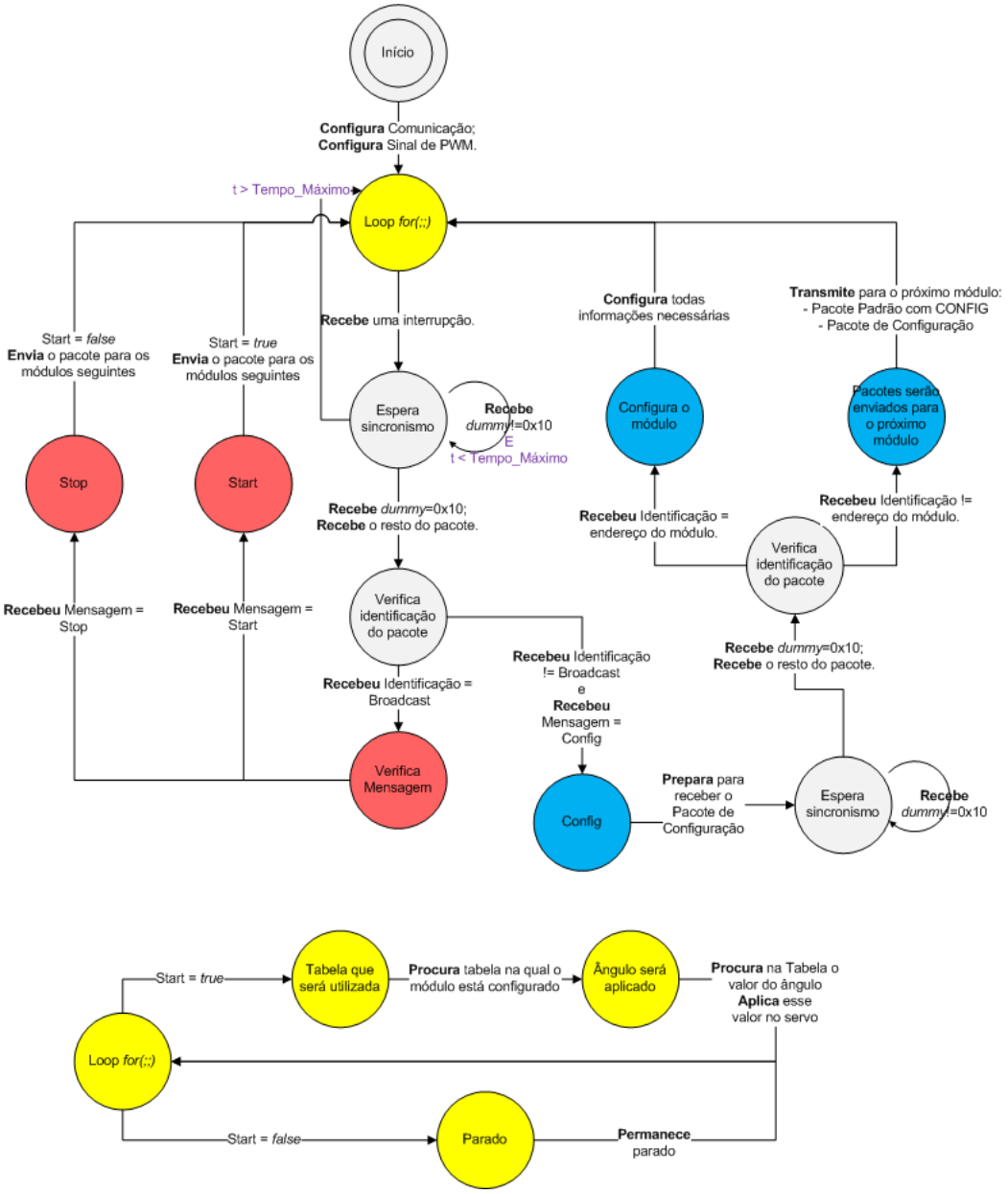


Figura 5.3: Terceiro Perfil Alterado (roxo)

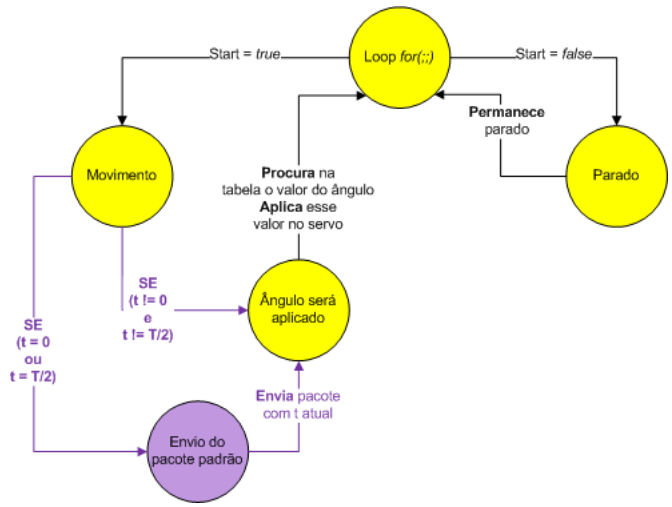


Figura 5.4: Alteração no algoritmo do Módulo Servidor de Tempo no Segundo Perfil (roxo)

5.3 Perda de Sincronismo

Após a recepção do sinal de início (*start*) no Segundo e Terceiro Perfis, os módulos começam o movimento seguindo suas Tabelas de Gaits internas. Na teoria, todos os módulos realizam o processamento com a mesma frequência de *clock* (6 MHz), mas, na prática, os osciladores internos dos Atmel® ATmega 8® são diferentes.

Então, com o tempo, os módulos perdem sincronismo: os que são executados no oscilador mais rápido aplicarão nos servos valores mais à frente da tabela e, os que são executados no mais lento, estarão mais atrás. Na robótica modular, é essencial que os módulos estejam em sincronismo para que o movimento seja executado corretamente.

Como todo o protocolo foi desenvolvido em malha aberta, ou seja, as informações são enviadas mas o *host* não possui garantia de que elas serão recebidas, é mais difícil controlar o sincronismo. A melhor solução encontrada para esse problema foi frequentemente "reiniciar" o sistema.

Um dos algoritmos mais simples para esse tipo de sincronização é o Algoritmo de Christian[23]. Ele pressupõe que uma das máquinas do sistema distribuído ajusta a hora das demais, que, periodicamente, a consultam para ajustar seus relógios. No caso do protocolo proposto para o Segundo e Terceiro Perfis, um dos módulos funciona como "servidor de tempo" e fornece a hora aos demais.

No Segundo Perfil, o algoritmo do *loop* do servidor de tempo é alterado, de forma que, quando ele procurar na tabela um $t = 0$ ou um $t = T/2$, envia para todos os demais um Pacote Padrão com MENSAGEM $= t$. A partir disso, os outros módulos atualizam sua posição na tabela para $t + t_{atraso}$, onde t_{atraso} é igual ao tempo que o módulo demora para receber o pacote [1].

A alteração no Terceiro Perfil é semelhante, sendo a única diferença o pacote enviado: um Pacote de Configuração, com t atualizado (Figura 5.5).

[1] O ideal: $t_{atraso} \ll t$

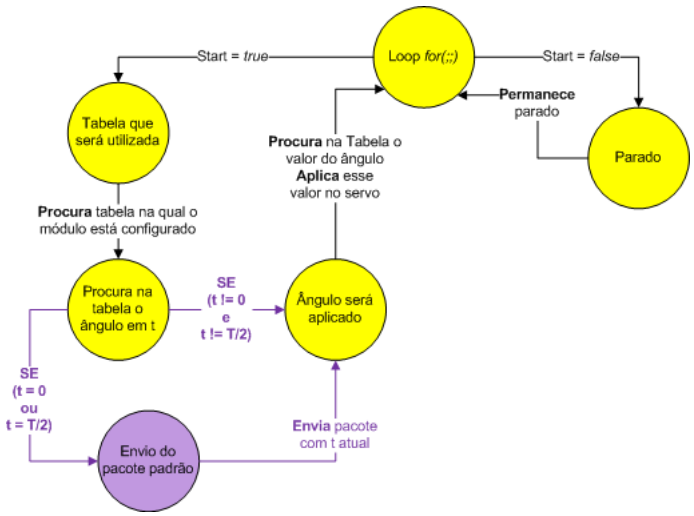


Figura 5.5: Alteração no algoritmo do Módulo Servidor de Tempo no Terceiro Perfil (roxo)

5.4 Teste de Confiabilidade de Entrega de Pacotes

A fim de mensurar a eficácia da transmissão de pacotes entre os módulos, foi realizado um teste de confiabilidade de entrega de pacotes. Esse teste foi preparado utilizando dois módulos, a interface RS-232 (Seção 4.1.1) e um computador que executa um programa em Java elaborado para o teste (Figura 5.6).

O primeiro módulo ($módulo_1$) foi programado para enviar, a cada $1000\,ms$, um Pacote Padrão com MENSAGEM = $\hat{A}ngulo$ para o segundo módulo ($módulo_2$) configurado no Primeiro Perfil. O endereço dentro desse Pacote Padrão é diferente do endereço do $módulo_2$, que, ao recebê-lo, de acordo com o algoritmo, transmite o pacote para frente.

O $módulo_2$ está conectado por meio da interface RS-232 ao computador, o qual executa um programa em Java que conta quantas vezes esse pacote foi recebido. Ao todo, 1000 pacotes com a mesma mensagem foram enviados pelo $módulo_1$.

O computador indicou que todos os pacotes enviados foram recebidos sem erros pelo computador. O teste foi, então, repetido para tempos mais arbitrários entre cada envio do pacote: 900, 800, 700, 600, 500, 400, 300, 200, 100, 90, 80, 70, 60, 50, 40, 30, 20, 10 e $0\,ms$. Os resultados podem ser encontrados no gráfico da Figura 5.7.

Do gráfico, pode-se perceber que a entrega dos pacotes é 100% eficiente para intervalos de tempo maiores que $200\,ms$. Para os módulos ErekoBots $\alpha\,v.2$, sabe-se que são necessários, no mínimo, $1500\,ms$ para que o servo mude entre as posições mais críticas, de $-90°$ a $90°$(ou vice-versa).

Como já existe essa restrição de intervalo mínimo de envio de transmissão, a confiabilidade da transmissão do Protocolo é eficiente.

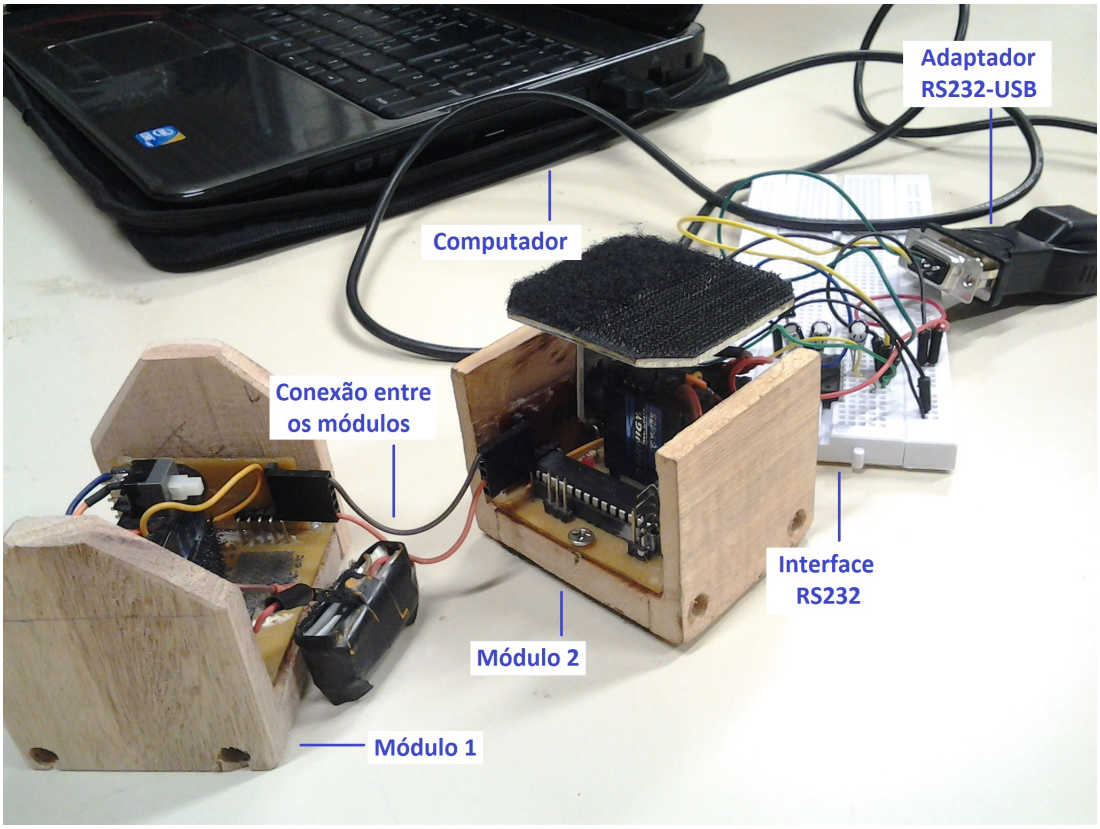


Figura 5.6: Preparo do Teste de Confiabilidade de Entrega de Pacotes

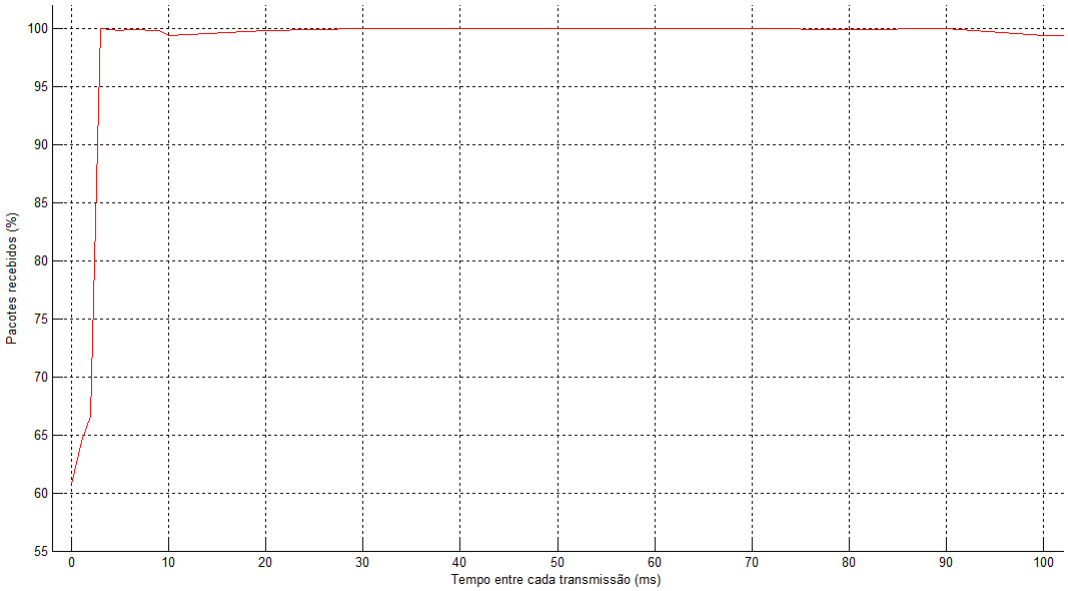


Figura 5.7: Gráfico dos Resultados do Teste de Confiabilidade de Entrega de Pacotes

5.5 Teste para a estimativa do atraso de transmissão

Alguns movimentos da robótica modular requerem que os módulos estejam sincronizados dentro de um intervalo de tempo específico e o protocolo desenvolvido assume que haverá uma falta de sincronização à medida em que mais módulos são conectados em linha reta, (ficando mais "distantes" do host). Existe um atraso de transmissão de um pacote que é diretamente proporcional ao número de módulos.

O teste da estimativa do atraso de transmissão foi realizado com o objetivo de observar o limite do número de módulos do sistema quando esses estão conectados em topologia linha. Esse teste foi preparado utilizando oito módulos, a interface RS-232 (Seção 4.1.1) e um computador que executa um programa semelhante ao Transmissor do Primeiro Perfil (Seção 4.1.7.1).

Os oito módulos estão roteados na topologia em linha e possuem o mesmo endereço ($endereço = 00H$). O teste é realizado oito vezes, cada vez com um dos oito módulos conectado à recepção da interface RS-232, ou seja, esse módulo reenvia qualquer pacote com $endereço \neq 00H$ não só para os módulos seguintes, mas também para o computador central. Em todos os testes, o programa envia pacotes com $endereço \neq 00H$.

No programa executado no computador, toda vez que um pacote é enviado e recebido pela interface RS-232, o programa lê o tempo de relógio (t_i e t_f, respectivamente). O resultado $atraso_t = t_f - t_i$ é o intervalo de tempo para que o pacote seja enviado e recebido de volta pelo programa, passando pelo(s) módulo(s).

Em cada um dos oito testes foram enviados 50 pacotes e foi calculada a média de todos os valores t_f encontrados. O propósito do teste é estimar o tempo que um pacote demora para ser transmitido entre os módulos (atraso de propagação).

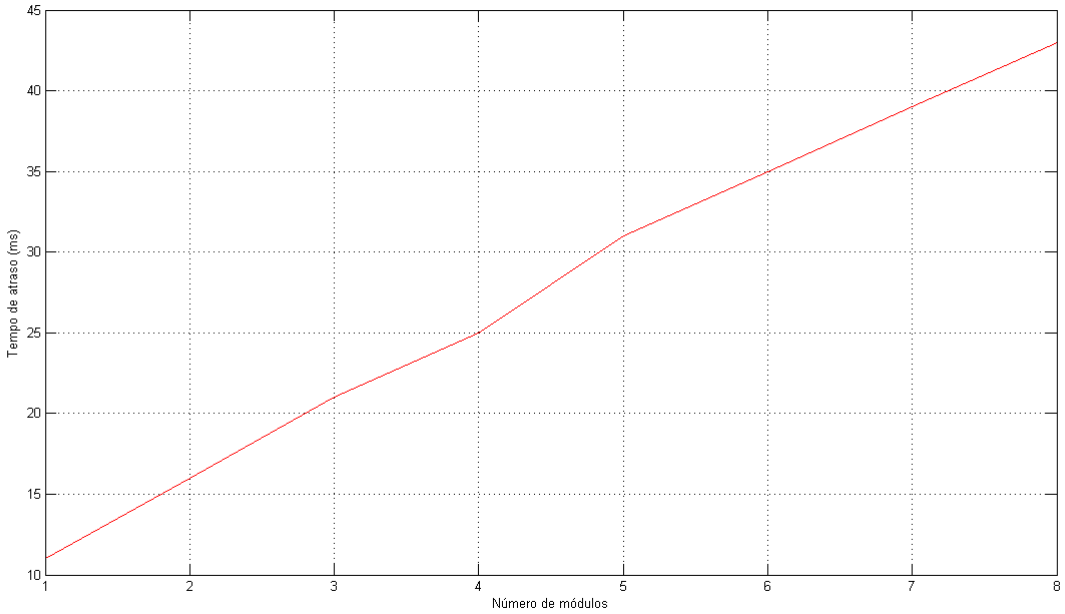
Os resultados do teste podem ser encontrados no gráfico da Figura 5.8(a).

Utilizando a ferramenta *cftool* do Matlab® é possível realizar uma regressão linear e encontrar uma função polinominal linear de $atraso_t$ em função do número de módulos (n), em milisegundos (Figura 5.8(b)):

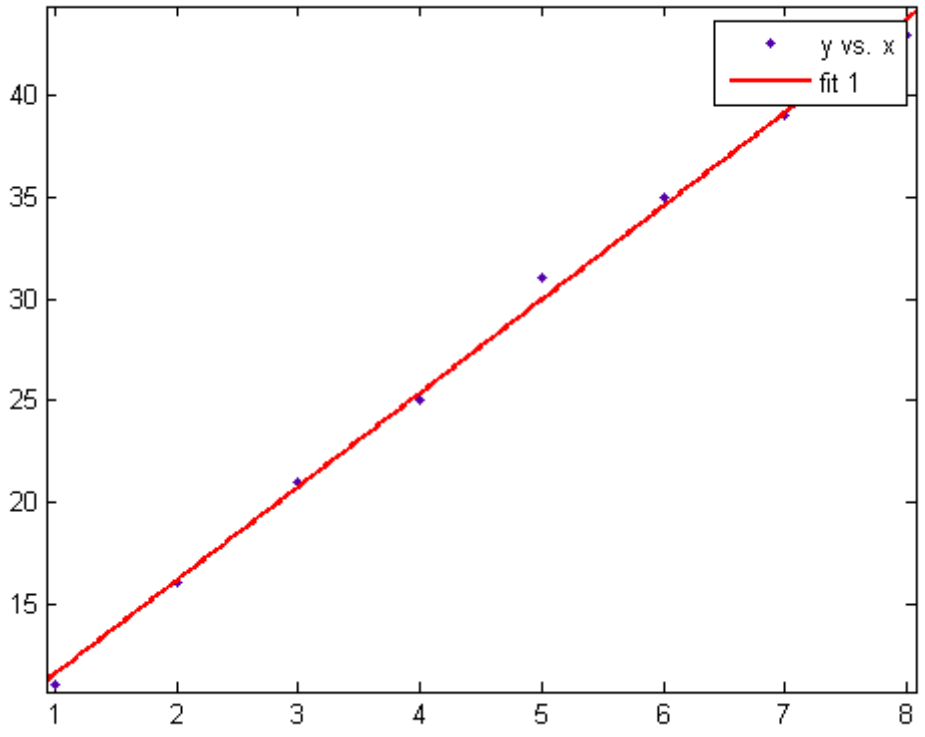
$$atraso_t = 4.607n + 6.893 \tag{5.1}$$

Para encontrar o atraso por módulo, divide-se a Equação 5.1 pelo número de módulos e calcula-se o limite quando $n \to \infty$:

$$atraso_{mod} = \frac{4.607n + 6.893}{n} \tag{5.2}$$



(a)



(b)

Figura 5.8: (a) Gráfico dos Resultados do Teste de Estimativa do Atraso de Propagação (b) Regressão Linear

$$\lim_{n\to\infty} atraso_{mod} = \lim_{n\to\infty} \frac{4.607n + 6.893}{n} = 4.607\,ms \tag{5.3}$$

Então, a estimativa é que o atraso por módulo é em torno de $4.607\,ms$.

O atraso téorico por módulo ($atrasoTeórico_{mod}$) corresponde a:

$$atrasoTeórico_{mod} = t_{trans} + t_{prop} + t_{proc} \tag{5.4}$$

Onde:

t_{trans}:	tempo necessário para transmitir um pacote
t_{prop}:	tempo necessário para que a informação se propague no meio físico
t_{proc}	tempo necessário para o módulo processar a informação

O tempo de transmissão é calculado dividindo o tamanho do pacote pela velocidade de transmissão $\frac{tamanho_{pac}}{v_{trans}}$. Um Pacote Padrão possui $40\,bits$ (Seção 4.1.6.2) e a velocidade foi configurada em $9600\,bps$ (Seção 4.1.1).

O tempo de propagação é calculado dividindo o tamanho do fio pela velocidade de transmissão no meio $\frac{tamanho_{fio}}{v_{meioFísico}}$. O fio tem $0.15\,m$ e a velocidade de transmissão nos fios de cobre é $150000000\,m/s$.

O tempo de processamento é calculado multiplicando o número de ciclos de clock da parte do programa correspondente ao processamento da informação pela frequência do microcontrolador $\frac{c}{f_{\mu controlador}}$. Utilizando o Atmel® AVR Studio® 5, foi possível contar o número de instruções dessa parte do programa e como cada instrução no Atmel® ATMega8® corresponde a um ciclo de clock, foi possível contabilizar $143\,ciclos$. E os microcontroladores dos módulos foram configurados a $8\,MHz$.

Então, temos que:

$$atrasoTeórico_{mod} = \frac{tamanho_{pacote}}{v_{trans}} + \frac{tamanho_{fio}}{v_{meioFísico}} + \frac{c}{f_{\mu controlador}} \tag{5.5}$$

$$atrasoTeórico_{mod} = 4 + 0.000001 + 0.017875 = 4.017876\,ms \tag{5.6}$$

Esse teste possui alguns problemas que justificam a diferença de 15% do atraso teórico para o atraso estimado:

- O computador executa um programa em Java™no sistema operacional Windows 7 Professional©, o que impede uma leitura de relógio em tempo real;
- A precisão da leitura é em milisegundos, o que dá uma margem de erro de $\pm 0.5\,ms$;
- O teste foi realizado com poucos módulos (8), o que impede uma regressão linear mais precisa.

Apesar desses problemas e dessa grande diferença, é possível prever a ordem de grandeza do atraso e com isso fazer uma estimativa do quanto um sistema perde em sincronia quando um movimento é realizado. E, por fim, com as especificações do movimento, definir se é possível sua realização.

Por exemplo, um movimento requer que 5 módulos tenham uma descincronização máxima de $50\,ms$. Se conectados em topologia linha, o atraso máximo estimado é de $23.035\,ms$, então, nessas especificações, é possível realizar o movimento.

Capítulo 6

Conclusões

Apresentação das conclusões do trabalho.

Nas últimas décadas, houve um grande interesse em implementar a reconfiguração em sistemas de robótica modular, visando a um avanço significativo no campo da robótica em geral. A promessa da versatilidade pode levar a uma mudança radical na automação e, por essa razão, existe um grande número de pesquisas nesse ramo atualmente. Apesar dos progressos realizados, é claro que desafios ainda existem, desde o acionamento de motores à conexão entre os módulos.

O presente trabalho representou a continuidade da frente de pesquisa de robótica modular existente no Grupo Ereko. Os trabalhos já realizados pelo grupo e a absorção de conhecimentos de artigos científicos, livros e outras formas de literatura especializada foram de fundamental importância para embasar as decisões tomadas em praticamente todas as etapas do projeto.

Para cada uma dessas etapas, procurou-se justificar a sua realização, apresentar a fundamentação teórica necessária para a sua compreensão e discorrer acerca dos resultados experimentais obtidos. A base de todo o desenvolvimento foi a revisão dos conceitos de robótica modular e de transmissão de dados. Foi por meio deles que pôde-se desenvolver o Protocolo para o ErekoBot $\alpha\, v.2$, com o levantamento de todas as limitações do sistema.

Conclui-se dos resultados apresentados no Capítulo 5 que este trabalho de graduação teve sucesso em atender os objetivos propostos no Capítulo 1. Ao final do período de duração, obteve-se um protocolo funcional, que pode ser útil a futuros projetos na área de robótica modular. Nos experimentos do Capítulo 5 foram feitas a verificação e validação dos pacotes enviados aos módulos de acordo com aquilo que era teoricamente esperado, o que atestou o bom funcionamento do protocolo. As diferentes interfaces de comunicação foram testadas, fornecendo aos seus usuários um sistema de fácil utilização para futuros testes de movimentação.

6.1 Perspectivas Futuras

Apesar da validação do Protocolo neste relatório, existem ainda alguns desafios para o sistema de comunicação do ErekoBot $\alpha\, v.2$.

O primeiro desses desafios é alterar a topologia totalmente em série para uma topologia em

barramento, que não só minimiza o tempo de transmissão (diminuindo o tempo de atraso), mas também simplifica os diagramas do sistema.

Outro grande desafio é a implementação de comunicação sem fios, uma vez que eles limitam a movimentação do robô móvel. O alcance máximo ao qual o robô pode chegar é o comprimento do fio, que pode atrapalhar o movimento. Como o ErekoBot é um robô móvel e o atual Protocolo não prevê uma comunicação sem fios, é necessário que, em trabalhos futuros, ela seja desenvolvida. E, para isso, as funções da Camada de Transporte e da Camada de Sessão também devem ser implementadas. Quando um sistema utiliza a tecnologia *wireless*, é importante autenticar o sistema para que usuários indesejáveis não possam também controlar o sistema.

Para a robótica modular, o desafio mais importante é o envio de novas formas de movimentação aos módulos em tempo real, em outras palavras, reconfigurá-los. Se isso fosse possível, os módulos seriam capazes de, por exemplo, iniciar uma movimentação em linha reta e, a partir de um comando do usuário, realizar uma curva para a direita. A implementação do Quarto Perfil no Protocolo possibilita essa reconfiguração em tempo real.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] YIM, M. *Locomotion With A Unit-Modular Reconfigurable Robot*. Tese (Doutorado) — Stanford University, 1994.

[2] YIM, M. et al. Modular self-reconfigurable robot systems – challenges and opportunities for the future. *IEEE Robotics and Autonomation Magazine*, March, p. 43–53, 2007.

[3] YIM, M.; DUFF, D.; ROUFAS, K. Polybot: A modular reconfigurable robot. In: *ICRA*. [S.l.: s.n.], 2000. p. 514–520.

[4] FUKUDA, T. et al. Self organizing robots based on cell structures - cebot. *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, p. 145–150, 1988.

[5] CHIRIKJIAN, G. Kinematics of a metamorphic robotic system. *IEEE International Conference on Robotics & Automation (ICRA)*, p. 449–55, 1994.

[6] MURATA, S.; KUROKAWA, H.; KOKAJI, S. Self-assembling machine. *IEEE International Conference on Robotics & Automation (ICRA)*, p. 441–448, 1994.

[7] KOTAY, K. et al. The self-reconfigurable robotic molecule. *IEEE International Conference on Robotics & Automation (ICRA)*, p. 424–431, 1998.

[8] CASTANO, A.; SHEN, W.-M.; WILL, P. M. Conro: Towards deployable robots with inter-robots metamorphic capabilities. *Auton. Robots*, v. 8, n. 3, p. 309–324, 2000.

[9] DUFF, D. G.; YIM, M.; ROUFAS, K. Evolution of polybot: A modular reconfigurable robot. *Harmonic Drive International Symposium*, 2001.

[10] KUROKAWA, H. et al. Self-reconfigurable modular robot m-tran: Distributed control and communication. In: *Proceedings of the 1st international conference on Robot communication and coordination*. Piscataway, NJ, USA: IEEE Press, 2007. (RoboComm '07), p. 21:1–21:7. ISBN 978-963-9799-08-0. Disponível em: <http://dl.acm.org/citation.cfm?id=1377868.1377895>.

[11] LUND, H. H.; BECK, R.; DALGAARD, L. Atron hardware modules for self-reconfigurable robotics. *10th International Symposium on Artificial Life and Robotics (AROB'10)*, 2005.

[12] SALEMI, B.; MOLL, M.; SHEN, W.-M. SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system. In: . Beijing, China: [s.n.], 2006.

[13] SOUZA, N. C. A.; KOIKE, C. M. C. e C. Robótica modular - desenvolvimento do projeto erekobot alfa. *XVI Congresso de Iniciação Ciêntífica da UnB*, 2010.

[14] GONZALEZ-GOMEZ, J. *Modular Robotics and Locomotion: Application to Limbless Robot.* Tese (Doutorado) — University of Madrid, 2008.

[15] SOUZA, N. C. A. Erekobot alfa project: Design and construction of a modular robot prototype. *21st Brazilian Congress of Mechanical Engineering*, v. 1, 2011.

[16] SOUZA, N. C. A.; KOIKE, C. M. C. e C.; VIANA, D. M. Projeto e prototipagem de módulo para robô modular reconfigurável. *XVII Congresso de Iniciação Ciêntífica da UnB*, 2011.

[17] YIM, M. et al. Modular self-reconfigurable robots. In: *Encyclopedia of Complexity and Systems Science*. [S.l.: s.n.], 2009. p. 5618–5631.

[18] BUTLER, Z. J. et al. Generic decentralized control for lattice-based self-reconfigurable robots. *I. J. Robotic Res.*, v. 23, n. 9, p. 919–937, 2004.

[19] RUBENSTEIN, M. et al. Docking among independent and autonomous CONRO self-reconfigurable robots. In: . New Orleans, USA: [s.n.], 2004. p. 2877–2882.

[20] SPROEWITZ, A. et al. Adaptive Locomotion Control in Modular Robotics. In: *Workshop on Self-Reconfigurable Robots/Systems and Applications IROS07*. [S.l.: s.n.], 2007. p. 81–84.

[21] GIOZZA, W. F. et al. *Redes Locais de Computadores - Tecnologia e Aplicações*. [S.l.: s.n.], 1986.

[22] ZUCCHI, W. L. *Transmissão de Dados em Redes de Computadores*. [S.l.: s.n.], 1986.

[23] CRISTIAN, F. Probabilistic clock synchronization. *Distributed Computing (Springer)*, v. 3, p. 146–158, 1989.

ANEXOS

I. DIAGRAMAS ESQUEMÁTICOS

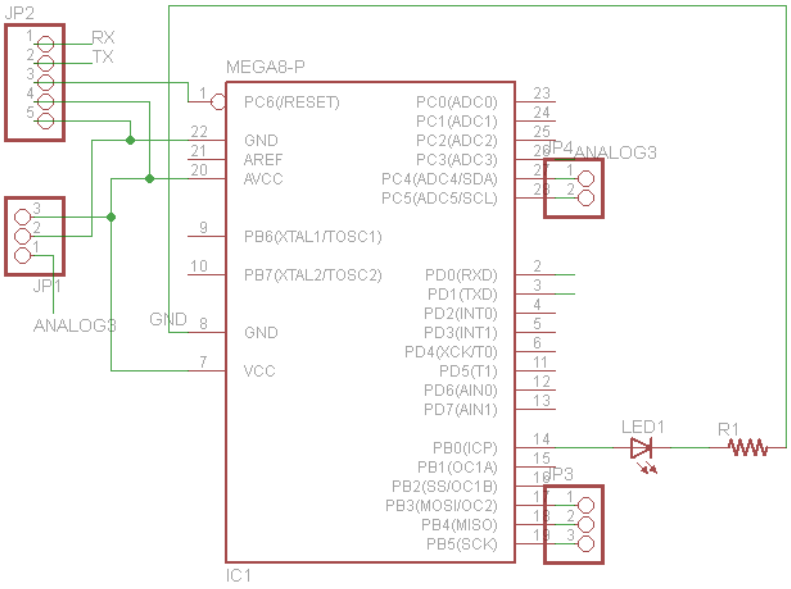


Figura I.1: Circuito da placa da base

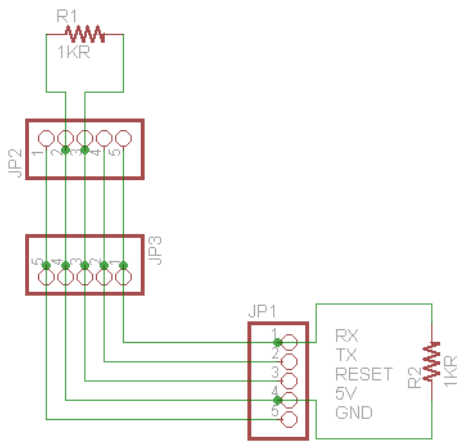


Figura I.2: Circuito da placa lateral esquerda

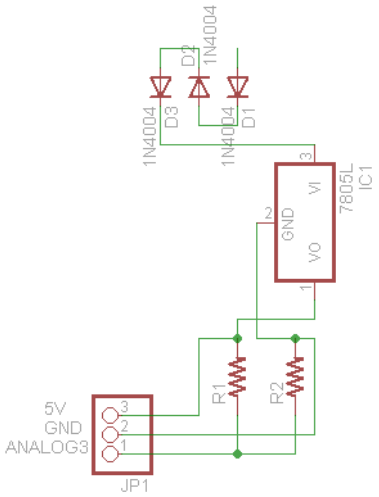


Figura I.3: Circuito da placa lateral direita

II. DESCRIÇÃO DO CONTEÚDO DO CD

O CD segue a seguinte estrutura:

- leiame.pdf: descrição do CD;
- \apresentacao: todos os arquivos da apresentação do relatório de graduação, já ajustados e configurados para uso no Lyx®;
  - \figs: todas as figuras da apresentação;
  - \videosLucas: todos os vídeos utilizados na apresentação.
- \pdf: relatório e resumo completos em pdf.
- \relatorio: todos os arquivos do relatório de graduação, já ajustados e configurados para uso no Lyx®.
  - \figs: todas as figuras do relatório.
    - \Diagramas: todos os diagramas do relatório, já ajustados e configurados para uso no Microsoft Office Visio Professional® 2007.

III. TABELA COM OS VALORES RESERVADOS

Valor (decimal)	Valor (hexadecimal)	Correspondência no Protocolo
0 a 11	00 a $0B$	Reservado
12	$0C$	*Broadcast*
13	$0D$	*Stop*
14	$0E$	*Config*
15	$0F$	*Start*
16	10	*Sync*
17 a 66	11 a 42	$endereço_{envio}$
67 a 74	43 a $4A$	Reservado
75 a 315	$4B$ a $13B$	$\hat{A}ngulo$
316 a 333	$13C$ a $14D$	Reservado
334 a 338	$14E$ a 152	Reservado para perfis
339 a 368	153 a 170	$modo\_operação_{envio}$
369 a 438	171 a $1B6$	Reservado
439 a 488	$1B7$ a $1E8$	T_{envio}
489 a 65535	$1EC$ a $FFFF$	Reservado

IV. SOFTWARES DE APOIO AO DESENVOLVIMENTO

Para facilitar a realização do Projeto, foram utilizados vários programas durante o desenvolvimento deste Trabalho de Graduação:

- Escrita do relatório:
  - LYX®: como ambiente de desenvolvimento para criar o texto com TEX, utilizou-se o software gratuito Lyx. Com ele é possível criar-se documentos sem a preocupação " com detalhes muito pequenos que existe no TEX "puro" com detalhes muito pequenos. O gerenciamento de tabelas, matrizes, figuras, rótulos (dentre outros) é muito facilitado por meio desse programa[1].
  - JABREF®: programa gratuito que facilita muito a organização das referências bibliográfica[2].
  - MICROSOFT OFFICE VISIO PROFESSIONAL® 2007: a maioria dos diagramas de blocos foram desenhados utilizando o Visio.
  - PAINT®: recurso do Windows 7 PROFESSIONAL© que permitiu desenhos rápidos de diagramas.
  - MATLAB®: software interativo de alta performance voltado para o cálculo numérico utilizado para a construção dos gráficos do relatório.
- Ambiente de programação:
  - UBUNTU 11.04[3]:
    - GEDIT: editor oficial de texto para o GNOME, foi utilizado, pois é um programa gratuito simples que possui compatibilidade com a Linguagem C.
    - MAKE: programa gratuito muito utilizado em programação que lê o *script* dentro de um arquivo nomeado Makefile. Esse *script* facilita a compilação, montagem, ligação durante o desenvolvimento de um projeto.
    - AVR-GCC: compilador de código aberto do AVR, utilizado para a compilação do projeto para o Atmel ATmega 8®[4].
    - AVRDUDE: programa de código aberto utilizado para gravar nos microcontroladores os arquivos executáveis gerados pelo avr-gcc[5].
  - WINDOWS 7 PROFESSIONAL©:

[1]Disponível em: http://www.lyx.org/Download
[2]Disponível em: http://jabref.sourceforge.net/download.php
[3]Disponível em: http://www.ubuntu-br.org/download
[4]Linha de comando para a instalação no Ubuntu: *sudo apt-get install gcc-avr*
[5]Linha de comando para a instalação no Ubuntu: *sudo apt-get install avrdude*

    * NETBEANS IDE 7.1®: ambiente de desenvolvimento integrado (IDE) gratuito e de código aberto para desenvolver os programas em Java.
    * ATMEL® AVR STUDIO® 5: IDE de desenvolvimento e depuração de aplicações embarcadas de ATMEL® AVR, para montar os programas executáveis e contar as instruções.

- Algumas bibliotecas específicas:
  - avr-libc: biblioteca de código aberto para a geração dos arquivos objetos e executáveis do Atmel ATmega 8®[6].
  - rxtx: API distribuída pela Sun© para manipulação da porta serial. Essa biblioteca foi ligada ao projeto no Netbeans para possibilitar a comunicação via serial do computador com os módulos[7].

[6]Linha de comando para a instalação no Ubuntu: *sudo apt-get install avr-libc*
[7]Disponível em: http://rxtx.qbang.org/wiki/index.php/Download