FÁBIO DE ALMEIDA GUIMARÃES

DESENVOLVIMENTO DE ROBÔ MÓVEL UTILIZADO PARA A EXPLORAÇÃO DE AMBIENTES HOSTIS

SÃO CAETANO DO SUL

2007

FÁBIO DE ALMEIDA GUIMARÃES

DESENVOLVIMENTO DE ROBÔ MÓVEL UTILIZADO PARA A EXPLORAÇÃO DE AMBIENTES HOSTIS

Dissertação apresentada à Escola de Engenharia Mauá do Centro Universitário do Instituto Mauá de Tecnologia para a obtenção do Título de Mestre em Engenharia de Processos Químicos e Bioquímicos.

Linha de Pesquisa: Análise e Controle de Processos Químicos.

Orientador: Prof. Dr. Wânderson de Oliveira Assis

SÃO CAETANO DO SUL

2007

Guimarães, Fábio de Almeida

Desenvolvimento de Robô Móvel Utilizado para a Exploração de Ambientes Hostis / Fábio de Almeida Guimarães – São Caetano do Sul, SP: CEUN-EEM, 2007. 227 p.

Dissertação de Mestrado em Engenharia de Processos Químicos e Bioquímicos – Escola de Engenharia Mauá do Centro Universitário do Instituto Mauá de Tecnologia, São Caetano do Sul, SP, 2007.

Palavras Chaves: Robôs Autônomos. Odometria. Comunicação por meio de Radiofreqüência.

AGRADECIMENTOS

Chego ao fim de mais uma etapa na minha vida, a conclusão do meu Mestrado, e aqui quero deixar registrado os meus agradecimentos a todos que de uma certa forma fizeram parte do desenvolvimento desse projeto.

Aos meus pais Anamaria e Nivaldir, quero agradecer por toda a dedicação, por todo o carinho, por todos os ensinamentos e pelo empenho para que os estudos sempre fizessem parte da minha vida.

Aos meus irmãos Alexandre e Marcelo que se tornaram uma grande fonte inspiradora, não só pela inteligência e sabedoria, mas principalmente pela força de vontade tanto nos estudos, quanto no trabalho.

As minhas cunhadas, Ana Paula e Amanda, que mesmo de longe fizeram parte desse projeto.

A minha amada esposa Daniela, pela paciência, compreensão, interesse e incentivo desde o início do desenvolvimento deste trabalho.

Ao meu professor Wânderson, que sempre me ajudou, apoiou e esteve presente nos momentos de maior dificuldade, e que além de orientador, tornou-se um grande amigo.

Aos meus grandes amigos Peterson e sua esposa Mari, que sempre estiveram ao meu lado, apoiando, incentivando e ajudando sempre que possível.

Aos meus amigos da GM, Leonardo e Fernando, pela compreensão e apoio nos momentos mais difíceis.

E por último, a alguém muito especial, minha filha Mariana, que se tornou a maior incentivadora para a conclusão deste trabalho.

A todos vocês que são a minha grande família, Muito Obrigado.

RESUMO

Este trabalho aborda diversos conceitos relacionados à robótica móvel, onde se procura apresentar as principais etapas envolvidas no projeto de um robô.

A abordagem consiste no desenvolvimento de um robô móvel autônomo, capaz de explorar ambientes hostis e com a capacidade de detectar obstáculos, apresentando resultados confiáveis, com qualidade e baixo custo.

A solução proposta é simples e flexível, possibilitando a adição de novos componentes ao sistema, apresenta o desenvolvimento de um algorítimo de cálculo para a aquisição da velocidade dos motores através do uso de sensores de velocidade, além de um protocolo simples e eficiente, incluindo sistema de codificação dos dados para minimização de erros na comunicação.

Diversos testes foram executados ao longo do desenvolvimento do trabalho e a maioria deles são apresentados detalhadamente, possibilitando a sua inteira e precisa reprodução.

Palavras-chave: Robótica, Mapeamento, Odometria, Detecção de Obstáculos, Sensores, Protocolos de Comunicação, Radiofreqüência e Motores de Corrente Contínua.

ABSTRACT

This work approaches concepts related to mobile robotic, presenting the main involved stages in a robot project.

The approaches consists in an autonomous mobile robotic development, capable to explore hostile enviroments and with the capability to detect obstacles, presenting reliable results, with quality and low cost.

The proposal solution is simple and flexible, providing capability to add new components in the system, presenting an algorithm development to acquire the engines speed through speed sensors, as well as an simple and efficient protocol, including a data codification system to minimize the communication fails.

Several tests had been executed in the development of this work and the majority of them, are presented in details, making possible its entire and accurate reproduction.

Keywords: Robotic, Mapping, Odometry, Obstacles Detection, Sensors, Communication Protocols, Radio Frequency and DC Motors.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SIGLA	SIGNIFICADO ORIGINAL	TRADUÇÃO
AF	Audio Frequency	Freqüência de Áudio
AIBO	Artificially Intelligent roBOt	Robô com Inteligência Artificial
ALU	Arithmetic Logic Unit	Unidade Lógica Aritmética
AM	Amplitude Modulation	Modulação por Amplitude
API	Application Program Interface	Interface de Programação de Aplicativos
AWG	American Wire Gauge	Calibrador de Fio Americano
BIN	Binary	Binário
BPS	Bits Per Second	Bits Por Segundo
CC	Direct Current (DC)	Corrente Contínua
CI	Integrated Circuit	Circuito Integrado
DEC	Decimal	Decimal
EEPROM	Electrically Erasable Programmable Read-Only Memory	Memória Somente de Leitura Programável Apagável Eletricamente
EHF	Ultra High Frequency	Freqüência Extremamente Alta
ELF	Extremely Low Frequency	Freqüência Extremamente Baixa
EOD	Explosive Ordnance Disposal	Arma de Eliminação de Explosivo
E/S	Entrada/ Saída	---
GND	Ground	Terra
HEX	Hexa	Hexadecimal
HF	High Frequency	Freqüência Alta
IHM	Interface Human-Machine	Interface Homem-Máquina
IR	Infrared	Infravermelho
I/O	Input/Output	Entrada/Saída

SIGLA	SIGNIFICADO ORIGINAL	TRADUÇÃO
LED	Light Emitting Diode	Diodo Emissor de Luz
LF	Low Frequency	Freqüência Baixa
MF	Medium Frequency	Freqüência Média
MIPS	Million Instructions Per Second	Milhões de Instruções Por Segundo
MTTF	Mean Time to Failure	Tempo Médio para Falha
NASA	National Aeronautics and Space Administration	Administração Nacional de Aeronáutica e Espaço
PC	Personal Computer	Computador Pessoal
PCI	Placa de Circuito Impresso	---
PI	Proportional Integral	Proporcional Integral
PWM	Pulse Width Modulation	Modulação por Largura de Pulso
QTD	Quantidade	---
RF	Radio Frequency	Radiofreqüência
RISC	Reduced Instruction Set Computer	Computador com Conjunto de Instruções Reduzido
RPM	Rotations Per Minute	Rotações Por Minuto
SHF	Super High Frequency	Freqüência Super Alta
SONAR	Sound Navigation And Ranging	Navegação e Alcance do Som
SPI	Serial Peripheral Interface	Interface Serial Periférica
SRAM	Static Random Access Memory	Memoria estática de acesso aleatório
UART	Universal Asynchronous Receiver and Transmitter	Receptor e Transmissor Assíncrono Universal
UBRR	UART Baud Rate Register	Registrador de Taxa de Transmissão da UART
UDR	UART I/O Data Register	Registrador de E/S de Dados da UART
USR	UART Status Register	Registrador de Status da UART
UHF	Ultra High Frequency	Freqüência Ultra Alta
UV	Ultraviolet	Ultravioleta

SIGLA	SIGNIFICADO ORIGINAL	TRADUÇÃO
VHF	Very High Frequency	Freqüência Muito Alta
VLF	Very Low Frequency	Freqüência Muito Baixa
XTAL	Cristal (Oscillator)	Cristal (Oscilador)

SUMÁRIO

1 INTRODUÇÃO

De uma forma geral, é difícil associar a palavra simplicidade quando as pessoas são estimuladas a tentar compreender a teoria associada à construção e ao próprio funcionamento dos robôs. Afinal, a robótica envolve os mais variados conceitos nas áreas de mecânica, eletrônica e informática. A complexidade do sistema está intrinsecamente ligada ao objetivo para o qual o robô está sendo projetado.

Hoje em dia, os conceitos de robótica são utilizados em uma série de aplicações na indústria, educação e também em pesquisas aeroespaciais. São exemplos de aplicações:

1. Situações que possam acarretar algum perigo à saúde humana: exploração de ambientes hostis onde há presença de gases tóxicos [1], inspeção de tubulações visando detecção de vazamentos e medições de concentração de gases [2] [3] [4], temperatura elevada ou exposição à radiação [5] [6], detecção de bombas em zonas de guerra e limpeza de reatores nucleares [7];
2. Organizações que realizam competições de Futebol de Robôs, com o objetivo de incentivar o desenvolvimento da robótica, da visão computacional e da inteligência artificial [8] [9] [10] [11];
3. Processos de manufatura com o objetivo de evitar desperdícios, aumentando a eficiência do sistema.

Com o intuito de auxiliar de uma forma prática no desenvolvimento e projeto de robôs móveis, apresentando um sistema flexível, que possibilite a adição de novos componentes ao sistema, expandindo assim a funcionalidade e aplicabilidade do robô, como por exemplo, sensores de proximidade (para a detecção de obstáculos), sensores de temperatura, sensores ultra-sônicos, sensores para a distinção de odores, entre outros [12] [13]. O trabalho foi organizado da seguinte maneira:

- Capítulo 1: além de um breve resumo do trabalho, é dada uma visão geral das principais etapas envolvidas no desenvolvimento e projeto de um robô, assim como alguns exemplos de aplicações bem sucedidas nessa área de estudo. Os objetivos do sistema projetado, também são apresentados.
- Capítulo 2: é mostrado uma breve revisão bibliográfica, procurando demonstrar alguns dos trabalhos mais importantes e que de alguma forma contribuíram para o desenvolvimento desse projeto

- Capítulo 3: além de uma breve descrição do funcionamento do sistema, é mostrado também o diagrama geral do sistema, tanto o diagrama em blocos como o circuito elétrico completo proposto.
- Capítulo 4: os componentes utilizados na construção do sistema são descritos e justificados em sua maioria, assim como é dada uma breve introdução sobre sensores e uma abordagem mais aprofundada nos tipos de sensores utilizados no projeto.
- Capítulo 5: neste capítulo é abordado o conceito utilizado para realizar o mapeamento do caminho percorrido pelo robô (odometria).
- Capítulo 6: é explicado o funcionamento do software de controle e os principais conceitos envolvidos na lógica de programação utilizada, tanto para o desenvolvimento da interface homem-máquina, quanto para o controle do robô.
- Capítulo 7: é apresentado todo o sistema de comunicação utilizado, assim como os conceitos adotados no protocolo de comunicação.
- Capítulo 8: o conceito da estrutura adotada, além dos tipos de materiais utilizados na construção do robô. O custo total do sistema também é abordado neste capítulo.
- Capítulo 9: a conclusão geral do projeto proposto é apresentada.
- Capítulo 10: é disponibilizada a listagem completa dos programas desenvolvidos ao longo do trabalho.
- Capítulo 11: as referências bibliográficas, que muito colaboraram na conceituação e em todo o desenvolvimento do projeto.

1.1 VISÃO GERAL

A Figura 1.1 tenta exemplificar em etapas e de uma forma bem ampla, o que está relacionado ao projeto de qualquer robô.

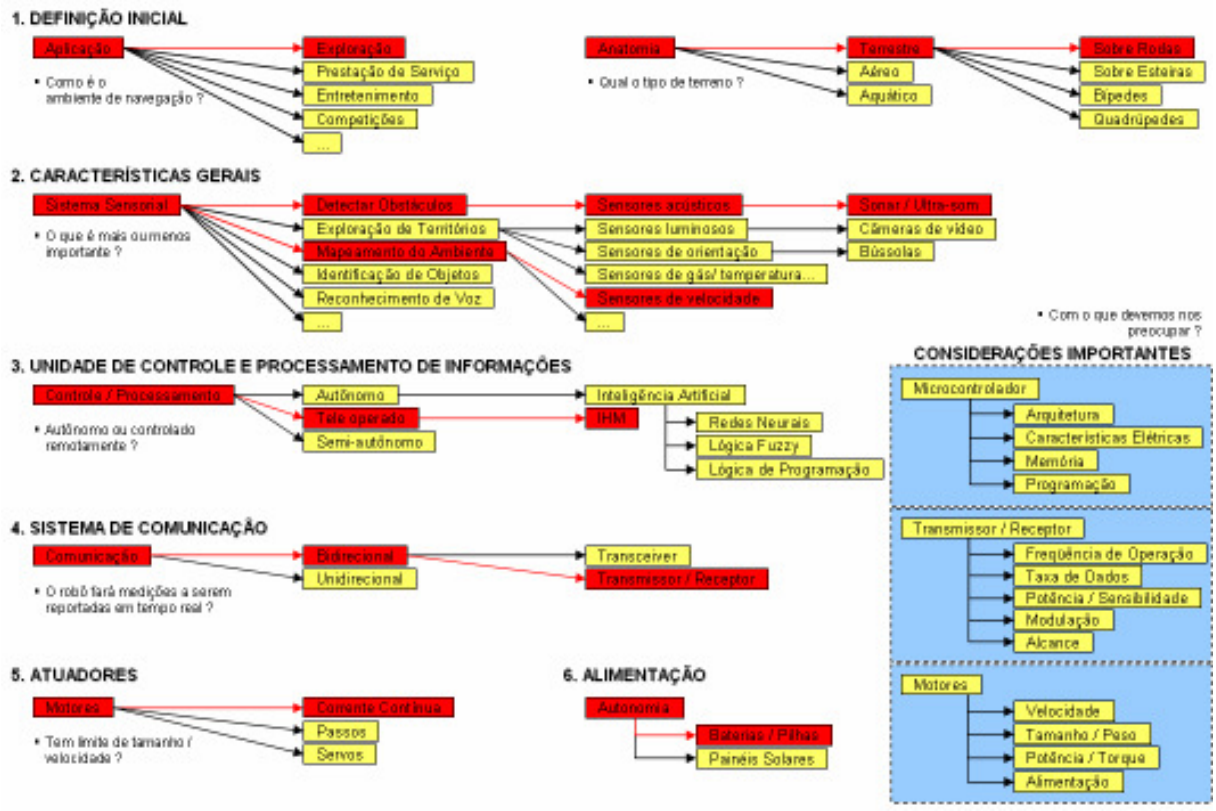


FIGURA 1.1 – ETAPAS ENVOLVIDAS NA CONSTRUÇÃO DE UM ROBÔ

A primeira etapa, a definição inicial, é extremamente importante. Antes de qualquer coisa é preciso ter em mente qual será a aplicação do robô e consequentemente, qual a sua estrutura. Nesta etapa serão respondidas perguntas como: qual o ambiente de navegação? qual o tipo do terreno?

A Figura 1.2 apresenta alguns exemplos reais de aplicações na área científica. Nessa figura, é possível observar como é vasta a aplicação da robótica e em alguns casos extremamente importante, visto que se torna em algumas aplicações como uma segurança essencial à saúde humana. Abaixo é apresentada uma breve explicação sobre cada uma das aplicações demonstradas:

- **Desarmamento de Bombas *(iRobot PackBot EOD)*[(1)]:** Sua missão é salvar vidas. É um robô resistente e leve, projetado para ajudar na eliminação de explosivos, procura de sobreviventes, resgate de reféns, entre outras tarefas.

- **Transporte *(R-Gator – Autonomous Unmanned Ground Vehicle)*[(2)]:** O R-Gator é um robô que tem como objetivo servir a diversas atividades militares,

[1] FONTE: http://www.irobot.com

como por exemplo, ao transporte de equipamentos militares, exploração e até mesmo resgate.

- **Prestação de Serviços *(Trilobite 2.0 Vacuum Cleaner)*[3]:** Outra aplicação bastante interessante que vem aumentando hoje em dia é a utilização da robótica no setor de prestação de serviços, esse robô tem a funcionalidade de um aspirador de pó autônomo.

- **Exploração *(Mars Exploration Rover Mission)*[4]:** Esse é o melhor exemplo de uma aplicação de um robô explorador de ambientes desconhecidos e hostis. É um projeto da NASA que tem como objetivo principal pesquisar a possível existência de água no planeta Marte. Para isso, dois robôs foram enviados em 2003 para lá e ainda hoje estão explorando o Planeta Vermelho.

- **Entretenimento *(AIBO® Entertainment Robot)*[5]:** AIBO o cãozinho desenvolvido pela Sony com o objetivo de divertir as pessoas.

- **Competições *(Robocup* – Futebol de Robôs)[6]:** O objetivo principal dessa aplicação é o de aumentar o interesse de estudantes, proporcionando-lhes a possibilidade do aprendizado e da diversão ao mesmo tempo.

---

[2] FONTE: http://www.irobot.com

[3] FONTE: http://trilobite.electrolux.co.uk

[4] FONTE: http://marsrovers.jpl.nasa.gov

[5] FONTE: http://www.sony.net/Products/aibo/index.html

[6] FONTE: http://www.robocup.org

FIGURA 1.2 – EXEMPLOS DE APLICAÇÕES

Definida a aplicação, o passo seguinte é definir o que o robô será capaz de fazer, ou seja, como será o seu sistema sensorial. Nessa etapa deve-se avaliar, para essa aplicação, se é essencial que:

- o robô tenha sensores que possibilitem o mapeamento do ambiente;
- que detecte objetos com o objetivo de evitar colisões;
- que tenha um sistema de orientação próprio.

Também nessa etapa é importante definir se o robô deve apresentar sistema de visão embarcado [14]. Portanto, serão definidas as características básicas do robô.

Outra etapa importante é definir como será o controle e o processamento dos dados do robô: ele será autônomo, semi-autônomo, ou tele-operado? Basicamente, o nível de complexidade da lógica de programação do robô será definido nessa etapa. Definido o controle, a definição do sistema de comunicação é conseqüência.

A próxima etapa é a definição dos atuadores, toda a movimentação do robô vai depender do tipo de atuador escolhido [15].

Por último, o sistema de alimentação, que precisa ser muito bem dimensionado, para que dê uma boa autonomia de funcionamento ao robô.

Algumas considerações que são importantes nessas etapas de definição estão relacionadas com as características físicas de determinados componentes chaves do sistema, como microcontrolador (ou microprocessador), os módulos de comunicação e os motores. Apenas como exemplo, o microcontrolador é o "cérebro" do robô, sendo importante que a escolha desse componente seja bem estudada, para que no futuro não represente uma limitação na implantação física de técnicas de controle ou características específicas desejáveis para o robô na aplicação.

Na Figura 1.1, apenas para dar uma noção incial, foi destacado em vermelho as principais características que o robô desenvolvido contém, onde todos detalhes estarão sendo apresentados no decorrer desse trabalho.

1.2 OBJETIVO

O objetivo principal desse trabalho é o desenvolvimento de um robô móvel capaz de explorar ambientes hostis (ambientes que sejam prejudiciais à saúde do ser humano) sem a intervenção humana, com a capacidade de detectar obstáculos, flexível tanto em sua estrutura mecânica, quanto eletrônica, e que apresentasse resultados confiáveis, com qualidade e baixo custo.

2 REVISÃO BIBLIOGRÁFICA

O objetivo deste capítulo é o de expor de uma maneira macro e clara, alguns dos trabalhos considerados importantes e que de alguma forma contribuíram para o desenvolvimento desse projeto, de onde se extraiu algumas idéias e soluções para os diversos módulos desse trabalho.

Através da Tabela 2.1, procurou-se sumarizar o conteúdo dos principais artigos, demonstrando a aplicação para o qual ele é proposto e os principais componentes envolvidos na aplicação prática das propostas desenvolvidas.

Apenas como referência, foi adicionada nessa mesma tabela, as características equivalentes desse projeto proposto.

TABELA 2.1 – RESUMO DOS PRINCIPAIS ARTIGOS ANALISADOS

#	Título do Trabalho	Resumo do Trabalho	Aplicação Principal			Locomoção		Localização					Unidade de Controle		Comunicação		Sensores					
			Exploração	Futebol de Robôs	Guerra de Robôs	Motor de CC	Motor de Passo	Bussola Digital	Odometria	Câmera	GPS	Outros	Microtontrolador	PC	RF	Por fio	Ultrasom	Encoders/ Velocidade	Infravermelho	Contato	Temperatura	Outros
	Trabalho Proposto (referência)		✓			✓			✓				✓		✓		✓	✓				
1	**Construindo robôs autônomos para partidas de futebol: o Time Guaraná. [8]**	São apresentados os algorítmos usados na implementação do software de controle, iniciando pelo sistema de visão, partindo para o módulo responsável pela decisão da estratégia de jogo e por último, a plataforma de hardware utilizado.		✓		✓			✓	✓			✓		✓				✓			
2	**Construção de robôs jogadores de futebol. [16]**	São apresentados os conceitos utilizados na construção de um robô utilizado em competições de futebol de robôs, além de um detalhamento dos sistemas de comunicação e movimentação do robô.		✓		✓				✓			✓		✓							
3	**Micro-Robot Soccer Team - Mechanical and Hardware Implementation. [11]**	Semelhante ao artigo anterior, os conceitos aplicados na construção de um robô utilizado em competições de futebol de robôs, como por exemplo, os conceitos da arquitetura, do sistema de movimentação e comunicação são apresentados.		✓		✓				✓			✓		✓							
4	**Controles de velocidade e trajetória para o robô Sonic Shark e sensoriamento por ultra-som. [17]**	Este trabalho consiste no desenvolvimento de um sistema de controles de velocidade, trajetória e sensoriamento por ultra-som para uma plataforma móvel rádio comandada.			✓	✓			✓				✓		✓		✓	✓				
5	**Navegação de robôs móveis em ambientes desconhecidos utilizando sonares de ultra-som. [18]**	São apresentados métodos para a navegação autônoma de robôs móveis em ambientes sobre os quais possui-se pouca ou nenhuma informação. O sistema opera em tempo real, utilizando os sonares do robô para fazer a detecção dos obstáculos, e interage com o sistema de controle do robô para rastreamento das trajetórias desejadas.	✓			✓	✓	✓	✓	✓				✓	✓		✓	✓				
6	**Implementation of an Intelligent Roving Robot Using Multiple Sensors. [12]**	Conceitos de ativação por meio de comando de voz, além de um interfaceamento com diversos sensores ao mesmo tempo, apresentam através deste artigo, um robô flexível e simples ao mesmo tempo.	✓			✓							✓				✓			✓	✓	✓
7	**Comportamentos reativos para seguir pistas em um robô móvel guiado por visão. [19]**	Este artigo descreve o projeto e a implementação de um sistema de controle para um robô móvel autônomo dotado de percepção, capaz de seguir um caminho desenhado na superfície sobre a qual o robô se move e evita colisões.	✓			✓			✓				✓			✓		✓	✓			
8	**Sistema para navegação e guiagem de robôs móveis autônomos. [20]**	Neste artigo, é apresentado um sistema para navegação e guiagem de robôs móveis autônomos, utilizando-se encoders e bússola digital. A informação de posição e atitude do veículo é então utilizada por um sistema de guiagem, baseado em controlador tipo fuzzy.	✓			✓		✓	✓				✓			✓		✓				

Os trabalhos 1, 2 e 3, conforme sumarizados na Tabela 2.1, estão relacionados com aplicação e desenvolvimento de robôs para serem utilizados em competições de futebol de

robôs. O artigo 1, descreve de uma maneira bem completa os conceitos utilizados em uma aplicação desse tipo, como por exemplo, o sistema de visão utilizado para a localização dos robôs e da bola, são realizados através de uma câmera externa ao robô. Nesse tipo de aplicação além da importância do hardware, em termos de velocidade de resposta e controle dos robôs, o software utilizado para a identificação e rastreamento em tempo real dos robôs e da bola, assim como a estratégia do jogo responsável pela determinação do novo caminho a ser perseguido pelo robô, são igualmente importantes para que o sistema apresente um bom resultado.

Os artigos 2 e 3, apresentam conceitos mais detalhados em termos de hardware e estrutura física dos robôs. Existe uma grande semelhança entre o sistema apresentado principalmente no artigo 2, com o deste trabalho, visto que os conceitos adotados em termos de controle (microcontrolador), comunicação RF e circuito de potência (ponte H), são basicamente os mesmos.

O trabalho 4, é uma monografia que consiste no desenvolvimento de um sistema de controle de velocidade, trajetória e sensoriamento por ultra-som para uma plataforma móvel rádio comandada. Este robô foi desenvolvido para participar de competições de guerra de robôs, onde é necessário possuir a capacidade de operar em situações de alto risco de colisões, onde ocasionalmente poderão ser exigidas manobras ágeis em modalidades de ataque e defesa. Os conceitos de localização, definição de trajetória e detecção de obstáculos foram bem detalhados neste trabalho escrito.

O trabalho apresentado através do artigo 5, procurou descrever um método relativamente simples de planejamento de trajetórias de robôs móveis em ambientes desconhecidos. Para isso foi utilizado sensores de ultrasom para a detecção de obstáculos, e para a localização do robô, além de encoders, sendo que a posição e orientação inicial eram conhecidas. Apesar do principal problema encontrado que foi referente ao uso dos sonares para detecção dos obstáculos, dada a sua imprecisão inerente, os resultados apresentados foram bastante satisfatórios.

O artigo 6, exemplifica uma aplicação simples e flexível, onde uma grande quantidade de sensores é utilizada em um mesmo robô, entre eles, sensores de ultrasom e de contato, para a detecção de obstáculos, sensores de corrente para a proteção dos motores e sensores de temperatura para a detecção de super aquecimento.

O artigo 7, descreve o projeto e a implementação de um sistema de controle para um robô móvel autônomo dotado de percepção, onde baseado em informações recebidas de sensores infravermelhos e informações previamente definidas em hardware no robô, ele é

capaz de seguir um caminho desenhado na superfície e evitar colisões. A arquitetura usada é composta por quatro módulos, sendo: módulo de aquisição de imagens, módulo de detecção de cores, módulo de identificação da posição, direção e trajetória do robô, e por último o módulo de controle do robô.

O artigo 8, apresenta um sistema para navegação e guiagem de robôs móveis autônomos. Para a navegação, utilizou-se integração via filtro de Kalman, conceitos de odometria, pelo uso de encoders e bússola digital.

3 SISTEMA

3.1 DIAGRAMA GERAL DO SISTEMA

O sistema projetado pode ser dividido em duas partes. A Figura 3.1 ilustra o diagrama em blocos da primeira parte.

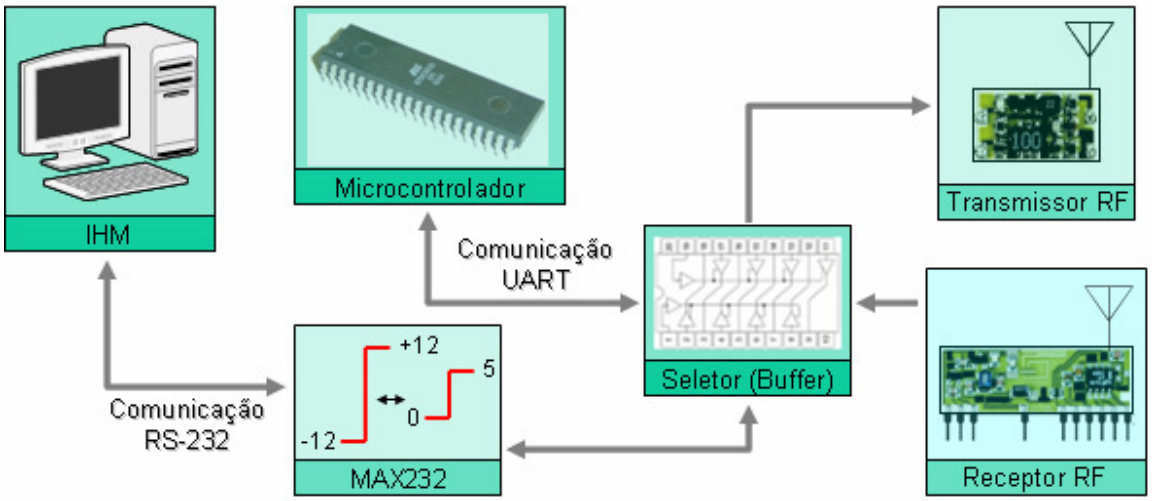


FIGURA 3.1 – DIAGRAMA EM BLOCOS DO SISTEMA DE INTERFACE COM O USUÁRIO

Essa primeira parte do sistema está relacionada com o módulo de interface com o usuário. Contempla uma Interface Homem-Máquina (IHM), ou seja, um computador pessoal, microcontrolador e os circuitos de transmissão e recepção.

O diagrama elétrico da primeira parte do sistema é apresentado na Figura 3.2. Como pode ser observado, existe uma separação em seis blocos.

No primeiro bloco temos o circuito de alimentação constituído de regulador de tensão que converte a tensão de entrada de 9V em nível TTL, ou seja, 5V.

O segundo bloco, o microcontrolador, tem apenas a utilização de alguns componentes passivos, necessários para o funcionamento do microcontrolador, que são utilizados para a alimentação e geração de clock.

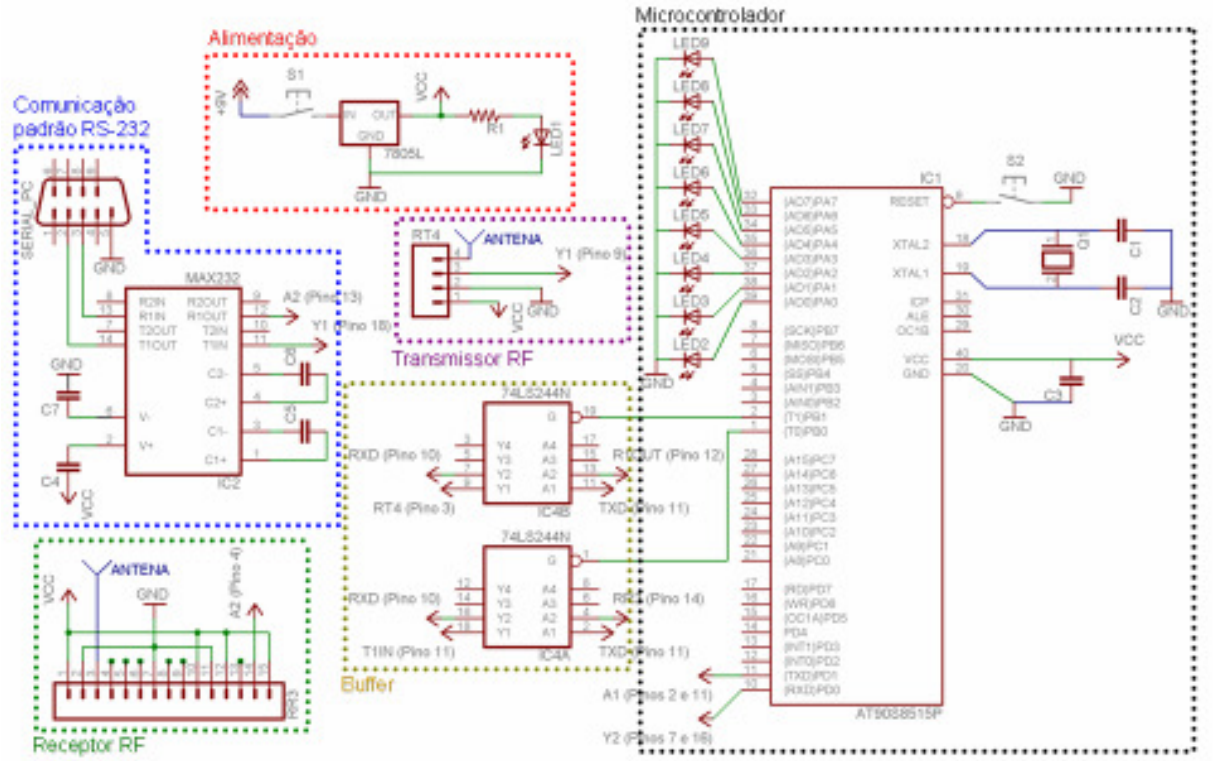


FIGURA 3.2 – DIAGRAMA ELÉTRICO DO SISTEMA DE INTERFACE COM O USUÁRIO

O terceiro bloco, circuito de comunicação padrão RS-232, é responsável pelo interfaceamento entre o computador pessoal (PC) e o microcontrolador, o qual utiliza um circuito integrado (CI) MAX232, que estabelece a conversão dos níveis de tensão do microcontrolador (0V/+5V) para os níveis de tensão da porta serial do computador, padrão RS232 (-12V/+12V) e vice-versa [21].

O quarto bloco, receptor de radiofreqüência (RF), contempla o módulo de recepção de dados por meio de radiofreqüência, que como pode ser observado, possui apenas as ligações para a alimentação, terra, antena externa e dados.

O quinto bloco, chamado de buffer, possui um circuito integrado 74244, que nada mais é do que um buffer de três-estados, utilizado para controlar o fluxo de dados entre o receptor RF e o PC, e entre o PC e o microcontrolador [22].

E por último o sexto bloco, transmissor RF, contempla o módulo de transmissão de dados por meio de radiofreqüência, que assim como o receptor RF é extremamente simples, possuindo apenas as ligações para a alimentação, terra, antena externa e dados, sendo que esta última é ligada diretamente ao microcontrolador.

Como explicado anteriormente, o sistema projetado pode ser dividido em duas partes, a Figura 3.3 ilustra o diagrama em blocos da segunda parte.

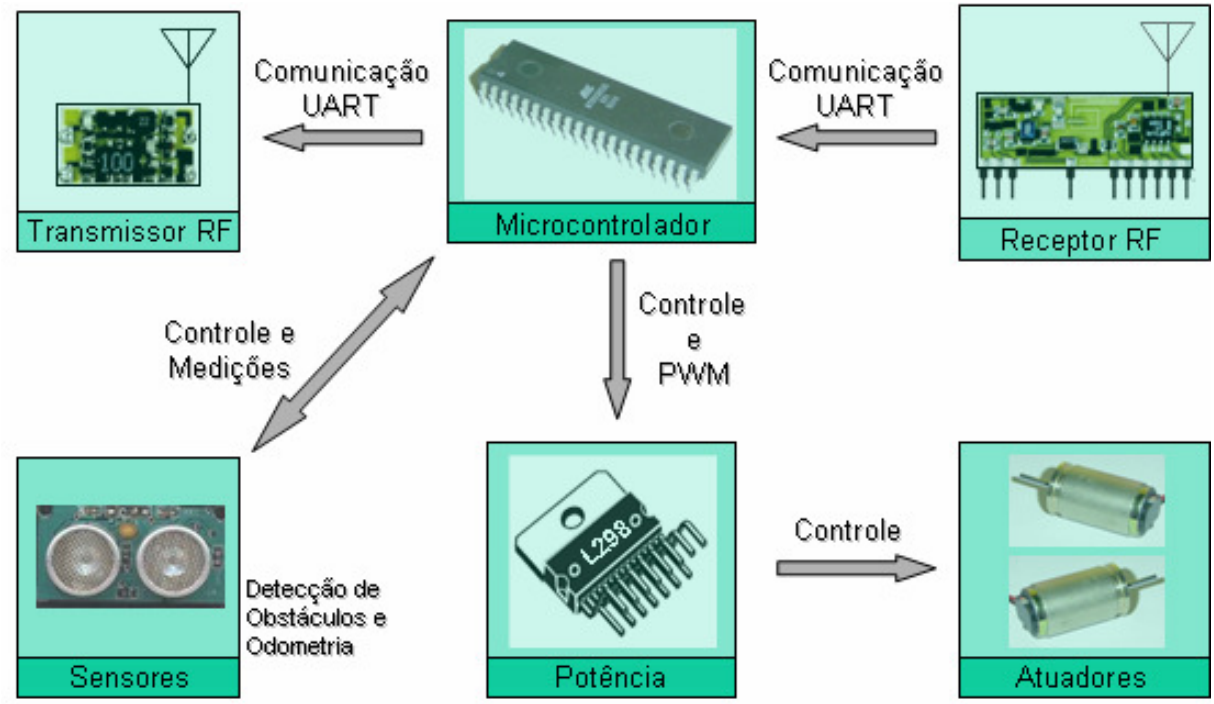


FIGURA 3.3 – DIAGRAMA EM BLOCOS DO ROBÔ

Essa segunda parte do sistema está relacionada com o circuito do robô, que contempla um microcontrolador, os módulos de transmissão e recepção, circuito de potência, os atuadores e os sensores.

O diagrama elétrico da segunda parte do sistema é apresentado na Figura 3.4. Nessa figura, da mesma forma como explicado anteriormente, pode ser observado que existe uma separação em seis blocos. Os blocos de: alimentação, microcontrolador, receptor e transmissor RF, possuem a mesma função que a apresentada no sistema de interface com o usuário.

As exceções nessa parte do sistema, estão no circuito de potência e módulo dos sensores.

O circuito de potência, apresenta basicamente uma ponte H em circuito integrado com o objetivo de fornecer corrente suficiente para acionar os atuadores.

O módulo dos sensores apresenta o circuito de interface para o sensor de ultra-som adicionado ao sistema e os sensores de velocidade, utilizados para o mapeamento do caminho percorrido pelo robô. Juntamente com os sensores de velocidade, é utilizado um buffer (CI 74244), utilizado para o controle liga/ desliga dos sensores, este que é comandado pelo microcontrolador.

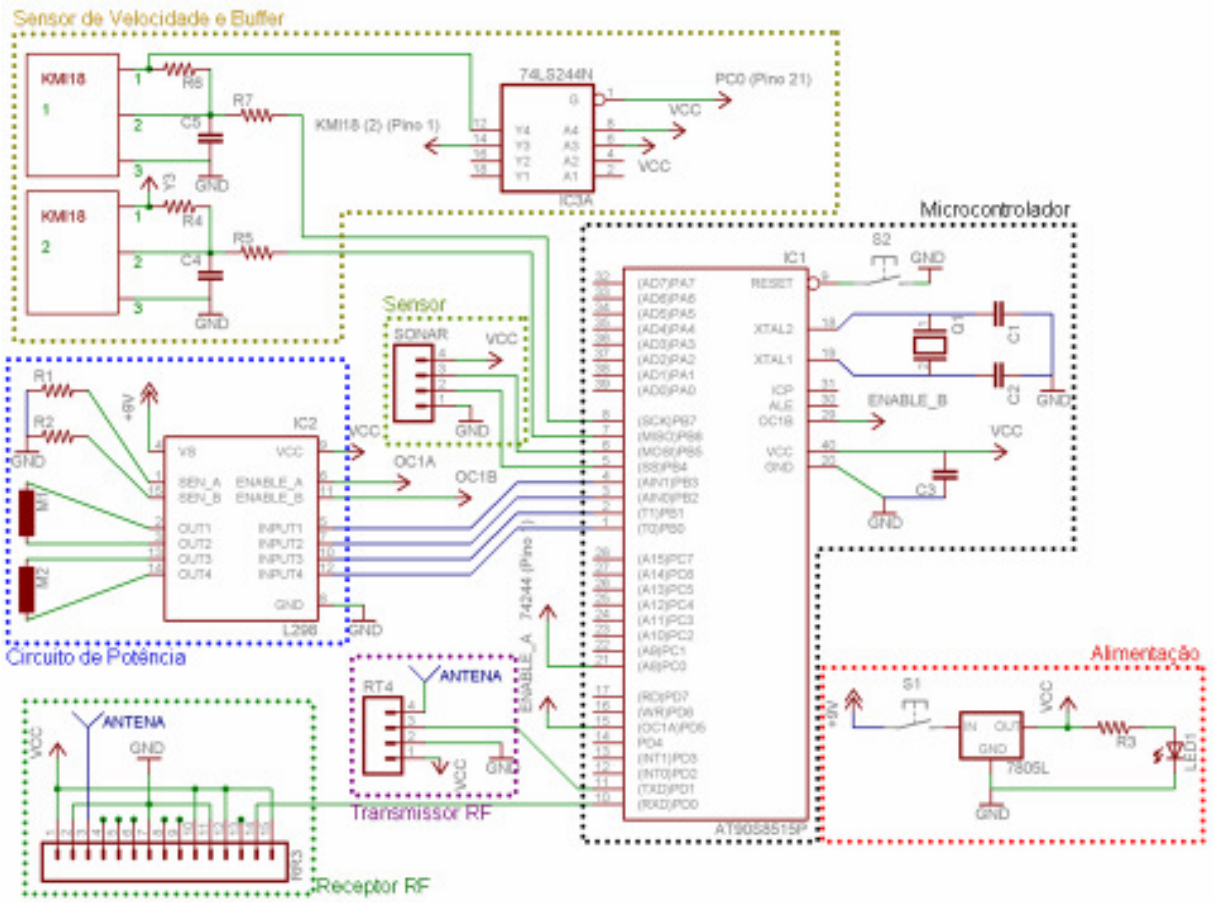


FIGURA 3.4 – DIAGRAMA ELÉTRICO DO ROBÔ

3.2 BREVE DESCRIÇÃO DO FUNCIONAMENTO DO SISTEMA

Para dar uma visão geral do funcionamento do sistema, a Figura 3.5 tenta ilustrar o conceito adotado.

Após iniciado o funcionamento do sistema, com base nas informações obtidas pelos sensores e transmitidas ao microcontrolador do robô, essas informações são processadas, codificadas e enviadas para o circuito de interface do usuário com o PC, através de uma comunicação por meio de radiofreqüência. Para isso, os dados enviados pelo microcontrolador são decodificados e posteriormente enviados ao PC através de um circuito integrado que faz a conversão dos níveis de tensão do microcontrolador para os níveis de tensão da porta serial do PC. As informações recebidas são apresentadas ao usuário, neste instante o sistema fica aguardando uma nova tomada de decisão do usuário, e assim que um

novo comando é selecionado e transmitido pela porta serial ao circuito de IHM, o microcontrolador codifica os dados recebidos e envia ao robô através de uma comunicação por meio de radiofreqüência. Por último, da mesma forma como no caso anterior, o microcontrolador processa e decodifica as informações recebidas, acionando os sensores de velocidade e de obstáculo, assim como o circuito de potência, para que esse possa fornecer energia suficiente para o acionamento dos atuadores. E assim, esse ciclo se repete continuamente.

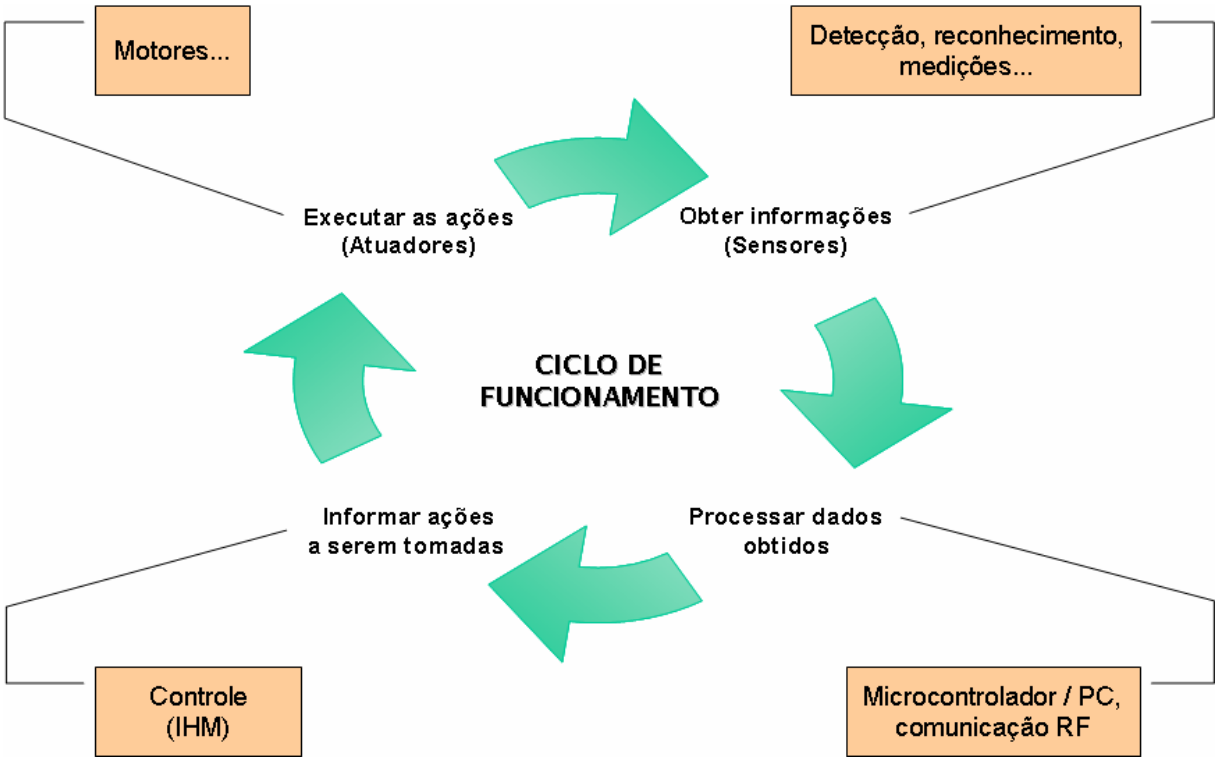


FIGURA 3.5 – CICLO DE FUNCIONAMENTO

4 COMPONENTES DO SISTEMA

A seguir é dada uma breve descrição dos componentes utilizados no projeto, assim como uma breve explicação a respeito da escolha dos principais componentes.

A Tabela 4.1 lista os componentes utilizados no circuito de interface com o usuário, conforme já ilustrados no circuito elétrico apresentado na Figura 3.2.

TABELA 4.1 – COMPONENTES DO MÓDULO DE INTERFACE COM O USUÁRIO

Designação	**Nomenclatura**[7]	**Fabricante**	**Valor/Unid.**	**Qtd**	**Custo**[8] **(R$)**
Microcontrolador	AT90S8515	Atmel	-	1	15,00
RS232 Transceiver	MAX232	Maxim	-	1	2,50
Buffer de Três-Estados	HD74HC244	Hitachi	-	1	1,30
Receptor RF	RR3	Telecontrolli	433,92MHz	1	17,50
Transmissor RF	RT4	Telecontrolli	433,92MHz	1	17,50
Cristal (Xtal)	Q1	KDS	4MHz	1	1,30
Capacitor (Cerâmico)	C1 e C2	Thonson	27pF	2	0,40
Capacitor (Cerâmico)	C3	Thonson	100nF	1	0,20
Capacitor (Eletrolítico)	C4, C5, C6 e C7	Thonson	1μF	4	0,80
Resistor	R1	Acel	300Ω	1	0,05
Regulador de Tensão	7805L	STMicroelectronics	5V	1	1,00
LED	LED1	Cons.	Vermelho	9	10,80

A Tabela 4.2 lista os componentes utilizados no circuito do robô, conforme já ilustrados no circuito elétrico apresentado na Figura 3.4.

TABELA 4.2 – COMPONENTES DO MÓDULO DO ROBÔ

Designação	**Nomenclatura**[9]	**Fabricante**	**Valor/Unid.**	**Qtd**	**Custo**[10] **(R$)**
Microcontrolador	AT90S8515	Atmel	-	1	15,00
Ponte H (Driver)	L298N	STMicroelectronics	-	1	12,30
Receptor RF	RR3	Telecontrolli	433,92MHz	1	17,50

[7] Nomenclatura baseada no circuito elétrico apresentado na Figura 3.2

[8] Valor médio aproximado encontrado na Região Sudeste (São Paulo, 2006)

[9] Nomenclatura baseada no circuito elétrico apresentado na Figura 3.4

[10] Valor médio aproximado encontrado na Região Sudeste (São Paulo, 2006)

Designação	Nomenclatura[9]	Fabricante	Valor/Unid.	Qtd	Custo[10] (R$)
Transmissor RF	RT4	Telecontrolli	433,92MHz	1	17,50
Cristal (Xtal)	Q1	KDS	4MHz	1	1,30
Capacitor (Cerâmico)	C1 e C2	Thonson	27pF	2	0,40
Capacitor (Cerâmico)	C3	Thonson	100nF	1	0,20
Capacitor (Cerâmico)	C4 e C5	Thonson	33pF	2	0,40
Resistor	R1 e R2	Acel	27Ω	2	0,10
Resistor	R3	Acel	300Ω	1	0,05
Resistor	R4 e R6	Acel	2,7kΩ	2	0,10
Resistor	R5 e R7	Acel	10kΩ	2	0,10
Regulador de Tensão	7805L	STMicroelectronics	5V	1	1,00
LED	LED1	Cons.	Vermelho	1	1,20
Motores CC	M1 e M2	Faulhaber	1516E012S	2	340,00[11]
Redutor		Faulhaber	15/3K 41:1	2	510,00[11]
Buffer Três-Estados	HD74HC244	Hitachi	-	1	1,30
Sensor de Velocidade	KMI18/2	NXP	-	2	30,00[12]
Sensor de Ultrasom	SONAR	Tato Eletrônicos	-	1	50,00[13]

A seguir são apresentados maiores detalhes dos principais componentes e as principais razões que levaram à escolha desses componentes.

4.1 MÓDULOS DE COMUNICAÇÃO

Em momento algum do projeto foi considerada a possibilidade de se desenvolver os módulos de comunicação, mesmo no início do desenvolvimento do projeto, onde se enfrentou muita dificuldade para encontrar no mercado doméstico os módulos que seriam viáveis para a execução do projeto a um custo acessível. A necessidade de operar em freqüências altas torna o circuito muito crítico e de difícil manuseio sem o equipamento necessário [23].

Abaixo, são listados alguns dos fatores que foram considerados importantes durante a escolha dos módulos de comunicação:

[11] FONTE: motores@martebal.com.br (Divisão Faulhaber do Brasil, Santa Rita do Sapucaí, Minas Gerais, 2006)

[12] FONTE: http://www.nxp.com/ (NXP Semiconducters, 2007)

[13] FONTE: http://www.tato.ind.br/ (Tato Eletrônicos, São Paulo, 2006)

- baixo custo;
- freqüência alta (possibilitando uma boa imunidade a ruídos e interferências);
- não requer nenhum tipo de regulagem;
- baixos níveis de tensão e corrente;
- formato de transmissão/ recepção dos dados (digital);
- tamanho/ peso;
- alcance/ potência/ sensibilidade;
- confiabilidade;
- velocidade de transmissão/ recepção (taxa de dados).

Na Tabela 4.3, é mostrado um comparativo de algumas características dos principais módulos de comunicação analisados no mercado.

TABELA 4.3 – PRINCIPAIS CARACTERÍSTICAS TÉCNICAS DOS MÓDULOS DE COMUNICAÇÃO

Fabricante	**Modelo**	**Descrição**	**VDC (V)**	**Is (mA)**	**Freq. (MHz)**	**Potência/ Sensibilid. (dBm)**	**Taxa de Dados (kbit/s)**	**Custo (R$)**
Telecontrolli	RT4[14]	Transmis.	2-14	4,0	433,92	+7	9,6	17,50
Telecontrolli	RR3[14]	Receptor	5,0	2,5	433,92	-103	9,6	17,50
Radiometrix	TX2[15]	Transmis.	4-6	10,0	433,92	+9	40,0	59,55
Radiometrix	RX2[15]	Receptor	5,0	13,0	433,92	-100	40,0	129,03
Wenshing Electronics	TWS[16]	Transmis.	3-12	9,5	433,92	+8,8	8,0	30,00
Wenshing Electronics	RWS[16]	Receptor	5,0	7,5	433,92	-116	4,8	30,00
Keymark Technology	TXB[17]	Transmis.	3,0	10-20,0	434,00	+6,0	1,0	65,00
Keymark Technology	RXC[17]	Receptor	3,0	9,0	434,00	-100	1,0	65,00

[14] FONTE: http://www.telecontrolli.com (Telecontrolli)

[15] FONTE: http://www.radiometrix.com (Radiometrix Ltd.)

[16] FONTE: http://www.wenshing.com.tw (Wenshing Electronics Co., Ltd.)

[17] FONTE: http://www.keymark.com.tw (Keymark Technology Co., Ltd.)

Fabricante	Modelo	Descrição	VDC (V)	Is (mA)	Freq. (MHz)	Potência/ Sensibilid. (dBm)	Taxa de Dados (kbit/s)	Custo (R$)
Linx Technology	TXM[18]	Transmis.	6-9,0	3-17,0	433,92	+4,0	10,0	250,00
Linx Technology	RXM[18]	Receptor	5,0	14,0	433,92	-100	10,0	330,00
Radiometrix	BIM[19]	Transceptor	4-5,5	14,0 (Tx) 18,0 (Rx)	433,92	+10 (Pot.) -101 (Sen.)	64,0	156,17
Laipac Tech. Inc.	TRF[20]	Transceptor	2-3,6	13,0 (Tx) 19,0 (Rx)	2.400	+4 (Pot.) -80 (Sen.)	1.000	70,00

Desde o início do projeto, tinha-se como objetivo, buscar um módulo de comunicação que fosse *full-duplex* (Transceptor, *"Transceiver"*), entretanto, os módulos disponíveis no mercado doméstico são muito caros. Dessa forma, optou-se em obter os pares de módulos de transmissão/ recepção independentes e assim desenvolver um sistema *half-duplex*.

Com base nos dados informados anteriormente, os módulos de comunicação adotados para o projeto, foram os módulos da Telecontrolli de 433,92MHz, apesar das características técnicas serem claramente inferiores ao da Radiometrix, o custo também é, e a princípio as características técnicas atendem ao objetivo para o qual o projeto está sendo desenvolvido.

4.1.1 Módulo de Transmissão

FIGURA 4.1 – MÓDULO DE TRANSMISSÃO[21]

A Figura 4.1 apresenta uma imagem do módulo de transmissão adotado e a Figura 4.2, apresenta o equivalente circuito esquemático.

O modelo utilizado é o RT4-433 da Telecontrolli, circuito híbrido que permite realizar uma transmissão por meio de radiofreqüência completa. Apresenta características elétricas

[18] FONTE: http://www.linxtechnologies.com (Linx Technologies, Inc.)

[19] FONTE: http://www.radiometrix.com (Radiometrix Ltd.)

[20] FONTE: http://www.laipac.com (Laipac Technology Inc.)

[21] FONTE: http://www.telecontrolli.com

estáveis, graças à utilização da tecnologia de circuito híbrido a filme espesso *"Thick film Hybrid"*.

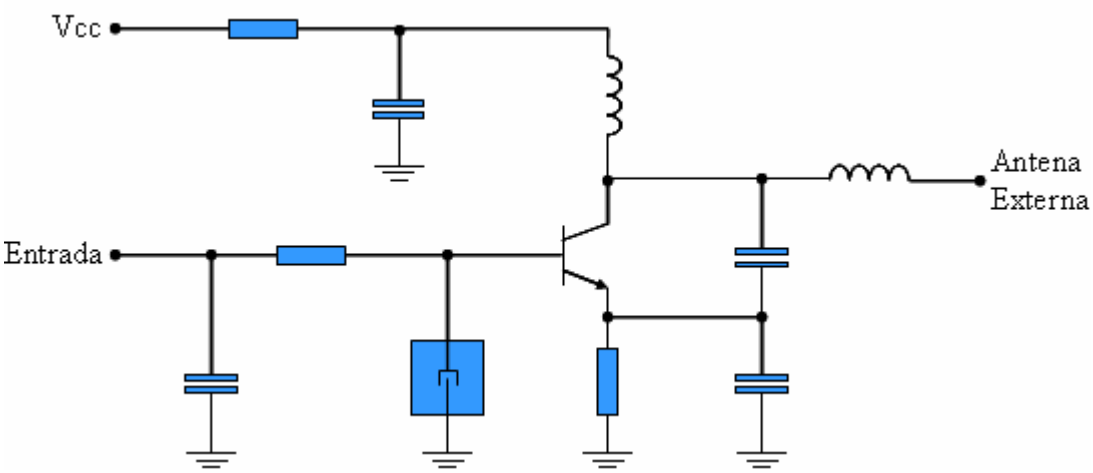


FIGURA 4.2 – CIRCUITO ESQUEMÁTICO DO MÓDULO DE TRANSMISSÃO[22]

4.1.2 Módulo de Recepção

FIGURA 4.3 – MÓDULO DE RECEPÇÃO[22]

A Figura 4.3 apresenta uma imagem do módulo de Recepção adotado e a Figura 4.4, apresenta o equivalente diagrama em blocos.

O modelo utilizado é o RR3-433 da Telecontrolli, receptor de dados de radiofreqüência super regenerativo. Apresenta alta estabilidade da freqüência, mesmo na presença de vibrações mecânicas, no manuseamento e em ambientes com grandes variações de temperatura. A exatidão da freqüência é muita alta, graças ao processo de ajuste a laser *"laser trimming process"*.

[22] FONTE: http://www.telecontrolli.com

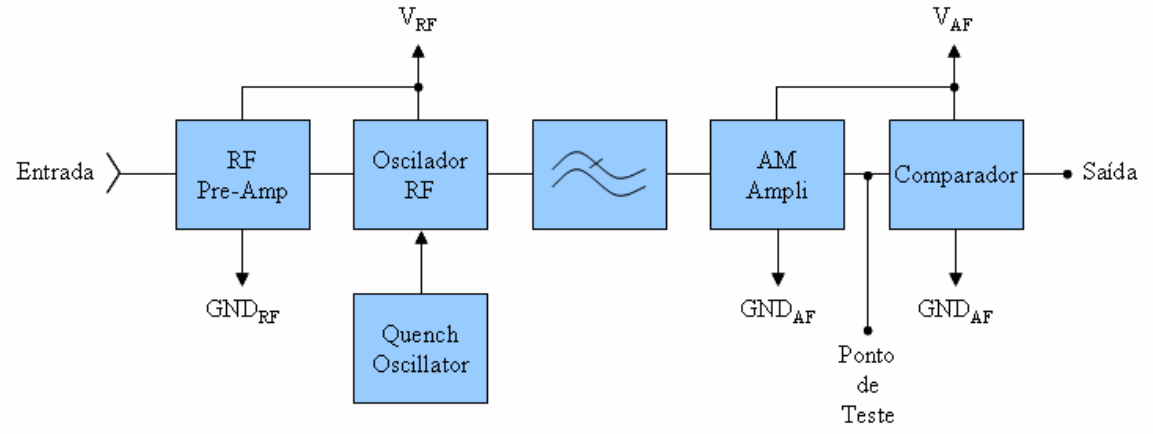


FIGURA 4.4 – DIAGRAMA EM BLOCOS DO MÓDULO DE RECEPÇÃO[23]

Apenas para ilustrar, a Figura 4.5 situa a freqüência de operação desses módulos da Telecontrolli no espectro eletromagnético.

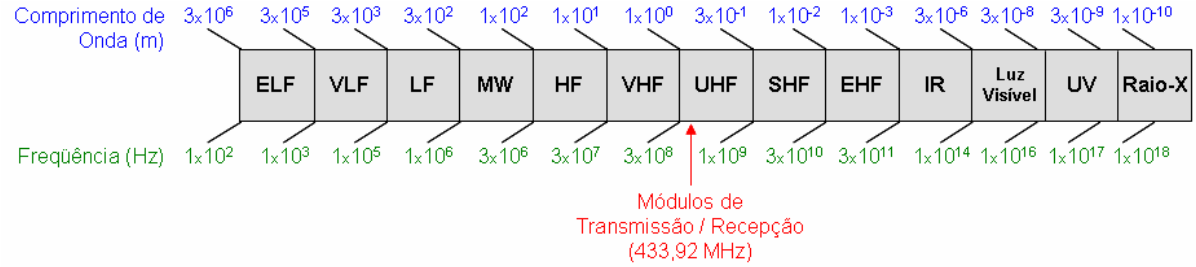


FIGURA 4.5 – ESPECTRO ELETROMAGNÉTICO[24]

4.1.3 Antena

O alcance da comunicação por meio de radiofreqüência é amplamente variável e depende tanto do tipo de antena utilizado, quanto do ambiente em que o sistema está operando.

Os três estilos de antenas de baixa potência mais comuns utilizados em projetos compactos de radiofreqüência são:

- Laço (*"Loop"*);
- Helicoidal (*"Helical"*);

---

[23] FONTE: http://www.telecontrolli.com

[24] FONTE: http://www.linxtechnologies.com (Linx Technologies, Inc. *Application Note: AN-00100*)

- Chicote (*"Whip"*).

A Tabela 4.4, ilustra um comparativo entre esses três estilos de antenas, que foi um dos determinantes utilizados para a seleção do tipo de antena adotado.

TABELA 4.4 – COMPARATIVO (PRINCIPAIS TIPOS DE ANTENAS EM PROJETOS COMPACTOS)[25]

Parâmetro	**Laço** ***"Loop"***	**Helicoidal** ***"Helical"***	**Chicote** ***"Whip"***
Desempenho final	●	●●	●●●
Facilidade da instalação do projeto	●	●●	●●●
Tamanho	●●	●●●	●
Imunidade a efeitos de proximidade	●●●	●●	●

● Regular ●● Bom ●●● Excelente

A antena adotada no projeto foi a do tipo "Whip", por possuir um desempenho excepcional e de ser facilmente desenhada e integrada ao projeto. No caso dos módulos de comunicação da Telecontrolli, o próprio fabricante recomenda esse tipo de antena. A Figura 4.6 apresenta o esquema de confecção da antena e a Tabela 4.5, mostra os parâmetros associados a esse esquema.

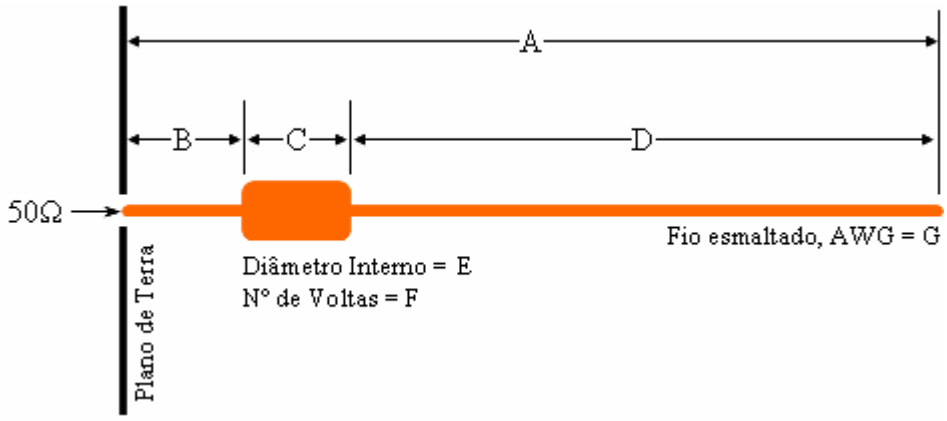


FIGURA 4.6 – ESQUEMA DE CONFECÇÃO DA ANTENA[26]

[25] FONTE: http://www.linxtechnologies.com (Linx Technologies, Inc. *RM Series Receiver Module Data Guide*)

[26] FONTE: http://www.rfm.com (RF Monolithics, Inc. *The RX Series SMT Hybrid ASH Receivers*)

TABELA 4.5 – PARÂMETROS RELACIONADOS AO ESQUEMA DE CONFECÇÃO DA ANTENA[27]

Dimensão	Unidade	Freqüência (MHz)			
		303,825	**418,000**	**433,920**	**916,500**
A	Milímetros	135,89	99,06	90,17	40,13
B	Milímetros	25,40	15,24	15,24	10,16
C	Milímetros	6,35	15,24	13,97	4,57
D	Milímetros	93,98	68,58	60,96	25,40
E	Milímetros	9,53	2,54	2,54	2,54
F	Número de Voltas	4	16	15	7
G	AWG	#20	#22	#22	#24

Durante alguns testes, a antena utilizada foi apenas um pedaço de fio de cobre com comprimento igual a ¼ do comprimento de onda [24], calculado conforme fórmula abaixo:

$$L_{(cm)} = \frac{\lambda}{4} \qquad \rightarrow \qquad \lambda = \frac{c}{f}$$

onde:

λ = comprimento de onda de uma onda sonora ou onda eletromagnética;

c = velocidade da luz no vácuo ≈ 300,000 km/s;

f = freqüência da onda 1/s = Hz.

Logo:

$$L_{(m)} = \frac{3\times10^{8}{}_{(m/s)}}{4\times433{,}92_{(MHz)}} \qquad \therefore \qquad \underline{L_{(cm)} = 17{,}28_{(cm)}}$$

4.2 MICROCONTROLADOR

O microcontrolador utilizado foi o AVR AT90S8515[28] da Atmel, sendo que suas principais características são mostradas na Tabela 4.6 [25].

[27] FONTE: http://www.rfm.com (RF Monolithics, Inc. *The RX Series SMT Hybrid ASH Receivers*)

[28] Não recomendado mais pelo fabricante (Atmel Corporation), substituído pelo ATmega8515.

TABELA 4.6 – PRINCIPAIS CARACTERÍSTICAS DO MICROCONTROLADOR AT90S8515[29]

Principais Características	Descrição
Arquitetura RISC	• Clock mais rápido, melhor desempenho, melhor otimização de compilação (número reduzido de instruções) • Utiliza arquitetura Harvard (memória e barramentos separados para dados e programa)
Alto Desempenho e Baixo Consumo	• 118 poderosas instruções, sendo as operações realizadas em sua maioria, em um ciclo de clock (busca de operandos, execução de instruções e armazenamento de resultados) • 32 registradores de trabalho de propósitos gerais, conectados diretamente com a ALU • Alcança aproximadamente 1MIPS por MHz
Dados e Memória Permanente do Programa	• Memória de programa Flash reprogramável de 8 Kbytes (Resistência: 1.000 Ciclos) • Memória EEPROM de 512 bytes (Resistência: 100.000 Ciclos) • Memória interna SRAM de 512 bytes
Características Periféricas	• Temporizadores/Contadores flexíveis, com modos de comparação • UART serial programável • *Watchdog Timer* programável com oscilador interno • Interface serial SPI • Comparador Analógico • Interrupções internas e externas
Outras Características	• Dois modos de *Power-Saving* selecionados por software • 32 linhas de I/O de propósitos gerais • Duplo PWM

Além dessas características apresentadas, outro fator importante que levou à escolha desse microcontrolador, foi a facilidade na construção de um circuito eletrônico para realizar a programação da memória e a possibilidade de escrever o código do programa, compilá-lo e fazer a simulação do seu funcionamento, economizando assim tempo na procura de erros de lógica de programação e tempo de gravação.

O fabricante fornece as ferramentas de desenvolvimento. Abaixo são listados os softwares utilizados no desenvolvimento do projeto:

- **AVR Studio 4:** Ambiente de Desenvolvimento Integrado (IDE) Profissional para a escrita e eliminação de erros (criação e simulação).
- **AVR In-System Programmer:** é o software utilizado para a atualização do programa desenvolvido para o microcontrolador.

Com intuito de garantir um melhor entendimento com relação à comunicação de dados adotada no trabalho, abaixo é dada uma breve explicação sobre a comunicação UART.

[29] FONTE: http://www.atmel.com (Atmel Corporation. *"8 Bit AVR Microcontroller with 8K Bytes In-System Programmable Flash AT90S8515"*).

4.2.1 Receptor e Transmissor Assíncrono Universal (UART)

No microcontrolador AT90S8515, dados seriais podem ser recebidos e transmitidos por meio de um UART *full-duplex*, com diferentes registradores para transmissão e recepção.

As suas principais características são:

- grande variedade de taxas de transmissão (bps);
- alta taxa de transmissão a baixas frequências do XTAL;
- 8 ou 9 bits de dados;
- filtragem de ruídos;
- detecção de *Overrun;*
- detecção de Erro de *Framing;*
- detecção de Falso Bit de Início;
- três interrupções distintas para TX Completo, Registrados TX de Dados Vazio e RX Completo.

4.2.1.1 Transmissão de Dados

A transmissão de dados é iniciada escrevendo o dado a ser transmitido no registrador de E/S de dados da UART (UDR). O dado é transferido do UDR para o registrador de deslocamento quando:

- um novo caractere seja escrito no UDR depois que o bit de parada do caractere anterior tenha sido enviado. Nesse caso, o registrador de deslocamento é carregado imediatamente;
- um novo caractere seja escrito no UDR antes que o bit de parada do caractere anterior tenha sido enviado. Então, o registrador de deslocamento é carregado quando o bit de parada do caractere que está sendo transmitido tenha sido enviado.

4.2.1.2 Recepção de Dados

Quando o receptor da UART detecta um bit de início válido, é iniciada a amostragem dos bits de dados recebidos. O valor lógico encontrado em pelo menos duas das três amostras, é tomado como o valor do bit. Todos os bits são então transferidos para o registrador de deslocamento, da forma como eles são amostrados.

Quando um bit de parada entra no receptor da UART, a maioria dos três bits amostrados deve ser de nível lógico igual a "1", para que o bit de parada seja aceito. Se duas ou mais amostras forem de nível lógico igual a "0", um erro de *framing* (dado recebido em tamanho diferente do esperado) é setado no registrador de *status* da UART (USR). Outros erros, como por exemplo, quando um byte é recebido antes que o dado anterior tenha sido lido, o USR é setado com um erro de *overrun*.

4.2.1.3 Gerador de Taxa de Transmissão

O gerador de taxa de transmissão é um divisor de freqüência que gera a taxa de transmissão, conforme a seguinte equação:

$$BAUD = \frac{f_{CK}}{16 \cdot (UBRR + 1)}$$

onde:

BAUD = Taxa de Transmissão;

f_{CK} = Freqüência do Cristal utilizado no projeto (no caso foi 4MHz);

UBRR = Conteúdo do registrador da taxa de transmissão da UART.

Logo:

$$BAUD_{(bps)} = \frac{4MHz}{16 \cdot (103 + 1)} \qquad \therefore \qquad \underline{BAUD_{(bps)} \approx 2400_{(bps)}}$$

A velocidade adotada para a comunicação com os módulos de comunicação RF foi de 2.400bps. Dessa forma, analisando a tabela de correspondência de dados entre os valores de UBRR em relação às diversas freqüências de cristal disponíveis no mercado e a taxa de erro,

para que se tenha uma taxa de transmissão na ordem de 2.400bps e utilizando um cristal de 4MHz, o valor de UBRR é de 103 e a taxa de erro é de aproximadamente 0,2%.

4.3 MÓDULO DE POTÊNCIA

O CI L298N é o responsável pelo módulo de potência no circuito do robô. Esse componente nada mais é do que uma ponte H em um circuito integrado. As maiores vantagens de se utilizar um componente como esse, está na redução do espaço ocupado no circuito e na simplicidade da montagem.

A Figura 4.7 apresenta o diagrama em blocos do CI L298N [26].

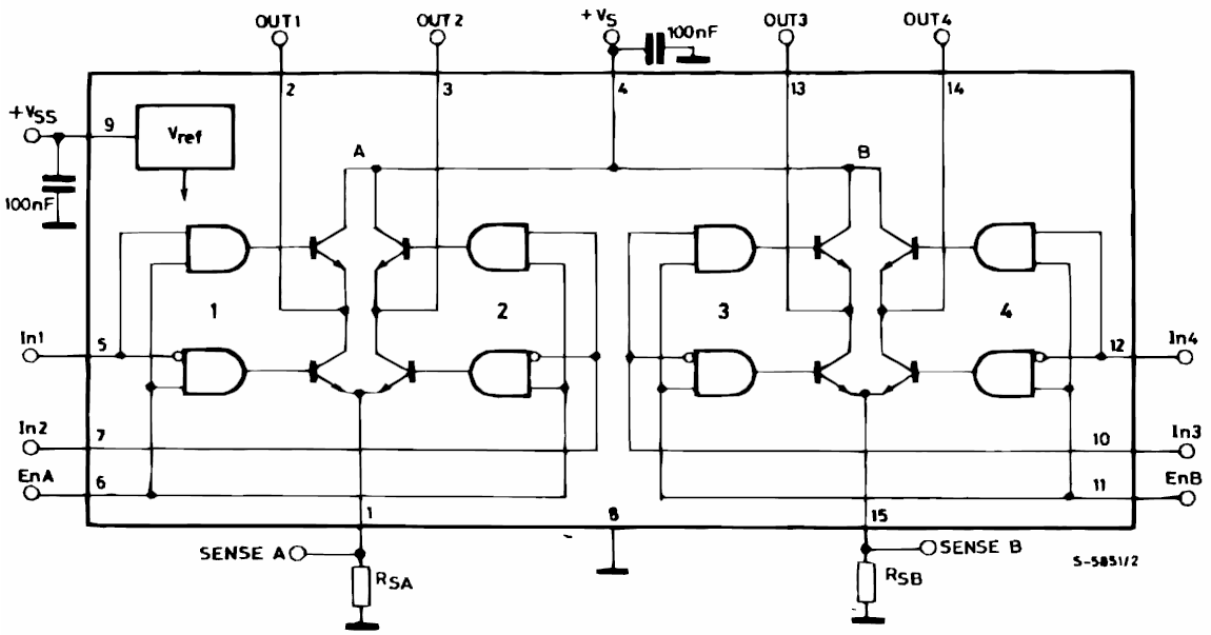


FIGURA 4.7 – DIAGRAMA EM BLOCOS DO CI L298N[30]

O L298 integra dois estágios de potência de saída (A e B). O estágio de potência de saída, dependendo do estado da entrada, pode movimentar uma carga indutiva, como por exemplo, relés, solenóides, motores de passo e motores de corrente contínua.

Na Tabela 4.7 são mostradas as funções dos principais pinos de conexão do L298.

[30] FONTE: http://www.st.com (STMicroelectronics)

TABELA 4.7 – FUNÇÕES DOS PRINCIPAIS PINOS DO L298[31]

Nome	Função
Sense A; Sense B	Entre esse pino e o Terra é conectado um resistor sensitivo para controlar a corrente da carga.
Out 1; Out 2	Saídas da Ponte A. A corrente que flui através da carga conectada entre esses dois pinos, é monitorada pelo pino 1.
Input 1; Input 2	Entradas da Ponte A (compatíveis com nível TTL).
Enable A; Enable B	Entradas (compatíveis com nível TTL); nível baixo desabilita a Ponte A (*enable A*) e/ou a Ponte B (*enable B*).
Input 3; Input 4	Entradas da Ponte B (compatíveis com nível TTL).
Out 3; Out 4	Saídas da Ponte B. A corrente que flui através da carga conectada entre esses dois pinos, é monitorada pelo pino 15.

4.4 MOTORES DE CORRENTE CONTÍNUA

4.4.1 Procedimento Para Escolha dos Motores

Com o objetivo de confirmar se as características técnicas do motor a ser utilizado, atenderiam os requisitos mínimos do projeto, uma verificação conforme procedimento[32] listado abaixo, foi realizado:

1. estimar o peso do robô;
2. estimar ou determinar empiricamente os coeficientes de fricção;
3. escolher a velocidade máxima desejada, tamanho da roda e aceleração;
4. a partir da velocidade máxima e do tamanho da roda definido, determinar a velocidade angular do motor;
5. a partir do peso estimado e da aceleração requerida, determinar o torque mínimo do motor;
6. verificar se o motor escolhido apresenta especificação de torque e velocidade que excedam os valores calculados.

[31] FONTE: http://www.st.com (STMicroelectronics)

[32] FONTE: http://academics.vmi.edu/ee/robocup.htm (*Virginia Military Institute*)

4.4.1.1 Velocidade Linear

A velocidade linear é determinada pelo tamanho da roda, pela velocidade angular do motor para um determinado torque, e por alguma engrenagem que possa existir entre o motor e a roda. Iniciando com o tamanho da roda, a distância (*d*) percorrida pelo robô para cada rotação da roda, é dada pela equação:

$$d = 2.\pi.r$$

onde *r* é o raio da roda. A velocidade linear média (*v*) da translação é simplesmente essa distância dividida pelo tempo percorrido:

$$v = d.f = 2.\pi.r.f$$

onde *f* é a velocidade angular. Fazendo a velocidade angular como uma função da velocidade linear, a equação fica:

$$f = \frac{v}{2.\pi.r}.$$

Multiplicando por 60 segundos, a velocidade angular é expressa em unidade de Rotações Por Minuto (RPM):

$$f_{(RPM)} = \frac{60.v}{2.\pi.r} = \frac{30.v}{\pi.r}$$

Sabendo-se que o diâmetro da roda utilizada no robô é de 6,5cm e que é desejado que a velocidade linear do robô seja de pelo menos 0,5m/s, temos que a velocidade angular *f* em RPM será de:

$$f_{(RPM)} = \frac{30_{(s)}.v_{(m/s)}}{\pi.r_{(m)}}$$

$$f_{(RPM)} = \frac{30_{(s)}.0{,}5_{(m/s)}}{3{,}1416 \times 0{,}0325_{(m)}}$$

$$f_{(RPM)} = 146{,}91 \approx 147_{(RPM)}$$

4.4.1.2 Torque

Para determinar quanto torque será necessário, é preciso saber quais forças estão envolvidas para movimentar o robô. Considerando que o robô projetado estará percorrendo ambientes planos, apenas duas forças devem ser consideradas, aquela devido à inércia quando em aceleração e aquela devido à força de fricção.

A força de fricção é difícil de ser calculada, pois depende dos tipos de materiais a que o robô terá contato. Geralmente os coeficientes de fricção entre o pneu e o concreto/asfalto, variam entre 0,001 e 0,03[33]. A força inercial é produto da massa pela aceleração.

Sabendo-se que o peso do robô projetado é de aproximadamente 400g, a conexão entre o motor e a roda é direta, ou seja, sem nenhum tipo de engrenagem, a aceleração desejada seja de 0,5m/s^2 e que o robô possui duas rodas/motores, o torque pode ser calculado, multiplicando-se a soma da força inercial e a força de fricção pelo raio da roda, conforme é ilustrado na Figura 4.8.

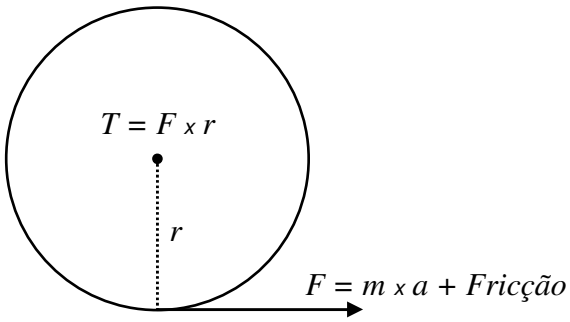


FIGURA 4.8 – FORÇA x RAIO = TORQUE[34]

Com base nos valores descritos anteriormente, sabendo-se que g = 9,80m/s^2 (aceleração da gravidade) e considerando que o coeficiente de fricção é no pior caso de 0,03, o valor do torque mínimo requerido para movimentar o robô é calculado, conforme apresentado logo abaixo:

$$F_{(N)} = m_{(kg)} \times a_{(m/s^2)} + 0{,}03 \times m_{(kg)} \times g_{(m/s^2)}$$

$$F_{(N)} = 0{,}40_{(kg)} \times 0{,}5_{(m/s^2)} + 0{,}03 \times 0{,}40_{(kg)} \times 9{,}8_{(m/s^2)}$$

[33] FONTE: http://en.wikipedia.org/wiki/Rolling_resistance

[34] FONTE: http://lancet.mit.edu/motors/index.html (*Designing with D.C. Motors*)

$$\underline{F_{(N)} = 0{,}3176_{(N)}}$$

$$T_{(N.m)} = F_{(N)} \times r_{(m)}$$

$$T_{(N.m)} = 0{,}3176_{(N)} \times 0{,}0325_{(m)}$$

$$\underline{T_{(mN.m)} \cong 10_{(mN.m)}}$$

Como são dois motores no robô, o valor de torque requerido deve ser dividido por dois, portanto:

$$\underline{T_{(mN.m)} = 5_{(mN.m)}}$$

4.4.2 Características Técnicas dos Motores Utilizados

Os motores utilizados no projeto são motores de corrente contínua fabricados pela FAULHABER modelo 1516E012S 15/3K. A Tabela 4.8, apresenta as principais características técnicas desse motor, assim como na Figura 4.9 e Figura 4.10, são apresentadas as curvas características do Torque e Potência versus a Velocidade Angular sem e com redução interna respectivamente.

TABELA 4.8 – PRINCIPAIS CARACTERÍSTICAS TÉCNICAS DOS MOTORES[35]

Características	Unidade	Valores Especificados (Motor sem Redução)	Valores Especificados (Motor com Redução)
Tensão Nominal	Volt	12	12
Resistência do Terminal	Ω	117	117
Potência de Saída	W	0,27	0,27
Eficiência	%	61	73
Velocidade Sem Carga	RPM	16.200	395
Corrente Sem Carga	A	0,006	0,006
Torque	mN.m	0,5	20,5
Stall Torque	mN.m	0,64	26,24
Peso	g	10	26

[35] FONTE: http://www.faulhaber-group.com (*Faulhaber Group*)

Resumindo, esse modelo é controlado por uma tensão contínua de no máximo 12V, com uma redução interna de 41:1, resultando num torque máximo em operação contínua de 20,5mN.m e com uma rotação máxima sem carga de aproximadamente 395RPM.

A Tabela 4.9, ilustra a comparação entre os valores especificados pelo fabricante e os valores mínimos requeridos, comprovando-se o atendimento aos requisitos do sistema.

TABELA 4.9 – CARACTERÍSTICAS TÉCNICAS DO MOTOR VS. VALORES MÍNIMOS REQUERIDOS

Características	**Unidade**	**Valores Especificados (Motor com Redução)**	**Valores Mínimos Requeridos**
Potência de Saída	W	0,27	0,092
Velocidade Angular	RPM	395	147
Torque	mN.m	20,5	5,0

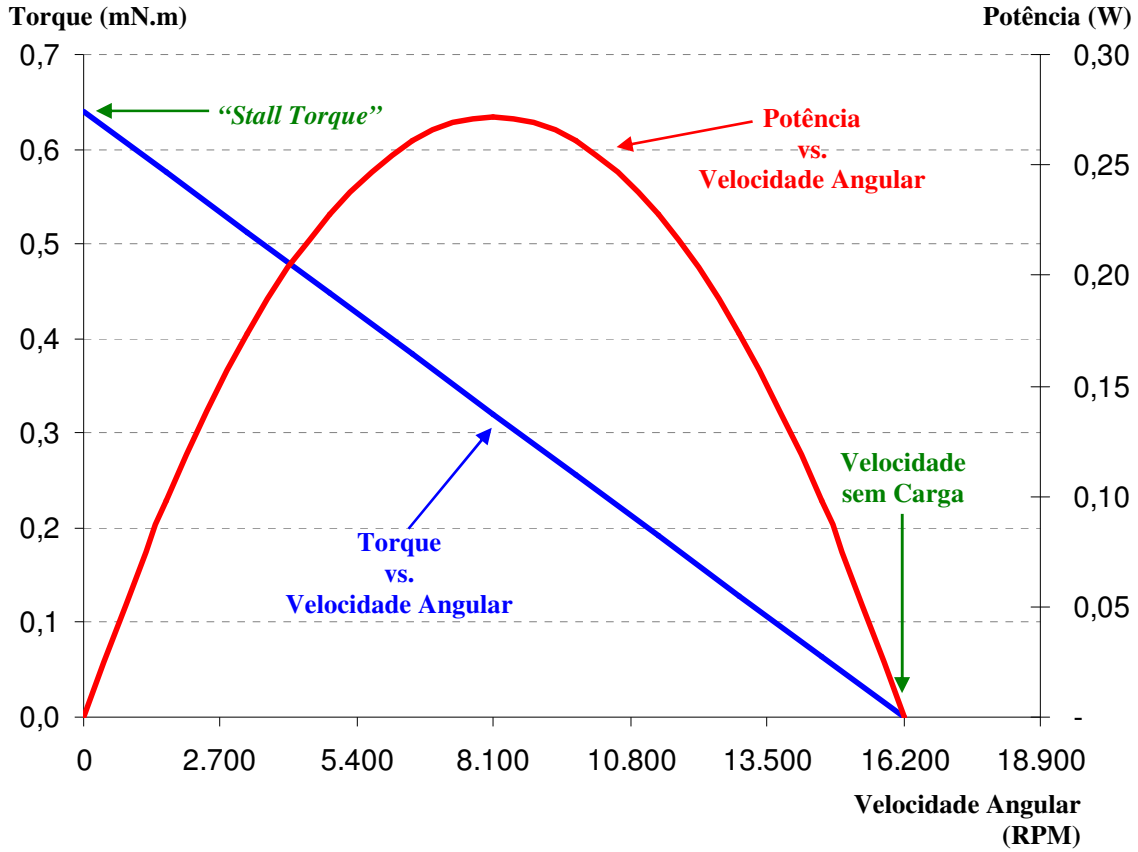


FIGURA 4.9 – CURVA DE TORQUE E POTÊNCIA VS. VELOCIDADE ANGULAR (SEM REDUÇÃO)

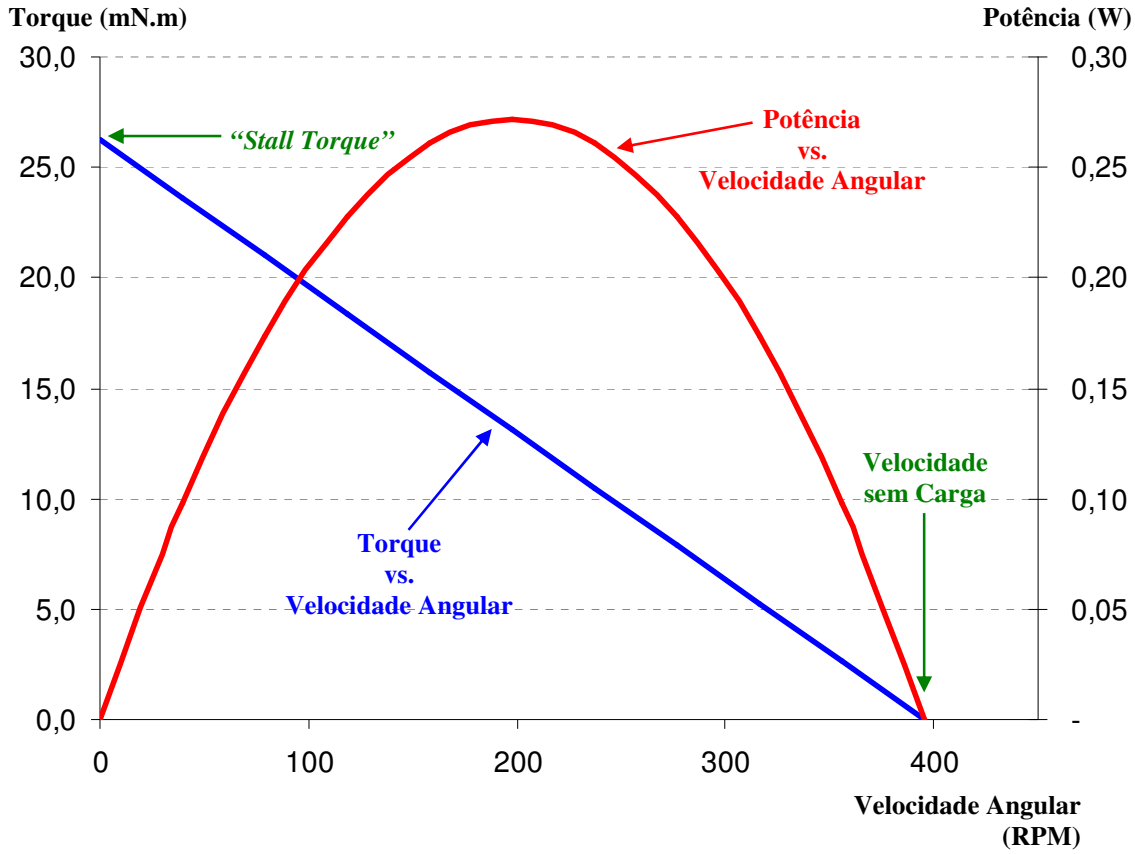


FIGURA 4.10 – CURVA DE TORQUE E POTÊNCIA VS. VELOCIDADE ANGULAR (COM REDUÇÃO)

4.5 SENSORES

Podem-se definir sensores, como sendo dispositivos que mudam seu comportamento sob a ação de uma grandeza física, podendo fornecer diretamente ou indiretamente um sinal que indica essa grandeza.

A seguir, são apresentadas algumas características que diferenciam os tipos de sensores encontrados e que são importantes na definição de qual tipo de sensor é mais apropriado para um determinado sistema [27]:

- sensibilidade: taxa de variação da leitura em relação ao estímulo;
- linearidade: medida de constância da sensibilidade;
- faixa de medida: diferença entre o mais intenso e o menos intenso estímulo mensurável;
- tempo de resposta: tempo necessário para uma mudança de estímulo se refletir na leitura;

- precisão: diferença entre leitura e estímulo;
- repetibilidade: diferença entre medidas sucessivas do mesmo estímulo;
- resolução: menor variação de estímulo mensurável;
- tipo de saída: movimento mecânico, corrente, tensão, pressão, nível de luminosidade, etc.

Algumas características físicas também podem ser destacadas [27]:

- tamanho e peso: dependendo da aplicação, pode ser um aspecto muito importante;
- confiabilidade: tempo médio para falha (MTTF "*Mean Time to Failure*"), número médio de horas entre falhas que causam impossibilidade de operação do sensor;
- interfaceamento: necessidade de conexões não padronizadas, tensões não-padronizadas, etc.
- aumento de complexidade e custo.

Os principais tipos de sensores são: sensores térmicos (resistivos, termopar), sensores mecânicos (deslocamento, posição, nível, tensão), sensores ópticos e sensores de campos magnéticos.

Existe uma infinidade de tipos de sensores e cada um pode ser mais bem aplicado para determinadas aplicações.

4.5.1 Sensores de Ultrasom

Os sensores de ultrasom (sonares) possuem propriedades únicas que são muito úteis na robótica móvel. Esse tipo de sensor possui diversas vantagens, sendo relativamente barato, robusto, preciso e capaz de detectar uma ampla variedade de objetos. O importante é que estimativas robustas de distância podem ser extraídas dos dados fornecidos pelo sensor. Essas informações podem ser utilizadas para evitar obstáculos, o que geralmente são difíceis de se obter por outros métodos.

4.5.1.1 Princípio de Funcionamento

O sensor emite um pulso acústico (onda sonora) e aguarda o retorno de um eco por um determinado período de tempo. Assim que o eco é detectado, é possível medir a distância entre um ponto observado, simplesmente multiplicando a velocidade do som pela metade do tempo medido (metade do tempo de vôo da onda sonora).

4.5.1.2 Considerações Importantes

Apesar das vantagens citadas, existem alguns problemas que precisam ser levados em consideração para a medição das distâncias [28]:

- **velocidade de propagação:** a velocidade de propagação das ondas sonoras é fortemente influenciada por variações de temperatura e de uma forma mais suave pela umidade atmosférica.
- **incerteza na detecção:** a energia refletida por um alvo depende do tipo de material do qual é composta a superfície desse material, o qual vai influenciar a intensidade do sinal refletido.
- **interação com as superfícies:** o eco recebido contém apenas uma fração da energia contida no sinal enviado. A energia restante é perdida em efeitos de absorção, difração ou refração. É freqüente, nesses tipos de sistemas, o sensor não receber qualquer sinal refletido, mesmo existindo um objeto na frente dele. Esse fenômeno deve-se ao fato de muitas superfícies se comportarem como espelhos para as ondas ultra-sonoras. Assim, se o ângulo de emissão, em relação à superfície do alvo, exceder um determinado valor crítico, a energia refletida será desviada para fora da zona de detecção do receptor. O que pode ocorrer, é que esses sinais sendo multiplamente refletidos, regressem ao receptor produzindo uma medida errada que é igual à soma das distâncias percorridas pelas ondas.

4.5.1.3 Descrição do Módulo de Sonar Adotado

O sensor de ultrasom, conforme ilustrado na Figura 4.11, possui quatro pinos, sendo dois da alimentação, Vcc (5V) e Terra, um pino de saída que fornece um pulso em nível alto cuja duração é o tempo para o som ir até o obstáculo e voltar e um pino de entrada que quando em nível baixo repete a medição.

FIGURA 4.11 – MÓDULO DE ULTRASOM ADOTADO (SONAR)[36]

De acordo com as especificações do fabricante[36], o módulo sonar tem um consumo médio de 2mA, alcance mínimo de 20cm e máximo de 1,5m. No final deste Capítulo, serão mostrados testes realizados que mostram que por meio do sistema projetado, foi possível detectar objetos a uma distância mínima de até 4cm.

4.5.2 Sensores de Velocidade/ Posição

Os sensores de velocidade e/ou de posição, são normalmente utilizados para a determinação do deslocamento do robô, na qual são acoplados nas rodas e possuem a capacidade de medir a rotação das rodas.

Segue abaixo alguns tipos de sensores de deslocamento e velocidade em uso atualmente e uma breve explicação sobre o funcionamento de cada um deles [28] [29].

- Encoder óptico incremental: um disco com janela a intervalos regulares é iluminado por uma fonte de luz, sendo que do outro lado do disco, fica um elemento fotosensível. Ao girar, alterna-se as áreas com janela e sem janela, gerando no elemento fotosensível uma seqüência de pulsos. A contagem dos pulsos, permite a avaliação do deslocamento angular.

[36] FONTE: http://www.tato.ind.br/ (Tato Eletrônicos)

- Encoder óptico absoluto: fornece um valor numérico específico (valor codificado) para cada posição angular. O disco codificado é firmemente montado ao eixo. Esse disco é dividido em segmentos separados que são alternadamente transparentes ou opacos. A fonte de luz emite um feixe de luz orientado paralelamente que ilumina todos os segmentos do disco codificado. Foto-unidades recebem a luz modulada e convertem-na em sinais que serão posteriormente digitalizados e fornecidos como uma saída.
- Encoder indutivo: sensor bastante utilizado para medida de pequenas distâncias, sendo baseado em correntes que são induzidas numa superfície condutora quando as linhas de fluxo magnético interceptam essa superfície. As correntes produzidas na superfície do material condutor é uma função da distância da bobina ativa e a superfície. Essas correntes aumentam quando a distância diminui.
- Encoder capacitivo: semelhante ao indutivo, a diferença entre eles reside no fato de que no sensor capacitivo o princípio de funcionamento está baseado na variação do dielétrico no meio em que o sensor está inserido. Quando nesta região penetrar algum objeto, este provoca a variação do dielétrico e, conseqüentemente a variação da freqüência do oscilador, variação esta que é detectada e transformada em um nível de tensão.
- Potenciométrico: este tipo de sensor é bastante utilizado em impressoras jato de tinta para controle do posicionamento do carro que contém os cartuchos de tinta.

O mais popular tipo de encoder para robôs móveis é o encoder óptico (incremental ou absoluto).

4.5.2.1 Descrição do Sensor de Velocidade Adotado

O sensor adotado é o KMI 18/2, que é um sensor integrado, utilizado para a medição de velocidade rotacional [30]. Conforme ilustrado na Figura 4.12, o sensor possui apenas três pinos, sendo dois da alimentação, Vcc (5V) e Terra, e um pino de saída.

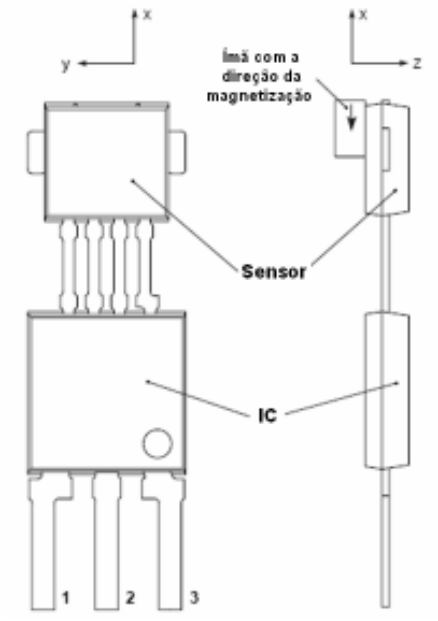


FIGURA 4.12 – SENSOR DE VELOCIDADE ADOTADO[37]

4.5.2.2 Princípio de Funcionamento

O sensor detecta a velocidade de rotação das rodas de acordo com os marcos magnéticos referenciados na roda. A freqüência do sinal de saída é proporcional à velocidade rotacional da roda.

O princípio funcional é mostrado na Figura 4.13.

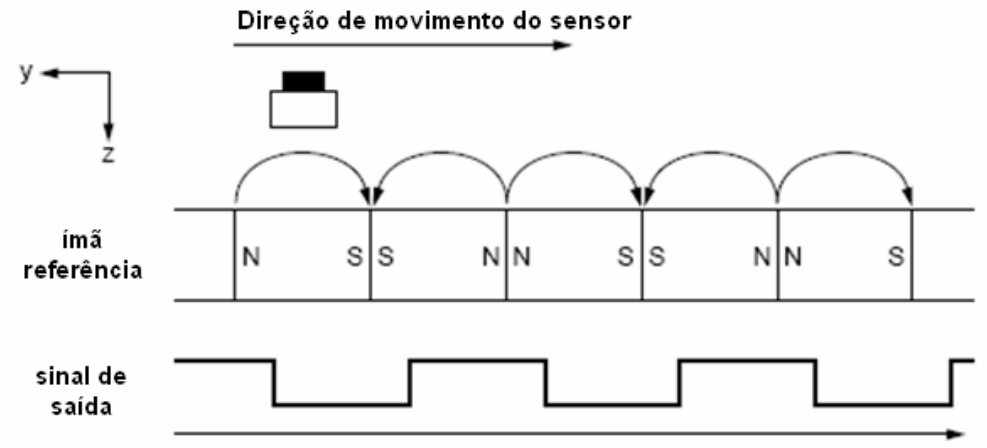


FIGURA 4.13 – PRINCÍPIO DE FUNCIONAMENTO DO SENSOR DE VELOCIDADE ADOTADO[37]

[37] FONTE: http://www.nxp.com/ (Philips Semiconductors)

4.5.3 Alguns Testes Realizados com os Sensores

4.5.3.1 Teste de Funcionamento do Sensor de Ultrasom (Parte 1)

O objetivo principal desse teste foi o de confirmar o funcionamento do sensor de ultrasom (Sonar), para tornar possível a detecção de obstáculos e conseqüentemente o mapeamento do ambiente envolvido.

Diversas medições foram realizadas através de um osciloscópio e os resultados obtidos são apresentados na Tabela 4.10, assim como a Figura 4.14, apresenta os dados obtidos pelo osciloscópio.

TABELA 4.10 – RESULTADOS OBTIDOS NO TESTE DO SONAR (PARTE 1)

Distância Medida (cm)	**Tempo Medido[38] (ms)**	**Distância Calculada (cm)**	**Diferença (cm)**	**Porcentagem de Erro**
2,5	0,30	5,1	2,6	51%
5,0	0,50	8,5	3,5	41%
10,0	0,80	13,6	3,6	26%
15,0	1,10	18,7	3,7	20%
20,0	1,45	24,7	4,7	19%
30,0	2,10	35,7	5,7	16%
45,0	3,10	52,7	7,7	15%

As distâncias calculadas, conforme apresentado na Tabela 4.10, foram obtidos a partir da seguinte fórmula:

$$d_{(cm)} = \frac{\left(t_{(s)}.c_{(cm/s)}\right)}{2}$$

onde:

$d_{(cm)}$ = distância calculada em centímetros;

$t_{(s)}$ = tempo em segundos em que o som emitido retornou ao sonar;

$c_{(cm/s)}$ = velocidade do som no ar em centímetros por segundo = 340m/s.

Por exemplo, para a distância medida de 45 cm, o valor calculado é:

$$d_{(cm)} = \frac{\left(3{,}1\times10^{-3}{}_{(s)}.34\times10^{+3}{}_{(cm/s)}\right)}{2} \qquad \therefore \qquad \underline{d_{(cm)} \approx 52{,}7_{(cm)}}$$

[38] Tempos medidos conforme valores obtidos na Figura 4.14

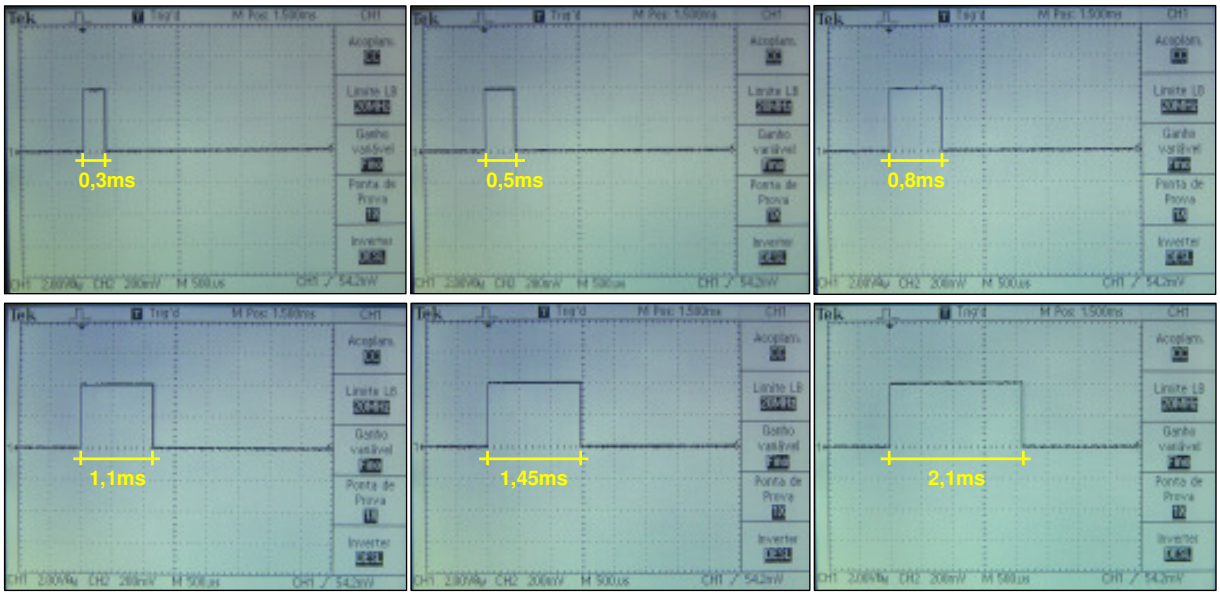


FIGURA 4.14 – CURVAS MEDIDAS PELO OSCILOSCÓPIO (TESTE DO SONAR) (PARTE 1)

A Figura 4.15 apresenta o circuito elétrico correspondente ao teste realizado.

FIGURA 4.15 – DIAGRAMA ELÉTRICO DO CIRCUITO DE TESTE DO SONAR (PARTE 1)

Nesse teste, não foi necessário nenhum tipo de desenvolvimento lógico, em nível de programação. O módulo não apresentou um resultado com uma precisão conforme o esperado, entretanto, o resultado obtido foi satisfatório e o nível de precisão é suficiente para atender os objetivos gerais do projeto.

4.5.3.2 Teste de Funcionamento do Sensor de Ultrasom (Parte 2)

O objetivo principal desse teste foi o de confirmar o funcionamento da lógica de programação desenvolvida para realizar o cálculo da distância entre um obstáculo qualquer e o sensor de ultrasom.

A Figura 4.16 apresenta o circuito elétrico correspondente ao teste realizado.

A Figura 4.17 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador.

Através do fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.5.

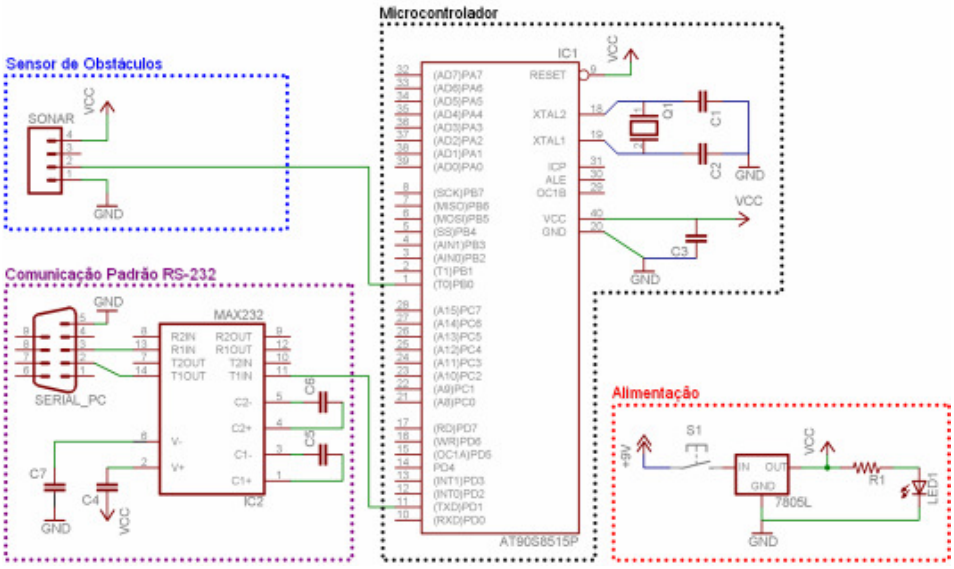
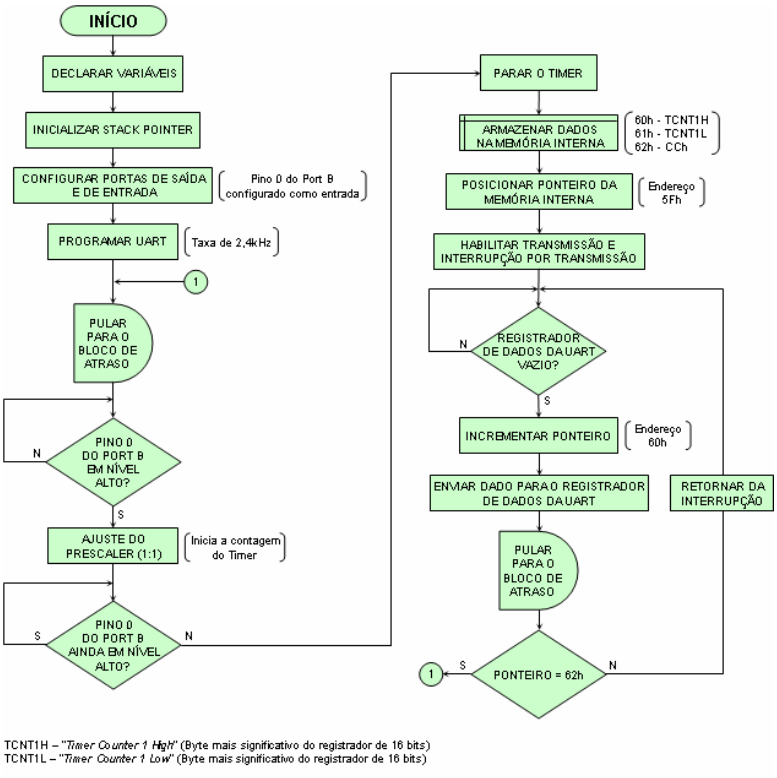


FIGURA 4.16 – DIAGRAMA ELÉTRICO DO TESTE DO SONAR (PARTE 2)



TCNT1H – "*Timer Counter 1 High*" (Byte mais significativo do registrador de 16 bits)
TCNT1L – "*Timer Counter 1 Low*" (Byte mais significativo do registrador de 16 bits)

FIGURA 4.17 – FLUXOGRAMA DA LÓGICA DE PROGRAMAÇÃO UTILIZADA

A lógica de programação adotada inicialmente para realizar a medição do tempo em alta do pulso do sensor de ultrasom, baseava-se em conceitos de Interrupção por Variação de Sinal de Entrada (*"Timer Input Capture Interrupt"*), entretanto, estava-se encontrando bastante dificuldade para obter êxito na medição correta da largura do pulso utilizando esse conceito. Dessa forma, outra técnica foi utilizada para realizar a medição do sinal, que pode ser observada conforme o fluxograma apresentado na Figura 4.17.

Sumarizando, o conceito do funcionamento do sistema é o seguinte:

1. o sensor de ultrasom emite uma onda sonora;
2. essa onda sonora é refletida pelo obstáculo;
3. o tempo que essa onda leva para retornar ao sensor é medida (registrador de 16 bits – TCNT1H e TCNT1L).
4. o valor do registrador de 16 bits é enviado para o computador pessoal através da comunicação serial. Primeiro o byte mais significativo, depois o menos significativo, e por último, o byte CCh (204 em decimal) apenas para delimitar uma separação entre o próximo valor lido pelo sensor que é submetido ao computador pessoal.
5. Os valores mostrados na Interface Homem-Máquina, estão em decimal. A Figura 4.18 apresenta um exemplo de resultado obtido e mostrado ao usuário. Pode ser observado que quatro leituras foram feitas pelo sensor e enviadas ao computador pessoal. As quatro leituras foram idênticas, o que mostra uma consistência nos valores medidos pelo microcontrolador com base na leitura dos dados recebidos pelo sensor.
6. Com base nos valores mostrados na Interface Homem-Máquina, é possível calcular o "tempo de vôo" da onda sonora. O esquema abaixo exemplifica o cálculo executado, com base nos valores apresentados na Figura 4.18:
   - Byte alto – 16 decimal = 10 hexadecimal;
   - Byte baixo – 89 decimal = 59 hexadecimal;
   - Timer = 1059 hexadecimal = 4185 decimal;
   - Cálculo do "tempo de vôo":

$$Tvoo_{(ms)} = Timer_{(decimal)} \times \frac{1}{Clock_{(Hz)}} \times \mathrm{P}rescaler$$

Onde:

$\mathrm{Tvoo}_{(\mathrm{ms})}$ = tempo de vôo em milisegundos;

$Timer_{(decimal)}$ = valor em decimal do Registrador de 16 bits;

$Clock_{(Hz)}$ = relógio do sistema (4MHz);

Prescaler = 1:1 (ou seja, acompanha o clock do sistema);

Portanto:

$$Tvoo_{(ms)} = 4185 \times \frac{1}{4 \times 10^{6}{}_{(Hz)}} \times 1$$

$$\underline{Tvoo_{(ms)} = 1{,}05_{(ms)}}$$

7. Sabendo-se o valor do "tempo de vôo", é possível calcular a distância em que o obstáculo encontra-se do sensor de ultrasom, utilizando-se a seguinte fórmula:

$$D_{(m)} = \frac{c_{(m/s)} \times Tvoo_{(s)}}{2}$$

Onde:

$D_{(cm)}$ = Distância calculada em cm;

$c_{(m/s)}$ = Velocidade do Som (temperatura ambiente);

$$c_{(m/s)} = 331{,}4_{(m/s)} \times \sqrt{\frac{Ta_{(K)}}{273_{(K)}}}$$

$Ta_{(K)}$ = Temperatura Ambiente em Kelvin

Logo:

$$Ta_{(K)} = 293{,}15_{(K)} = 20_{(^{\circ}C)}$$

$$c_{(m/s)} = 331{,}4_{(m/s)} \times \sqrt{\frac{293{,}15_{(K)}}{273_{(K)}}}$$

$$\underline{c_{(m/s)} = 343{,}41_{(m/s)}}$$

Substituindo pelos valores encontrados, temos que:

$$D_{(m)} = \frac{343{,}15_{(m/s)} \times 1{,}05_{(ms)}}{2} \qquad \therefore \qquad \underline{D_{(cm)} \cong 17{,}96_{(cm)}}$$

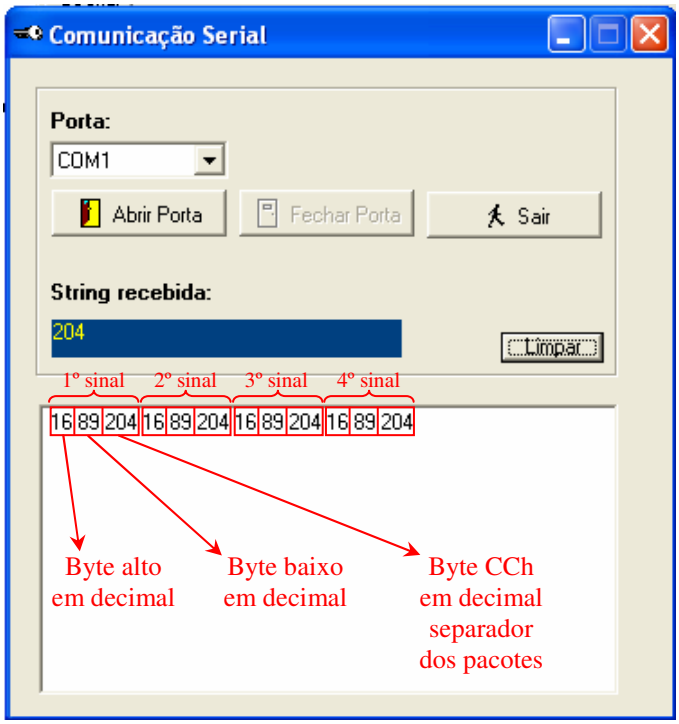


FIGURA 4.18 – TELA DO PROGRAMA EM EXECUÇÃO (TESTE DO SONAR) (PARTE 2)

A Tabela 4.11 apresenta outros resultados obtidos e comparados com as distâncias medidas.

TABELA 4.11 – RESULTADOS OBTIDOS NO TESTE DO SONAR (PARTE 2)

Distância Medida (cm)	**TCNT1H [39] (Decimal)**	**TCNT1L [40] (Decimal)**	**TCNT1 (Hexadecimal)**	**TCNT1 (Decimal)**	**Tempo de Vôo (ms)**	**Distância Calculada (cm)**
2,5	4	125	047D	1149	0,29	4,93
5,0	6	105	0669	1641	0,41	7,04
10,0	11	197	0BC5	3013	0,75	12,93
15,0	16	89	1059	4185	1,05	17,96
20,0	20	173	14AD	5293	1,32	22,72
25,0	25	161	19A1	6561	1,64	28,16
30,0	30	133	1E85	7813	1,95	33,54
40,0	39	25	2719	10009	2,50	42,97
50,0	48	241	30F1	12529	3,13	53,78
60,0	57	53	3935	14645	3,66	62,87
70,0	67	117	4375	17269	4,32	74,13

[39] TCNT1H – "*Timer Counter 1 High*" (Byte mais significativo do registrador de 16 bits)

[40] TCNT1L – "*Timer Counter 1 Low*" (Byte mais significativo do registrador de 16 bits)

Distância Medida (cm)	TCNT1H [39] (Decimal)	TCNT1L [40] (Decimal)	TCNT1 (Hexadecimal)	TCNT1 (Decimal)	Tempo de Vôo (ms)	Distância Calculada (cm)
80,0	75	245	4BF5	19445	4,86	83,47
130,0	119	89	7759	30553	7,64	131,15
150,0	138	245	8AF5	35573	8,89	152,70

Fazendo uma rápida análise dos resultados apresentados na Tabela 4.11, a distância calculada apresenta sempre um valor maior que a distância real medida, logo pode ser concluído que uma calibração pode ser realizada. Dessa forma, fez-se a seguinte análise:

- determinando-se a diferença entre a distância medida e a distância calculada, calculou-se a média desses valores (2,92cm), e por último, subtraiu-se o valor da média de cada um dos valores de distância calculada. Conforme apresentado na Tabela 4.12 e ilustrado graficamente na Figura 4.19. Pode ser observado que a diferença diminui consideralmente, mostrando um sistema muito mais preciso.

TABELA 4.12 – CALIBRAGEM DOS RESULTADOS OBTIDOS NO TESTE DO SONAR (PARTE 2)

Distância Medida (cm)	Distância Calculada (cm)	Distância Medida - Calculada (cm)	Distância Calculada - Média (2,92cm)	Distância Medida - Calculada "Calibrada" (cm)
2,5	4,93	2,43	2,01	-0,49
5,0	7,04	2,04	4,13	-0,87
10,0	12,93	2,93	10,01	0,01
15,0	17,96	2,96	15,05	0,05
20,0	22,72	2,72	19,80	-0,20
25,0	28,16	3,16	25,24	0,24
30,0	33,54	3,54	30,62	0,62
40,0	42,97	2,97	40,05	0,05
50,0	53,78	3,78	50,86	0,86
60,0	62,87	2,87	59,95	-0,05
70,0	74,13	4,13	71,21	1,21
80,0	83,47	3,47	80,55	0,55
130,0	131,15	1,15	128,23	-1,77
150,0	152,70	2,70	149,78	-0,22

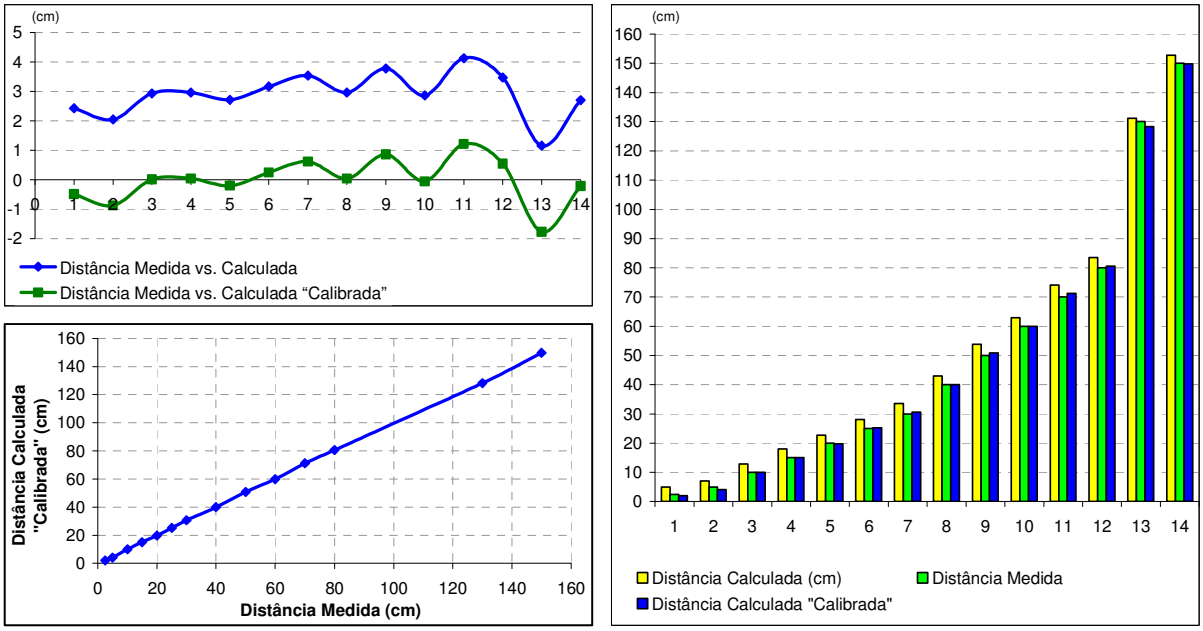


FIGURA 4.19 – DISTÂNCIA MEDIDA VS. DISTÂNCIA CALCULADA E CALIBRADA (GRÁFICOS)

Com base nesses novos valores obtidos, e sabendo-se que 2,92cm equivale a um “tempo de vôo” da onda sonora do sensor de ultrasom de aproximadamente 0,17ms, conclui-se que o sistema pode ser calibrado desconsiderando esse tempo adicional de 0,17ms do “tempo total de vôo” calculado para cada amostragem, tornando assim, os resultados apresentados pelo sistema mais precisos.

4.5.3.3 Teste de Funcionamento do Sensor de Velocidade

Esse teste teve como objetivo, testar o funcionamento dos sensores de velocidade.

Para que seja possível mapear o caminho percorrido pelo robô, é necessário que o sistema seja capaz de mensurar a quantidade de vezes que cada roda rotacionou e para isso utilizou-se sensores de velocidade com a capacidade de detectar a presença de metais imantados.

A Figura 4.20 apresenta o circuito elétrico que corresponde ao teste realizado.

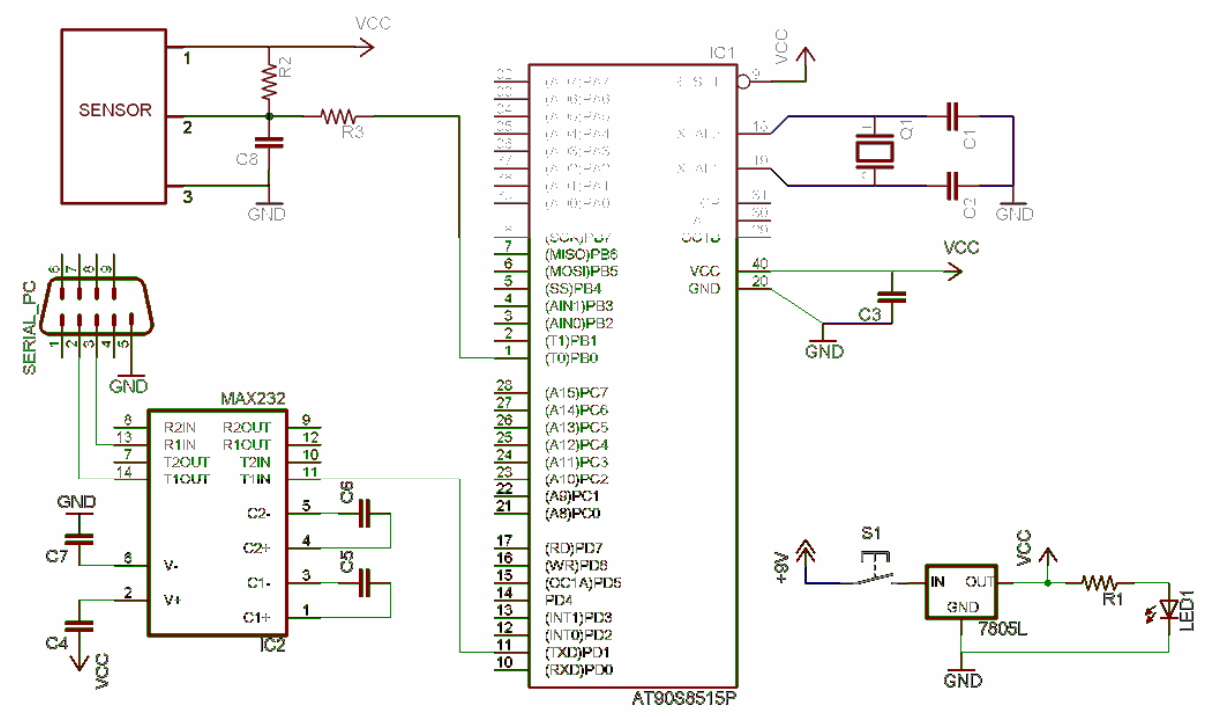


FIGURA 4.20 – DIAGRAMA ELÉTRICO DO CIRCUITO DE ODOMETRIA

A Figura 4.21 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador.

Através do fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.11.

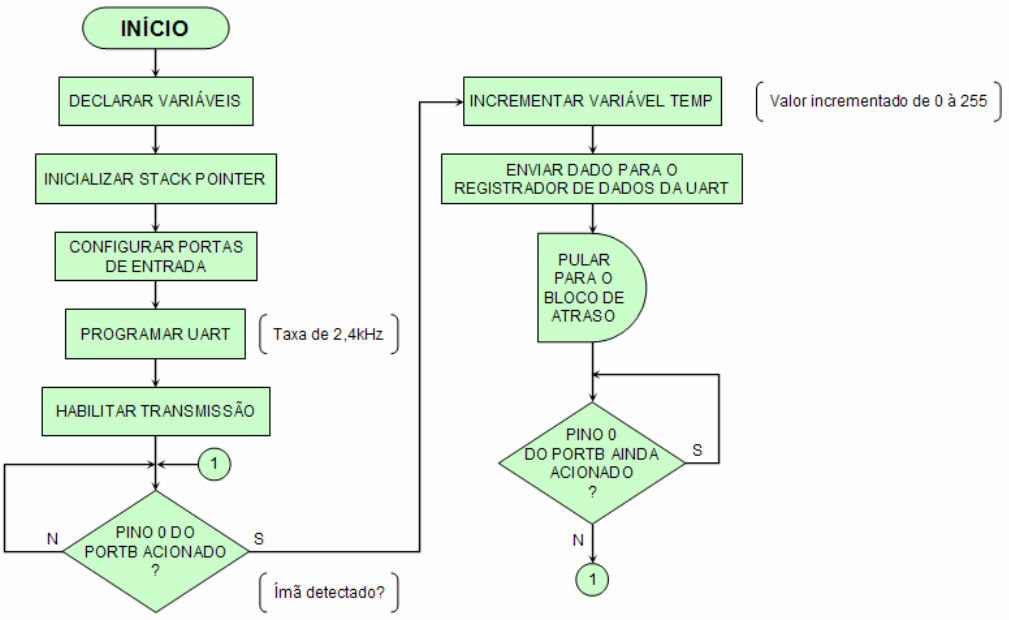


FIGURA 4.21 – FLUXOGRAMA DA LÓGICA DE PROGRAMAÇÃO UTILIZADA

A lógica de programação utilizada para a interface com o usuário, será melhor detalhada no Capítulo 7.3.3 (Comunicação Serial – Parte 3). A Figura 4.22 ilustra a tela do programa em execução.

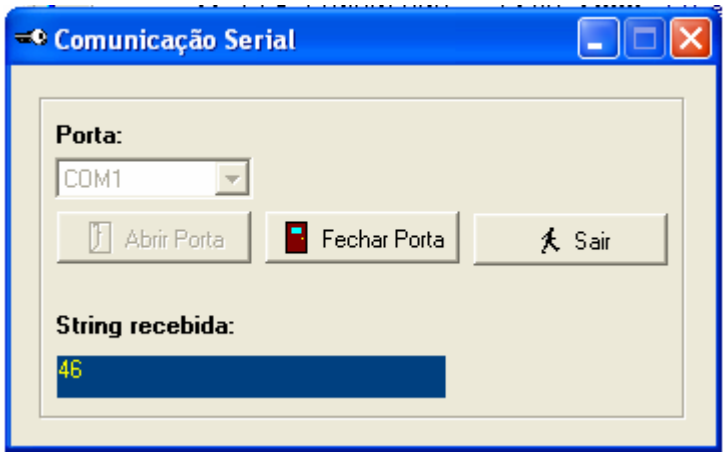


FIGURA 4.22 – TELA DO PROGRAMA DESENVOLVIDO EM EXECUÇÃO

É possível observar na Figura 4.22 a seqüência de números recebidos, que representam a quantidade de vezes (46) que o sensor detectou a presença de um metal imantado.

Os sensores apresentaram um funcionamento satisfatório e conforme o esperado, com base nos resultados obtidos, verificou-se que será possível adicionar ao robô um sistema de odometria e dessa forma, o ambiente percorrido pelo robô poderá ser mapeado.

5 ODOMETRIA

Uma das principais tarefas a serem desenvolvidas em um sistema de navegação de um robô móvel é a determinação da localização do robô, ou seja, a sua posição e orientação no ambiente analisado.

Conforme ilustrado na Figura 5.1, o modelo utilizado é conhecido como plataforma robótica com acionamento diferencial, ou seja, caracterizada pelo uso de duas rodas colocadas nas laterais do robô que são acionadas individualmente por dois motores independentes.

Para se determinar a localização desse tipo de plataforma, é necessário obter a posição do seu ponto central (x, y) e o ângulo θ de orientação do robô em relação a um referencial fixo no espaço analisado.

Sabendo-se que:

- L = comprimento do eixo;
- r_e, r_d = raios das rodas esquerda e direita, respectivamente;
- ω_e, ω_d = velocidades angulares das rodas esquerda e direita, respectivamente;
- v_e, v_d = velocidades lineares das rodas esquerda e direita, respectivamente;
- v = velocidade linear do robô;
- ω = velocidade angular do robô.

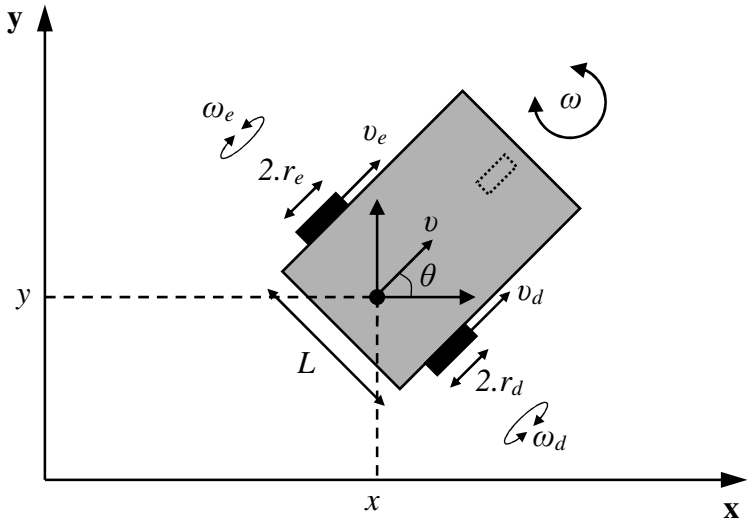


FIGURA 5.1 – ILUSTRAÇÃO DA PLATAFORMA ROBÓTICA UTILIZADA

O método de odometria consiste na determinação da posição e orientação do robô através da integração dos deslocamentos incrementais de suas rodas, medidos a partir de um referencial fixo.

O deslocamento do robô é medido por meio de sensores acoplados nas rodas com a capacidade de medir a rotação das rodas. Um tipo de sensor bastante utilizado nesse tipo de aplicação, conforme já explicado no capítulo anterior, é o encoder ótico, que funciona com base na transmissão e recepção de luz através de um disco perfurado, que gira acoplado ao eixo que movimenta a roda do robô.

O sensor utilizado no projeto (detalhado no Capítulo 4), é um sensor integrado de velocidade rotacional, dotado da capacidade de detectar referências magnéticas, basicamente, a freqüência da tensão do sinal de saída é proporcional a velocidade rotacional da roda [30]. Para a plataforma robótica apresentada na figura 5.1, as medidas de odometria são feitas a partir de dois sensores de velocidade acoplados a cada um dos motores, os quais permitem medir as rotações da roda correspondente.

A partir dos dados coletados dos sensores, conhecendo-se alguns parâmetros cinemáticos do robô e sabendo-se a localização inicial do robô, é possível deduzir as equações [31] que irão determinar a sua posição e orientação em um dado instante.

Considerando o movimento executado pelo robô semelhante a um arco de circunferência, conforme apresentado na Figura 5.2, e sabendo-se que *r* é o raio de giro do robô, teremos que a velocidade linear de cada roda e do robô, assim como a sua velocidade angular, serão:

$$v_d = \omega \cdot \left(r + \frac{L}{2} \right) \quad \rightarrow \quad \text{velocidade linear da roda direita.}$$

$$v_e = \omega \cdot \left(r - \frac{L}{2} \right) \quad \rightarrow \quad \text{velocidade linear da roda esquerda.}$$

$$v = \omega \cdot r = \frac{v_d + v_e}{2} \quad \rightarrow \quad \text{velocidade linear do robô.}$$

$$\omega = \frac{v}{r} = \frac{v_d - v_e}{L} \quad \rightarrow \quad \text{velocidade angular do robô.}$$

Como pode ser observado na Figura 5.2, considerando-se que o movimento executado pelo robô ocorreu em um intervalo de tempo infinitesimal *dt*, é possível decompor o deslocamento linear do robô, *dl* nas suas componentes horizontal (*dx*) e vertical (*dy*).

Discretizando as equações dessas componentes do deslocamento linear, do deslocamento angular $d\theta$ e das velocidades lineares de cada roda do robô, através do método de integração de Euler, obtém-se:

$$dx = dl \cdot \cos\theta = \upsilon \cdot dt \cdot \cos\theta \quad \rightarrow \quad x(t+\Delta t) = x(t) + \upsilon \cdot \Delta t \cdot \cos\theta(t)$$

$$dy = dl \cdot sen\theta = \upsilon \cdot dt \cdot sen\theta \quad \rightarrow \quad y(t+\Delta t) = y(t) + \upsilon \cdot \Delta t \cdot sen\theta(t)$$

$$d\theta = \omega \cdot dt \quad \rightarrow \quad \theta(t+\Delta t) = \theta(t) + \omega \cdot \Delta t$$

$$\upsilon_d = \frac{dl_d}{dt};\ \upsilon_e = \frac{dl_e}{dt} \quad \rightarrow \quad \frac{\Delta l_d}{\Delta t};\ \frac{\Delta l_e}{\Delta t}$$

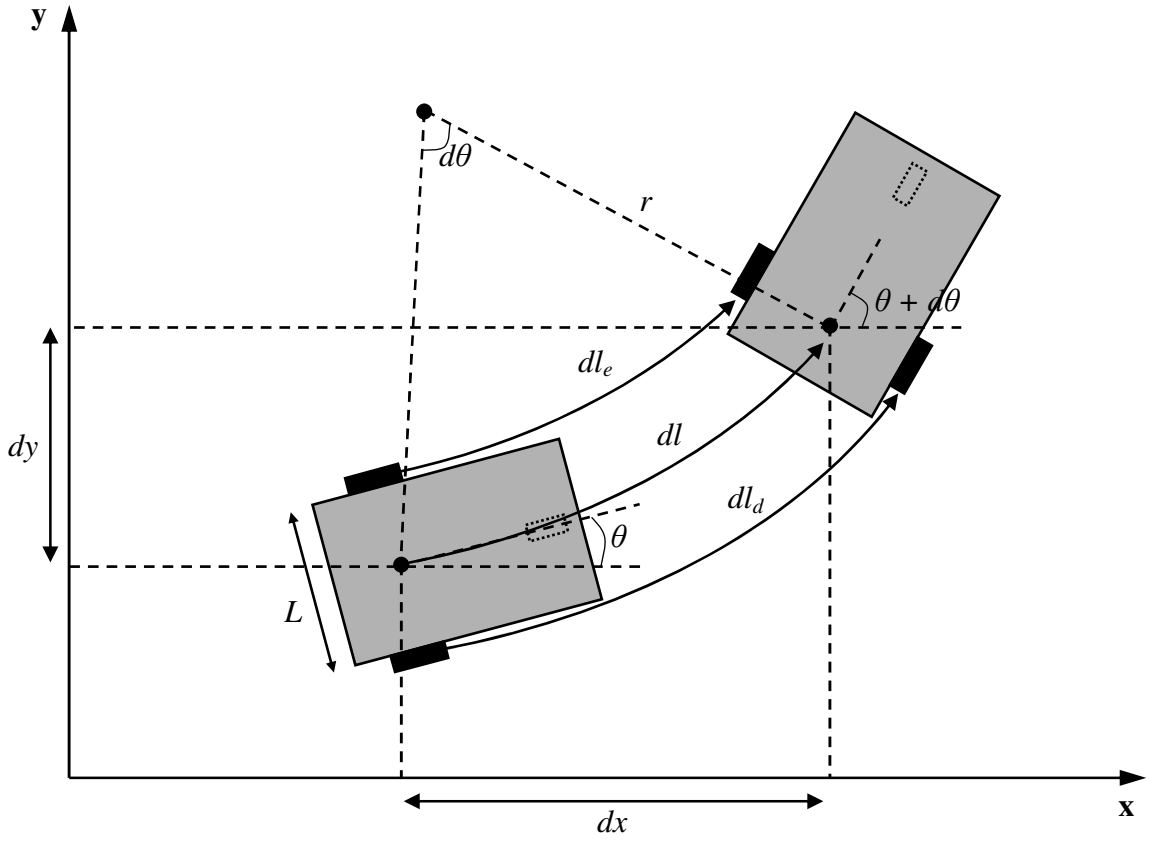


FIGURA 5.2 – ILUSTRAÇÃO DO MOVIMENTO INFINITESIMAL DO ROBÔ

Considerando N_d e N_e o número de vezes que os sensores de velocidade do lado direiro e esquerdo respectivamente, detectaram um marco magnético e N_{res} a quantidade total de marcos magnéticos localizados em cada uma das rodas, é possível obter o deslocamento de cada roda do robô, como sendo:

$$\Delta l_d = \frac{N_d}{N_{res}} \cdot 2 \cdot \pi \cdot r_d \qquad \rightarrow \quad \text{deslocamento da roda do lado direito.}$$

$$\Delta l_e = \frac{N_e}{N_{res}} \cdot 2 \cdot \pi \cdot r_e \qquad \rightarrow \quad \text{deslocamento da roda do lado esquerdo.}$$

Substituindo essas duas últimas equações às equações da velocidade linear de cada roda, obtém-se:

$$\upsilon_d = \frac{N_d}{N_{res}} \cdot \frac{2 \cdot \pi \cdot r_d}{\Delta t} \qquad \rightarrow \quad \text{velocidade linear da roda direita.}$$

$$\upsilon_e = \frac{N_e}{N_{res}} \cdot \frac{2 \cdot \pi \cdot r_e}{\Delta t} \qquad \rightarrow \quad \text{velocidade linear da roda esquerda.}$$

Com base nessas duas últimas equações, e substituindo-as às equações anteriormente apresentadas de deslocamento linear e angular, é possível obter a localização do robô em relação a um referencial inicialmente conhecido, através das seguintes equações:

$$x(t+\Delta t) = x(t) + (N_d \cdot r_d + N_e \cdot r_e) \cdot \frac{\pi}{N_{res}} \cdot \cos\theta(t)$$

$$y(t+\Delta t) = y(t) + (N_d \cdot r_d + N_e \cdot r_e) \cdot \frac{\pi}{N_{res}} \cdot sen\,\theta(t)$$

$$\theta(t+\Delta t) = \theta(t) + \frac{2 \cdot \pi}{L \cdot N_{res}} \cdot (N_d \cdot r_d - N_e \cdot r_e)$$

5.1 FONTES DE ERRO NA ODOMETRIA

Apesar de bastante utilizado, a odometria não é um método exato para determinar a localização de um robô móvel em um determinado espaço de tempo, este método apresenta erros que podem ser classificados de duas formas [32]:

- Erros sistemáticos: originados de características do robô e/ou sensores, por exemplo:
  - Diâmetro das rodas diferentes;
  - Desalinhamento das rodas;

- Incerteza sobre o ponto de contato da roda;
- Resolução finita dos sensores de velocidade;
- Comprimento do eixo diferente do comprimento nominal.

- Erros não sistemáticos: originados de características da relação do robô com o ambiente, por exemplo:
  - Movimento sobre solos não uniformes (rugosos);
  - Movimento sobre obstáculos inesperados no solo;
  - Escorregamento das rodas (solo escorregadio, aceleração acentuada, rotações rápidas).

Os erros sistemáticos, são causados por incertezas nos parâmetros do modelo cinemático do robô (por exemplo, comprimento do eixo e diâmetro das rodas), dessa forma, são graves, pois são uma fonte constante de erros aditivos para o sistema.

Os erros não sistemáticos são imprevisíveis, pois independem das características físicas do robô, mas sim do ambiente em que ele se encontra (por exemplo, imperfeições no solo e obstáculos inesperados), ao contrário dos erros sistemáticos, esse tipo de erro não ocorre durante toda a navegação do robô.

Através de alguns procedimentos de calibração [33], é possível reduzir os erros de odometria, entretanto, nenhum deles foram aplicados nesse trabalho.

5.2 CÁLCULOS E TESTES

Como explicado anteriormente, para iniciar os cálculos para a definição da posição do robô, é necessário o conhecimento dos seguintes parâmetros:

- Posição inicial do robô: $x(0)$, $y(0)$ e $\theta(0)$;
- Número de pulsos lidos em cada sensor de velocidade: N_d e N_e;
- Comprimento do eixo do robô: L;
- Raios de cada roda: r_d e r_e;
- Resolução dos sensores, que é o número de pulsos lidos em uma revolução das rodas: N_{res}.

Apenas como exemplo, considera-se que:

- $x(0) = y(0) = 0$; $\theta(0) = 0$;
- $L = 10cm$;
- $r_d = r_e = 3,2cm$;
- $N_{res} = 2$.

Sabendo-se que a quantidade de pulsos por revolução das rodas é igual a 2, que o raio de cada roda é de 3,2cm, pode-se concluir que a precisão do sistema é cerca de 10,05cm, que é o menor deslocamento mensurável pelos sensores de velocidade. O objetivo para este teste não é ter precisão, mas apenas um meio hipotético para mensurar o caminho percorrido pelo robô, considerando-se uma resolução relativamente baixa.

Com base nos valores apresentados, e utilizando-se das equações de odometria, conforme apresentado anteriormente, é possível estimar a posição atual do robô, variando-se apenas o número de pulsos lidos em cada sensor de velocidade (N_d e N_e).

A Tabela 5.1, procura ilustrar o que foi explicado acima.

TABELA 5.1 – CÁLCULO DA POSIÇÃO E DIREÇÃO DO ROBÔ

Parâmetro	**Unidade**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
N_d	[pulsos]	5,0	1,0	3,0	1,0	5,0	1,0	3,0	1,0	5,0
N_e	[pulsos]	5,0	0,6	3,0	0,6	5,0	0,6	3,0	0,6	5,0
V_d	[%]	100	100	100	100	100	100	100	100	100
V_e	[%]	100	61	100	61	100	61	100	61	100
V_d	[decimal]	0	0	0	0	0	0	0	0	0
V_e	[decimal]	0	100	0	100	0	100	0	100	0
$\boldsymbol{\theta}$	[rad]	0	1,57	1,57	3,14	3,14	4,71	4,71	6,28	6,28
$\boldsymbol{\theta}$	**[º]**	0	90	90	180	180	270	270	360	360
x	[cm]	50,27	55,04	55,04	49,53	-0,74	-5,51	-5,51	0,00	50,27
y	[cm]	0,00	5,51	35,67	40,44	40,44	34,93	4,77	0,00	0,00

Os parâmetros V_d e V_e apresentados na Tabela 5.1, estão relacionados com os parâmetros do PWM ("*Pulse Width Modulation*") para o controle de velocidade dos motores, onde 0 (100%), significa velocidade máxima de rotação e 255 (0%), significa que o motor está parado. No exemplo em questão, quando V_d está em 100%, e V_e está 61% da velocidade total, significa que o robô está fazendo uma curva para à esquerda, que teoricamente, quando o sensor de velocidade do motor do lado direito detectar o primeiro marco magnético (pulso),

o robô deve estar a aproximadamente 90º em relação a posição inicial. Maiores detalhes sobre o PWM, será abordado no Capítulo 6.3.1.

O gráfico apresentado na Figura 5.3 (destacado em vermelho), ilustra o caminho teoricamente percorrido pelo robô com base nos valores apresentados na Tabela 5.1.

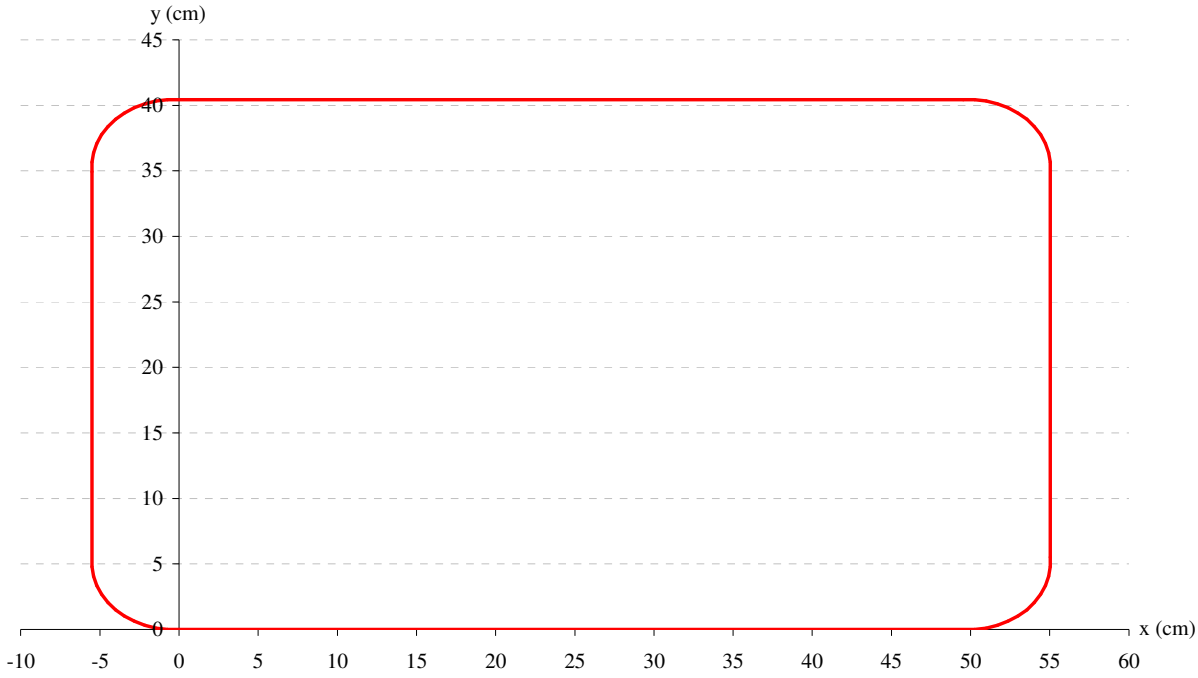


FIGURA 5.3 – ILUSTRAÇÃO DO CAMINHO TEÓRICO PERCORRIDO PELO ROBÔ

6 PROGRAMAÇÃO E SOFTWARE DE CONTROLE

O software de controle do robô, utilizado através da Interface Homem-Máquina (IHM) do projeto, foi desenvolvido em Delphi® [34].

Logo abaixo é apresentada uma breve descrição de cada um dos principais módulos criados durante o desenvolvimento deste software.

- Comunicação serial: para acessar a porta serial do microcomputador, foram utilizadas as funções da API do Windows. Na listagem do programa apresentada no Anexo 10.1.12, podem ser observados os comandos e conceitos adotados para executar essa comunicação.
- Parâmetros de ajustes do robô: são ajustes necessários que podem variar de acordo com as características físicas do robô e das condições do meio ambiente.
- Controle dos motores: ferramenta de controle independente para cada um dos motores. Conceito básico do controle utilizado.
- Leitura do sonar: descrição dos dados recebidos do robô com base na leitura do sensor de ultrasom.
- Interface gráfica: cálculos utilizados para representação gráfica do caminho percorrido pelo robô, sendo:
    - caminho aleatório: caminho desejado que seja percorrido pelo robô;
    - caminhos pré-definidos: testes realizados com base em alguns caminhos pré-definidos;
    - caminho aproximado: caminho possível de ser percorrido pelo robô, limitado às características físicas do robô;
    - entrada manual: teste da interface gráfica.
- Cálculo dos dados recebidos e enviados: descrição detalhada dos cálculos executados para o compilamento dos dados enviados e recebidos do robô.

A Figura 6.1 apresenta a tela do software desenvolvido utilizado para realizar o controle remoto do robô. Nesta figura, além dos principais módulos descritos acima, é possível observar também, que o programa informa o tempo decorrido entre o comando enviado e a equivalente resposta do robô (rodapé).

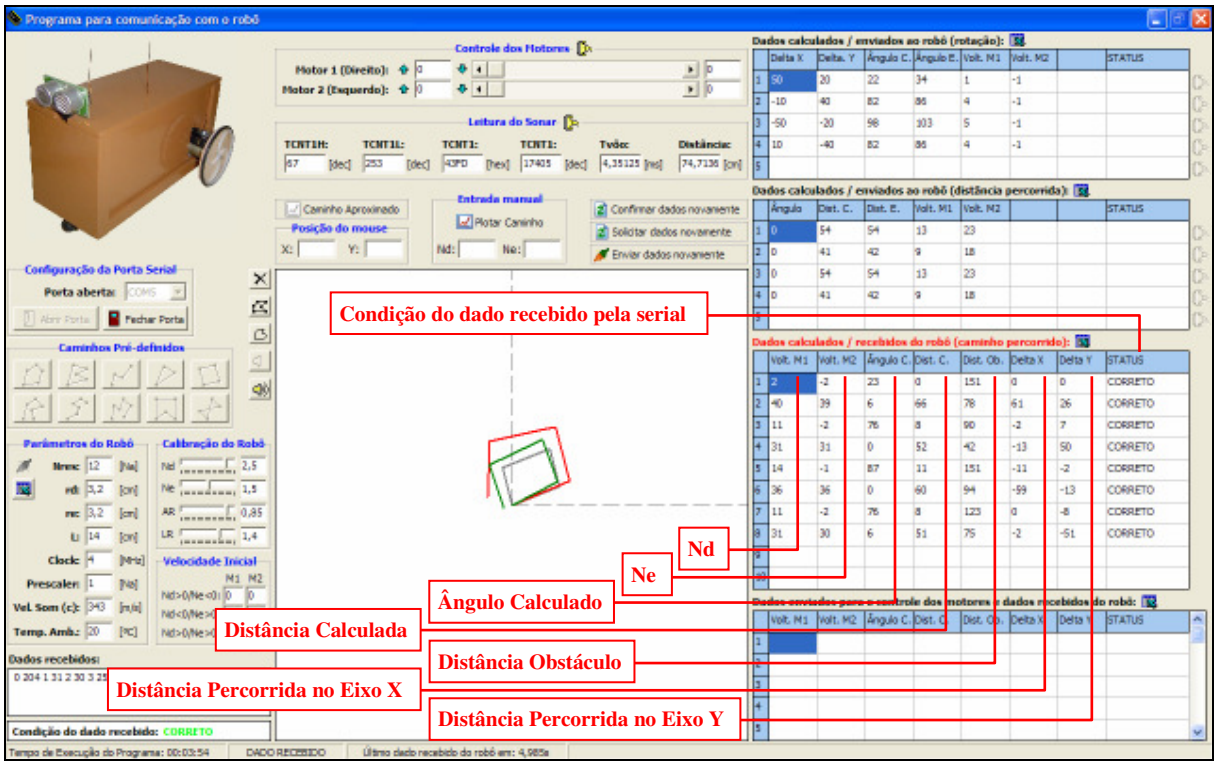


FIGURA 6.1 – IMAGEM DO SOFTWARE DE CONTROLE DESENVOLVIDO

A seguir serão abordados em maiores detalhes os principais módulos de controle utilizados neste software.

6.1 COMUNICAÇÃO SERIAL

A comunicação serial entre o microcomputador e o módulo de interface homem-máquina, foi realizada utilizando-se as funções da API do Windows.

Durante o desenvolvimento do trabalho, notou-se alguns erros no recebimento dos dados enviados do circuito de IHM para o microcomputador.

Esses erros acontecem com maior freqüência quando a porta serial do windows está em uso pelo sistema e neste instante o circuito de IHM é desligado e ligado novamente sem que a porta serial do windows seja fechada. O correto seria antes de desligar o circuito de IHM, fechar a porta serial (software) e somente após o circuito de IHM estiver ligado é que a porta serial deve ser aberta novamente.

Na Tabela 6.1, são apresentadas 50 leituras seqüenciais feitas pela porta serial, de acordo com o dado enviado pelo circuito de IHM.

TABELA 6.1 – DADOS CORRIGIDOS – RECEBIDOS PELA PORTA SERIAL

#	REFERÊNCIAS (ERROS)		INFORMAÇÕES RECEBIDAS QUE CONTÉM O DADO ENVIADO PELO ROBÔ								REFERÊNCIA UTILIZADA	DADO CORRIGIDO								NOMENCLATURA UTILIZADA
	1	2	1	2	3	4	5	6	7	8		1	2	3	4	5	6	7	8	
1	0	0	1	0	2	5	3	190	4	0	0	1	0	2	5	3	190	4	0	CORRETO
2	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-872415232	1	0	2	5	3	11	4	0	ERRO LEITURA
3	13369345	13369345	13369345	13369344	13369346	13369349	13369347	13369356	13369348	13369344	13369344	1	0	2	5	3	12	4	0	ERRO ZERO
4	0	0	1	0	2	5	3	211	4	0	0	1	0	2	5	3	211	4	0	CORRETO
5	0	0	1	0	2	5	3	207	4	0	0	1	0	2	5	3	207	4	0	CORRETO
6	0	0	1	0	2	5	3	103	4	0	0	1	0	2	5	3	103	4	0	CORRETO
7	-20122624	-20122624	-20122623	-20122624	-20122622	-20122619	-20122621	-20122601	-20122620	-20122624	-20122624	1	0	2	5	3	23	4	0	ERRO LEITURA
8	65536	65536	65537	65536	65538	65541	65539	65763	65540	65536	65536	1	0	2	5	3	227	4	0	ERRO LEITURA
9	0	204	1	0	2	5	3	97	4	0	0	1	0	2	5	3	97	4	0	CORRETO
10	0	0	1	0	2	5	3	80	4	0	0	1	0	2	5	3	80	4	0	CORRETO
11	33488896	33488896	33488897	33488896	33488898	33488901	33488899	33489063	33488900	33488896	33488896	1	0	2	5	3	167	4	0	ERRO LEITURA
12	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-872351744	1	0	2	5	3	158	4	0	ERRO LEITURA
13	0	254	1	0	2	5	3	164	4	0	0	1	0	2	5	3	164	4	0	CORRETO
14	0	0	1	0	2	5	3	126	4	0	0	1	0	2	5	3	126	4	0	CORRETO
15	65024	65024	65025	65024	65026	65029	65027	65232	65028	65024	65024	1	0	2	5	3	208	4	0	ERRO LEITURA
16	0	254	1	0	2	5	3	204	4	0	0	1	0	2	5	3	204	4	0	CORRETO
17	0	0	1	0	2	5	3	71	4	0	0	1	0	2	5	3	71	4	0	CORRETO
18	0	0	1	0	2	5	3	24	4	0	0	1	0	2	5	3	24	4	0	CORRETO
19	16678912	16678912	16678913	16678912	16678914	16678917	16678915	16679115	16678916	16678912	16678912	1	0	2	5	3	203	4	0	ERRO LEITURA
20	16777216	16777420	16777217	16777216	16777218	16777221	16777219	16777341	16777220	16777216	16777216	1	0	2	5	3	125	4	0	ERRO LEITURA
21	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-2,1E+08	-210239488	1	0	2	5	3	145	4	0	ERRO ZERO
22	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-872415232	1	0	2	5	3	72	4	0	ERRO LEITURA
23	13369344	13369344	13369345	13369344	13369346	13369349	13369347	13369399	13369348	13369344	13369344	1	0	2	5	3	55	4	0	ERRO LEITURA
24	-25165569	-25165569	-25165823	-25165824	-25165822	-25165819	-25165821	-25165631	-25165820	-25165824	-25165824	1	0	2	5	3	193	4	0	ERRO -254
25	126976	126976	126977	126976	126978	126981	126979	127000	126980	126976	126976	1	0	2	5	3	24	4	0	ERRO LEITURA
26	16646144	16646144	16646145	16646144	16646146	16646149	16646147	16646180	16646148	16646144	16646144	1	0	2	5	3	36	4	0	ERRO LEITURA
27	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-872415232	1	0	2	5	3	8	4	0	ERRO LEITURA
28	16515073	16515073	16515073	16515072	16515074	16515077	16515075	16515283	16515076	16515072	16515072	1	0	2	5	3	211	4	0	ERRO ZERO
29	-16777216	-16777216	-16777215	-16777216	-16777214	-16777211	-16777213	-16777031	-16777212	-16777216	-16777216	1	0	2	5	3	185	4	0	ERRO LEITURA
30	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2,15E+09	-2147418368	1	0	2	5	3	141	4	0	ERRO LEITURA
31	-16776960	-16776960	-16776959	-16776960	-16776958	-16776955	-16776957	-16776899	-16776956	-16776960	-16776960	1	0	2	5	3	61	4	0	ERRO LEITURA
32	16711681	16711681	16711681	16711680	16711682	16711685	16711683	16711833	16711684	16711680	16711680	1	0	2	5	3	153	4	0	ERRO ZERO
33	0	0	1	0	2	5	3	50	4	0	0	1	0	2	5	3	50	4	0	CORRETO
34	16252929	16252929	16252929	16252928	16252930	16252933	16252931	16253027	16252932	16252928	16252928	1	0	2	5	3	99	4	0	ERRO ZERO
35	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-1,34E+08	-134217472	1	0	2	5	3	220	4	0	ERRO LEITURA
36	0	0	1	0	2	5	3	175	4	0	0	1	0	2	5	3	175	4	0	CORRETO
37	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-2,01E+08	-201326336	1	0	2	5	3	99	4	0	ERRO LEITURA
38	-33554432	-33554432	-33554431	-33554432	-33554430	-33554427	-33554429	-33554215	-33554428	-33554432	-33554432	1	0	2	5	3	217	4	0	ERRO LEITURA
39	0	96	1	0	2	5	3	233	4	0	0	1	0	2	5	3	233	4	0	CORRETO
40	0	0	1	0	2	5	3	184	4	0	0	1	0	2	5	3	184	4	0	CORRETO
41	16662272	16662272	16662273	16662272	16662274	16662277	16662275	16662335	16662276	16662272	16662272	1	0	2	5	3	63	4	0	ERRO LEITURA
42	0	0	1	0	2	5	3	96	4	0	0	1	0	2	5	3	96	4	0	CORRETO
43	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-872415232	1	0	2	5	3	82	4	0	ERRO LEITURA
44	0	254	1	0	2	5	3	117	4	0	0	1	0	2	5	3	117	4	0	CORRETO
45	52224	52224	52225	52224	52226	52229	52227	52318	52228	52224	52224	1	0	2	5	3	94	4	0	ERRO LEITURA
46	16646399	16646399	16646145	16646144	16646146	16646149	16646147	16646239	16646148	16646144	16646144	1	0	2	5	3	95	4	0	ERRO -254
47	-25037824	-25037824	-25037823	-25037824	-25037822	-25037819	-25037821	-25037620	-25037820	-25037824	-25037824	1	0	2	5	3	204	4	0	ERRO LEITURA
48	-16777216	-16777216	-16777215	-16777216	-16777214	-16777211	-16777213	-16777106	-16777212	-16777216	-16777216	1	0	2	5	3	110	4	0	ERRO LEITURA
49	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-8,72E+08	-872351744	1	0	2	5	3	127	4	0	ERRO LEITURA
50	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1,64E+09	1644295168	1	0	2	5	3	158	4	0	ERRO LEITURA

Quando existe algum erro na comunicação serial, os dois últimos valores recebidos antes do envio dos dados pelo circuito de IHM ao microcomputador, são utilizados como referência para identificar o tipo de erro encontrado e assim corrigí-lo.

Abaixo é feita uma breve explicação sobre os três tipos de erros que foram identificados ao longo do desenvolvimento do trabalho.

- ERRO LEITURA: esse é o erro mais comum que acontece. Os valores recebidos são somados a um número de referência aleatório do sistema, para corrigir esse erro e obter a informação correta enviada pelo circuito de IHM, basta subtrair os dados recebidos por essa referência.
- ERRO ZERO: para corrigir esse erro, é necessário subtrair 1 do valor de referência e esse resultado ser subtraído dos dados recebidos. A nomenclatura utilizada como “ERRO ZERO” é devido ao fato de que se subtrairmos o valor

da referência pelo primeiro dado enviado pelo circuito de IHM, a diferença encontrada será igual a zero.

- ERRO -254: esse é um tipo de erro mais difícil de acontecer, para corrigí-lo, é necessário subtrair 255 do valor de referência e esse resultado ser subtraído dos dados recebidos. Nesse caso, a nomenclatura utilizada como "ERRO -254" é devido ao fato de que se subtrairmos o valor da referência pelo primeiro dado enviado pelo circuito de IHM, a diferença encontrada será de -254.

Para poder identificar os dados recebidos no meio dos erros apresentados pela comunicação serial, quando este ocorrer, foi incorporado no protocolo de comunicação entre o circuito de IHM e o microcomputador uma seqüência numérica de 1 a 4, a Figura 6.2 procura ilustrar o que está sendo explicado.

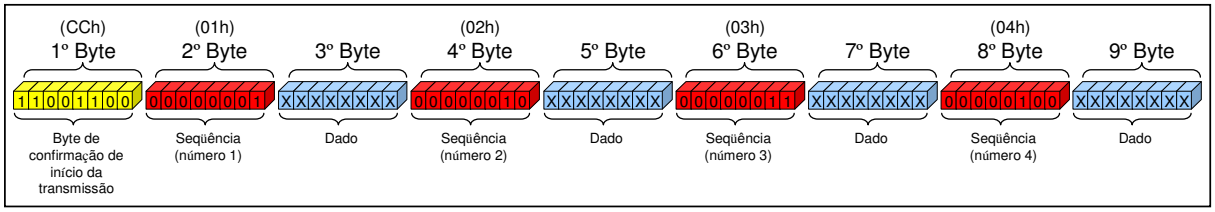


FIGURA 6.2 – SEQÜÊNCIA DE DADOS RECEBIDOS PELO PC

Sabendo-se que a seqüência de dados será conforme apresentado na Figura 6.2, é possível identificar o momento exato em que o dado está sendo recebido pela porta serial e se este dado possui ou não um dos erros apresentados anteriormente. Maiores detalhes com relação ao protocolo de comunicação utilizado entre o circuito de IHM e o microcomputador, será abordado no Capítulo 7.

Na Figura 6.3 é apresentado o fluxograma que procura ilustrar o conceito aplicado na programação para identificar essa seqüência de dados e corrigí-la quando for necessário.

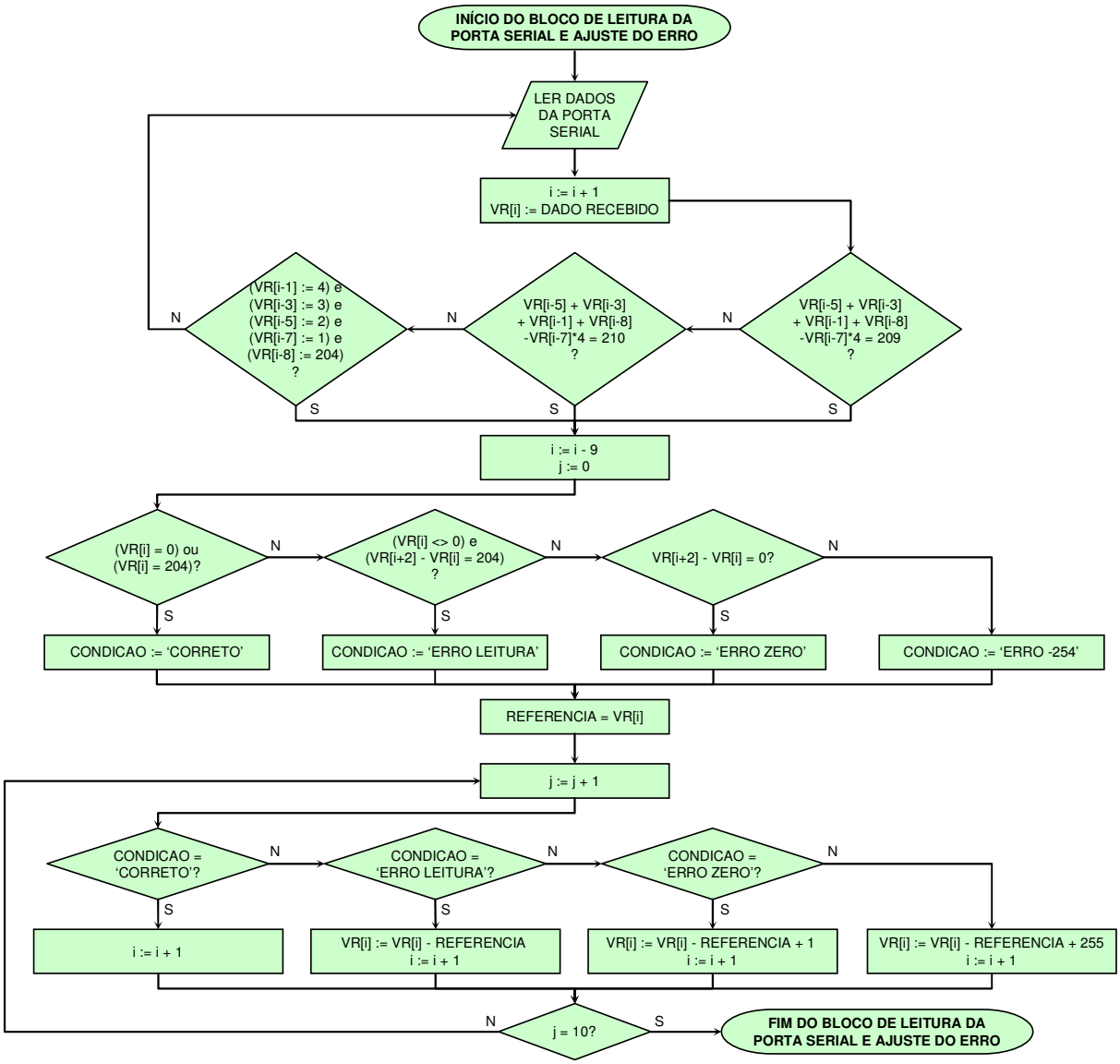


FIGURA 6.3 – FLUXOGRAMA DA LEITURA DA PORTA SERIAL E AJUSTE DO ERRO

6.2 PARÂMETROS DE AJUSTE DO ROBÔ

Na Figura 6.4, são mostrados os parâmetros que podem ser ajustados para a execução correta dos cálculos e medições com relação aos movimentos executados pelo robô.

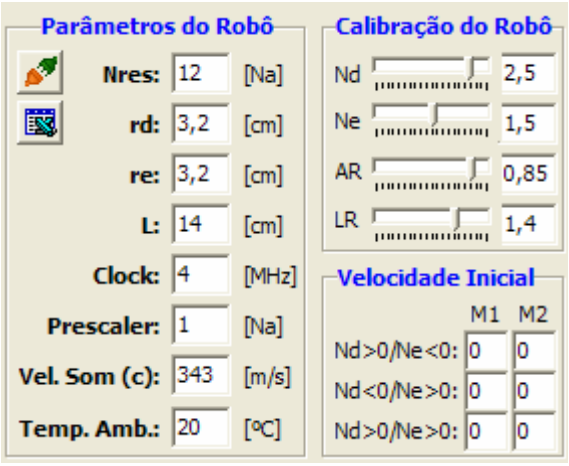


FIGURA 6.4 – PARÂMETROS DE AJUSTE DO ROBÔ

Segue abaixo uma breve descrição de cada um desses parâmetros:

- *Nres*: resolução dos sensores, que é o número de pulsos lidos em uma revolução das rodas;
- *rd* e *re*: são os raios de cada roda;
- *L*: é o comprimento do eixo do robô;
- *Clock*: é a freqüência de oscilação do microcontrolador;
- *Prescaler*: parâmetro de ajuste, caso o cálculo de tempo registrado no microcontrolador não esteja acompanhando o clock do sistema, ou seja, 1:1;
- *Vel. Som (c)*: velocidade do som calculada com base na temperatura do ambiente de trabalho (*Temp. Amb*);
- Calibração do robô: ajuste necessário de acordo com o nível de tensão da fonte de alimentação do robô;
- Ajuste inicial de velocidade: é possível definir qual a velocidade inicial de rotação para cada um dos motores do robô;
- informações mais detalhadas sobre os últimos dois parâmetros, serão abordados no decorrer deste capítulo.

Para que a movimentação do robô possa ser mapeada e de uma certa forma, bem controlada, em termos de direção e distância percorrida em relação a um referencial fixo, o conceito utilizado foi o seguinte:

- primeiro define-se o ângulo entre o ponto de partida e o ponto de chegada (desejado), considerando a localização e a direção inicial previamente

conhecidas, utilizando-se dessas informações, o movimento executado pelo robô deve ser circular, de acordo com os valores previamente calculados, com base nos parâmetros de ajustes do robô;

- por último a distância entre os pontos é calculada e assim como no caso anterior, o movimento executado pelo robô, neste caso em linha reta, estará de acordo com os valores previamente calculados, com base nos parâmetros de ajustes do robô.

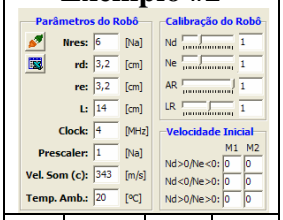
Nas Tabelas 6.2 e 6.3 são ilustrados alguns valores de ângulos calculados com base nos parâmetros de ajustes do robô, que são utilizados como referência para a definição do comando a ser enviado para o robô.

TABELA 6.2 – ÂNGULO CALCULADO VS. PARÂMETROS DE AJUSTES DO ROBÔ

Exemplo #1

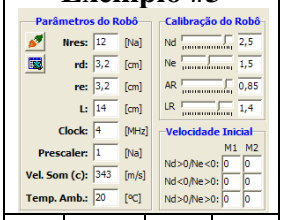
#	θ	Nd	Ne	#	θ	Nd	Ne
1	-178	-1	25	**26**	14	1	-1
2	-171	-1	24	**27**	21	2	-1
3	-165	-1	23	**28**	27	3	-1
4	-158	-1	22	**29**	34	4	-1
5	-151	-1	21	**30**	41	5	-1
6	-144	-1	20	**31**	48	6	-1
7	-137	-1	19	**32**	55	7	-1
8	-130	-1	18	**33**	62	8	-1
9	-123	-1	17	**34**	69	9	-1
10	-117	-1	16	**35**	75	10	-1
11	-110	-1	15	**36**	82	11	-1
12	-103	-1	14	**37**	89	12	-1
13	-96	-1	13	**38**	96	13	-1
14	-89	-1	12	**39**	103	14	-1
15	-82	-1	11	**40**	110	15	-1
16	-75	-1	10	**41**	117	16	-1
17	-69	-1	9	**42**	123	17	-1
18	-62	-1	8	**43**	130	18	-1
19	-55	-1	7	**44**	137	19	-1
20	-48	-1	6	**45**	144	20	-1
21	-41	-1	5	**46**	151	21	-1
22	-34	-1	4	**47**	158	22	-1
23	-27	-1	3	**48**	165	23	-1
24	-21	-1	2	**49**	171	24	-1
25	-14	-1	1	**50**	178	25	-1

Exemplo #2



#	θ	Nd	Ne
1	-178	-1	12
2	-165	-1	11
3	-151	-1	10
4	-137	-1	9
5	-123	-1	8
6	-110	-1	7
7	-96	-1	6
8	-82	-1	5
9	-69	-1	4
10	-55	-1	3
11	-41	-1	2
12	-27	-1	1
13	27	1	-1
14	41	2	-1
15	55	3	-1
16	69	4	-1
17	82	5	-1
18	96	6	-1
19	110	7	-1
20	123	8	-1
21	137	9	-1
22	151	10	-1
23	165	11	-1
24	178	12	-1

Exemplo #3



#	θ	Nd	Ne
1	-175	-1	16
2	-165	-1	15
3	-154	-1	14
4	-144	-1	13
5	-134	-1	12
6	-123	-1	11
7	-113	-1	10
8	-103	-1	9
9	-93	-1	8
10	-82	-1	7
11	-72	-1	6
12	-62	-1	5
13	-51	-1	4
14	-41	-1	3
15	-31	-1	2
16	-21	-1	1
17	34	1	-1
18	51	2	-1
19	69	3	-1
20	86	4	-1
21	103	5	-1
22	120	6	-1
23	137	7	-1
24	154	8	-1
25	171	9	-1

TABELA 6.3 – ÂNGULO CALCULADO VS. QUANTIDADE DE PULSOS LIDOS

Ne < 0 \ Nd > 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0	0	7	14	21	27	34	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178
-1	7	14	21	27	34	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185
-2	14	21	27	34	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192
-3	21	27	34	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199
-4	27	34	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206
-5	34	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213
-6	41	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219
-7	48	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226
-8	55	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233
-9	62	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240
-10	69	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247
-11	75	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254
-12	82	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261
-13	89	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267
-14	96	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274
-15	103	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281
-16	110	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288
-17	117	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295
-18	123	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302
-19	130	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309
-20	137	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315
-21	144	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315	322
-22	151	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315	322	329
-23	158	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315	322	329	336
-24	165	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315	322	329	336	343
-25	171	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315	322	329	336	343	350
-26	178	185	192	199	206	213	219	226	233	240	247	254	261	267	274	281	288	295	302	309	315	322	329	336	343	350	357

Ne > 0 \ Nd < 0	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24	-25	-26
0	0	-7	-14	-21	-27	-34	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178
1	-7	-14	-21	-27	-34	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185
2	-14	-21	-27	-34	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192
3	-21	-27	-34	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199
4	-27	-34	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206
5	-34	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213
6	-41	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219
7	-48	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226
8	-55	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233
9	-62	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240
10	-69	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247
11	-75	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254
12	-82	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261
13	-89	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267
14	-96	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274
15	-103	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281
16	-110	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288
17	-117	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295
18	-123	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302
19	-130	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309
20	-137	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315
21	-144	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315	-322
22	-151	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315	-322	-329
23	-158	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315	-322	-329	-336
24	-165	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315	-322	-329	-336	-343
25	-171	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315	-322	-329	-336	-343	-350
26	-178	-185	-192	-199	-206	-213	-219	-226	-233	-240	-247	-254	-261	-267	-274	-281	-288	-295	-302	-309	-315	-322	-329	-336	-343	-350	-357

Na Tabela 6.2, são apresentados três exemplos de configurações para os parâmetros de ajustes definidos para o robô, que resultam em seqüências diferenciadas de ângulos possíveis que podem ser atingidos no caminho percorrido pelo robô.

Os valores de *Nd* e *Ne* são a quantidade de pulsos lidos pelos sensores de velocidade, localizados nas rodas direita e esquerda respectivamente. Quando o valor de *Nd* ou *Ne* apresentado for um número negativo, significa que o sentido de giro do motor é o anti-horário, quando este número for positivo significa o oposto, ou seja, o sentido de giro do motor é o horário. No exemplo #1 da Tabela 6.2, apenas para ilustrar, para que o robô faça um giro de -178º, é necessário que a roda direita gire no sentido anti-horário até que 1 pulso seja detectado pelo sensor de velocidade e a roda esquerda precisa girar no sentido horário até que 25 pulsos sejam detectados pelo sensor de velocidade.

A Tabela 6.3 apresenta, de uma maneira mais abrangente, os ângulos possíveis que podem ser atingidos no caminho percorrido pelo robô, considerando os valores de $Nd > 0$ com $Ne < 0$ e $Nd < 0$ com $Ne > 0$ respectivamente, considerando a aplicação fiel dos parâmetros do robô, ou seja, sem ajustes de calibração, conforme exemplo #1 da Tabela 6.2.

O conceito de se manter praticamente uma roda parada e apenas a outra girando para se realizar a movimentação de direcionamento do robô, foi o que apresentou os melhores resultados e menores probabilidades de erro ao longo dos testes realizados, estes que serão apresentados posteriormente, ainda neste capítulo.

Um dos conceitos testados que poderia ser utilizado para esta função seria o de rotacionar os dois motores em sentidos opostos, na mesma velocidade e na mesma quantidade de pulsos, entretanto alguns problemas foram encontrados, conforme listado abaixo:

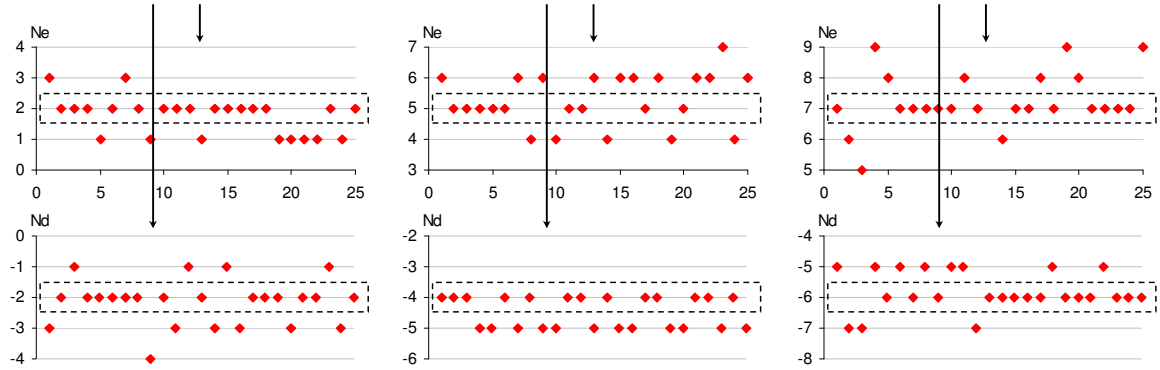
- velocidade de rotação dos motores diferenciada;
- dificuldade no ajuste de parada do robô (inércia);
- calibração variada em função do nível de tensão da bateria de alimentação do robô, dificultando a acertividade do comando executado com relação ao movimento desejado.

Ainda analisando a Tabela 6.2, o primeiro exemplo apresenta os valores de ângulos possíveis considerando a aplicação fiel dos parâmetros do robô, o segundo exemplo, apresenta os resultados caso a resolução dos sensores fosse reduzido pela metade, ou seja 6, nesse caso é possível observar que o passo entre os ângulos é o dobro quanto comparado ao exemplo anterior, e por último, o terceiro exemplo que apresenta um ajuste de calibração do robô, necessário devido ao caminho adicional percorrido pelo robô, mesmo após a execução do comando de desligamento dos motores, isso ocorre devido à inércia do robô.

Com relação ao fator de calibração do robô, este parâmetro foi obtido através de testes realizados, conforme alguns exemplos apresentados nas Tabelas 6.4 e 6.5.

TABELA 6.4 – TESTES DE VERIFICAÇÃO DO COMANDO ENVIADO VS. EXECUTADO PELO ROBÔ

	Enviado		Recebido		Enviado		Recebido		Enviado		Recebido		Enviado		Recebido		Enviado		Recebido		Enviado		Recebido	
	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne	Nd	Ne
1	-1	1	-3	3	1	-1	1	-2	-2	2	-4	6	2	-2	5	-5	-3	3	-5	7	3	-3	9	-5
2	-1	1	-2	2	1	-1	3	-3	-2	2	-4	5	2	-2	5	-4	-3	3	-7	6	3	-3	7	-6
3	-1	1	-1	2	1	-1	2	-4	-2	2	-4	5	2	-2	7	-4	-3	3	-7	5	3	-3	9	-5
4	-1	1	-2	2	1	-1	3	-2	-2	2	-5	5	2	-2	5	-4	-3	3	-5	9	3	-3	10	-6
5	-1	1	-2	1	1	-1	1	-3	-2	2	-5	5	2	-2	5	-5	-3	3	-6	8	3	-3	9	-5
6	-1	1	-2	2	1	-1	3	-2	-2	2	-4	5	2	-2	5	-5	-3	3	-5	7	3	-3	8	-5
7	-1	1	-2	3	1	-1	2	-4	-2	2	-5	6	2	-2	6	-3	-3	3	-6	7	3	-3	9	-7
8	-1	1	-2	2	1	-1	2	-4	-2	2	-4	4	2	-2	7	-4	-3	3	-5	7	3	-3	10	-5
9	-1	1	-4	1	1	-1	3	-3	-2	2	-5	6	2	-2	5	-7	-3	3	-6	7	3	-3	8	-5
10	-1	1	-2	2	1	-1	3	-2	-2	2	-5	4	2	-2	6	-3	-3	3	-5	7	3	-3	10	-6
11	-1	1	-3	2	1	-1	1	-3	-2	2	-4	5	2	-2	5	-3	-3	3	-5	8	3	-3	7	-5
12	-1	1	-1	2	1	-1	2	-2	-2	2	-4	5	2	-2	6	-4	-3	3	-7	7	3	-3	9	-5
13	-1	1	-2	1	1	-1	3	-3	-2	2	-5	6	2	-2	6	-4	-3	3	-6	10	3	-3	11	-6
14	-1	1	-3	2	1	-1	3	-2	-2	2	-4	4	2	-2	7	-6	-3	3	-6	6	3	-3	8	-4
15	-1	1	-1	2	1	-1	1	-3	-2	2	-5	6	2	-2	6	-5	-3	3	-6	7	3	-3	7	-6
16	-1	1	-3	2	1	-1	4	-2	-2	2	-5	6	2	-2	4	-3	-3	3	-6	7	3	-3	8	-6
17	-1	1	-2	2	1	-1	2	-2	-2	2	-4	5	2	-2	7	-4	-3	3	-6	8	3	-3	10	-4
18	-1	1	-2	2	1	-1	1	-2	-2	2	-4	6	2	-2	6	-5	-3	3	-5	7	3	-3	8	-5
19	-1	1	-2	1	1	-1	3	-2	-2	2	-5	4	2	-2	6	-4	-3	3	-6	9	3	-3	9	-5
20	-1	1	-3	1	1	-1	2	-2	-2	2	-5	5	2	-2	7	-3	-3	3	-6	8	3	-3	7	-5
21	-1	1	-2	1	1	-1	2	-4	-2	2	-4	6	2	-2	5	-4	-3	3	-6	7	3	-3	9	-6
22	-1	1	-2	1	1	-1	3	-3	-2	2	-4	6	2	-2	4	-6	-3	3	-5	7	3	-3	8	-6
23	-1	1	-1	2	1	-1	3	-3	-2	2	-5	7	2	-2	4	-3	-3	3	-6	7	3	-3	9	-4
24	-1	1	-3	1	1	-1	2	-4	-2	2	-4	4	2	-2	6	-4	-3	3	-6	7	3	-3	8	-6
25	-1	1	-2	2	1	-1	3	-2	-2	2	-5	6	2	-2	6	-4	-3	3	-6	9	3	-3	9	-6
	Média		-2,2	1,8	Média		2,3	-2,7	Média		-4,5	5,3	Média		5,6	-4,2	Média		-5,8	7,4	Média		8,6	-5,4



Foi solicitado ao robô que ele executasse 25 vezes o mesmo comando, e que esta seqüência de repetições acontecesse para 6 comandos diferentes.

O objetivo dos testes apresentados na Tabela 6.4 foi o de verificar se o comando executado pelo robô, independentemente de qual fosse, apresentasse um resultado constante. Nos três casos apresentados graficamente, existem pequenas dispersões em torno da média dos resultados obtidos, porém a maior incidência de resultados obtidos está na própria média. Conclui-se portanto, apenas como exemplo, que quando se desejar que o robô realize um giro de 27º, o comando que deve ser enviado para que ele execute algo próximo disso, deve ser o de *Nd = -1* e *Ne = 1*, sendo que o resultado obtido será algo em torno de *Nd = -2* e *Ne = 2*, equivalente a um ângulo de giro de 27º, conforme dados apresentados na Tabela 6.3.

Na Tabela 6.5, são apresentados os resultados de alguns testes que tiveram como objetivo confrontar o comando enviado para o robô versus o resultado do comando executado

pelo robô, versus o caminho real percorrido pelo robô. Para isso, o robô foi inicialmente posicionado sempre em uma mesma direção e comandos iguais foram enviados ao robô até que este completasse um giro de 360º, ou algo próximo disso.

TABELA 6.5 – TESTES DO COMANDO ENVIADO VS. EXECUTADO PELO ROBÔ (ÂNGULO DE GIRO)

Teste #1						
Enviado			Recebido			Real
Nd	Ne	θ	Nd	Ne	θ	θ
2	-1	21	4	-2	41	30
2	-1	21	4	-2	41	30
2	-1	21	4	-3	48	35
2	-1	21	4	-1	34	25
2	-1	21	4	-2	41	30
2	-1	21	4	-2	41	30
2	-1	21	3	-2	34	25
2	-1	21	3	-2	34	25
2	-1	21	4	-2	41	30
2	-1	21	5	-1	41	30
2	-1	21	4	-2	41	30
2	-1	21	5	-1	41	30
Total		252	Total		478	350
Média		21	Média		40	29
		190%				73%

Teste #2						
Enviado			Recebido			Real
Nd	Ne	θ	Nd	Ne	θ	θ
3	-1	27	6	-2	55	43
3	-1	27	6	-2	55	43
3	-1	27	5	-3	55	43
3	-1	27	6	-1	48	37
3	-1	27	5	-2	48	37
3	-1	27	6	-2	54	42
3	-1	27	7	-2	62	48
3	-1	27	7	-2	62	48
3	-1	27	6	-1	48	37
Total		243	Total		487	380
Média		27	Média		54	42
		200%				78%

Teste #3						
Enviado			Recebido			Real
Nd	Ne	θ	Nd	Ne	θ	θ
4	-1	34	7	-4	75	61
4	-1	34	9	-2	75	61
4	-1	34	8	-1	62	51
4	-1	34	8	-3	75	61
4	-1	34	7	-2	62	51
4	-1	34	7	-2	62	51
4	-1	34	7	-1	55	45
Total		238	Total		466	380
Média		34	Média		67	54
		196%				82%

Teste #4						
Enviado			Recebido			Real
Nd	Ne	θ	Nd	Ne	θ	θ
5	-1	41	10	-1	75	61
5	-1	41	9	-2	76	62
5	-1	41	9	-2	75	61
5	-1	41	10	-3	89	73
5	-1	41	9	-2	76	62
5	-1	41	9	-2	75	61
Total		246	Total		466	380
Média		41	Média		78	63
		189%				82%

Teste #5						
Enviado			Recebido			Real
Nd	Ne	θ	Nd	Ne	θ	θ
6	-1	48	11	-1	82	72
6	-1	48	12	-2	96	84
6	-1	48	11	-2	89	78
6	-1	48	10	-3	89	78
6	-1	48	11	-2	89	78
Total		240	Total		445	390
Média		48	Média		89	78
		185%				88%

Teste #6						
Enviado			Recebido			Real
Nd	Ne	θ	Nd	Ne	θ	θ
7	-1	55	12	-2	96	78
7	-1	55	11	-3	96	78
7	-1	55	12	-4	110	90
7	-1	55	10	-1	75	61
7	-1	55	11	-2	89	73
Total		275	Total		466	380
Média		55	Média		93	76
		169%				82%

Pegando como exemplo o teste número 4, pode-se observar que o comando enviado pelo computador ao robô, deveria resultar em um ângulo de giro do robô próximo de 246º, porém o resultado recebido do robô apresenta que o ângulo de giro do robô foi de 466º, isso equivale a 189% acima do valor esperado, além do mais é possível observar que apesar do valor recebido apontar um valor de 466º, o ângulo real de giro do robô foi aproximadamente 380º, isso equivale a 82% do valor recebido do robô.

É possível obsevar pelos demais testes apresentados nessa tabela, que essas diferenças não diferem muito de um comando para outro, mostrando uma certa precisão se utilizarmos esses parâmetros de ajustes ao enviar os dados para o robô.

Apesar de diversos outros testes terem sido executados, apenas 6 deles foram apresentados na Tabela 6.5, porém isso não interfere na conclusão final obtida, pois os resultados obtidos foram muito semelhantes.

Na Tabela 6.6 são apresentadas as distâncias calculadas em relação à quantidade de pulsos lidos pelo microcontrolador, com base na leitura dos sensores de velocidade.

TABELA 6.6 – DISTÂNCIA CALCULADA VS. QUANTIDADE DE PULSOS LIDOS

Ne > 0 \ Nd > 0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
0	0	1	2	3	3	4	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22
1	1	2	3	3	4	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23
2	2	3	3	4	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23
3	3	3	4	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24
4	3	4	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25
5	4	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26
6	5	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27
7	6	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28
8	7	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28
9	8	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29
10	8	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30
11	9	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31
12	10	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32
13	11	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33
14	12	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34
15	13	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34
16	13	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35
17	14	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36
18	15	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37
19	16	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38
20	17	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39
21	18	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39	39
22	18	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39	39	40
23	19	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39	39	40	41
24	20	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39	39	40	41	42
25	21	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39	39	40	41	42	43
26	22	23	23	24	25	26	27	28	28	29	30	31	32	33	34	34	35	36	37	38	39	39	40	41	42	43	44

Para que robô percorra um caminho em linha reta, é necessário que o giro dos motores esquerdo e direito sejam iguais, ou seja, a quantidade de pulsos lidos pelos sensores de ambos os lados deve ser igual (*Nd* = *Ne*).

Um dos problemas identificados é que ao mandar o robô percorrer um caminho em linha reta, quando os motores são desligados, o motor direito demora mais tempo para parar do que o do lado esquerdo, isso faz com que o robô tenha um ângulo de giro em relação ao ponto de partida.

A Tabela 6.7, apresenta alguns testes realizados, com o objetivo de fazer com que o robô percorra um caminho em linha reta.

Os testes 1 e 2 mostram as distâncias calculadas versus a distância percorrida e medida pelo robô, versus a distância real percorrida pelo robô. Foram feitas 10 medições para cada comando, a primeira com *Nd* = *Ne* = *20* pulsos e a segunda com *Nd* = *Ne* = *35* pulsos.

TABELA 6.7 – TESTES DO COMANDO ENVIADO VS. EXECUTADO PELO ROBÔ (LINHA RETA)

Teste #1								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
20	20	0	34	45	37	55	69	67
20	20	0	34	41	35	41	64	61
20	20	0	34	46	37	62	70	67
20	20	0	34	41	35	41	64	63
20	20	0	34	42	35	48	65	63
20	20	0	34	40	35	34	63	61
20	20	0	34	42	35	48	65	63
20	20	0	34	44	36	55	67	65
20	20	0	34	42	36	41	65	65
20	20	0	34	40	34	41	62	62
	Média		34	Média		47	65	63
	Diferença		31					97%

Teste #2								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
35	35	0	59	60	53	48	95	93
35	35	0	59	56	51	34	90	89
35	35	0	59	59	52	48	93	90
35	35	0	59	56	50	41	89	87
35	35	0	59	61	50	75	93	91
35	35	0	59	61	50	75	93	89
35	35	0	59	61	50	75	93	89
35	35	0	59	59	49	69	90	87
35	35	0	59	59	51	55	92	89
35	35	0	59	54	49	34	86	85
	Média		59	Média		55	91	89
	Diferença		32					97%

Teste #3								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
10	20	-69	25	29	29	0	49	47
10	20	-69	25	33	30	21	53	53
10	20	-69	25	32	30	14	52	50
10	20	-69	25	30	29	7	49	47
10	20	-69	25	30	29	7	49	48
10	20	-69	25	29	28	7	48	47
10	20	-69	25	33	29	27	52	50
10	20	-69	25	30	29	7	49	47
10	20	-69	25	31	29	14	50	50
10	20	-69	25	28	28	0	47	45
	Média		25	Média		10	50	48
	Diferença		25					97%

Teste #4								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
25	35	-69	50	48	46	14	79	77
25	35	-69	50	49	45	27	79	77
25	35	-69	50	49	46	21	80	79
25	35	-69	50	49	46	21	80	79
25	35	-69	50	47	46	7	78	76
25	35	-69	50	47	45	14	77	75
25	35	-69	50	46	45	7	76	75
25	35	-69	50	46	45	7	76	74
25	35	-69	50	46	44	14	75	75
25	35	-69	50	47	45	14	77	76
	Média		50	Média		15	78	76
	Diferença		28					98%

Analisando os resultados dos testes 1 e 2, é possível observar que independentemente da distância solicitada que o robô percorresse, quando *Nd* for igual a *Ne*, a diferença é de aproximadamente 31cm a mais do que o desejado, para valores de *Nd* e *Ne* abaixo de 20 pulsos, essa diferença tende a diminuir devido a diminuição da força de inércia. A diferença entre o valor real medido e o valor calculado com base na leitura dos pulsos foi extremamente pequena, de apenas 2cm, essa diferença se mostrou constante em praticamente todos os testes realizados. As principais causas para essa diferença podem ser:

- leitura adicional de um pulso pelos sensores de velocidade;
- erro devido a resolução de leitura dos pulsos (apenas 12 ímãs);
- diferenças entre os valores considerados para os cálculos versus as medidas reais, como por exemplo, raio das rodas.

As Figuras 6.5, 6.6, 6.7 e 6.8 mostram a imagem dos caminhos percorridos pelo robô, os 10 pontos de chegada, e a direção final do robô dos testes 1, 2, 3 e 4 respectivamente.

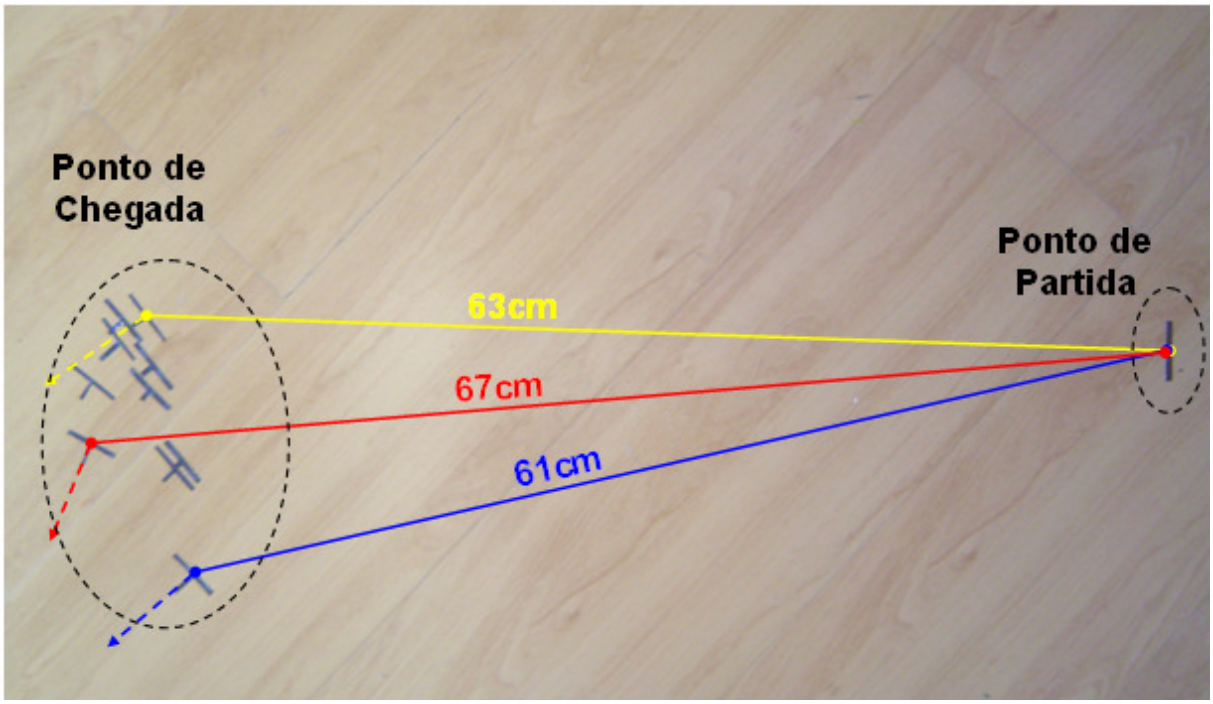


FIGURA 6.5 – TESTE #1 RELACIONADO AOS DADOS DA TABELA 6.7 (ESCALA 1:4,8)

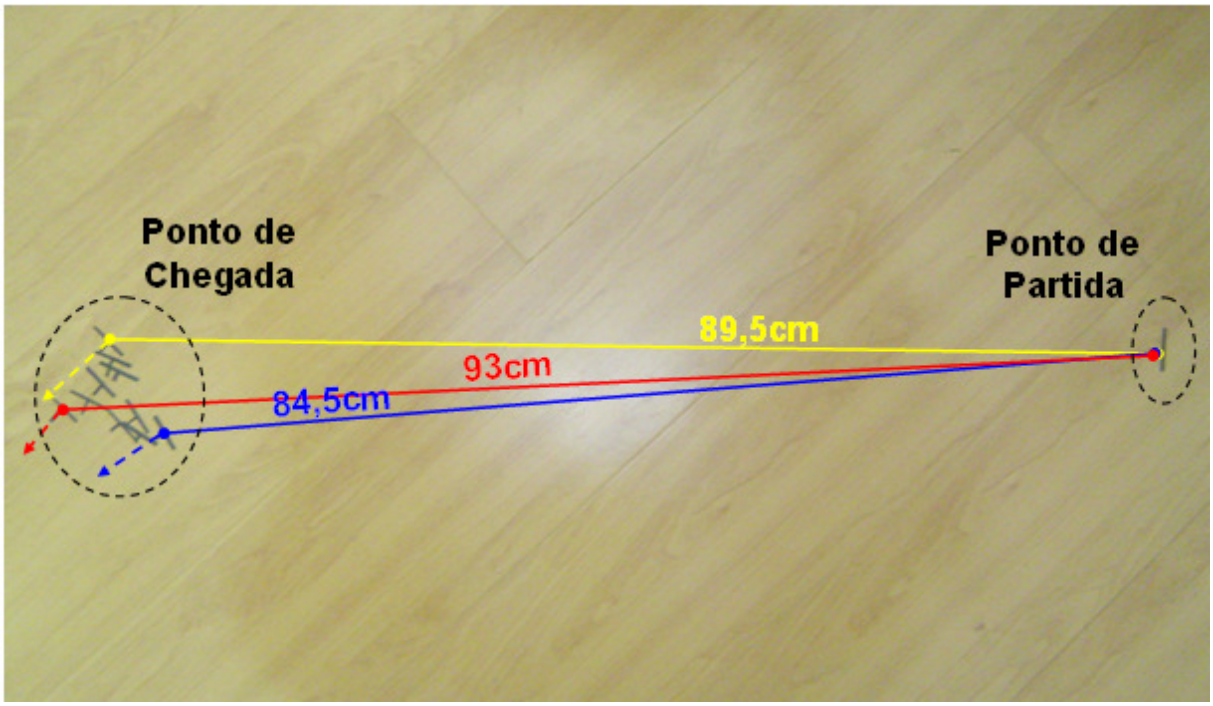


FIGURA 6.6 – TESTE #2 RELACIONADO AOS DADOS DA TABELA 6.7 (ESCALA 1:6,6)

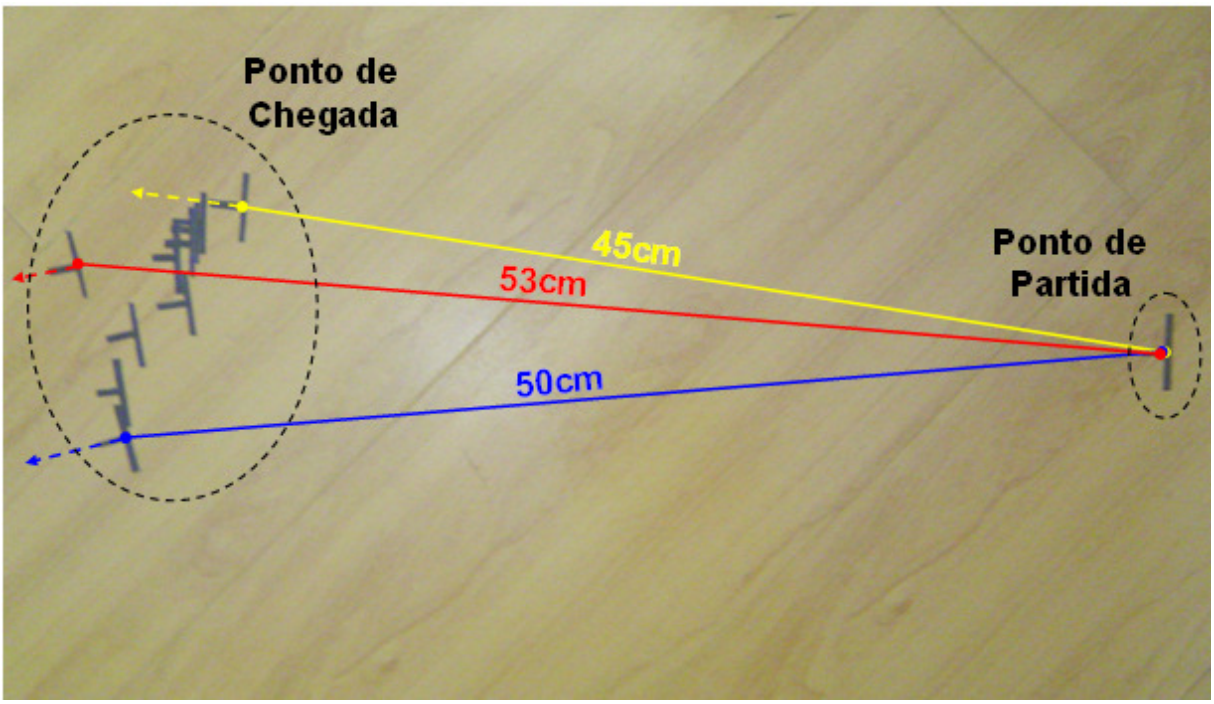


FIGURA 6.7 – TESTE #3 RELACIONADO AOS DADOS DA TABELA 6.7 (ESCALA 1:3,8)

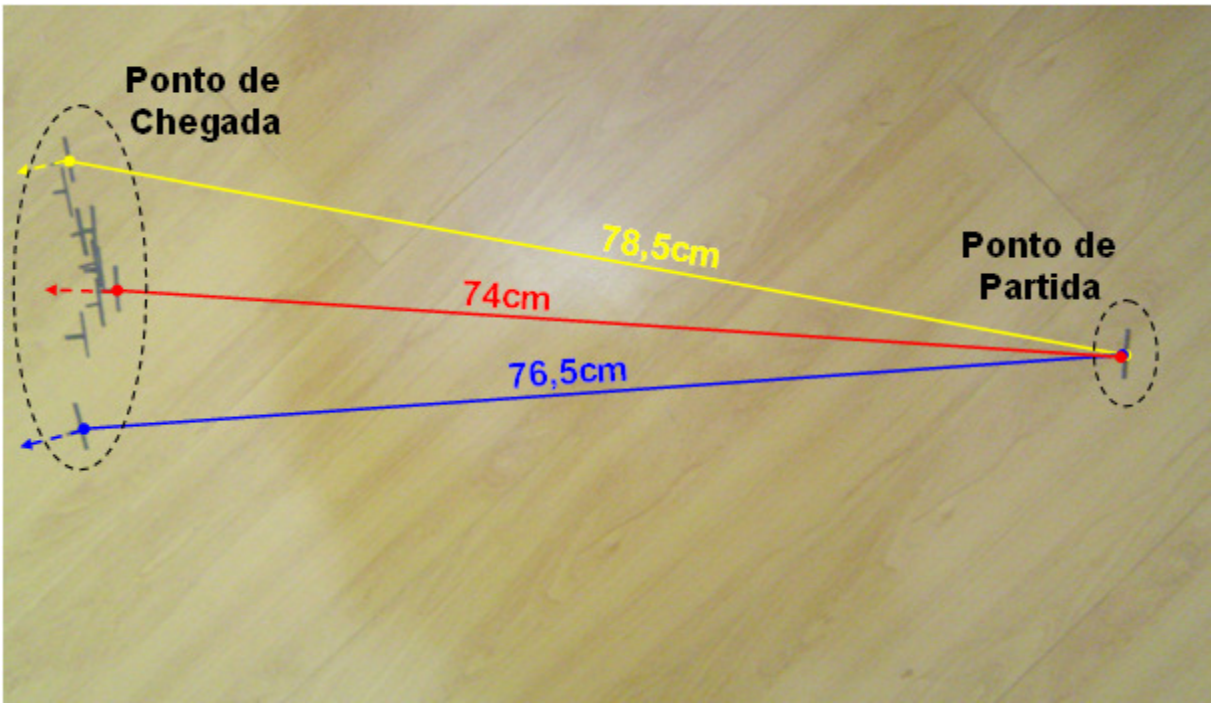


FIGURA 6.8 – TESTE #4 RELACIONADO AOS DADOS DA TABELA 6.7 (ESCALA 1:5,7)

Nas Figuras 6.5 e 6.6, o importante é observar que o problema não se encontra na distância percorrida, mas sim na direção final do robô.

Para minimizar esse erro, foram feitos alguns testes adicionais como os testes 3 e 4 apresentados na Tabela 6.7. A idéia foi a de reduzir a quantidade de giros que o motor direito deve executar em relação a quantidade de giros do motor esquerdo, e também, considerar uma velocidade inicial para o motor direito de 90% do total, ou seja, 10% mais lento que a velocidade do motor esquerdo.

Essa alternativa resultou em uma média de ângulo de giro bem menor quando comparada com a dos testes 1 e 2, conforme pode ser observado nas Figuras 6.7 e 6.8.

6.3 CONTROLE DOS MOTORES

6.3.1 Controle da Velocidade

Antes de explicar o conceito aplicado para melhorar a performance do robô com relação ao caminho percorrido em linha reta, cabe aqui uma breve explicação do conceito utilizado com relação à forma de controle de velocidade dos motores.

A velocidade de um motor de corrente contínua, pode ser controlada por meio do ajuste da sua tensão de armadura. Em vez de um sinal contínuo ser utilizado para realizar esse controle, pode ser utilizado um sinal PWM (*Pulse Width Modulation*) onde a largura dos pulsos controla a potência fornecida ao motor e por sua vez a velocidade de rotação.

A modulação PWM, consiste basicamente em aplicar uma onda quadrada de amplitude Vcc e freqüência alta (entre 10KHz e 20KHz)[41] no lugar da tensão contínua (Vcc). A tensão média varia em função do tempo que a onda fica em nível alto (Vcc) e do tempo que a onda fica em nível baixo (0V). A relação entre o tempo que a onda fica em nível alto e o período total é conhecido como *"Duty Cycle"* (Ciclo de Trabalho). O *Duty Cycle* é normalmente expresso em porcentagem, ou seja, para uma modulação PWM com *Duty Cycle* igual a 50%, metade do tempo a tensão fica em nível alto (Vcc) e metade em nível baixo (0V). Uma modulação PWM de amplitude 12V e *Duty Cycle* de 50% produz o mesmo efeito de uma tensão contínua de amplitude 6V, isto porque a tensão média nos dois casos é 6V.

A Figura 6.9, explica graficamente o que foi comentado.

[41] FONTE: http://en.wikipedia.org/wiki/Pulse-width_modulation

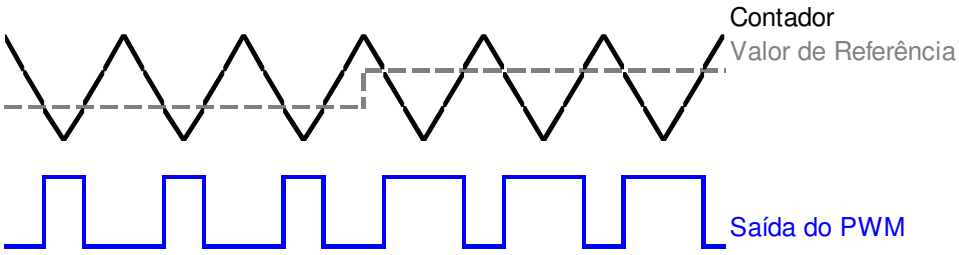


FIGURA 6.9 – VARIAÇÃO DO SINAL PWM

Neste projeto o PWM é gerado pelo próprio microcontrolador, onde um contador é incrementado de acordo com o clock adotado para o sistema (4MHz) e é resetado no final de cada período do PWM. Quando o valor do contador é maior que o valor da referência, a saída do PWM muda o estado de alto para baixo [25].

A resolução do PWM adotada é de 8 bits, o que significa um valor máximo de 255 em decimal (FFh) e a freqüência de operação é de:

$$Freqüência = \frac{f_{TCK}}{510} = \frac{4MHz}{510} = 7{,}843KHz$$

A Figura 6.10 apresenta um exemplo de ciclo de trabalho, considerando um valor de referência igual 50% do total (127,5 em decimal).

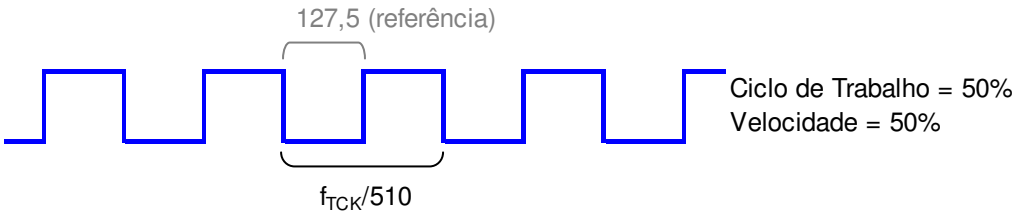


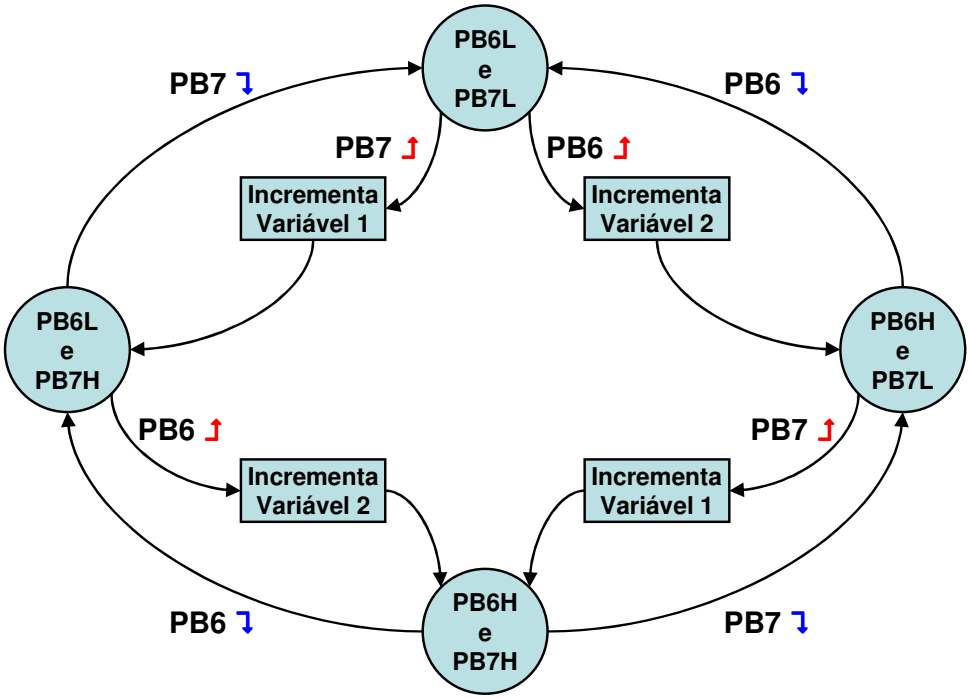
FIGURA 6.10 – EXEMPLO DE VARIAÇÃO DO SINAL PWM (50% DE CICLO DE TRABALHO)

6.3.2 Controle da Quantidade de Rotações

Como explicado anteriormente, os valores de *Nd* e *Ne* são a quantidade de pulsos lidos pelos sensores de velocidade, localizados nas rodas direita e esquerda respectivamente. Com base nos valores de *Nd* e *Ne*, o microcontrolador consegue definir o momento exato em que o motor deve ser desligado. O sinal recebido dos sensores de velocidade incrementam dois

contadores no microcontrolador que ele utiliza como referência para comparar com os valores recebidos de *Nd* e *Ne*.

A Figura 6.11, procura ilustrar o diagrama de estados baseado na lógica de programação utilizada para identificar a mudança de estado nas portas do microcontrolador, por onde o sinal dos sensores de velocidade será recebido.



Legenda:

PB6L = Pino 6 da porta B do microcontrolador está em nível baixo;
PB6H = Pino 6 da porta B do microcontrolador está em nível alto;
PB6 ↓ = Mudança de estado do pino 6 da porta B de nível alto para nível baixo;
PB6 ↑ = Mudança de estado do pino 6 da porta B de nível baixo para nível alto;
PB7L = Pino 7 da porta B do microcontrolador está em nível baixo;
PB7H = Pino 7 da porta B do microcontrolador está em nível alto;
PB7 ↓ = Mudança de estado do pino 7 da porta B de nível alto para nível baixo;
PB7 ↑ = Mudança de estado do pino 7 da porta B de nível baixo para nível alto.

FIGURA 6.11 – DIAGRAMA DE ESTADOS (SINAL DOS SENSORES DE VELOCIDADE)

6.3.3 Compensação da Velocidade

Para melhorar a performance do robô, com relação ao caminho percorrido em linha reta, foi incorporado na lógica de programação do microcontrolador um sistema de compensação de velocidade.

A Figura 6.12 apresenta o fluxograma dessa programação, onde é possível entender o conceito utilizado.

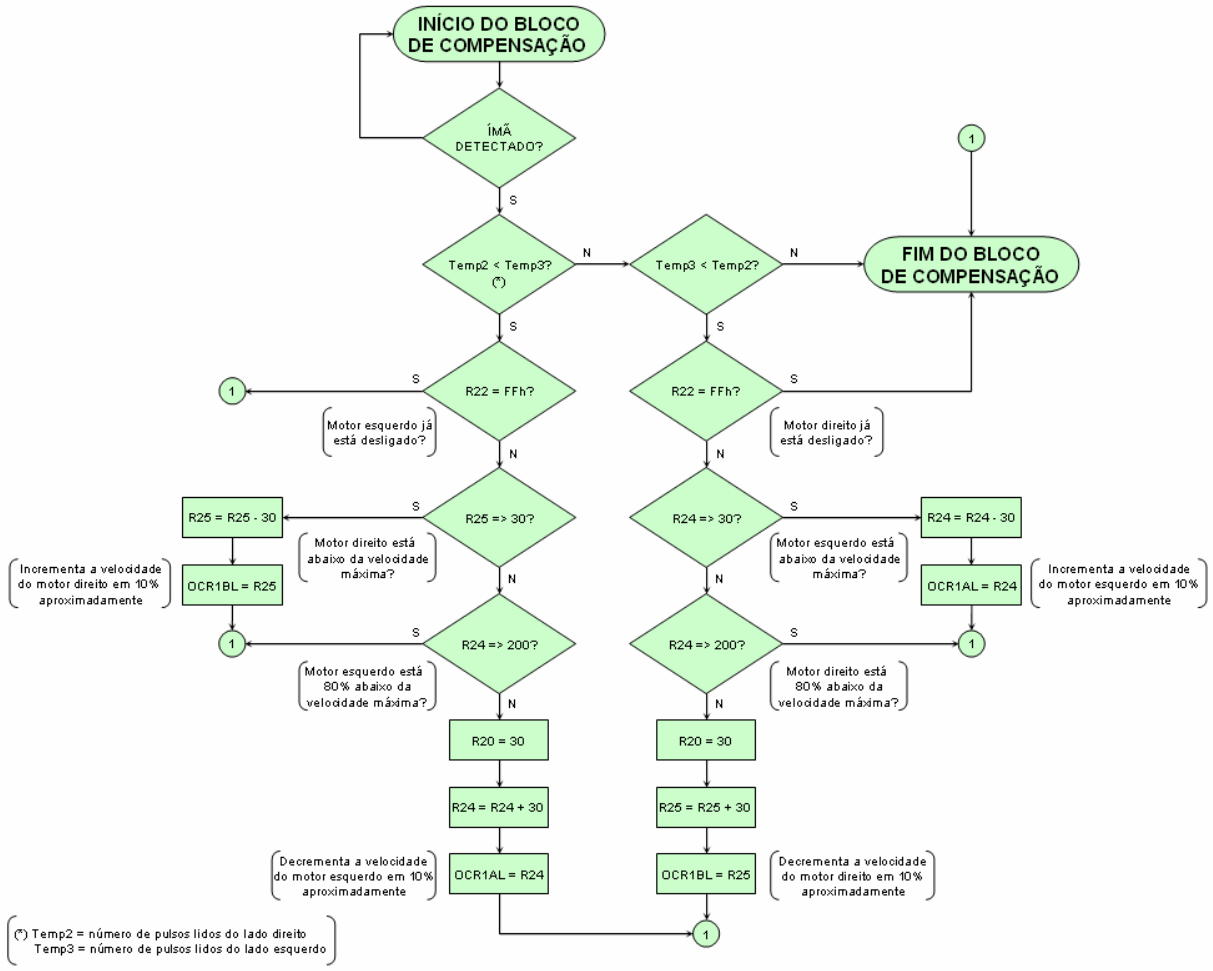


FIGURA 6.12 – FLUXOGRAMA DA LÓGICA DE COMPENSAÇÃO DE VELOCIDADE

Um conceito interessante que também poderia ser aplicado no controle de velocidade, é através do desenvolvimento de um algoritmo de controle PI (Proporcional Integral) no microcontrolador para garantir uma melhor precisão no controle de velocidade dos motores de corrente contínua, conforme apresentado no artigo [35].

Com o objetivo de ilustrar o conceito explicado no fluxograma acima, foram realizados oito testes, sendo quatro sem utilizar o sistema de compensação de velocidade e quatro utilizando-o.

A Tabela 6.8, apresenta os resultados dos quatro testes sem a utilização do sistema de compensação de velocidade e a Figura 6.13, mostra a imagem dos caminhos percorridos pelo robô, relacionados com estes testes. Já a Tabela 6.9 e a Figura 6.14, apresentam os resultados dos quatro testes utilizando o sistema de compensação de velocidade.

TABELA 6.8 – TESTES DE DESEMPENHO SEM SISTEMA DE COMPENSAÇÃO DE VELOCIDADE

Teste #1

Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
20	20	0	34	53	43	69	80	76
20	20	0	34	52	42	68	79	75
20	20	0	34	52	41	76	78	75
20	20	0	34	52	41	75	78	75
20	20	0	34	49	40	62	75	71
	Média		34		Média	70	78	74
	Diferença		44					95%

Teste #2

Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
10	20	-69	25	40	34	41	62	61
10	20	-69	25	40	34	41	62	62
10	20	-69	25	39	33	41	60	60
10	20	-69	25	39	33	42	60	59
10	20	-69	25	40	33	48	61	60
	Média		25		Média	43	61	60
	Diferença		36					98%

Teste #3

Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
45	45	0	75	79	66	89	121	115
45	45	0	75	80	66	96	122	116
45	45	0	75	80	67	89	123	117
45	45	0	75	79	66	89	121	114
45	45	0	75	78	66	83	121	115
	Média		75		Média	89	122	115
	Diferença		47					95%

Teste #4

Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
35	45	-69	67	76	64	82	117	114
35	45	-69	67	75	65	69	117	114
35	45	-69	67	74	64	68	116	111
35	45	-69	67	73	63	69	114	110
35	45	-69	67	75	63	82	116	112
	Média		67		Média	74	116	112
	Diferença		49					97%

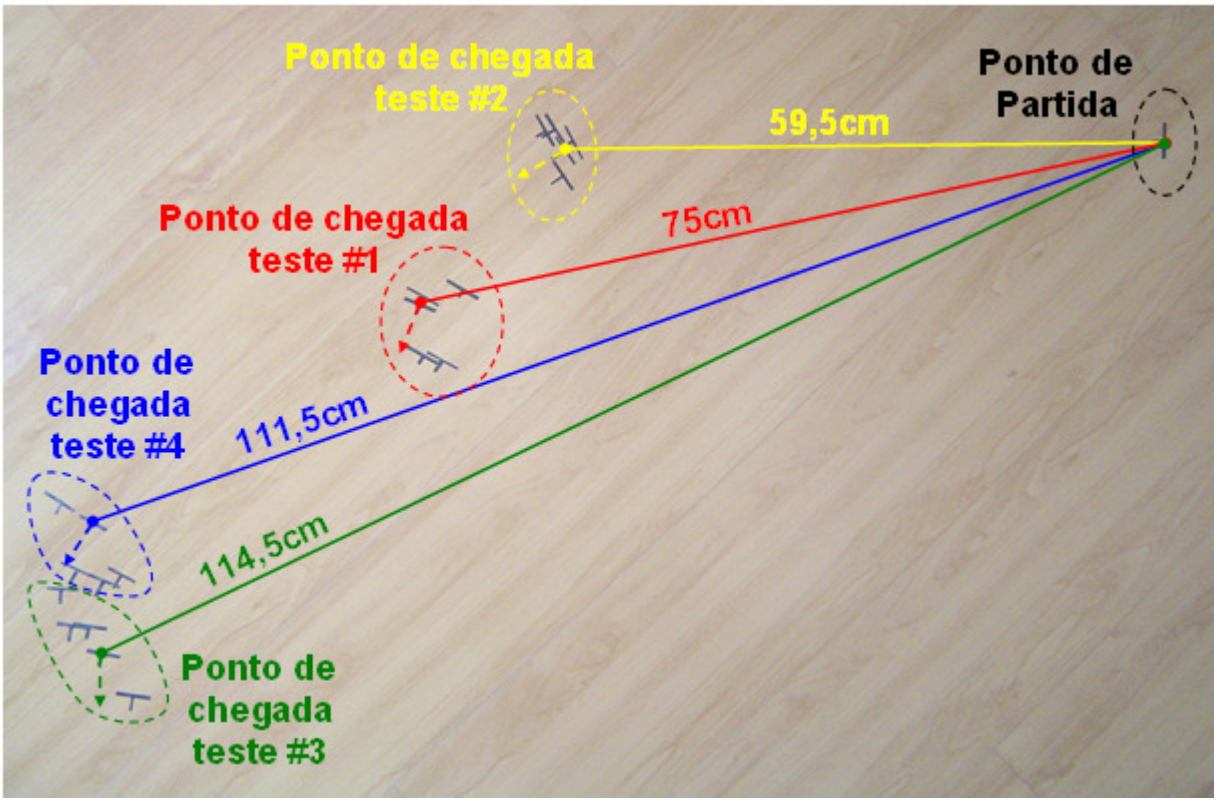


FIGURA 6.13 – TESTE RELACIONADO AOS DADOS DA TABELA 6.8 (ESCALA 1:7,6)

TABELA 6.9 – TESTES DE DESEMPENHO COM SISTEMA DE COMPENSAÇÃO DE VELOCIDADE

Teste #1								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
20	20	0	34	41	32	62	61	61
20	20	0	34	40	34	41	62	61
20	20	0	34	39	33	41	60	60
20	20	0	34	37	33	27	59	57
20	20	0	34	35	32	21	56	55
	Média		34		Média	38	60	58
Diferença			26					98%

Teste #2								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
10	20	-69	25	34	32	14	55	54
10	20	-69	25	30	30	0	50	48
10	20	-69	25	29	30	-7	49	49
10	20	-69	25	31	30	7	51	50
10	20	-69	25	29	29	0	49	48
	Média		25		Média	3	51	50
Diferença			26					98%

Teste #3								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
45	45	0	75	70	63	48	111	107
45	45	0	75	69	61	55	109	106
45	45	0	75	69	62	48	110	107
45	45	0	75	71	63	55	112	109
45	45	0	75	71	62	61	111	108
	Média		75		Média	53	111	107
Diferença			36					97%

Teste #4								
Enviado				Recebido				Real
Nd	Ne	θ	Dist.	Nd	Ne	θ	Dist.	Dist.
35	45	-69	67	67	62	34	108	107
35	45	-69	67	60	54	41	96	94
35	45	-69	67	62	60	14	102	101
35	45	-69	67	61	53	55	96	93
35	45	-69	67	65	60	34	105	103
	Média		67		Média	36	101	99
Diferença			34					98%

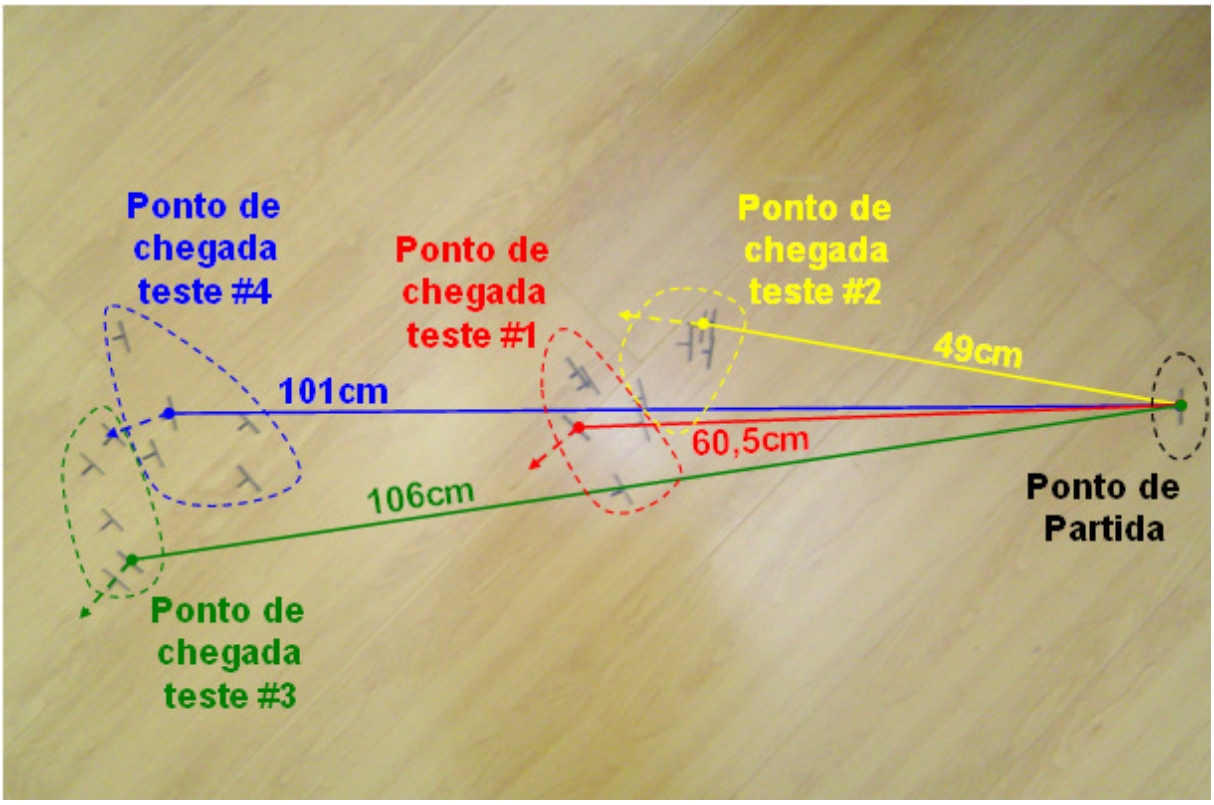


FIGURA 6.14 – TESTE RELACIONADO AOS DADOS DA TABELA 6.9 (ESCALA 1:7,7)

Comparando os resultados apresentados nas Tabelas 6.8 e 6.9, e analisando as imagens apresentadas nas Figuras 6.13 e 6.14, é possível concluir uma grande melhora em relação ao caminho percorrido, entretanto ainda existe um pequeno ângulo de giro no final do percurso.

Esse erro pode ser compensado descontando essa diferença do próximo comando a ser enviado ao robô.

6.4 INTERFACE GRÁFICA

Conforme já comentado anteriormente, a Figura 6.1, apresenta a tela do programa de controle em execução. É possível observar nessa imagem, que existe uma área gráfica onde é possível desenhar o caminho que se deseja que o robô percorra.

Após desenhado o caminho que o robô deve percorrer, essa informação gráfica, precisa ser convertida em valores de *Nd* e *Ne*, que é a linguagem de comando de movimentação do robô.

Para se definir os valores de *Nd* e *Ne*, é necessário saber qual a direção do caminho desenhado, em outras palavras, qual o ângulo de giro do robô em relação a posição anterior que ele se encontrava.

A definição desse ângulo, depende da direção que o robô se encontrava anteriormente, dependendo dessa posição, o cálculo para a definição do comando correto a ser escolhido pode variar.

Ao longo do desenvolvimento do projeto, observou-se que existem 81 possibilidades diferentes que precisam ser consideradas no cálculo para a definição do ângulo de giro do robô.

Para facilitar o entendimento, a Figura 6.15 apresenta algumas situações possíveis de movimentação do robô, onde cada cor representa um caminho simulado.

Partindo-se do princípio de que o ponto de origem nesse gráfico é realmente o ponto de partida do robô, e que o ângulo inicial considerado é igual a 0º, ou seja, a direção inicial do robô é no sentido do eixo $x > 0$, na Figura 6.16 é possível observar o resultado dos cálculos que teoricamente devem ser considerados no caminho a ser percorrido pelo robô.

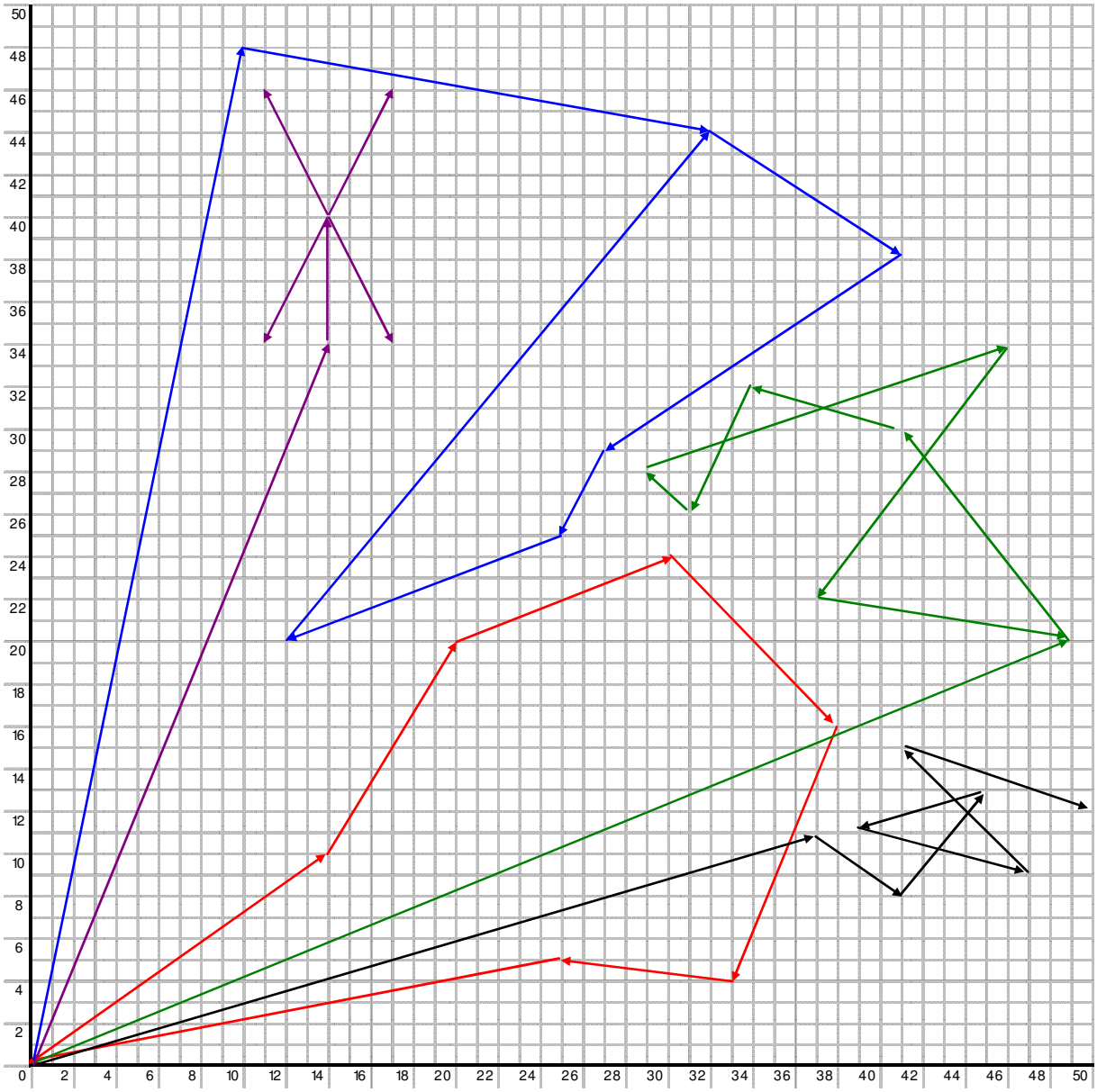


FIGURA 6.15 – SIMULAÇÃO DE CAMINHOS POSSÍVEIS A SEREM PERCORRIDOS PELO ROBÔ

	x	y	Dx	Dy	α	θ
x>0 y>0	14	10	14	10	36	
x>0 y>0	20	20	6	10	59	23
x>0 y>0	30	24	10	4	22	-37
x>0 y<0	38	16	8	-8	-45	-67
x<0 y<0	33	4	-5	-12	67	-68
x<0 y>0	25	5	-8	1	-7	-75
x<0 y>0	0	0	-25	-5	11	18

	x	y	Dx	Dy	α	θ
x>0 y>0	10	48	14	10	78	
x>0 y<0	32	44	22	-4	-10	-89
x>0 y<0	41	38	9	-6	-34	-23
x<0 y<0	27	29	-14	-9	33	-114
x<0 y<0	25	25	-2	-4	63	31
x<0 y<0	12	20	-13	-5	21	-42
x>0 y>0	32	44	20	24	50	-151

	x	y	Dx	Dy	α	θ
x>0 y>0	14	34	14	10	68	
x=0 y>0	14	40	0	6	90	22
x<0 y>0	11	46	-3	6	-63	27
x>0 y>0	17	46	3	6	63	-27
x<0 y<0	11	34	-3	-6	63	153
x>0 y<0	17	34	3	-6	-63	-153

	x	y	Dx	Dy	α	θ
x>0 y>0	37	11	14	10	17	
x>0 y<0	43	8	6	-3	-27	-43
x>0 y>0	47	13	4	5	51	78
x<0 y<0	39	11	-8	-2	14	143
x>0 y<0	47	9	8	-2	-14	152
x<0 y>0	41	15	-6	6	-45	149
x>0 y<0	50	12	9	-3	-18	207

	x	y	Dx	Dy	α	θ
x>0 y>0	49	20	14	10	22	
x<0 y>0	41	30	-8	10	-51	106
x<0 y>0	34	32	-7	2	-16	35
x<0 y<0	31	26	-3	-6	63	79
x<0 y>0	29	28	-2	2	-45	-108
x>0 y>0	46	34	17	6	19	-116
x<0 y<0	37	22	-9	-12	53	-146
x>0 y<0	49	20	12	-2	-9	117

FIGURA 6.16 – RESULTADO DOS CÁLCULOS BASEADOS NOS CAMINHOS DA FIGURA 6.15

Na Figura 6.16, é possível observar o ponto de chegada do robô (*x* e *y*), a distância percorrida pelo robô pode ser calculada, com base nos valores de *Dx* e *Dy* ($D = \sqrt{(D_x)^2 + (D_y)^2}$), o ângulo α, que é o ângulo de giro do robô, considerando o eixo *x* e *y* fixo e o ângulo θ, que é o ângulo de giro do robô, considerando no cálculo a direção anterior.

Os resultados apresentados na Figura 6.16, demonstram que o cálculo do ângulo θ, pode variar dependendo da direção que o robô se encontrava no instante anterior.

Abaixo são listados alguns exemplos de calculos possíveis, dependendo da direção do robô no instante anterior ao do novo comando:

Situação 1:

Estado anterior: $x \geq 0$ e $y \geq 0$

Estado novo: $x > 0$ e $y > 0$

$$\begin{cases} \alpha = \arctan\left(\dfrac{y - y\_anterior}{x - x\_anterior}\right) \\ \theta = \alpha - \alpha\_anterior \end{cases}$$

Situação 2:

Estado anterior: $x < 0$ e $y > 0$

Estado novo: $x > 0$ e $y > 0$

$$\begin{cases} \alpha = \arctan\left(\dfrac{y - y\_anterior}{x - x\_anterior}\right) \\ \theta = -180° + \alpha - \alpha\_anterior \end{cases}$$

Situação 3:

Estado anterior: $x > 0$ e $y > 0$

Estado novo: $x < 0$ e $y > 0$

$$\begin{cases} \alpha = \arctan\left(\dfrac{y - y\_anterior}{x - x\_anterior}\right) \\ \theta = 180° + \alpha - \alpha\_anterior \end{cases}$$

Situação 4:

Estado anterior: $x > 0$ e $y > 0$

Estado novo: $x = 0$ e $y > 0$

$$\begin{cases} \alpha = 90° \\ \theta = \alpha - \alpha\_anterior \end{cases}$$

Situação 5:

Estado anterior: $x > 0$ e $y > 0$

Estado novo: x = 0 e y < 0

$$\begin{cases} \alpha = -90° \\ \theta = \alpha - \alpha\_anterior \end{cases}$$

Situação 6:

Estado anterior: x = 0 e y > 0

Estado novo: x < 0 e y > 0

$$\begin{cases} \alpha = \arctan\left(\dfrac{y - y\_anterior}{x - x\_anterior}\right) \\ \theta = \alpha + \alpha\_anterior \end{cases}$$

A Figura 6.17, apresenta as 81 situações possíveis identificadas, que devem ser consideradas no cálculo para a definição do ângulo de giro do robô.

As situações destacadas em cinza, são as situações mais prováveis de acontecer e que foram introduzidas na lógica de programação do software de controle do robô.

Estado anterior	x>0 y>0	x<0 y>0	x>0 y>0	x<0 y>0	x>0 y>0	x<0 y>0	x=0 y>0	x=0 y>0	x=0 y>0
Estado novo	x>0 y>0	x<0 y>0	x<0 y>0	x>0 y>0	x=0 y>0	x=0 y>0	x<0 y>0	x>0 y>0	x=0 y>0
Estado anterior	x>0 y>0	x<0 y>0	x>0 y>0	x<0 y>0	x>0 y>0	x<0 y>0	x=0 y>0	x=0 y>0	x=0 y>0
Estado novo	x>0 y<0	x<0 y<0	x<0 y<0	x>0 y<0	x=0 y<0	x=0 y<0	x<0 y<0	x>0 y<0	x=0 y<0
Estado anterior	x>0 y<0	x<0 y<0	x>0 y<0	x<0 y<0	x>0 y<0	x<0 y<0	x=0 y<0	x=0 y<0	x=0 y<0
Estado novo	x>0 y<0	x<0 y<0	x<0 y<0	x>0 y<0	x=0 y<0	x=0 y<0	x<0 y<0	x>0 y<0	x=0 y<0
Estado anterior	x>0 y<0	x<0 y<0	x>0 y<0	x<0 y<0	x>0 y<0	x<0 y<0	x=0 y<0	x=0 y<0	x=0 y<0
Estado novo	x>0 y>0	x<0 y>0	x<0 y>0	x>0 y>0	x=0 y>0	x=0 y>0	x<0 y>0	x>0 y>0	x=0 y>0
Estado anterior	x>0 y>0	x<0 y>0	x>0 y>0	x<0 y>0	x>0 y>0	x<0 y>0	x=0 y>0	x=0 y>0	x=0 y>0
Estado novo	x>0 y=0	x<0 y=0	x<0 y=0	x>0 y=0	x=0 y=0	x=0 y=0	x<0 y=0	x>0 y=0	x=0 y=0
Estado anterior	x>0 y<0	x<0 y<0	x>0 y<0	x<0 y<0	x>0 y<0	x<0 y<0	x=0 y<0	x=0 y<0	x=0 y<0
Estado novo	x>0 y=0	x<0 y=0	x<0 y=0	x>0 y=0	x=0 y=0	x=0 y=0	x<0 y=0	x>0 y=0	x=0 y=0
Estado anterior	x>0 y=0	x<0 y=0	x>0 y=0	x<0 y=0	x>0 y=0	x<0 y=0	x=0 y=0	x=0 y=0	x=0 y=0
Estado novo	x>0 y>0	x<0 y>0	x<0 y>0	x>0 y>0	x=0 y>0	x=0 y>0	x<0 y>0	x>0 y>0	x=0 y>0
Estado anterior	x>0 y=0	x<0 y=0	x>0 y=0	x<0 y=0	x>0 y=0	x<0 y=0	x=0 y=0	x=0 y=0	x=0 y=0
Estado novo	x>0 y<0	x<0 y<0	x<0 y<0	x>0 y<0	x=0 y<0	x=0 y<0	x<0 y<0	x>0 y<0	x=0 y<0
Estado anterior	x>0 y=0	x<0 y=0	x>0 y=0	x<0 y=0	x>0 y=0	x<0 y=0	x=0 y=0	x=0 y=0	x=0 y=0
Estado novo	x>0 y=0	x<0 y=0	x<0 y=0	x>0 y=0	x=0 y=0	x=0 y=0	x<0 y=0	x>0 y=0	x=0 y=0

FIGURA 6.17 – SITUAÇÕES POSSÍVEIS PARA O CÁLCULO DO ÂNGULO DE GIRO DO ROBÔ

A seguir serão apresentados os resultados de alguns testes realizados, utilizando-se da interface gráfica do software de controle do robô.

Para facilitar o desenvolvimento do trabalho e análise dos resultados dos testes, criou-se no software de controle, 10 caminhos pré-definidos para que o robô percorra.

A Figura 6.18, apresenta a tela do software de controle, com o resultado do primeiro teste executado e a Figura 6.19, apresenta uma foto do caminho real percorrido pelo robô.

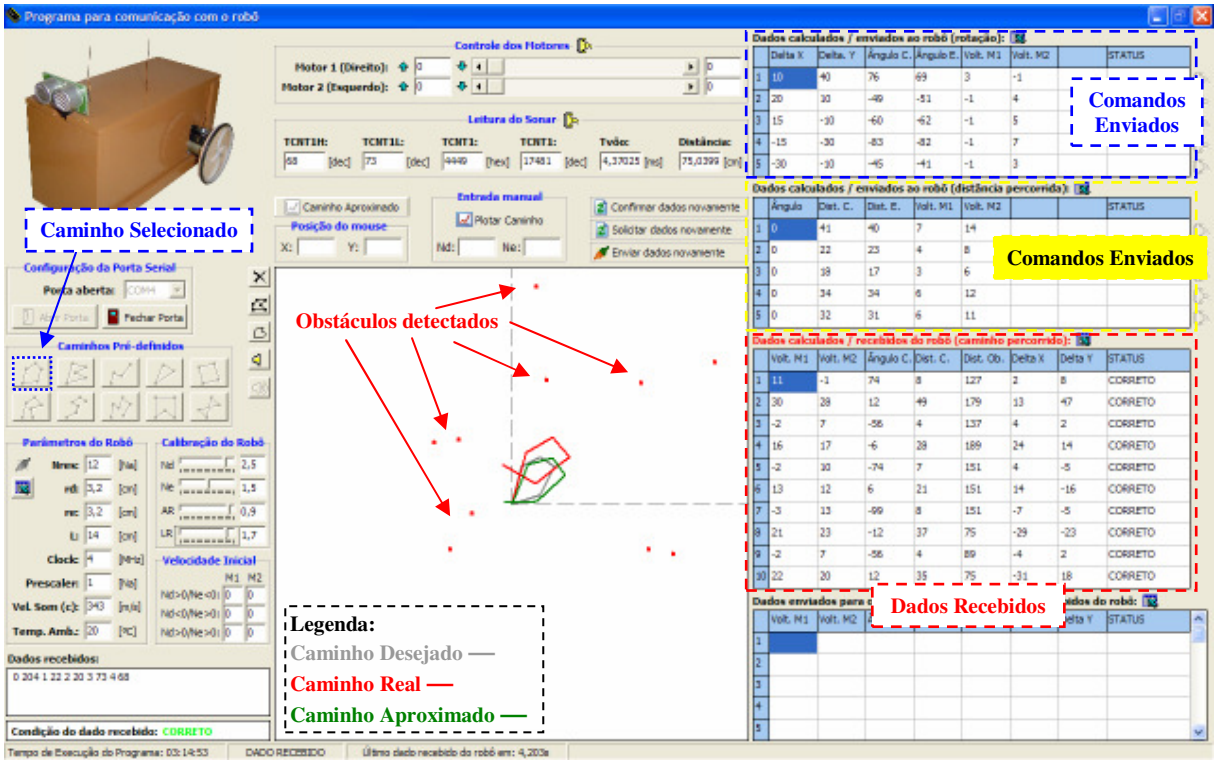


FIGURA 6.18 – TESTE #01 UTILIZANDO-SE A INTERFACE GRÁFICA

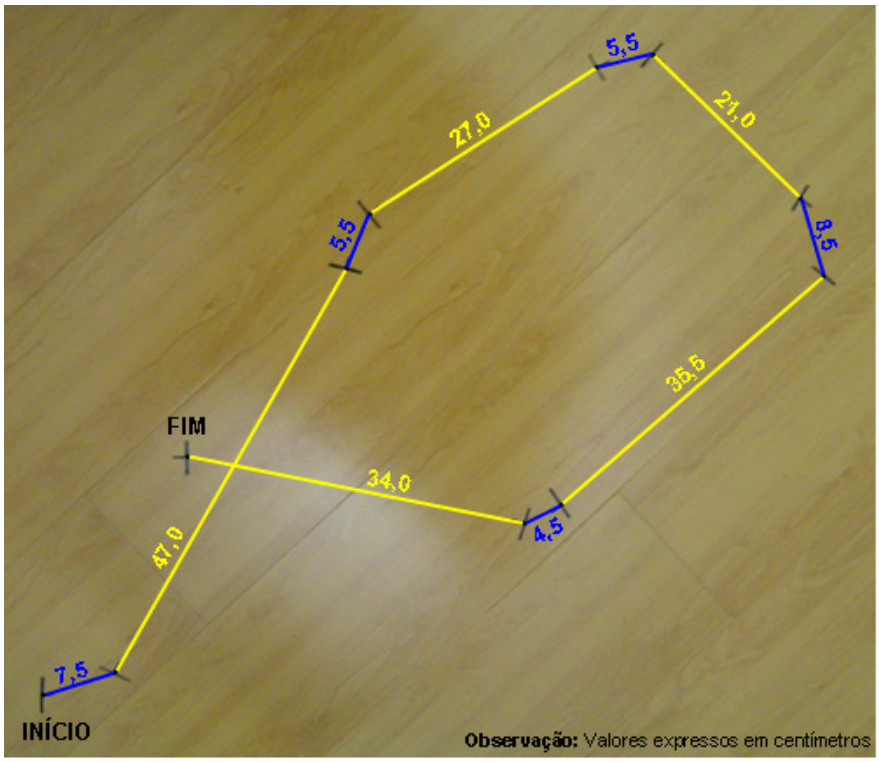


FIGURA 6.19 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #01) (ESCALA 1:7,8)

A Figura 6.20, apresenta a tela do software de controle, com o resultado do segundo teste executado e a Figura 6.21, apresenta uma foto do caminho real percorrido pelo robô.

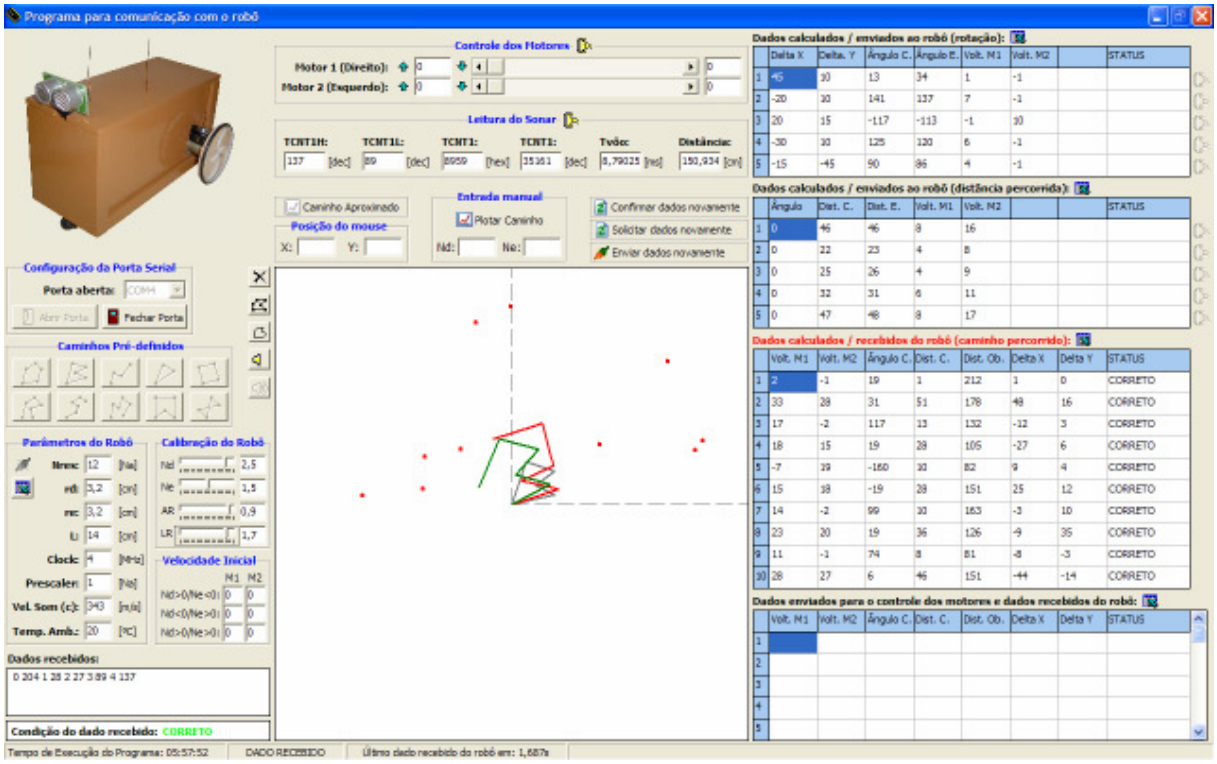


FIGURA 6.20 – TESTE #02 UTILIZANDO-SE A INTERFACE GRÁFICA

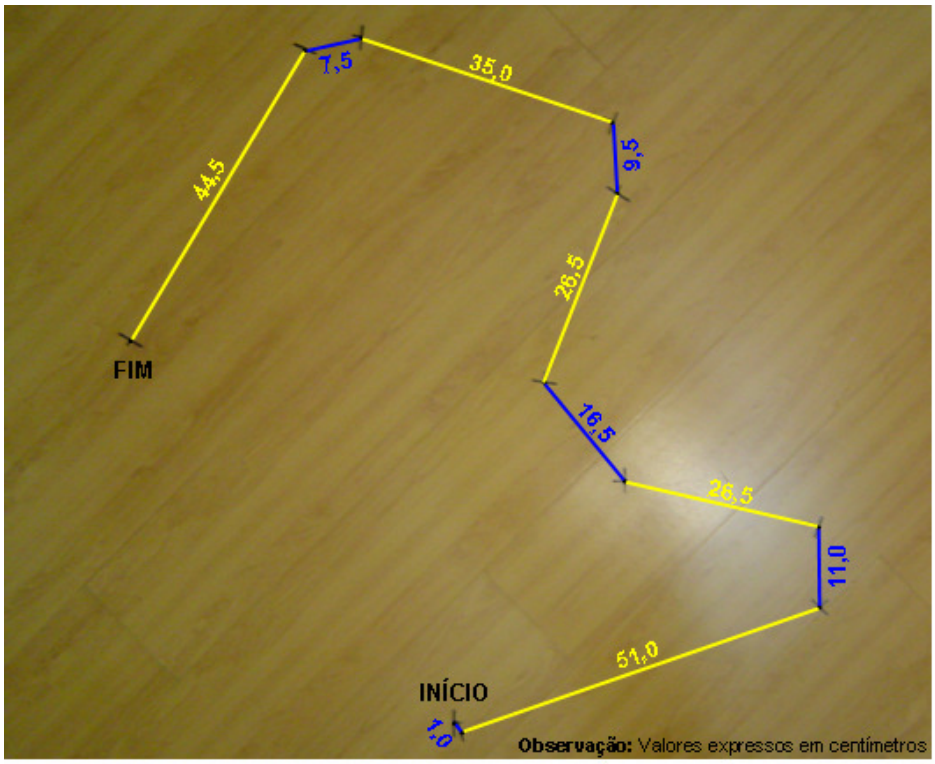


FIGURA 6.21 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #02) (ESCALA 1:10,4)

A Figura 6.22, apresenta a tela do software de controle, com o resultado do terceiro teste executado e a Figura 6.23, apresenta uma foto do caminho real percorrido pelo robô.

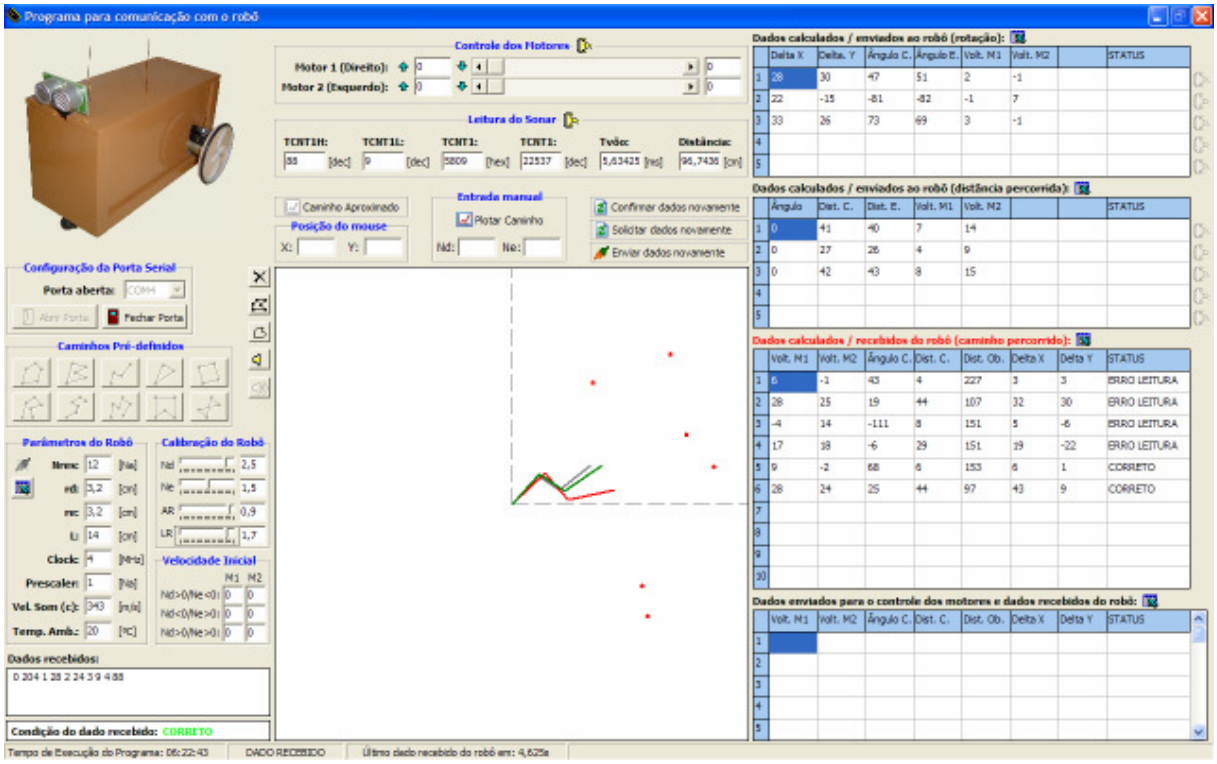


FIGURA 6.22 – TESTE #03 UTILIZANDO-SE A INTERFACE GRÁFICA

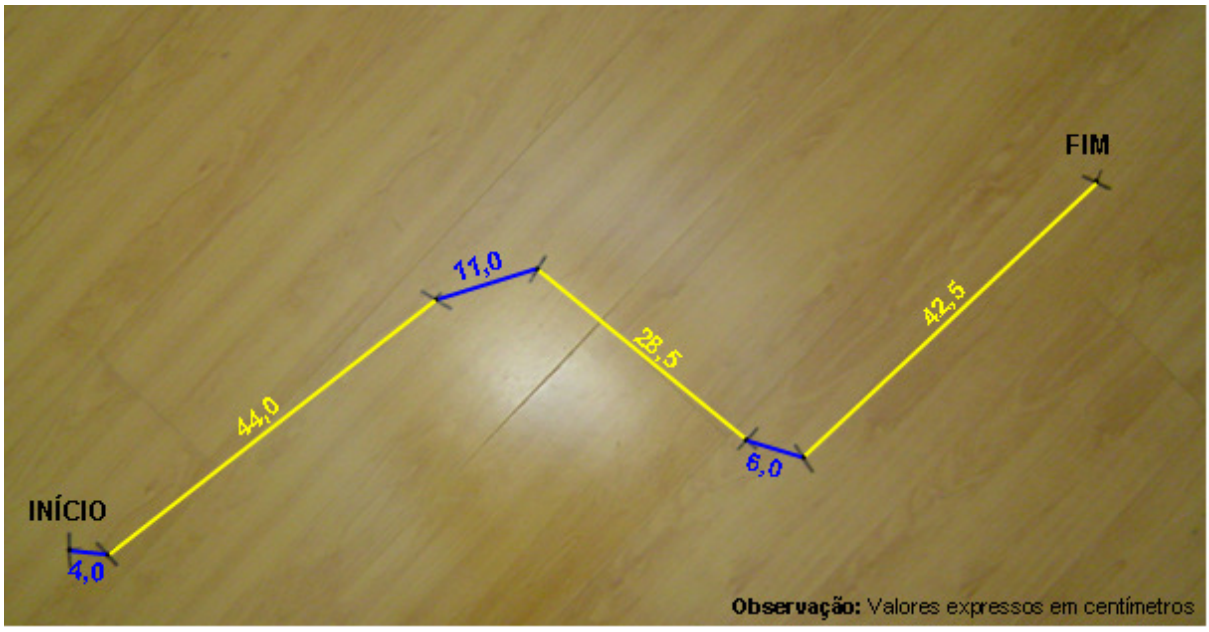


FIGURA 6.23 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #03) (ESCALA 1:8,15)

A Figura 6.24, apresenta a tela do software de controle, com o resultado do quarto teste executado e a Figura 6.25, apresenta uma foto do caminho real percorrido pelo robô.

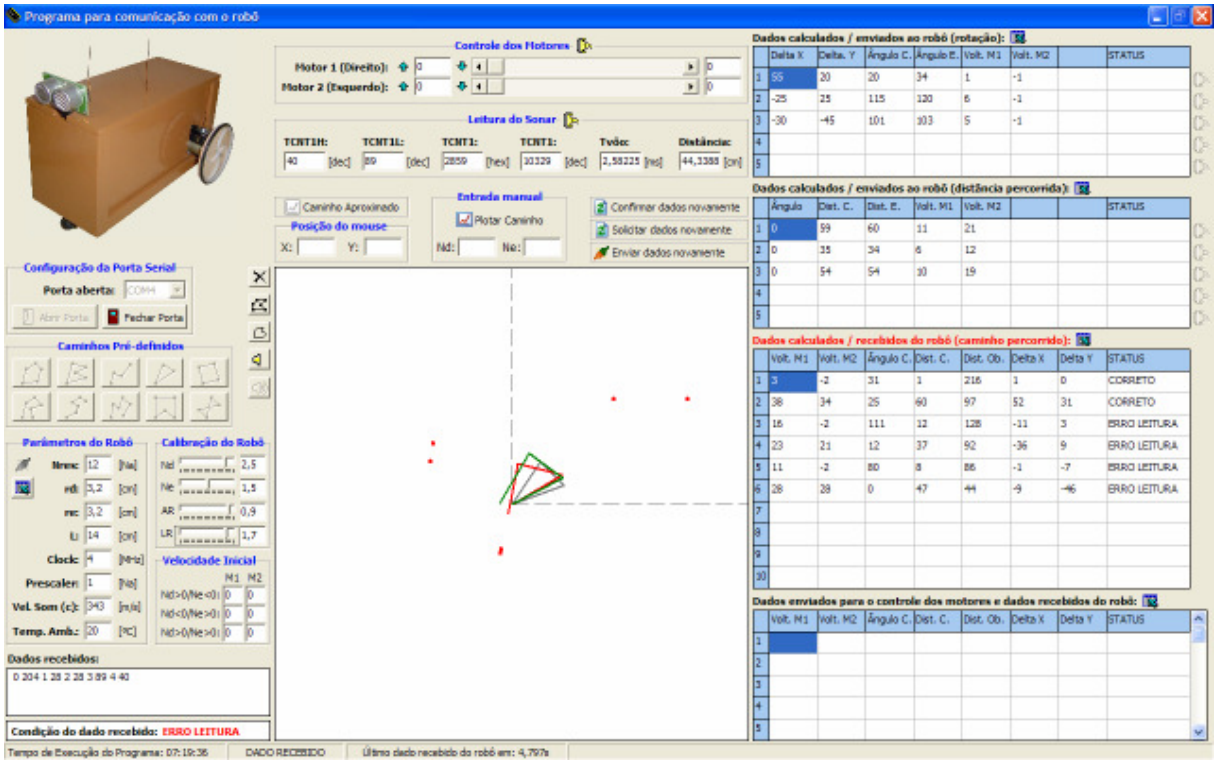


FIGURA 6.24 – TESTE #04 UTILIZANDO-SE A INTERFACE GRÁFICA

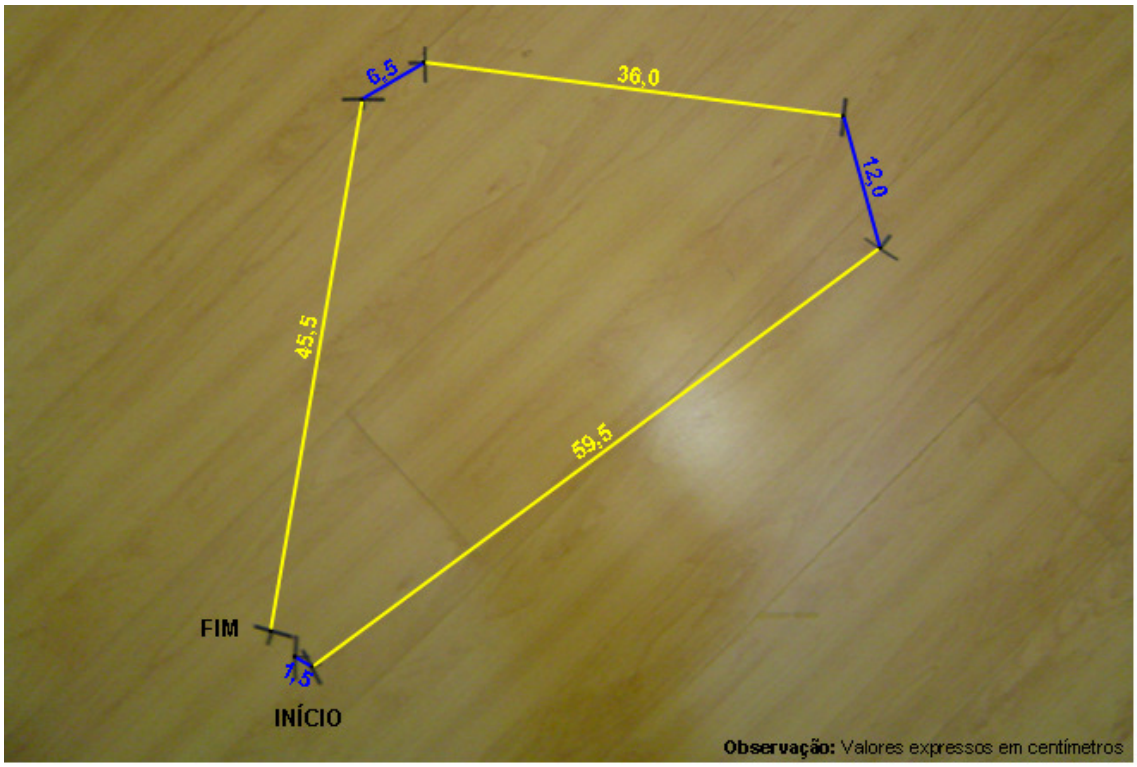


FIGURA 6.25 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #04) (ESCALA 1:6,54)

A Figura 6.26, apresenta a tela do software de controle, com o resultado do quinto teste executado e a Figura 6.27, apresenta uma foto do caminho real percorrido pelo robô.

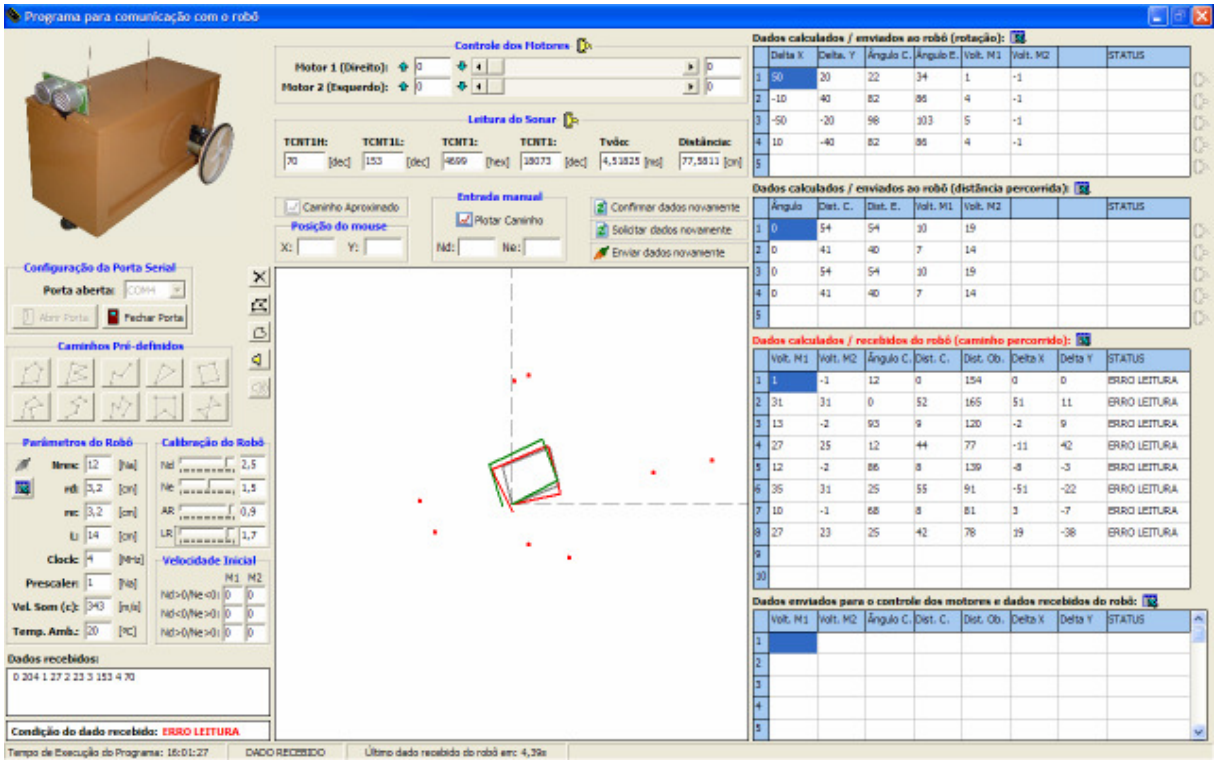


FIGURA 6.26 – TESTE #05 UTILIZANDO-SE A INTERFACE GRÁFICA

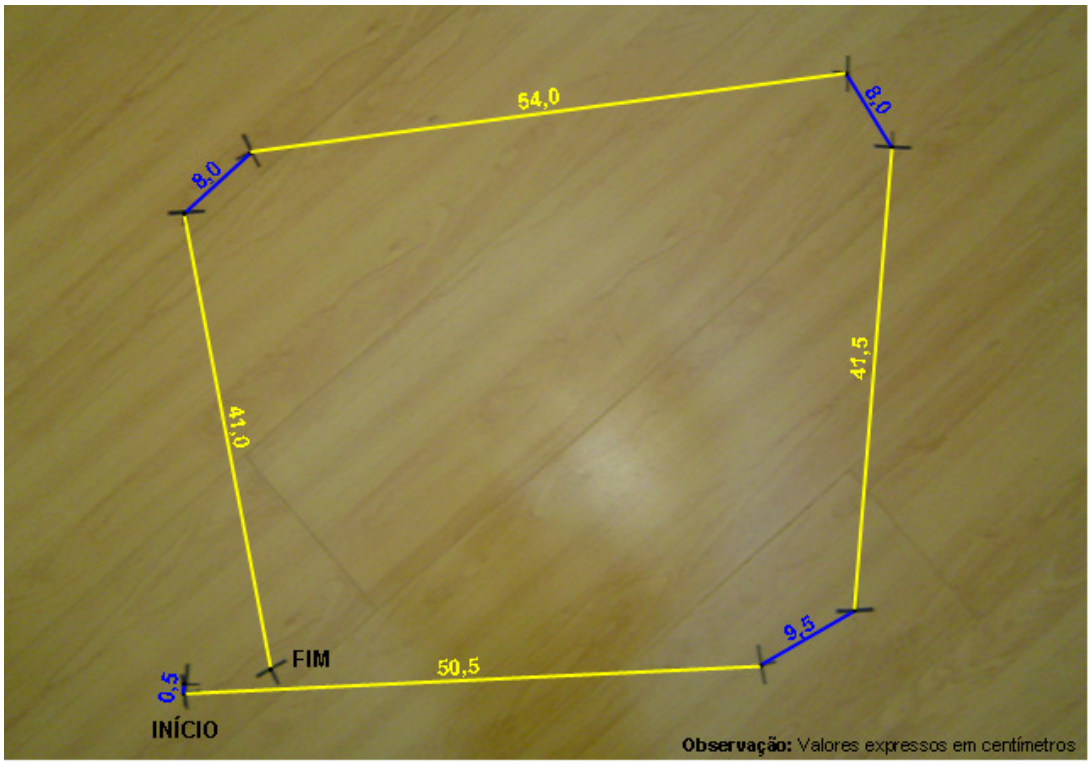


FIGURA 6.27 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #05) (ESCALA 1:6,82)

A Figura 6.28, apresenta a tela do software de controle, com o resultado do sexto teste executado e a Figura 6.29, apresenta uma foto do caminho real percorrido pelo robô.

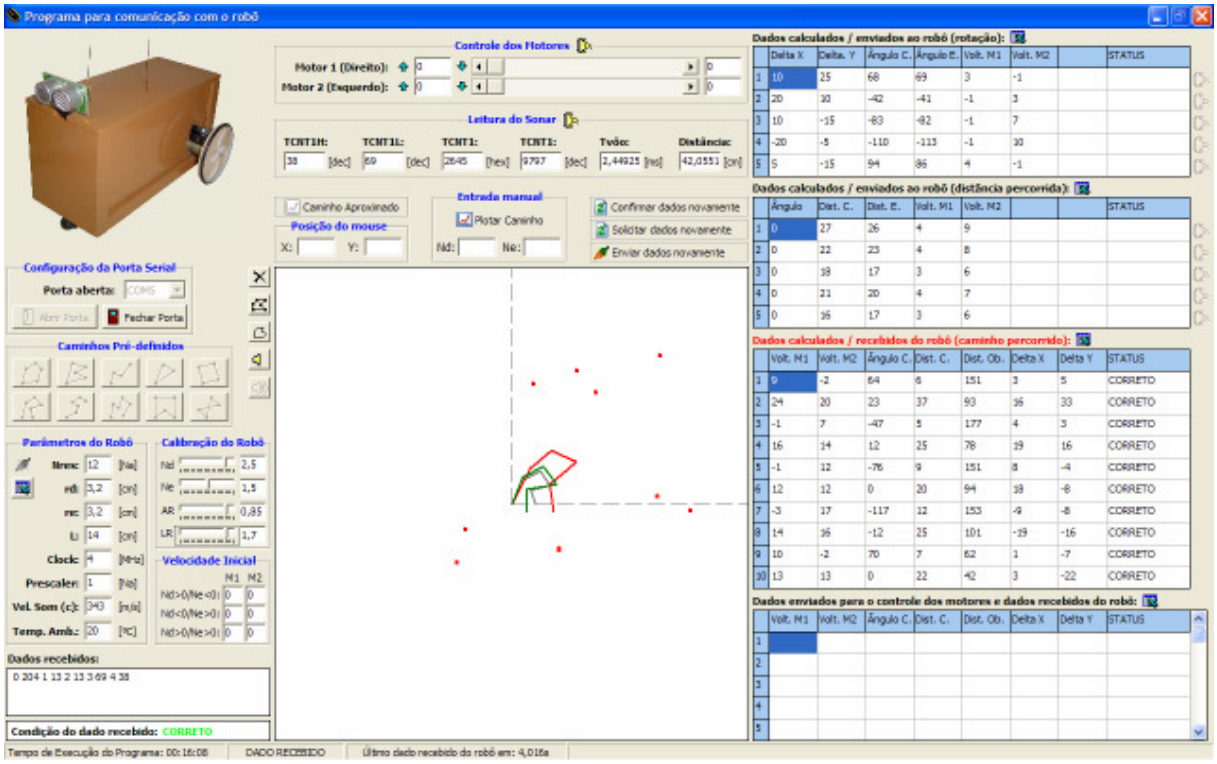


FIGURA 6.28 – TESTE #06 UTILIZANDO-SE A INTERFACE GRÁFICA

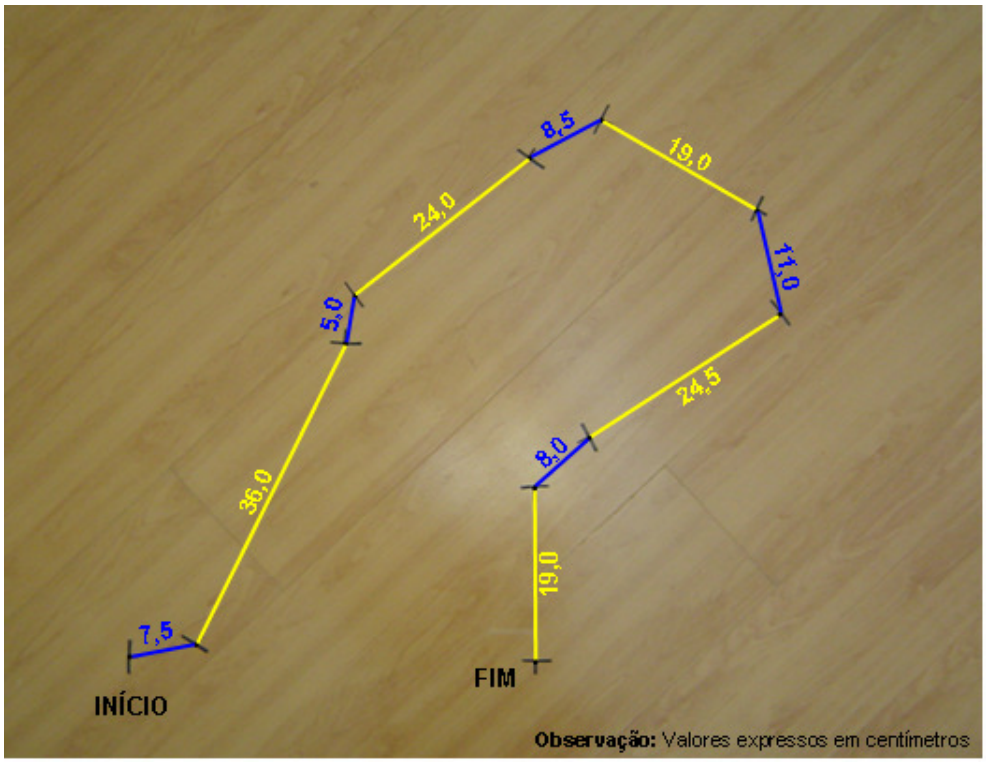


FIGURA 6.29 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #06) (ESCALA 1:8,37)

A Figura 6.30, apresenta a tela do software de controle, com o resultado do sétimo teste executado e a Figura 6.31, apresenta uma foto do caminho real percorrido pelo robô.

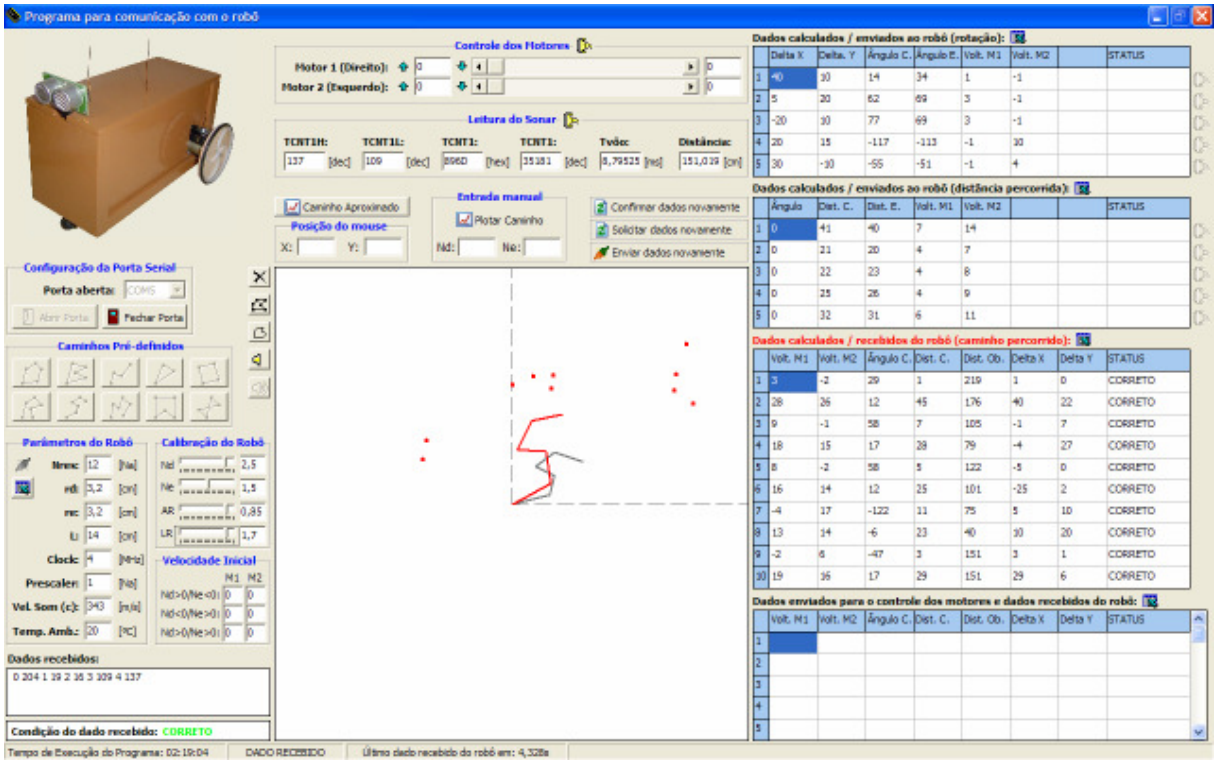


FIGURA 6.30 – TESTE #07 UTILIZANDO-SE A INTERFACE GRÁFICA

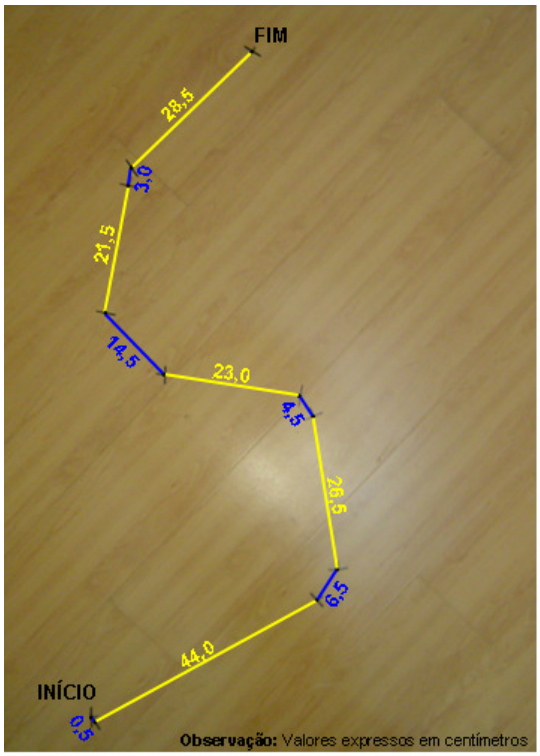


FIGURA 6.31 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #07) (ESCALA 1:13,33)

A Figura 6.32, apresenta a tela do software de controle, com o resultado do oitavo teste executado e a Figura 6.33, apresenta uma foto do caminho real percorrido pelo robô.

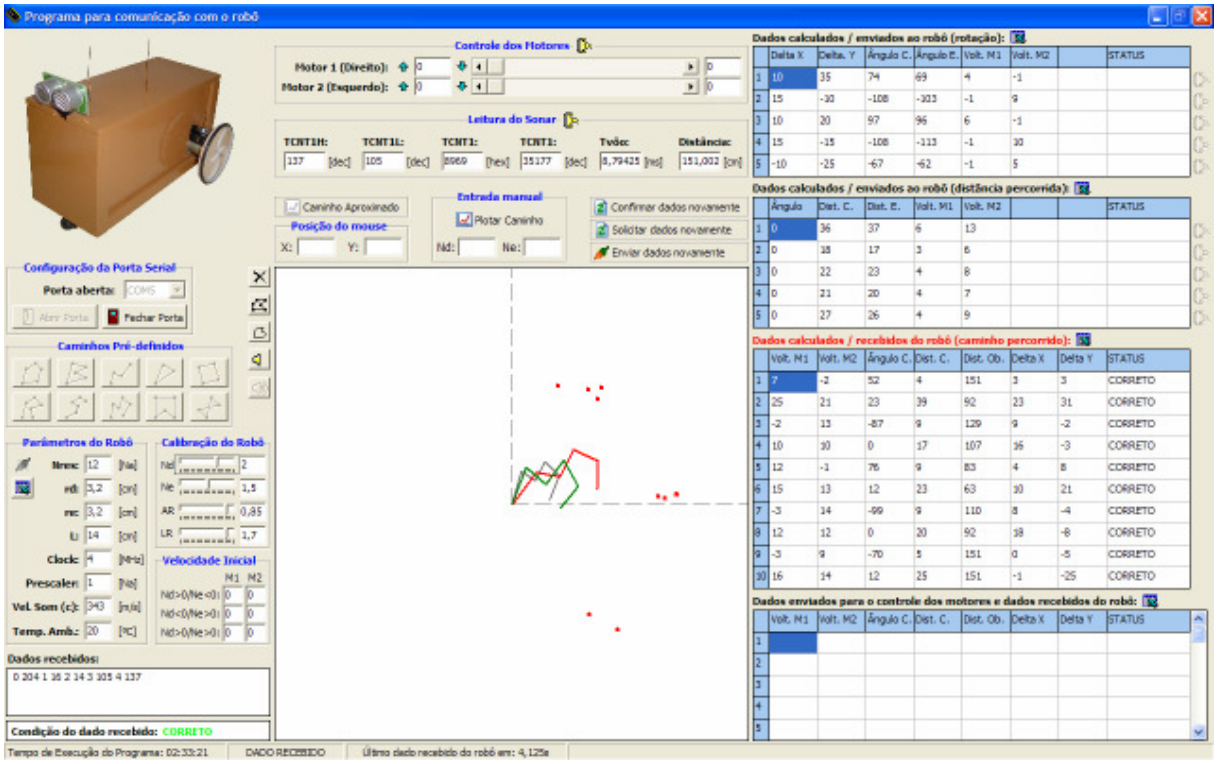


FIGURA 6.32 – TESTE #08 UTILIZANDO-SE A INTERFACE GRÁFICA

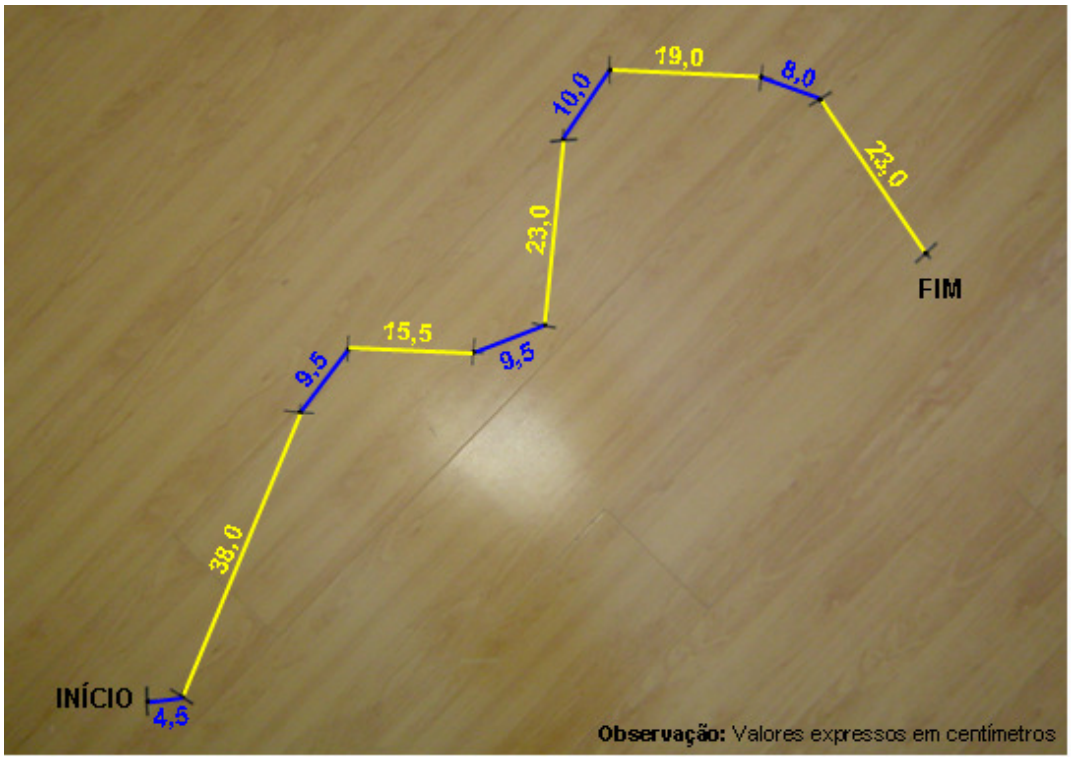


FIGURA 6.33 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #08) (ESCALA 1:9,5)

A Figura 6.34, apresenta a tela do software de controle, com o resultado do nono teste executado e a Figura 6.35, apresenta uma foto do caminho real percorrido pelo robô.

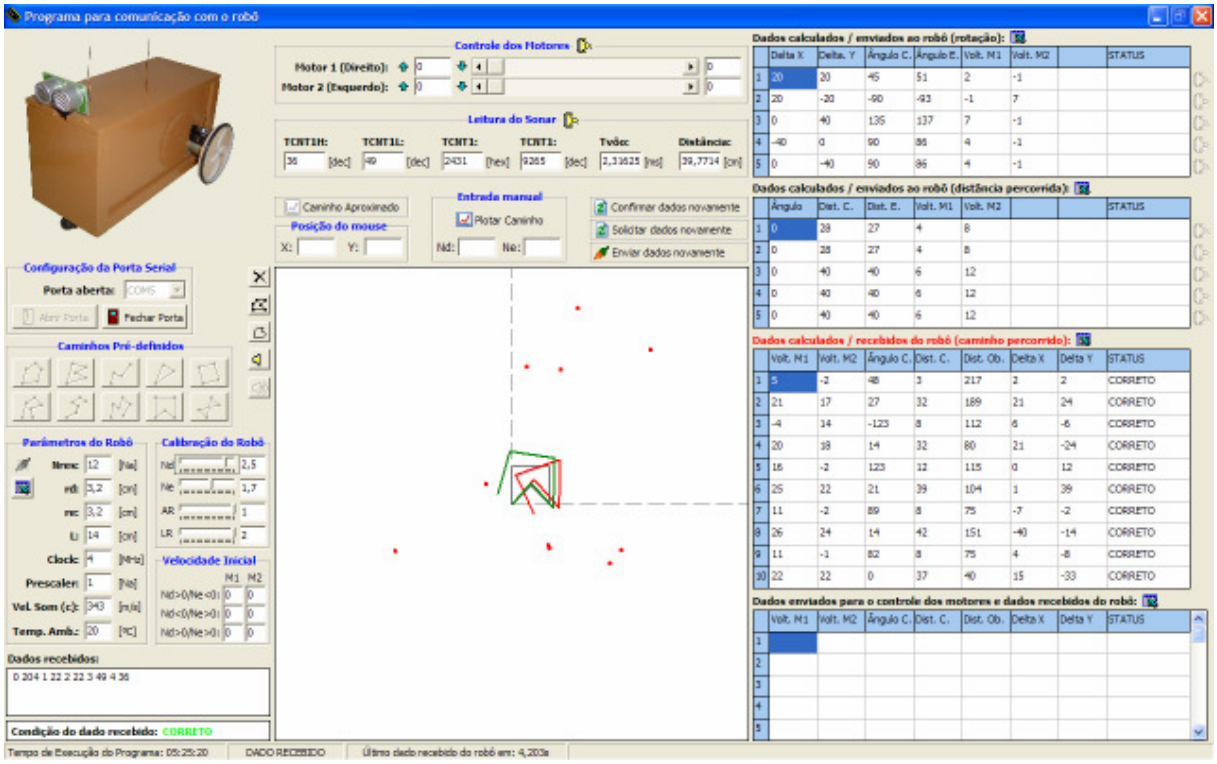


FIGURA 6.34 – TESTE #09 UTILIZANDO-SE A INTERFACE GRÁFICA

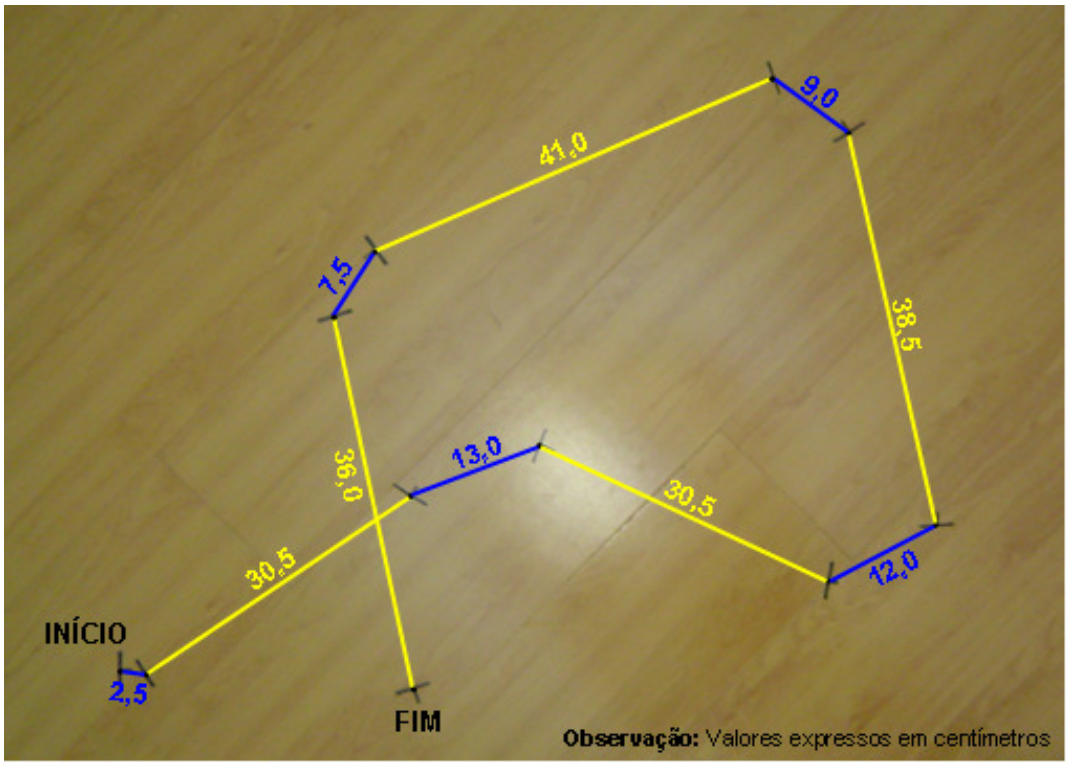


FIGURA 6.35 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #09) (ESCALA 1:7,32)

A Figura 6.36, apresenta a tela do software de controle, com o resultado do décimo teste executado e a Figura 6.37, apresenta uma foto do caminho real percorrido pelo robô.

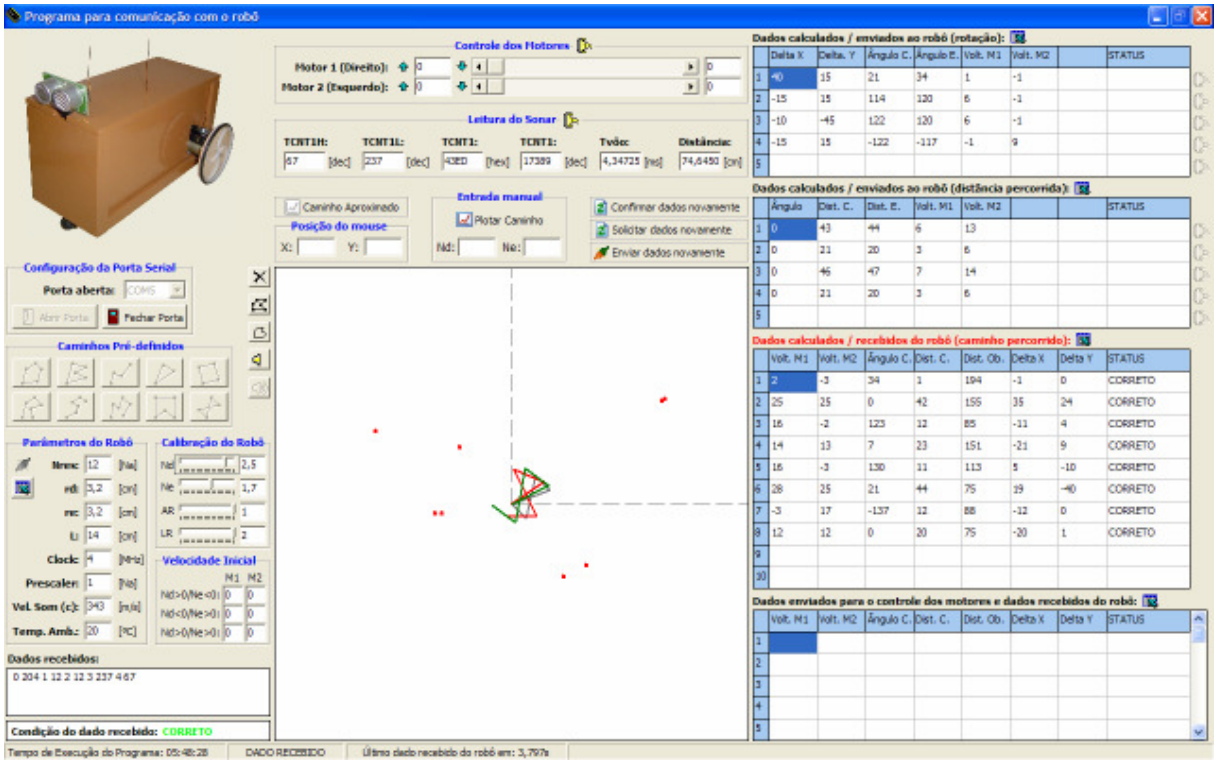


FIGURA 6.36 – TESTE #10 UTILIZANDO-SE A INTERFACE GRÁFICA

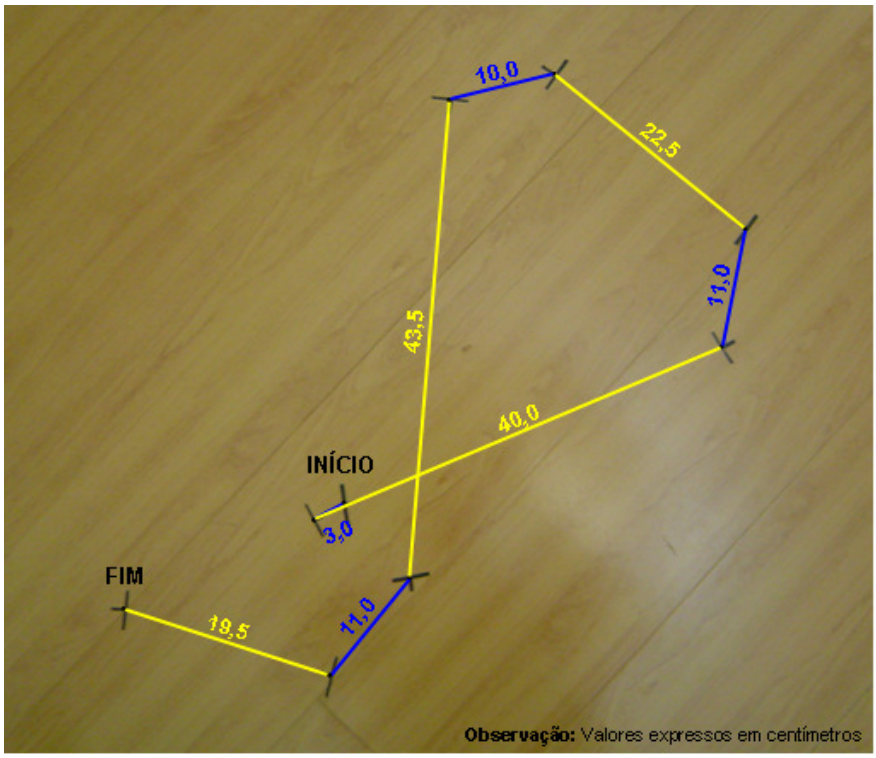


FIGURA 6.37 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #10) (ESCALA 1:7,02)

A Figura 6.38, apresenta a tela do software de controle, com o resultado do último teste executado e a Figura 6.39, apresenta uma foto do caminho real percorrido pelo robô.

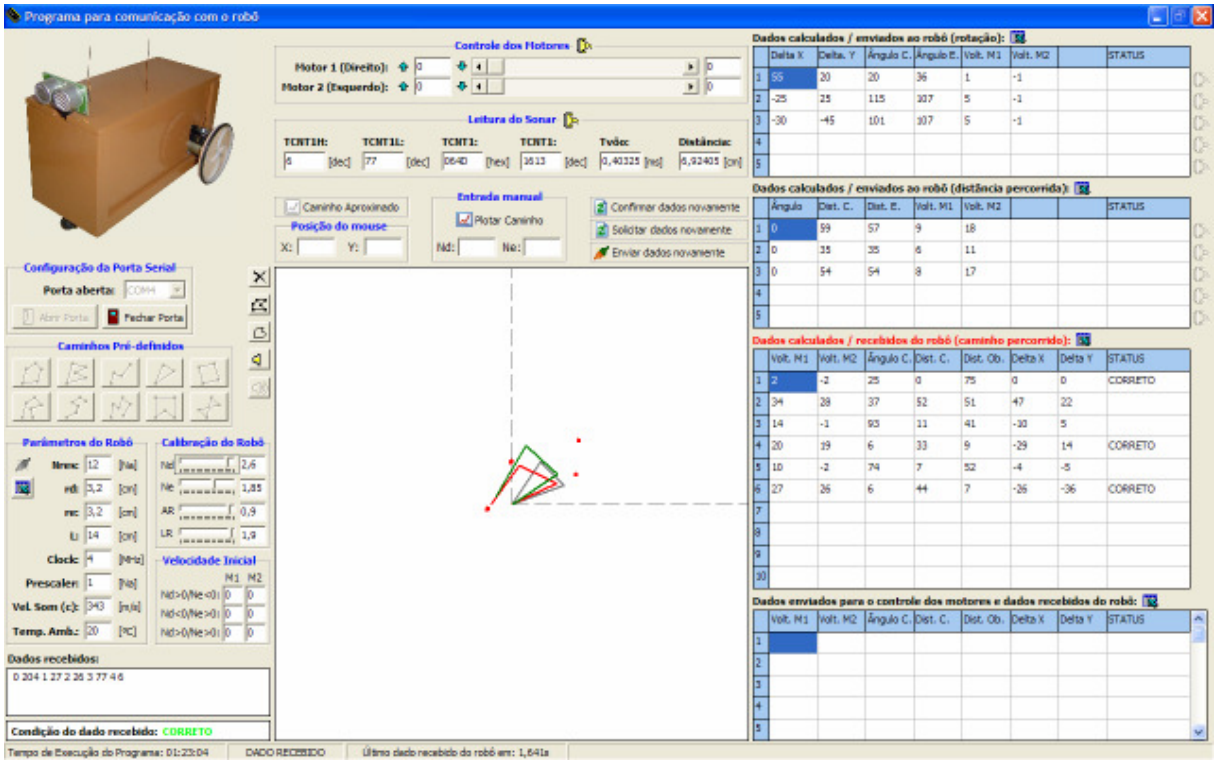


FIGURA 6.38 – TESTE #11 UTILIZANDO-SE A INTERFACE GRÁFICA

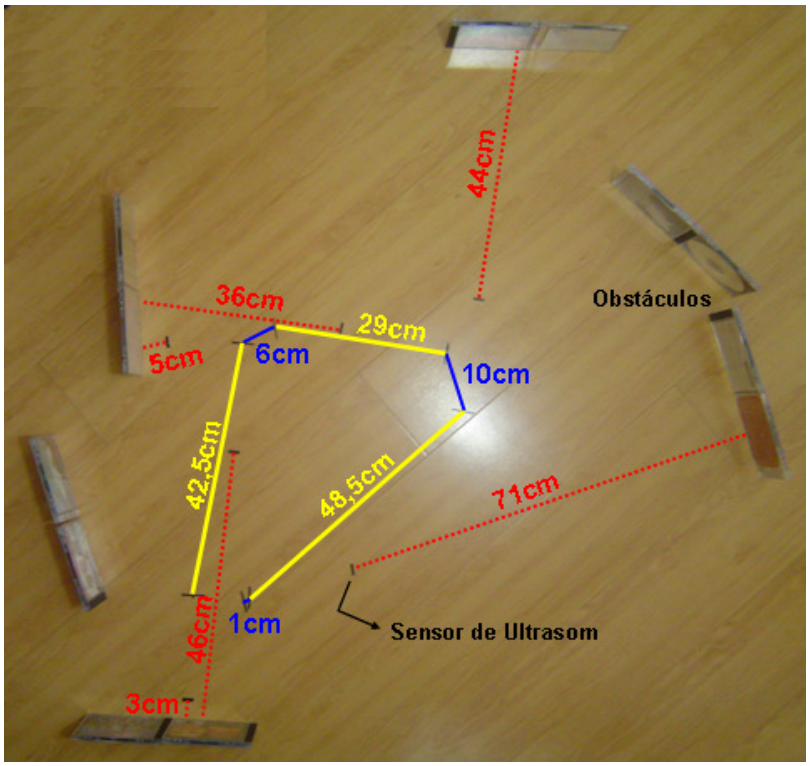


FIGURA 6.39 – CAMINHO REAL PERCORRIDO PELO ROBÔ (TESTE #11) (ESCALA 1:12,76)

A diferença entre este último teste e os demais testes apresentados anteriormente, está nos obstáculos adicionais incluídos no caminho do robô, com o objetivo de ilustrar o obstáculo detectado comparando os valores de distância calculados com base nos valores recebidos do robô, versus a distância real que o obstáculo se encontrava do robô (valores apresentados em vermelho na Figura 6.39).

6.5 PROGRAMAÇÃO DOS MICROCONTROLADORES

Apesar de algumas partes da lógica de programação utilizada nos microcontroladores já terem sido abordadas anteriormente, esta seção tem por objetivo sumarizar a lógica de programação utilizada tanto no circuito de IHM, quanto no circuito de controle do robô.

As Figuras 6.40 e 6.41, apresentam os fluxogramas correspondentes à lógica de programação do circuito de IHM e do robô respectivamente.

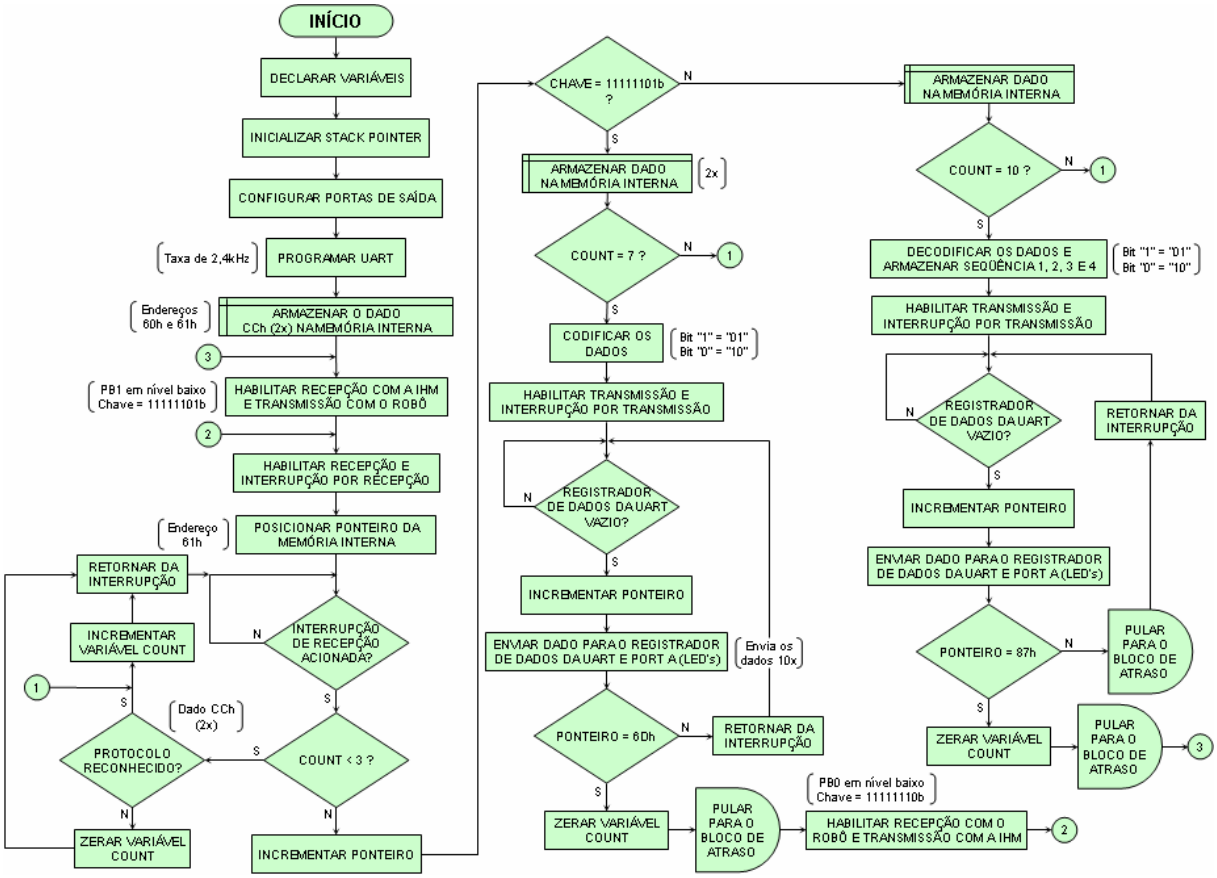


FIGURA 6.40 – FLUXOGRAMA DA LÓGICA DE PROGRAMAÇÃO DO CIRCUITO DE IHM

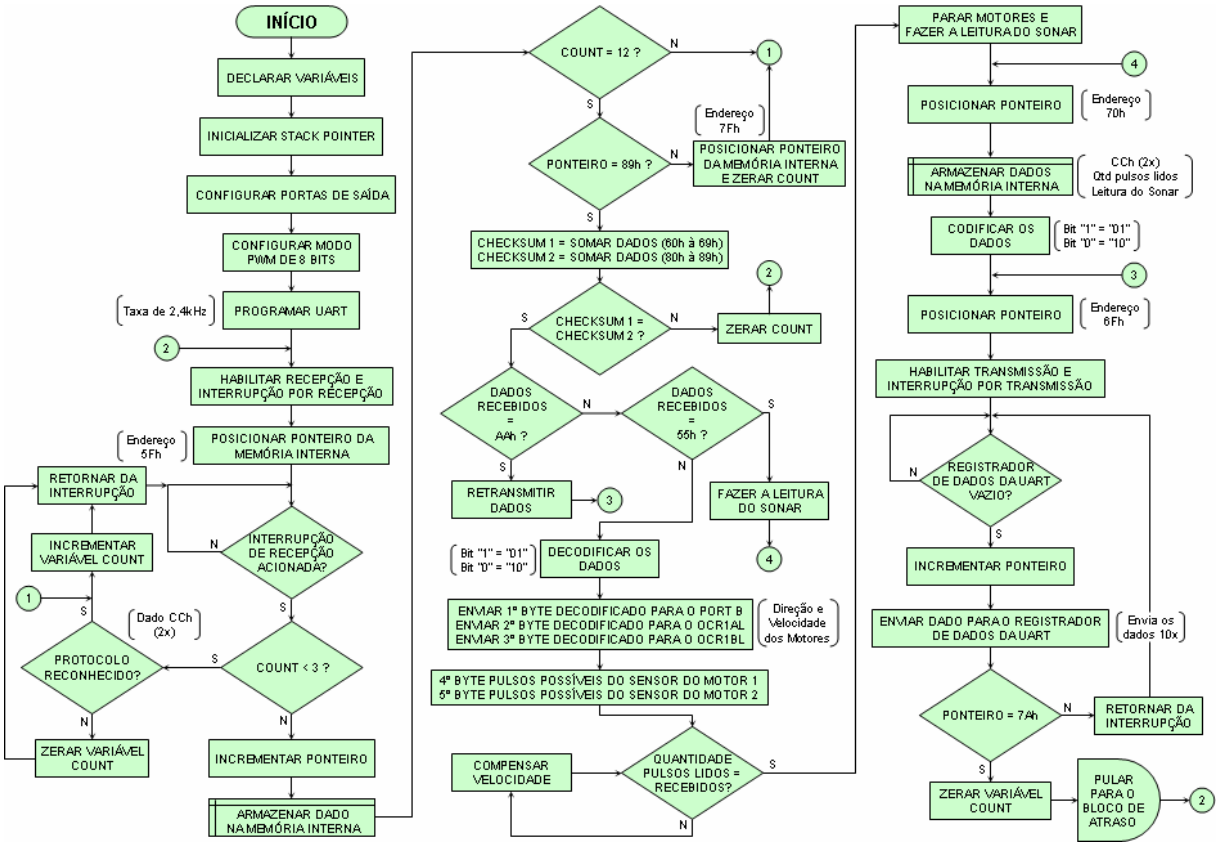


FIGURA 6.41 – FLUXOGRAMA DA LÓGICA DE PROGRAMAÇÃO DO CIRCUITO DO ROBÔ

Para maiores detalhes, os Anexos 10.1.13 e 10.1.14, apresentam a listagem final dos programas utilizados nos microcontroladores do circuito de IHM e do robô respectivamente.

7 SISTEMA DE COMUNICAÇÃO

O sistema de comunicação desenvolvido e utilizado no projeto pode ser dividido em duas partes:

1. c**omunicação serial:** entre a Interface Homem-Máquina (IHM), computador pessoal, e o circuito de IHM (com os módulos de comunicação por meio de radiofreqüência);
2. **comunicação bidirecional por meio de radiofreqüência:** é a comunicação utilizada entre o circuito de interface homem-máquina e o robô.

Basicamente, pode-se dizer que apenas duas informações trafegam entre o computador pessoal e o robô, são elas:

1. comandos enviados pelo computador pessoal para a movimentação do robô, ou para a leitura do sensor de ultrasom;
2. dados lidos pelos sensores que são enviados pelo robô ao computador pessoal, para o processamento desses dados.

7.1 COMUNICAÇÃO SERIAL

Como explicado anteriormente, a comunicação realizada entre o computador pessoal e o circuito de IHM é serial.

A taxa de transmissão utilizada é de 2.400 bits por segundo.

Nessa comunicação não houve preocupação em utilizar nenhum tipo de codificação que evitasse qualquer tipo de erro no envio/ recebimento dos dados. Apesar de alguns problemas de recepção de dados, conforme já abordado no capítulo anterior, existe sempre um padrão na informação recebida que pode ser identificado e o dado correto que deve ser considerado é possível de ser extraído. Outros testes realizados também com a comunicação serial, serão apresentados no final deste Capítulo.

7.1.1 Protocolo de Comunicação (PC ↔ Circuito de IHM)

A seguir, será dada uma breve explicação do protocolo de comunicação utilizado na comunicação serial entre o computador pessoal e o circuito de IHM.

7.1.1.1 Sentido do fluxo de dados (PC → Circuito de IHM)

Um total de sete bytes são enviados ao circuito de IHM através do computador pessoal.

Os dois primeiros bytes, CCh (11001100 em binário), são utilizados para identificar o início da comunicação entre o computador pessoal e o circuito de IHM.

O *nibble* menos significativo do terceiro byte é utilizado para a definição do sentido de rotação dos motores, mais precisamente, os bits 0 e 1 definem o sentido de rotação do motor 1 (esquerdo) e os bits 2 e 3 definem o sentido de rotação do motor 2 (direito). A Tabela 7.1 apresenta as possibilidades de movimentação do robô de acordo com os valores definidos por esses quatro bits e a Figura 7.1, procura ilustrar de uma maneira geral o explicado na tabela.

TABELA 7.1 – AÇÃO DOS MOTORES/ ROBÔ DE ACORDO COM O COMANDO RECEBIDO

Bits								**Motor 1 (Esquerdo)**	**Motor 2 (Direito)**	**Movimento do Robô[42]**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**			
X	X	X	X	0	0	0	0	Parado	Parado	Parado
X	X	X	X	0	0	0	1	Sentido horário	Parado	Gira para Esquerda
X	X	X	X	0	0	1	0	Sentido anti-horário	Parado	Gira para Direita
X	X	X	X	0	0	1	1	Parado	Parado	Parado
X	X	X	X	0	1	0	0	Parado	Sentido horário	Gira para Esquerda
X	X	X	X	0	1	0	1	Sentido horário	Sentido horário	Gira para Esquerda
X	X	X	X	0	1	1	0	Sentido anti-horário	Sentido horário	Frente
X	X	X	X	0	1	1	1	Parado	Sentido horário	Gira para Esquerda
X	X	X	X	1	0	0	0	Parado	Sentido anti-horário	Gira para Direita
X	X	X	X	1	0	0	1	Sentido horário	Sentido anti-horário	Trás
X	X	X	X	1	0	1	0	Sentido anti-horário	Sentido anti-horário	Gira para Direita
X	X	X	X	1	0	1	1	Parado	Sentido anti-horário	Gira para Direita

[42] Considerando a mesma velocidade de rotação dos motores 1 e 2

Bits								Motor 1 (Esquerdo)	Motor 2 (Direito)	Movimento do Robô[42]
7	6	5	4	3	2	1	0			
X	X	X	X	1	1	0	0	Parado	Parado	Parado
X	X	X	X	1	1	0	1	Sentido horário	Parado	Gira para Esquerda
X	X	X	X	1	1	1	0	Sentido anti-horário	Parado	Gira para Direita
X	X	X	X	1	1	1	1	Parado	Parado	Parado

Pelo fato de existir três estados possíveis para o motor (parado, girando no sentido horário ou girando no sentido anti-horário), leva-se à necessidade da utilização de pelo menos dois bits para cada motor, por essa razão é necessário o uso de pelo menos 4 bits para a definição correta do movimento a ser executado pelo robô.

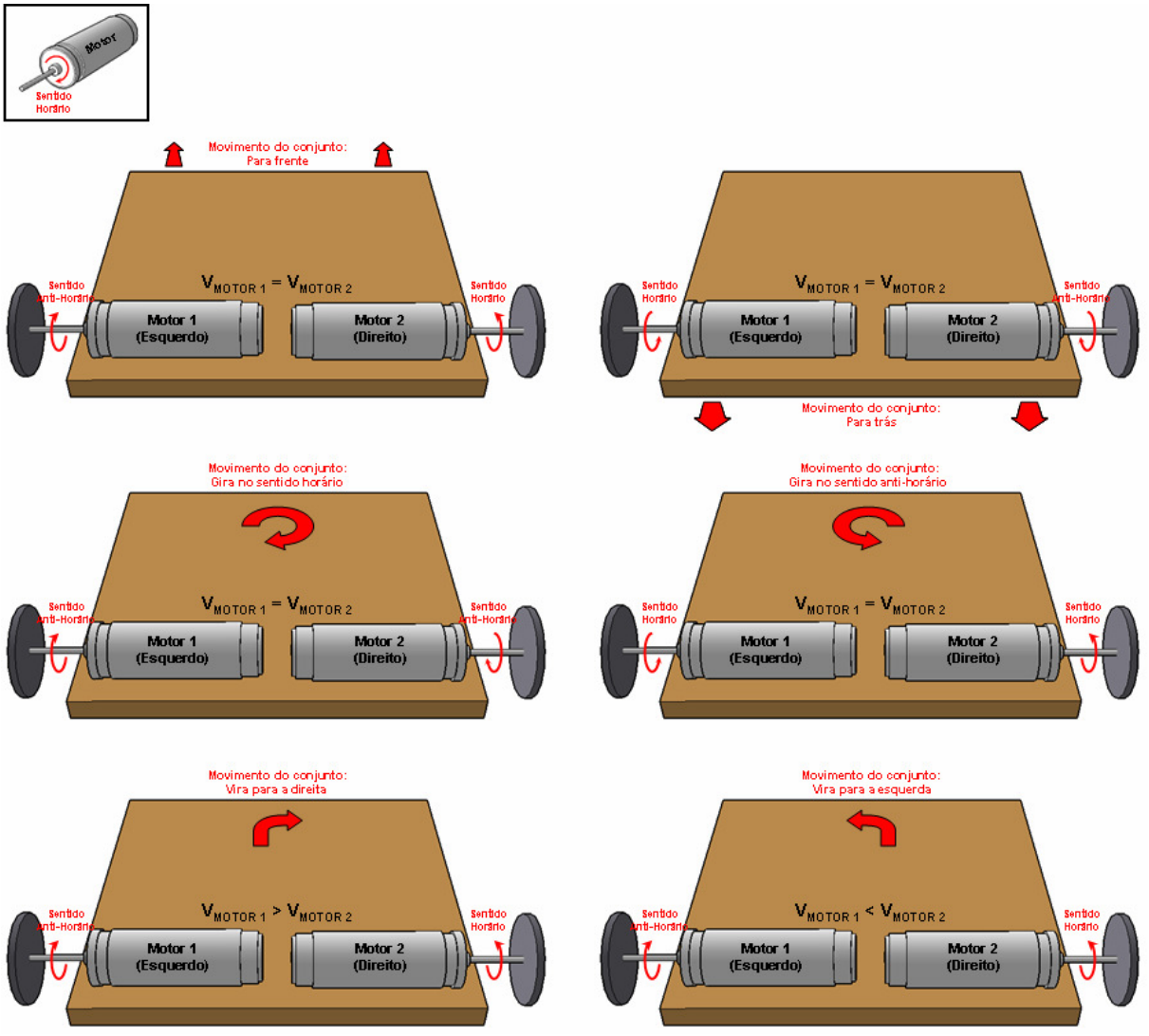


FIGURA 7.1 – AÇÃO DOS MOTORES/ ROBÔ DE ACORDO COM O COMANDO RECEBIDO

O quarto e quinto bytes definem a velocidade de rotação dos motores 1 e 2 respectivamente e o sexto e sétimo bytes, definem a quantidade de pulsos que os sensores de velocidade localizados próximo às rodas devem detectar.

Depois de confirmado o início da recepção (byte CChx2 reconhecido), o microcontrolador irá armazenar os cinco próximos bytes enviados pelo computador pessoal.

A Figura 7.2 ilustra o pacote de informações transmitido do computador pessoal ao circuito de interface homem-máquina.

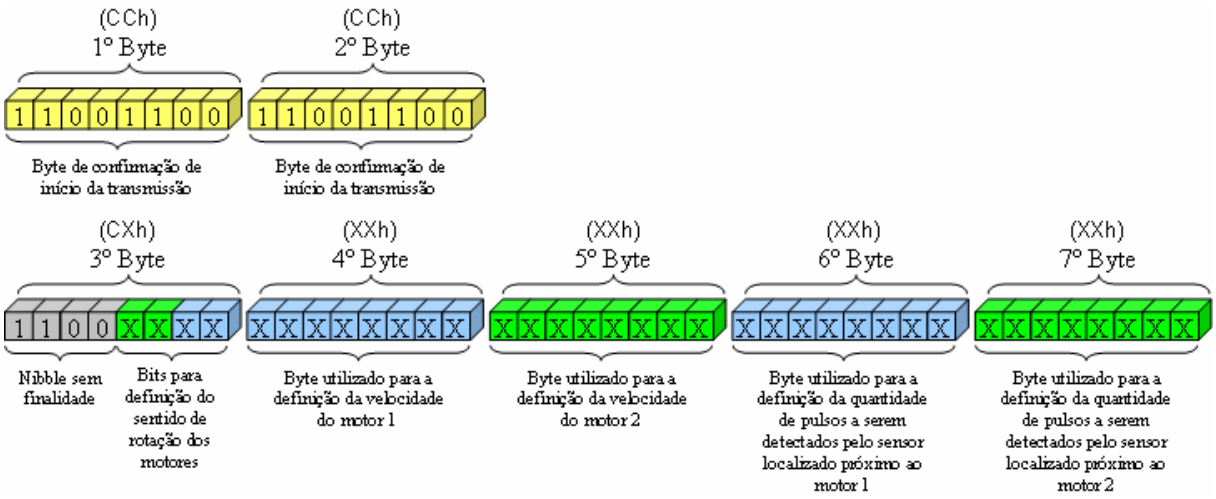


FIGURA 7.2 – PACOTE DE DADOS TRANSMITIDO DO PC AO CIRCUITO DE IHM (PADRÃO #01)

Além desse padrão de dados a serem transmitidos entre o computador pessoal e o circuito de IHM, existem outros dois padrões definidos, sendo:

1. leitura do sensor de ultrasom, utilizado para confirmar a distância que o obstáculo está do robô. A Figura 7.3 ilustra o protocolo de comunicação utilizado;
2. para que seja solicitado ao robô o reenvio do último dado transmitido por ele, este que tem como objetivo confirmar os dados recebidos anteriormente, ou que por algum motivo, não foram detectados pelo circuito de recepção do módulo de IHM. A Figura 7.4 ilustra neste caso, o protocolo de comunicação utilizado.

Pela Figura 7.3, é possível observar que o padrão utilizado para solicitar ao robô a leitura do sensor de ultrasom é através de uma seqüência de 5 bytes iguais à FF em hexadecimal, ou seja, 255 em decimal.

Pela Figura 7.4, é possível observar que o padrão utilizado para solicitar ao robô o reenvio do último dado transmitido por ele, é através de uma seqüência de 5 bytes iguais à 00 em hexadecimal, ou seja, 0 em decimal.

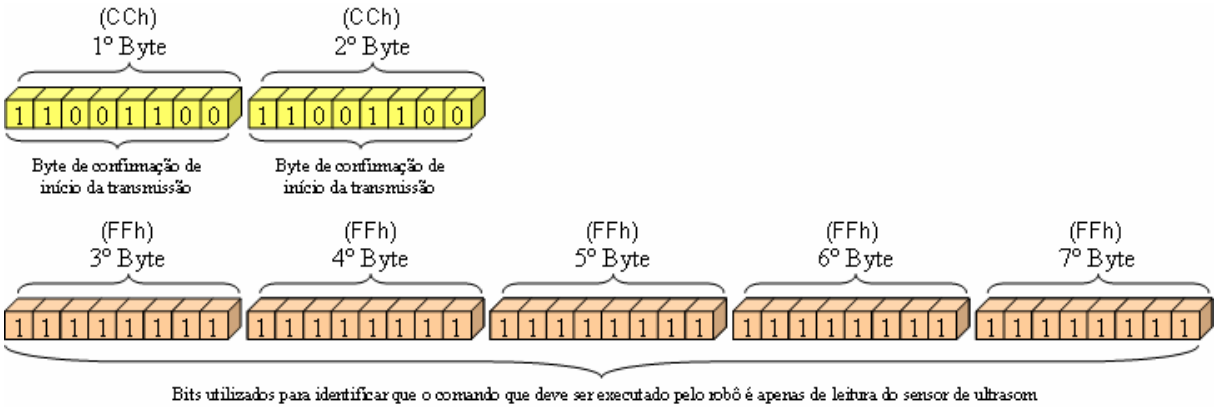


FIGURA 7.3 – PACOTE DE DADOS TRANSMITIDO DO PC AO CIRCUITO DE IHM (PADRÃO #02)

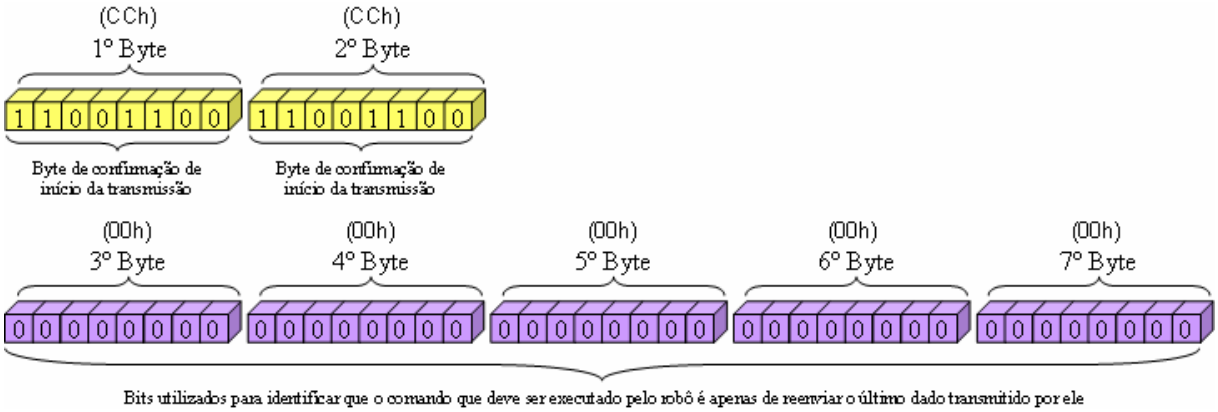


FIGURA 7.4 – PACOTE DE DADOS TRANSMITIDO DO PC AO CIRCUITO DE IHM (PADRÃO #03)

7.1.1.2 Sentido do fluxo de dados (PC ← Circuito de IHM)

Nesse caso, um total de nove bytes são enviados ao computador pessoal.

O primeiro byte é utilizado para identificar o início da transmissão entre o computador pessoal e o circuito de interface homem-máquina.

O segundo, quarto, sexto e oitavo bytes, são os números 1, 2, 3 e 4 respectivamente, pois conforme já explicado no capítulo anterior, essa seqüência numérica é importante para identificar o dado correto enviado pelo circuito de IHM, quando existe alguma falha durante a comunicação serial entre este circuito e o microcomputador.

O terceiro e quinto bytes estão relacionados com a quantidade de pulsos lidos pelos sensores de velocidade localizados próximos aos motores 1 e 2 respectivamente.

O sétimo e nono bytes, contemplam respectivamente o valor do byte mais e menos significativo do registrador de 16 bits, com o tempo medido pelo microcontrolador, referente ao "tempo de vôo" da onda sonora emitida pelo sensor de ultrasom.

A Figura 7.5 ilustra o pacote de informações transmitido do circuito de interface homem-máquina ao computador pessoal.

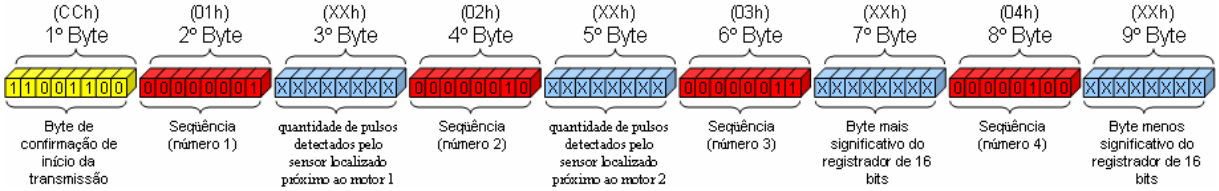


FIGURA 7.5 – PACOTE DE DADOS TRANSMITIDO DO CIRCUITO DE IHM AO PC

7.2 COMUNICAÇÃO BIDIRECIONAL VIA RF

Nessa comunicação, apesar da freqüência de trabalho dos módulos de comunicação ser alta, 433,92MHz, o que proporciona uma boa imunidade a ruídos e interferências, constatou-se que o envio de dados com seqüências de bits 1 ou 0, maior do que 4 bits apresentavam uma grande quantidade de erro na recepção dos dados.

Com o intuito de minimizar esse erro, utilizou-se uma técnica de codificação dos dados conforme apresentado na Tabela 7.2.

TABELA 7.2 – RELAÇÃO ENTRE BITS DE DADOS ORIGINAIS E CODIFICADOS

Bit Original	**Bit Codificado**
0	10
1	01

Pode ser observado que um bit 0 é substituído por dois bits 10 e um bit 1 é substituído por dois bits 01, logo é possível observar que a quantidade de bits a ser enviado é duplicada.

Esse tipo de codificação é bem semelhante ao método de codificação de dados do tipo Manchester[43], muito utilizado na comunicação de dados.

Na codificação Manchester, cada bit é dividido em dois intervalos. Um bit 1 é codificado utilizando uma transição no meio do intervalo, iniciando com nível alto e, após a transição, nível baixo. Na codificação do bit 0 ocorre o inverso, iniciando com nível baixo e, após a transição, nível alto. Esse esquema garante que todo bit tenha uma transição no meio do intervalo total, tornando fácil à estação receptora a sincronização com a estação transmissora.

A Figura 7.6, exemplifica a codificação conforme explicado anteriormente.

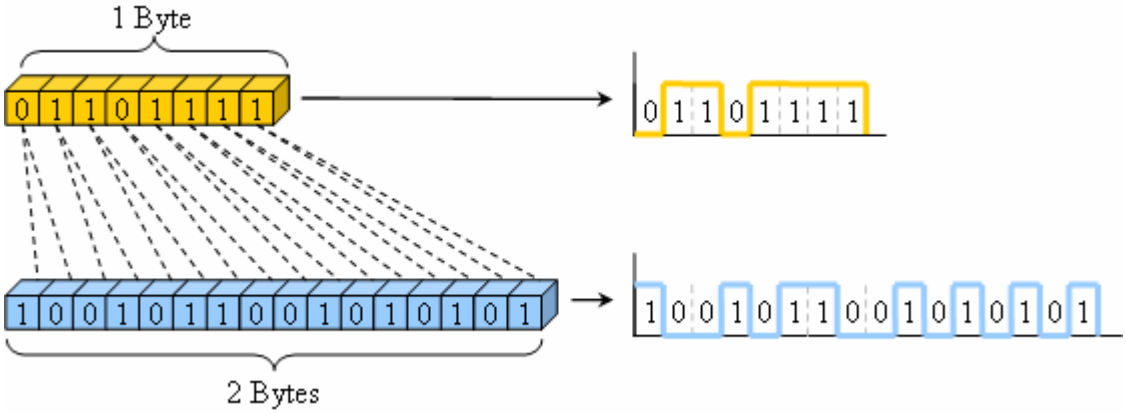


FIGURA 7.6 – EXEMPLO DE CODIFICAÇÃO DOS DADOS

É possível observar que utilizando-se essa técnica de codificação a maior seqüência possível de bits 0's ou 1's é igual a 2, dessa forma, o problema identificado anteriormente na comunicação devido a seqüências de bits iguais maiores do que 4 bits deixam de existir.

Na Tabela 7.3, é possível observar a relação entre todos os valores codificados e decodificados, ou seja, em sua forma original.

A Figura 7.7, tem como objetivo ilustrar o processo de codificação e decodificação (número 8 como exemplo), adotado na lógica de programação dos microcontroladores, tanto do circuito do robô, quanto do circuito de IHM. Neste exemplo, é possível observar a transição do valor inicial em sua forma original, para a forma final codificada e vice-versa.

[43] FONTE: http://en.wikipedia.org/wiki/Manchester_code

TABELA 7.3 – VALORES CODIFICADOS VS. VALORES DECODIFICADOS

DECODIFICADO/ ORIGINAL				CODIFICADO			
DEC	HEX	BIN	Curva	DEC	HEX	BIN	Curva
0	0	0000	0 0 0 0	170	AA	10101010	1 0 1 0 1 0 1 0
1	1	0001	0 0 0 1	169	A9	10101001	1 0 1 0 1 0 0 1
2	2	0010	0 0 1 0	166	A6	10100110	1 0 1 0 0 1 1 0
3	3	0011	0 0 1 1	165	A5	10100101	1 0 1 0 0 1 0 0
4	4	0100	0 1 0 0	154	9A	10011010	1 0 0 1 1 0 1 0
5	5	0101	0 1 0 1	153	99	10011001	1 0 0 1 1 0 0 1
6	6	0110	0 1 1 0	150	96	10010110	1 0 0 1 0 1 1 0
7	7	0111	0 1 1 1	149	95	10010101	1 0 0 1 0 1 0 1
8	8	1000	1 0 0 0	106	6A	01101010	0 1 1 0 1 0 1 0
9	9	1001	1 0 0 1	105	69	01101001	0 1 1 0 1 0 0 1
10	A	1010	1 0 1 0	102	66	01100110	0 1 1 0 0 1 1 0
11	B	1011	1 0 1 1	101	65	01100101	0 1 1 0 0 1 0 1
12	C	1100	1 1 0 0	90	5A	01011010	0 1 0 1 1 0 1 0
13	D	1101	1 1 0 1	89	59	01011001	0 1 0 1 1 0 0 1
14	E	1110	1 1 1 0	86	56	01010110	0 1 0 1 0 1 1 0
15	F	1111	1 1 1 1	85	55	01010101	0 1 0 1 0 1 0 1

PROCESSO DE CODIFICAÇÃO

Comando Assembler		Variável	BIN								HEX	DEC
		temp =	0	0	0	0	1	0	0	0	8	8
andi	temp,0x0F	temp =	0	0	0	0	1	0	0	0	08	8
clr	final	final =	0	0	0	0	0	0	0	0	00	0
mov	temp1,temp	temp1 =	0	0	0	0	1	0	0	0	08	8
com	temp	temp =	1	1	1	1	0	1	1	1	F7	247
lsl	temp	temp =	1	1	1	0	1	1	1	0	EE	238
mov	desloc,temp	desloc =	1	1	1	0	1	1	1	0	EE	238
mov	desloc1,temp1	desloc1 =	0	0	0	0	1	0	0	0	08	8
andi	desloc1,0x01	desloc1 =	0	0	0	0	0	0	0	0	00	0
andi	desloc,0x02	desloc =	0	0	0	0	0	0	1	0	02	2
add	desloc,desloc1	desloc =	0	0	0	0	0	0	1	0	02	2
add	final,desloc	final =	0	0	0	0	0	0	1	0	02	2
lsl	temp	temp =	1	1	0	1	1	1	0	0	DC	220
lsl	temp1	temp1 =	0	0	0	1	0	0	0	0	10	16
mov	desloc,temp	desloc =	1	1	0	1	1	1	0	0	DC	220
mov	desloc1,temp1	desloc1 =	0	0	0	1	0	0	0	0	10	16
andi	desloc1,0x04	desloc1 =	0	0	0	0	0	0	0	0	00	0
andi	desloc,0x08	desloc =	0	0	0	0	1	0	0	0	08	8
add	desloc,desloc1	desloc =	0	0	0	0	1	0	0	0	08	8
add	final,desloc	final =	0	0	0	0	1	0	1	0	0A	10
lsl	temp	temp =	1	0	1	1	1	0	0	0	B8	184
lsl	temp1	temp1 =	0	0	1	0	0	0	0	0	20	32
mov	desloc,temp	desloc =	1	0	1	1	1	0	0	0	B8	184
mov	desloc1,temp1	desloc1 =	0	0	1	0	0	0	0	0	20	32
andi	desloc1,0x10	desloc1 =	0	0	0	0	0	0	0	0	00	0
andi	desloc,0x20	desloc =	0	0	1	0	0	0	0	0	20	32
add	desloc,desloc1	desloc =	0	0	1	0	0	0	0	0	20	32
add	final,desloc	final =	0	0	1	0	1	0	1	0	2A	42
lsl	temp	temp =	0	1	1	1	0	0	0	0	70	112
lsl	temp1	temp1 =	0	1	0	0	0	0	0	0	40	64
mov	desloc,temp	desloc =	0	1	1	1	0	0	0	0	70	112
mov	desloc1,temp1	desloc1 =	0	1	0	0	0	0	0	0	40	64
andi	desloc1,0x40	desloc1 =	0	1	0	0	0	0	0	0	40	64
andi	desloc,0x80	desloc =	0	0	0	0	0	0	0	0	00	0
add	desloc,desloc1	desloc =	0	1	0	0	0	0	0	0	40	64
add	final,desloc	final =	0	1	1	0	1	0	1	0	**6A**	**106**

PROCESSO DE DECODIFICAÇÃO

Comando Assembler		Variável	BIN								HEX	DEC
ld	temp,X	temp =	0	1	1	0	1	0	1	0	6A	106
clr	final	final =	0	0	0	0	0	0	0	0	00	0
mov	temp1,temp	temp1 =	0	1	1	0	1	0	1	0	6A	106
andi	temp,0x01	temp =	0	0	0	0	0	0	0	0	00	0
add	final,temp	final =	0	0	0	0	0	0	0	0	00	0
lsr	temp1	temp1 =	0	0	1	1	0	1	0	1	35	53
mov	temp,temp1	temp =	0	0	1	1	0	1	0	1	35	53
andi	temp,0x02	temp =	0	0	0	0	0	0	0	0	00	0
add	final,temp	final =	0	0	0	0	0	0	0	0	00	0
lsr	temp1	temp1 =	0	0	0	1	1	0	1	0	1A	26
mov	temp,temp1	temp =	0	0	0	1	1	0	1	0	1A	26
andi	temp,0x04	temp =	0	0	0	0	0	0	0	0	00	0
add	final,temp	final =	0	0	0	0	0	0	0	0	00	0
lsr	temp1	temp1 =	0	0	0	0	1	1	0	1	0D	13
mov	temp,temp1	temp =	0	0	0	0	1	1	0	1	0D	13
andi	temp,0x08	temp =	0	0	0	0	1	0	0	0	08	8
add	final,temp	final =	0	0	0	0	1	0	0	0	**08**	**8**

FIGURA 7.7 – PROCESSO DE CODIFICAÇÃO E DECODIFICAÇÃO DOS DADOS

7.2.1 Protocolo de Comunicação (Circuito de IHM ↔ Robô)

A seguir, será dada uma breve explicação do protocolo de comunicação utilizado na comunicação RF entre o circuito de interface homem-máquina e o robô.

7.2.1.1 Sentido do fluxo de dados (Circuito de IHM → Robô)

Além do processo de codificação aplicado no dado trafegado por meio de radiofreqüência, para evitar que o robô executasse algum comando que não fosse exatamente o comando requisitado pelo circuito de IHM, foi adicionado à lógica de programação do robô um sistema "*checksum*" [(44)].

O conceito é o seguinte:

1. O circuito de IHM envia o comando para o robô no mínimo duas vezes;
2. Após o primeiro pacote de dados ser recebido pelo robô, os valores são somados e armazenados em uma variável;
3. Após o segundo pacote de dados ser recebido pelo robô, da mesma forma como anteriormente os valores são somados e posteriormente comparados com o resultado da soma do primeiro pacote.
4. Caso o resultado da comparação seja de valores iguais o robô executa o comando recebido, caso contrário, aguarda uma nova seqüência de dados do circuito de IHM.

No fluxo de dados do circuito de IHM para o robô, um total de doze bytes codificados são enviados ao robô.

Os dois primeiros bytes, CCh (11001100 em binário), são utilizados para identificar o início da comunicação entre o circuito de IHM e o robô.

O terceiro e o quarto bytes, são utilizados para a definição do sentido de rotação dos motores, mais precisamente, o *nibble* menos significativo do terceiro byte define o sentido de rotação do motor 1 (direito) e o *nibble* mais significativo define o sentido de rotação do motor 2 (esquerdo).

---

[44] FONTE: http://en.wikipedia.org/wiki/Checksum (O termo *checksum* é geralmente utilizado, quando pretende se referir a algum tipo de cheque redundante.

O quinto e sexto bytes, definem a velocidade de rotação do motor 1, enquanto que o sétimo e oitavo bytes definem a velocidade de rotação do motor 2.

O nono e décimo bytes, definem o valor de *Nd*, ou seja, a quantidade de variações (ímãs detectados) que o sensor do lado direito deve detectar para que em seguida, o comando de parada do motor direito seja executado.

Os dois últimos bytes, definem o valor de *Ne*, ou seja, a quantidade de variações (ímãs detectados) que o sensor do lado esquerdo deve detectar para que em seguida, o comando de parada do motor esquerdo seja executado.

Depois de confirmado o início da recepção (byte CCh reconhecido) e o processo de *checksum*, os dados recebidos serão decodificados.

A Figura 7.8 ilustra o pacote de informações original recebido do PC e o correspondente pacote de informações codificado a ser transmitido do circuito de interface homem-máquina ao circuito de comunicação do robô.

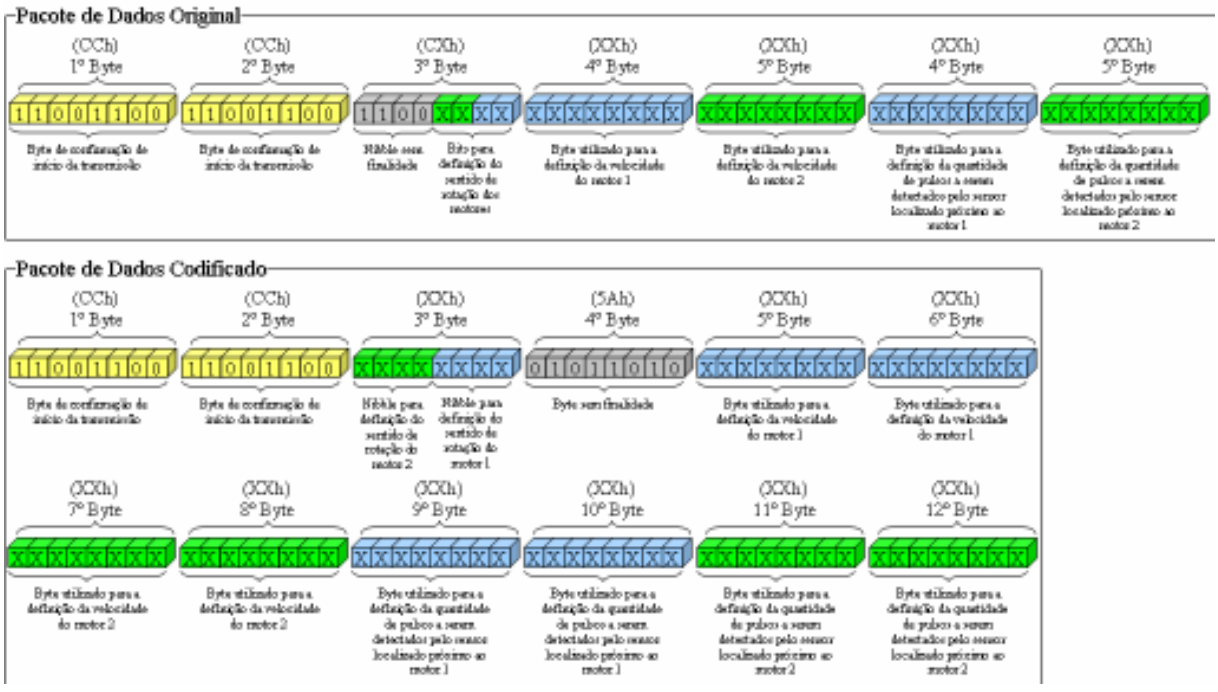


FIGURA 7.8 – PACOTE DE DADOS TRANSMITIDO DO CIRCUITO DE IHM AO ROBÔ (PADRÃO #01)

Da mesma forma que na trasmissão de dados entre o PC e o circuito de IHM, além desse padrão de dados a serem transmitidos, existem outros dois padrões definidos, sendo:

1. leitura do sensor de ultrasom, na qual a Figura 7.9, procura ilustrar os dados trafegados nessa situação, tanto em sua forma original, quanto codificados.
2. reenvio dos dados pelo robô, apresentados pela Figura 7.10.

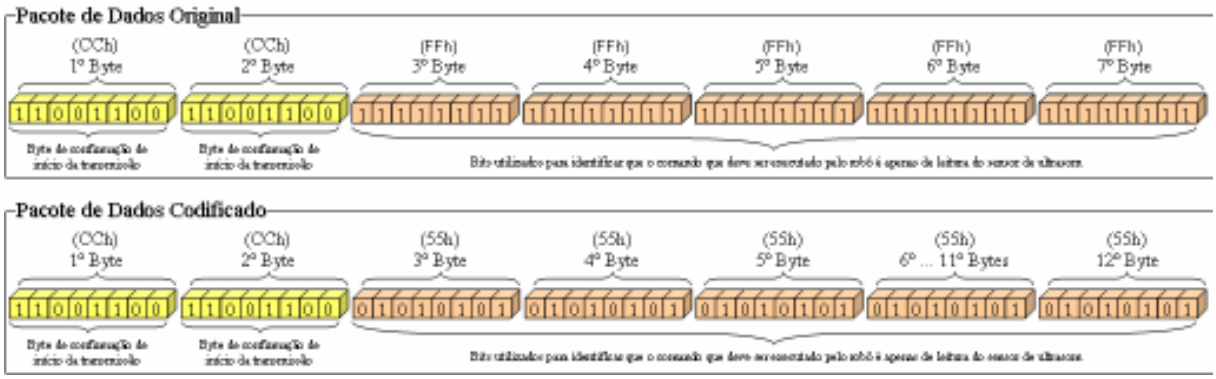


FIGURA 7.9 – PACOTE DE DADOS TRANSMITIDO DO CIRCUITO DE IHM AO ROBÔ (PADRÃO #02)

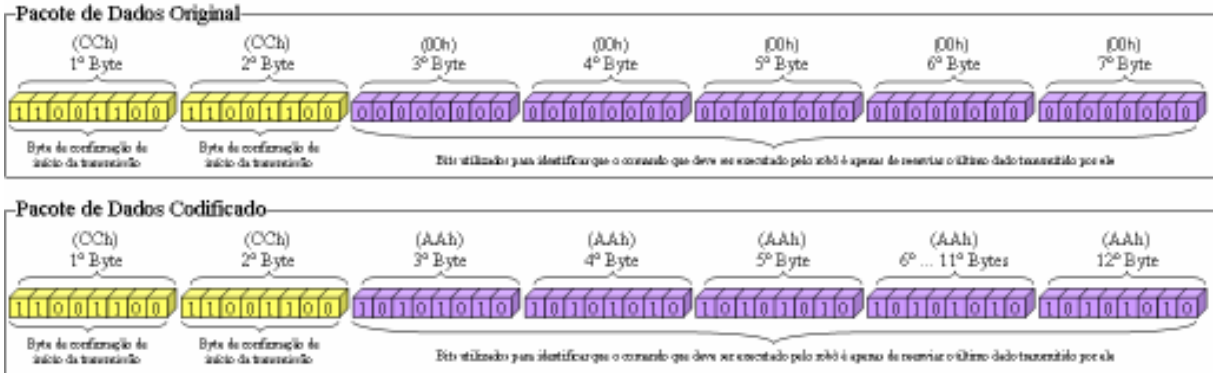


FIGURA 7.10 – PACOTE DE DADOS TRANSMITIDO DO CIRCUITO DE IHM AO ROBÔ (PADRÃO #03)

7.2.1.2 Sentido do fluxo de dados (Circuito de IHM ← Robô)

Um total de dez bytes codificados são enviados ao circuito de interface homem-máquina.

Os dois primeiros bytes, CCh (11001100 em binário), são utilizados para identificar o início da comunicação entre o circuito de IHM e o robô.

O terceiro e o quarto bytes estão relacionados com a quantidade de pulsos, ímãs detectados, pelo sensor de velocidade do lado direito, assim como o quinto e sexto bytes estão relacionados com o sensor do lado esquerdo.

O sétimo e oitavo bytes contemplam o valor do byte mais significativo do registrador de 16 bits, assim como o nono e décimo bytes contemplam o valor do byte menos significativo do registrador de 16 bits, este que possui o tempo medido pelo microcontrolador, referente ao “tempo de vôo” da onda sonora emitida pelo sensor de ultrasom.

A Figura 7.11 ilustra o pacote de informações original e o correspondente pacote de informações codificado a ser transmitido do circuito de transmissão do robô ao circuito de interface homem-máquina.

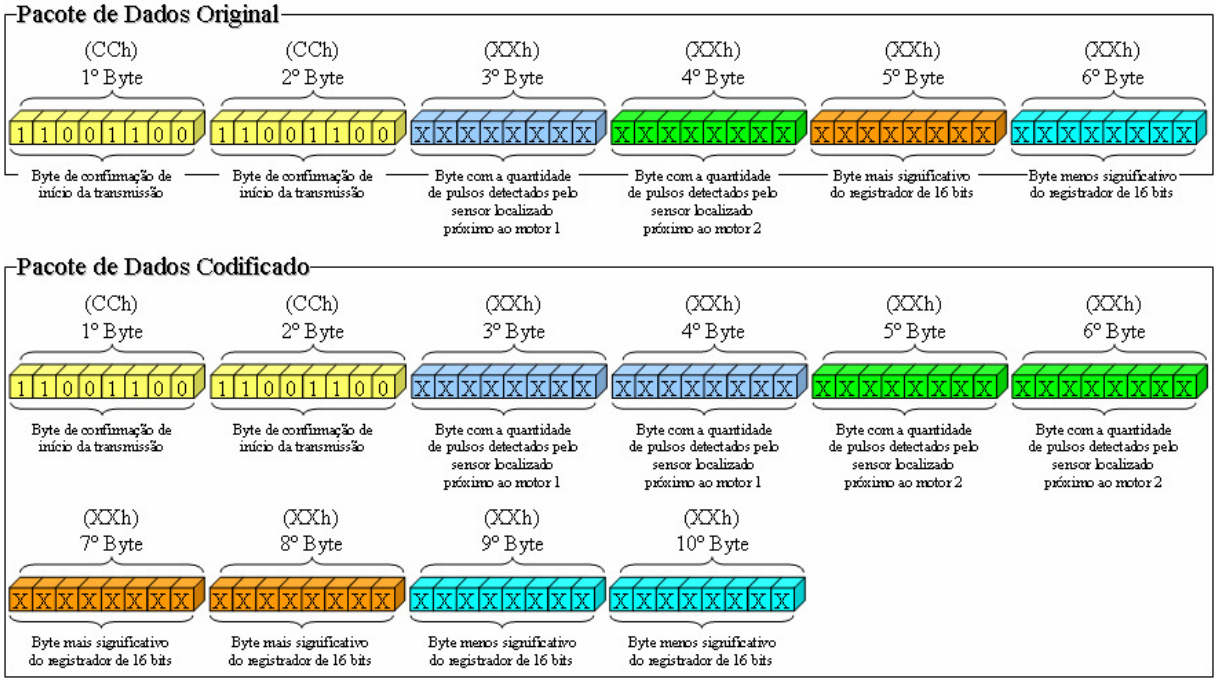


FIGURA 7.11 – PACOTE DE DADOS TRANSMITIDO DO ROBÔ AO CIRCUITO DE IHM

É importante esclarecer que o byte CCh (11001100 em binário, ou 204 em decimal), está sendo utilizado para o reconhecimento do pacote de dados, pois não existe a possibilidade de se ter um caractere codificado igual ao caractere CCh, além de que a seqüência máxima de bits 0's ou 1's nesse byte é igual a 2. Será possível observar que após diversos testes realizados e apresentados no final deste Capítulo, enviando uma seqüência de dois bytes igual a CCh para identificar o início da transmissão de dados entre os módulos de comunicação RF, e aplicando a codificação nos demais bytes, conforme explicado anteriormente, é mais do que suficiente para garantir uma comunicação por meio de radiofreqüência confiável.

Para se ter uma visão geral de todo o sistema de comunicação, a Figura 7.12 procura ilustrar o fluxo de dados trafegados entre o computador pessoal, o circuito de interface homem-máquina e o robô.

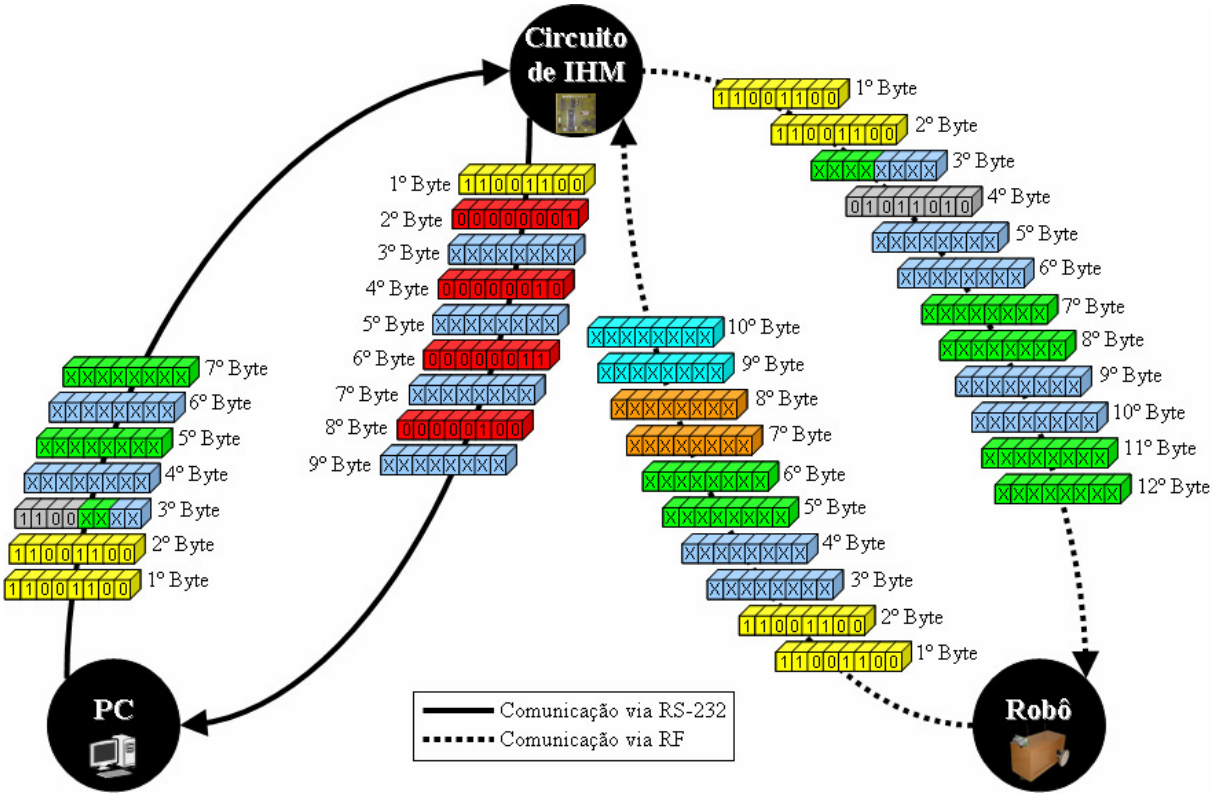


FIGURA 7.12 – VISÃO GERAL DO FLUXO DE DADOS TRAFEGADOS NO SISTEMA

7.3 ALGUNS TESTES DE COMUNICAÇÃO REALIZADOS

7.3.1 Comunicação Serial (Parte 1)

O objetivo principal desse teste foi o de confirmar o funcionamento do circuito elétrico proposto e da porta serial do computador, para tornar possível a comunicação entre o microcontrolador do circuito de transmissão/ recepção de dados para o robô e o computador pessoal (IHM).

O teste foi realizado, utilizando-se o Software RComSerial v1.1, obtido na Internet[45], a Figura 7.13, apresenta o software utilizado, assim como o resultado do teste executado.

[45] FONTE: http://www.rogercom.com (*Freeware*)

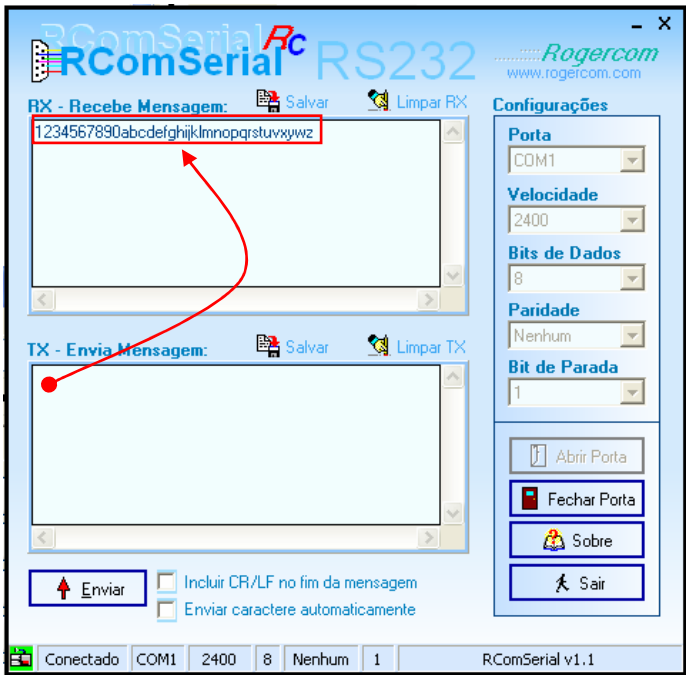


FIGURA 7.13 – TELA DO SOFTWARE RCOMSERIAL V1.1 E O RESULTADO OBTIDO (PARTE 1)

A Figura 7.14 apresenta o circuito elétrico correspondente ao teste realizado.

Nesse teste, não foi necessário nenhum tipo de desenvolvimento lógico, a nível de programação. Adotou-se o conceito de que o caractere a ser enviado, deveria ser o mesmo a ser recebido.

O resultado obtido foi satisfatório e conforme o esperado.

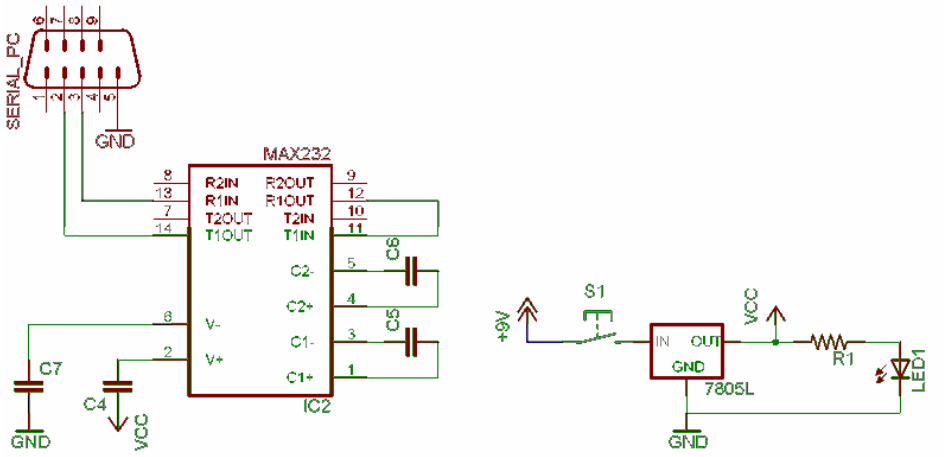


FIGURA 7.14 – DIAGRAMA ELÉTRICO DO CIRCUITO TESTE DA COMUNICAÇÃO SERIAL (PARTE 1)

7.3.2 Comunicação Serial (Parte 2)

O objetivo principal desse teste foi o de confirmar o funcionamento do circuito elétrico proposto e da lógica de programação desenvolvida para o microcontrolador, tornando possível a comunicação entre o microcontrolador do circuito de transmissão/ recepção de dados para o robô e o computador pessoal (IHM).

O teste foi realizado, utilizando-se o software RComSerial v1.1, o mesmo utilizado no teste anterior. A Figura 7.15, apresenta o software utilizado, assim como o resultado do teste executado.

FIGURA 7.15 – TELA DO SOFTWARE RCOMSERIAL V1.1 E O RESULTADO OBTIDO (PARTE 2)

A Figura 7.16 apresenta o circuito elétrico correspondente ao teste realizado, assim como na Figura 7.17 é apresentado o fluxograma correspondente à lógica de programação utilizada no teste executado.

Através do fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.1.

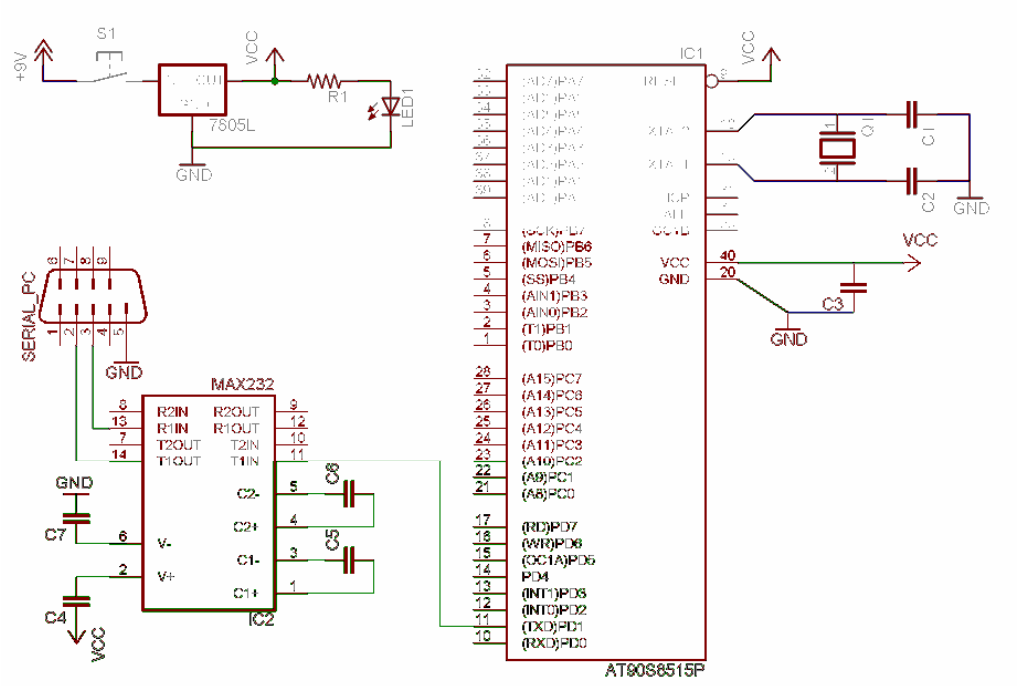


FIGURA 7.16 – DIAGRAMA ELÉTRICO DO CIRCUITO TESTE DA COMUNICAÇÃO SERIAL (PARTE 2)

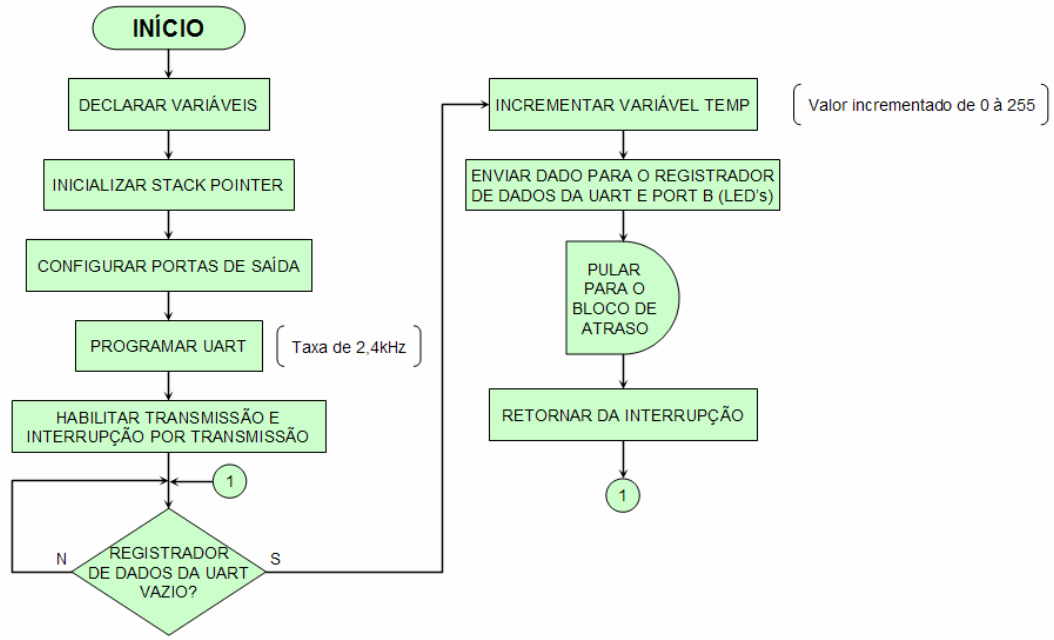


FIGURA 7.17 – FLUXOGRAMA DA LÓGICA UTILIZADA NA COMUNICAÇÃO SERIAL (PARTE 2)

Nesse teste, foi desenvolvida a primeira parte da lógica de programação do circuito de interface com o usuário, é através dessa comunicação serial, que o computador pessoal receberá os dados dos sensores do robô, para que essas informações sejam processadas e novos dados enviados ao robô para que ele realize determinada ação.

O resultado obtido foi satisfatório e conforme o esperado.

7.3.3 Comunicação Serial (Parte 3)

O objetivo principal desse teste foi o de confirmar o funcionamento da lógica de programação desenvolvida em Delphi® para o computador pessoal, para tornar possível a visualização e confirmação dos dados enviados pelo microcontrolador.

O circuito elétrico e a lógica de programação desenvolvida para o microcontrolador não sofreram alteração em relação ao teste apresentado anteriormente (Comunicação Serial – Parte 2).

A Figura 7.18 apresenta o fluxograma correspondente à lógica de programação utilizada no teste executado. A Figura 7.19 ilustra a tela do programa em execução.

No fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.2.

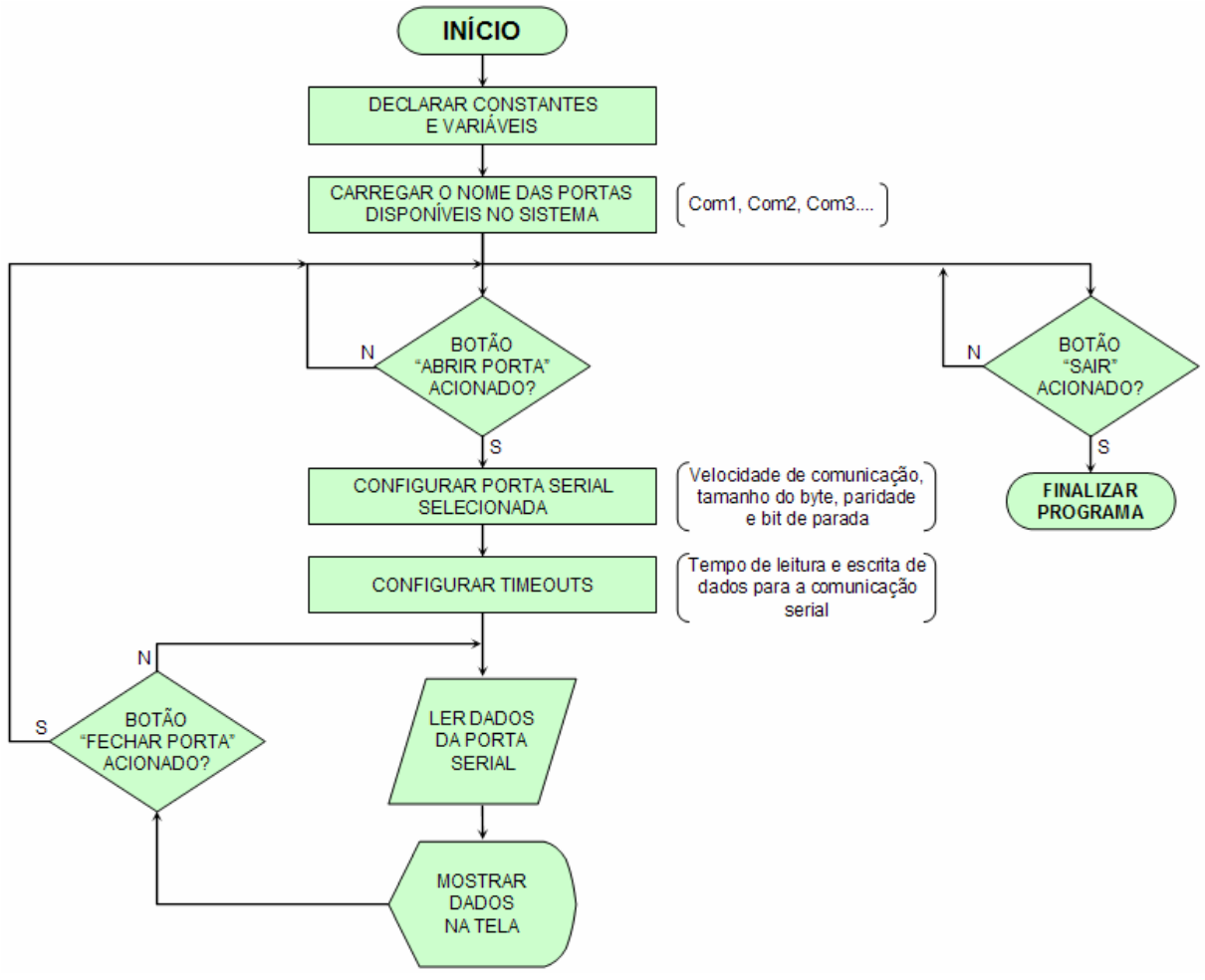


FIGURA 7.18 – FLUXOGRAMA DA LÓGICA UTILIZADA NA COMUNICAÇÃO SERIAL (PARTE 3)

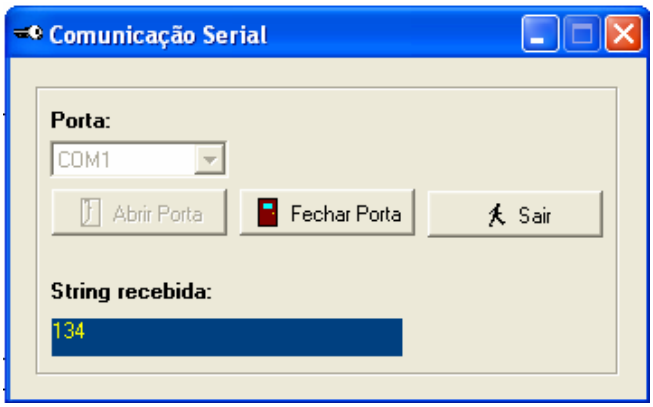


FIGURA 7.19 – TELA EM EXECUÇÃO DO PROGRAMA DESENVOLVIDO

Esse teste foi importante para confirmar que os dados que estavam sendo enviados pelo microcontrolador seriam apresentados fielmente ao usuário. A lógica de programação desenvolvida no Delphi® apresentou um resultado satisfatório e conforme o esperado.

7.3.4 Comunicação Unidirecional Via RF (Parte 1)

Esse teste teve como objetivo, a confirmação do funcionamento da transmissão de dados entre os módulos de comunicação.

Inicialmente, utilizando-se um gerador de funções, uma onda quadrada foi transmitida pelo módulo de transmissão e recebida pelo módulo de recepção.

A onda quadrada transmitida, juntamente com a onda quadrada recebida, pode ser verificada pelo uso de um osciloscópio, conforme ilustrado na Figura 7.20.

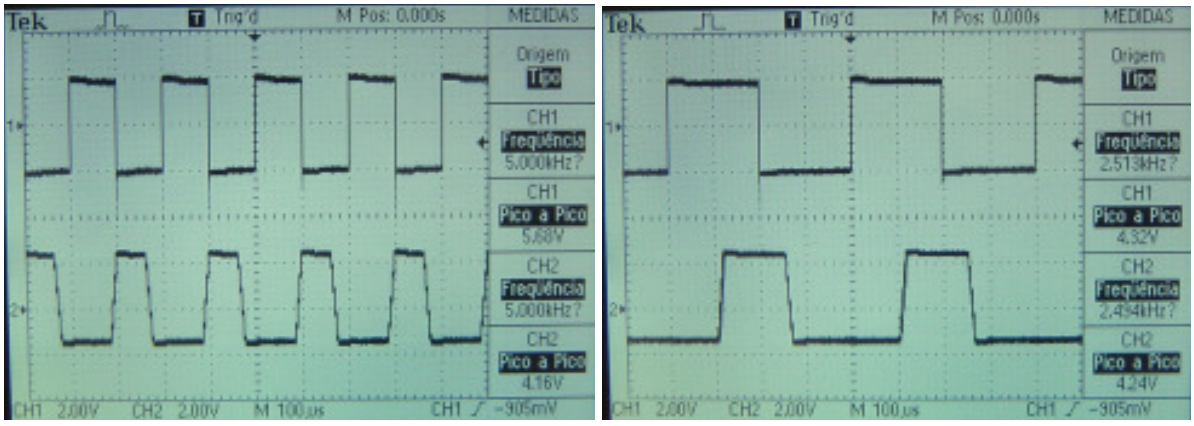


FIGURA 7.20 – ONDA QUADRADA TRANSMITIDA (CH1) VS. ONDA QUADRADA RECEBIDA (CH2)

A Figura 7.21 apresenta o circuito elétrico correspondente ao teste realizado.

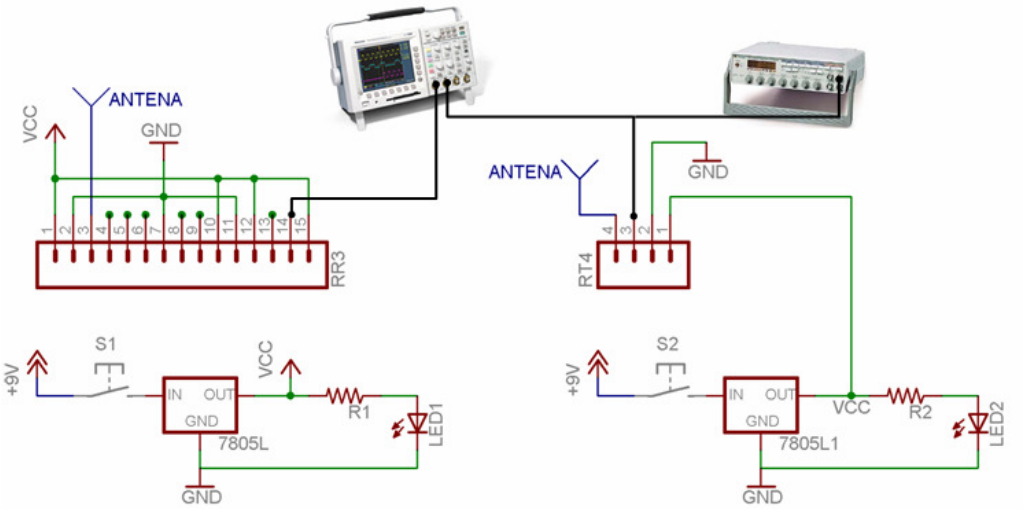


FIGURA 7.21 – DIAGRAMA ELÉTRICO DO CIRCUITO COMUNICAÇÃO UNIDIRECIONAL (PARTE 1)

Foram realizados alguns testes com esse sistema e a Figura 7.20 apresenta as duas principais medições realizadas, uma onda quadrada na freqüência de 5,0MHz foi enviada pelo módulo de transmissão e perfeitamente recebida pelo módulo de recepção, assim como uma onda quadrada na freqüência de aproximadamente 2,5MHz foi enviada pelo módulo de transmissão e perfeitamente recebida pelo módulo de recepção.

Portanto, o resultado obtido foi satisfatório e conforme o esperado.

7.3.5 Comunicação Unidirecional Via RF (Parte 2)

Esse teste teve como objetivo, a confirmação do funcionamento da transmissão de dados entre os módulos de comunicação e das lógicas de programação desenvolvidas para cada um dos módulos.

A Figura 7.22 apresenta o circuito elétrico dos dois módulos (receptor e transmissor) que correspondem ao teste realizado e a Figura 7.23 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador do módulo do transmissor.

Através do fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.3.

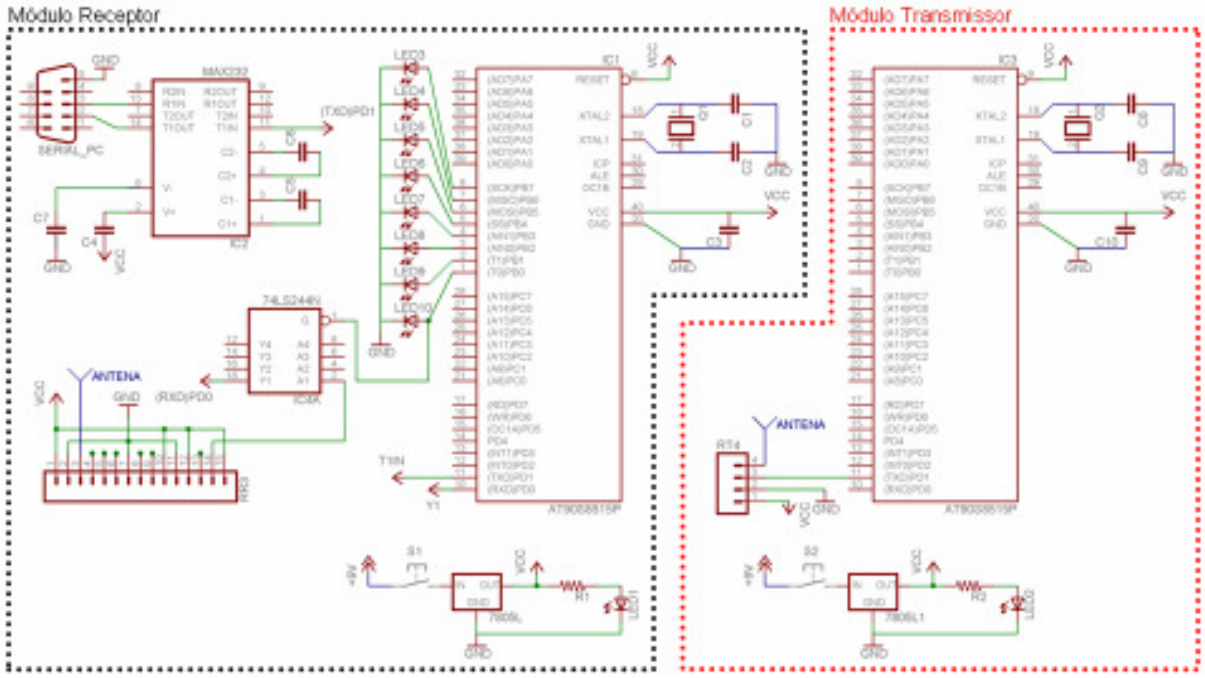


FIGURA 7.22 – DIAGRAMA ELÉTRICO DO CIRCUITO COMUNICAÇÃO UNIDIRECIONAL (PARTE 2)

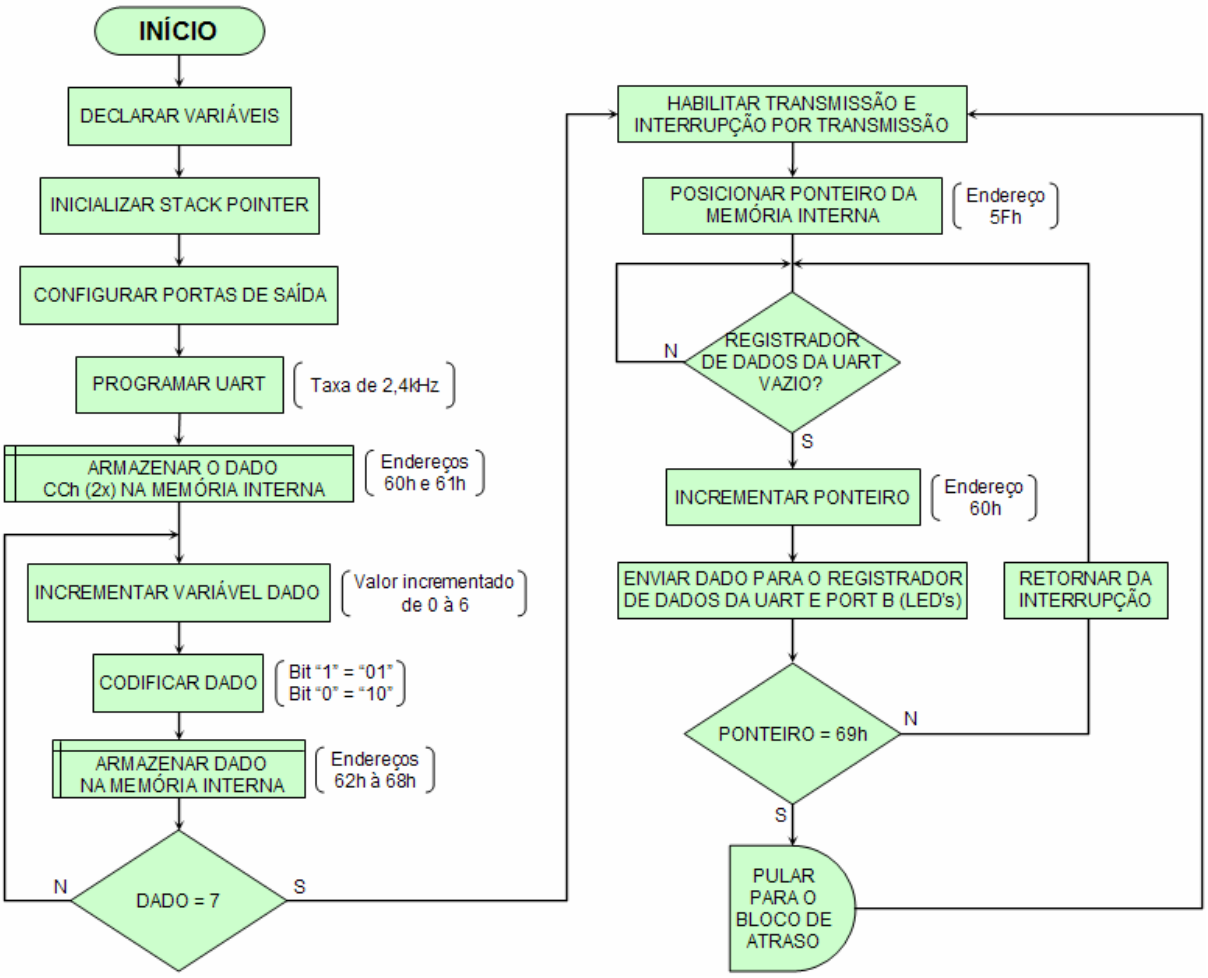


FIGURA 7.23 – FLUXOGRAMA DA LÓGICA UTILIZADA PARA O MÓDULO DO TRANSMISSOR

A Figura 7.24 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador do módulo do receptor.

No fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.4.

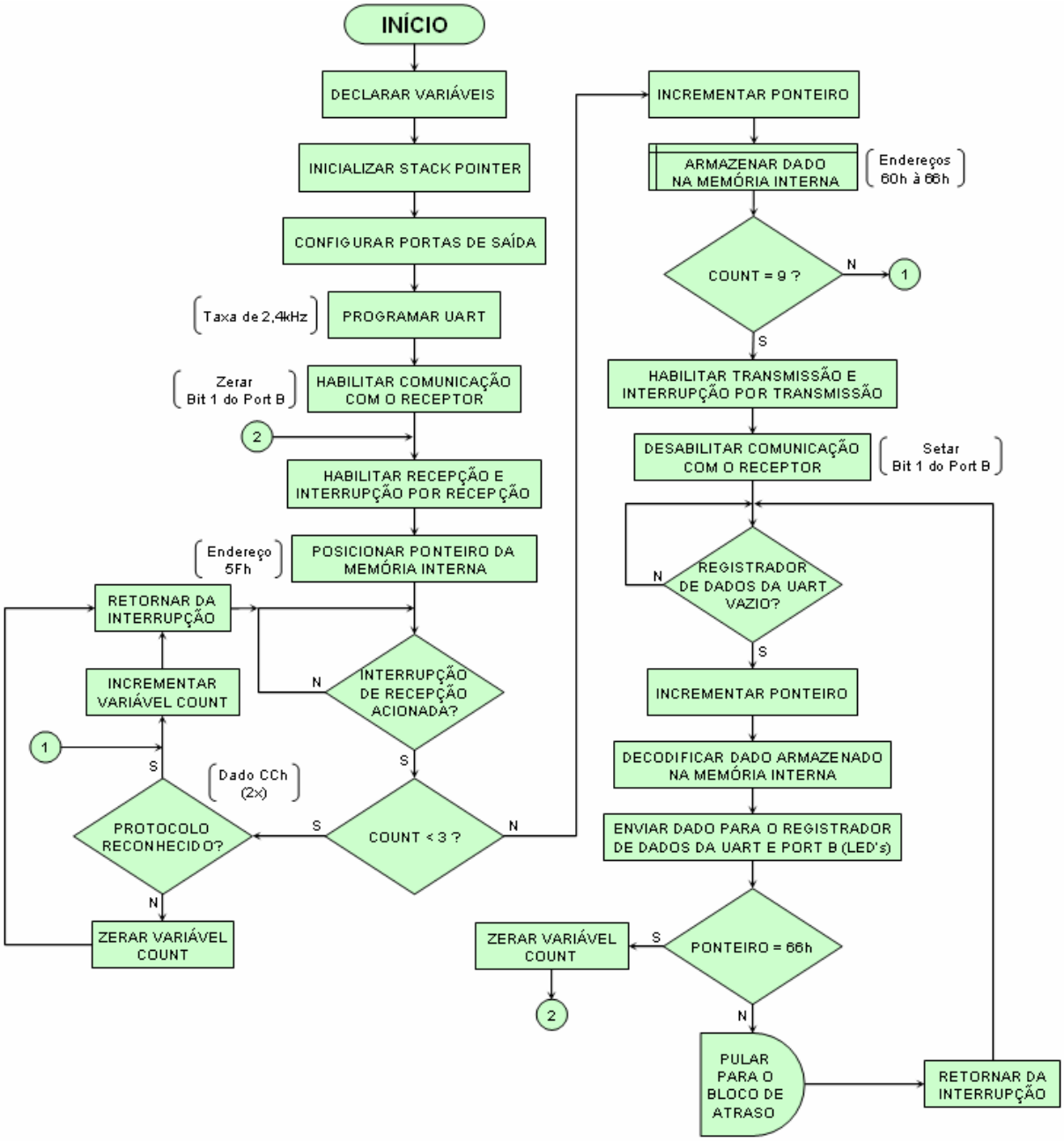


FIGURA 7.24 – FLUXOGRAMA DA LÓGICA UTILIZADA PARA O MÓDULO DO RECEPTOR

A lógica de programação desenvolvida para a interface com o usuário, praticamente não sofreu alteração em relação ao teste apresentado anteriormente (Comunicação Serial – Parte 3). A Figura 7.25 ilustra a tela do programa em execução.

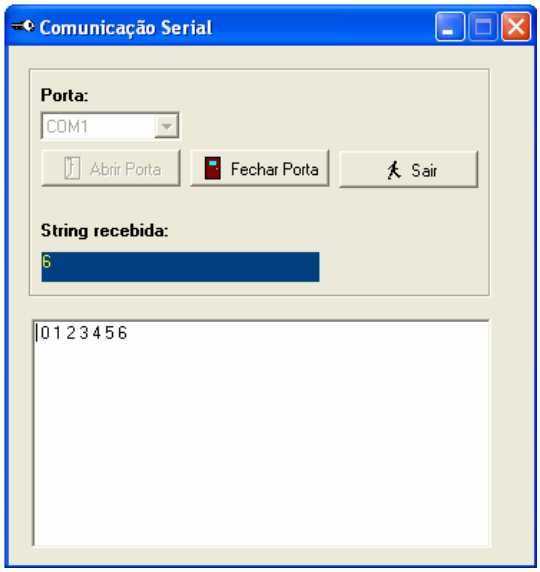


FIGURA 7.25 – TELA DO PROGRAMA DESENVOLVIDO EM EXECUÇÃO

Inicialmente estava-se encontrando bastante dificuldade para obter êxito no resultado desse teste. A comunicação não constante entre os módulos de transmissão e recepção, sem que haja um protocolo de comunicação entre eles torna quase impossível de comprovar o recebimento exato da mensagem que está sendo transmitida pelo transmissor.

A simulação apresentava os resultados conforme o esperado, mas o teste real falhava.

Dessa forma, esse teste foi dividido em duas partes, a primeira parte, focando em testar fisicamente a lógica de programação desenvolvida para cada um dos módulos, ignorando assim os módulos de comunicação, ou seja, o circuito elétrico anteriormente apresentado na Figura 7.22, foi concebido inicialmente, conforme apresentado na Figura 7.26. Após os ajustes necessários na lógica de programação, essa parte do teste passou a apresentar o funcionamento correto, portanto, deu-se prosseguimento ao objetivo principal do teste, que após mais alguns ajustes no protocolo de comunicação (conforme explicado anteriormente neste Capítulo), o resultado obtido foi satisfatório e conforme o esperado.

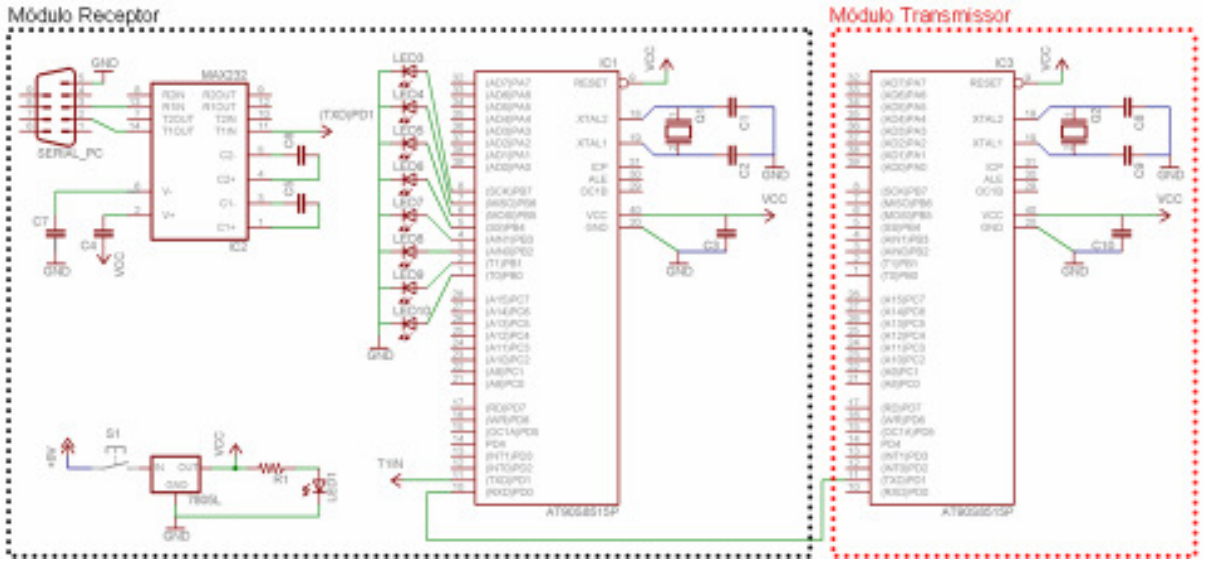


FIGURA 7.26 – DIAGRAMA ELÉTRICO DO TESTE DE COMUNICAÇÃO POR MEIO DE UART

7.3.6 Acionamento dos Motores (Parte 1)

Esse teste teve como objetivo, a verificação do funcionamento dos motores, assim como das lógicas de programação desenvolvidas (módulos de transmissão/ recepção), para que os motores pudessem ser precisamente controlados através do computador pessoal.

A Figura 7.27 apresenta o circuito elétrico dos dois módulos (receptor e transmissor) que correspondem ao teste realizado.

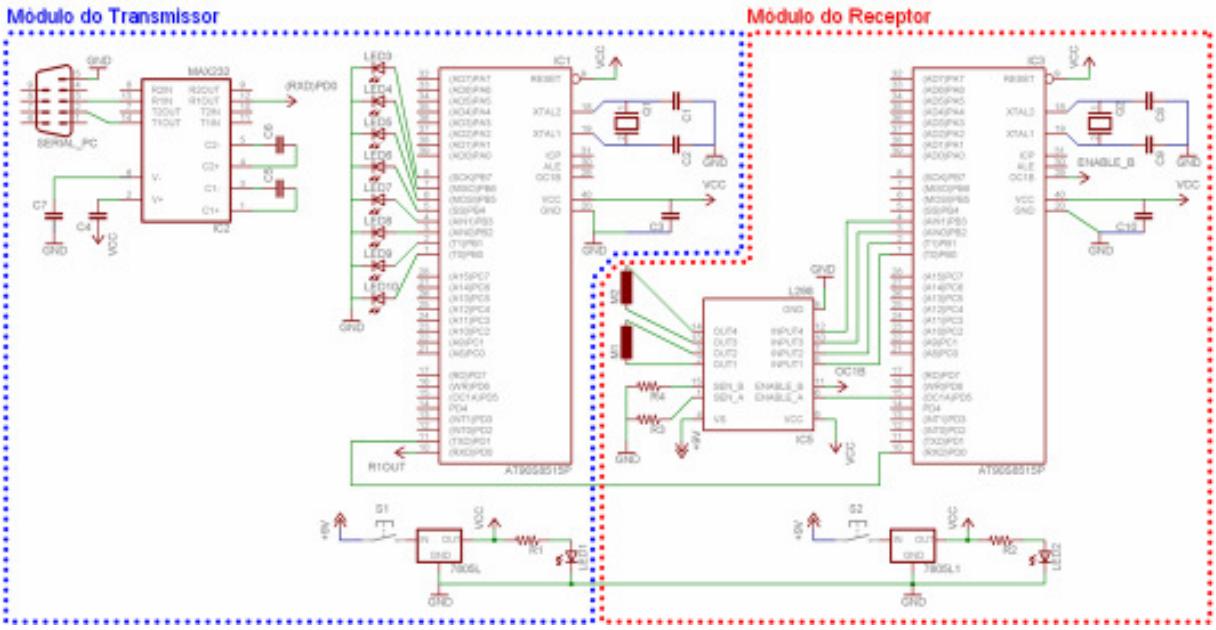


FIGURA 7.27 – DIAGRAMA ELÉTRICO DO CIRCUITO ACIONAMENTO DOS MOTORES (PARTE 1)

A Figura 7.28 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador do módulo do transmissor.

Através do fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.6.

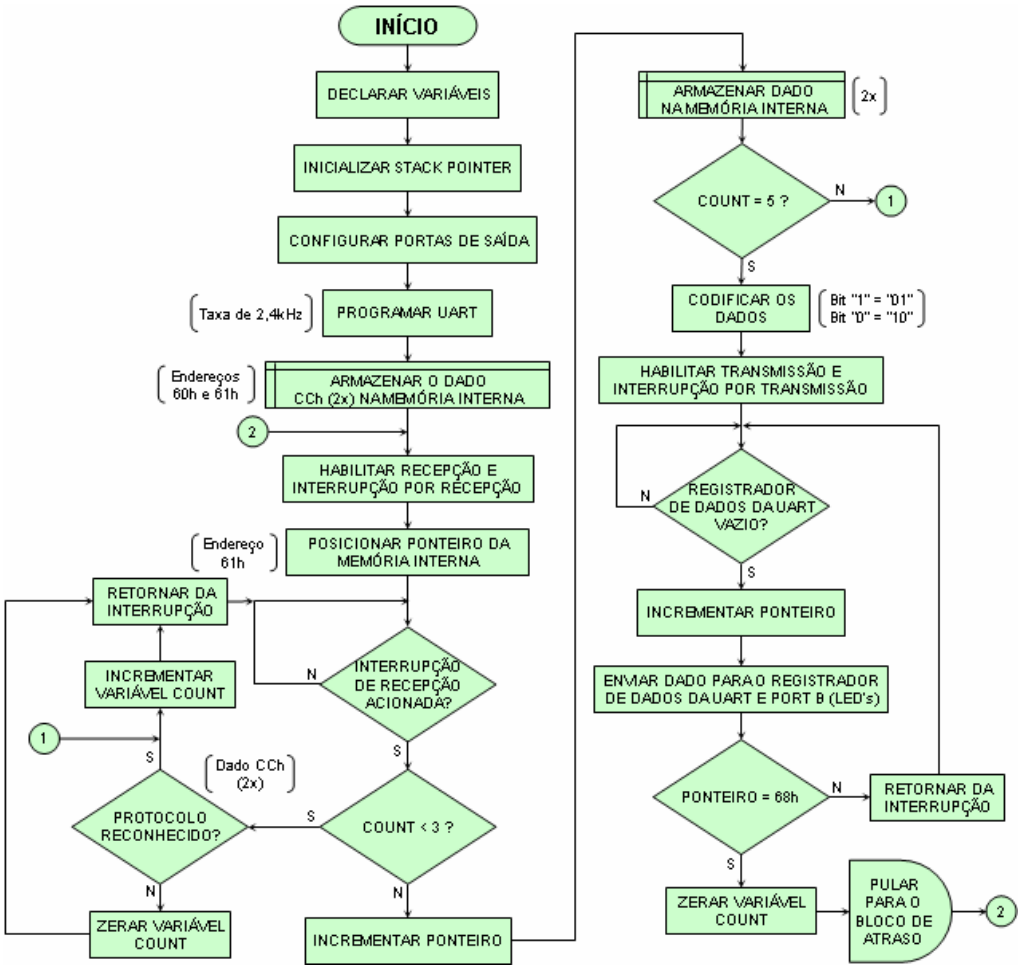


FIGURA 7.28 – FLUXOGRAMA DA LÓGICA UTILIZADA PARA O MÓDULO DO TRANSMISSOR

A Figura 7.29 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador do módulo do receptor.

No fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.7.

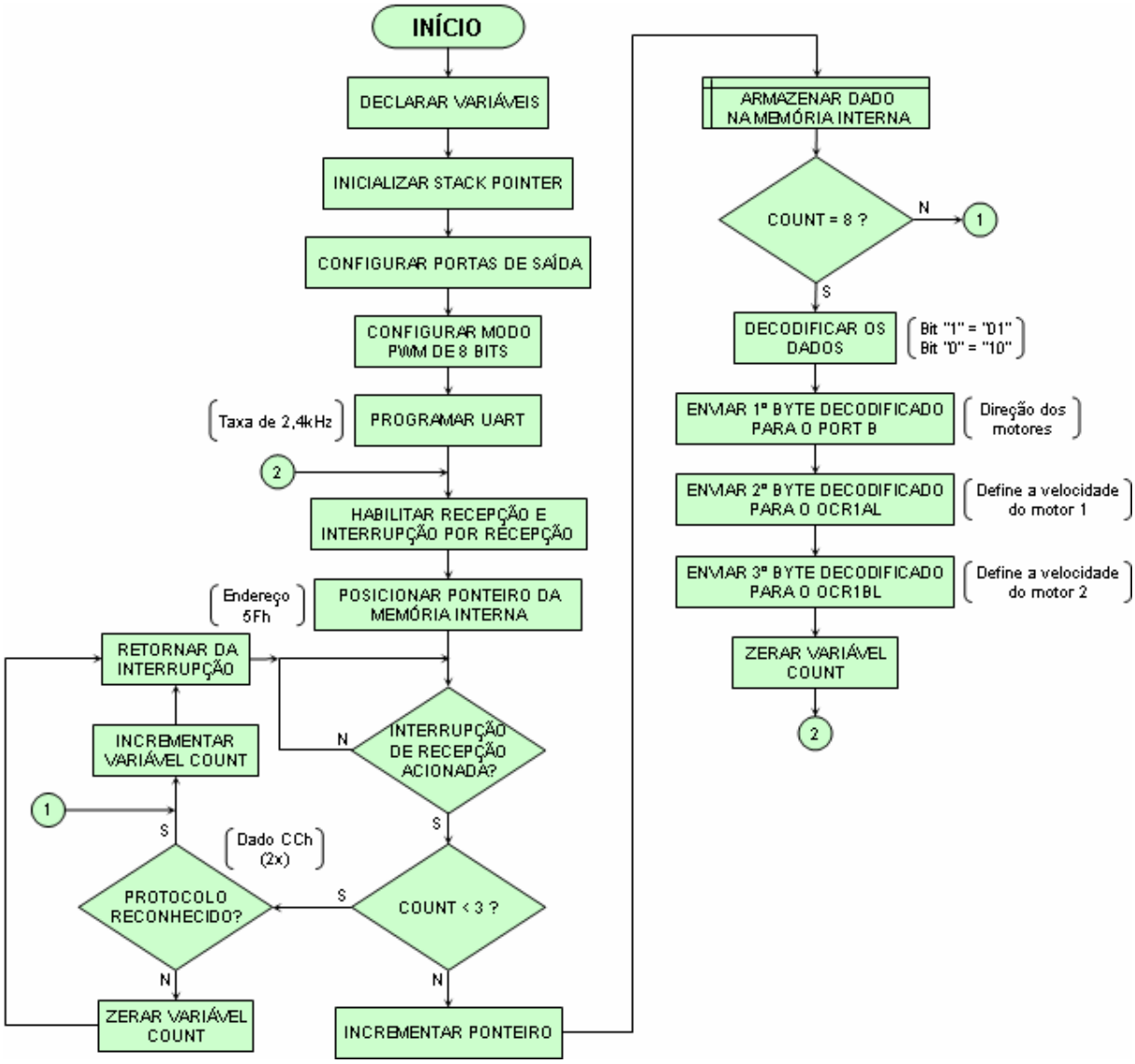


FIGURA 7.29 – FLUXOGRAMA DA LÓGICA UTILIZADA PARA O MÓDULO DO RECEPTOR

A lógica de programação desenvolvida para a interface com o usuário, sofreu algumas alterações em relação ao teste apresentado anteriormente (Comunicação Serial – Parte 3).

Foram adicionados os comandos necessários que permitissem o controle preciso dos motores. A Figura 7.30 ilustra a tela do programa em execução.

Para maiores detalhes, a listagem do programa teste (apenas o adicional em relação ao apresentado no Anexo 10.1.2), pode ser encontrada no Anexo 10.1.8.

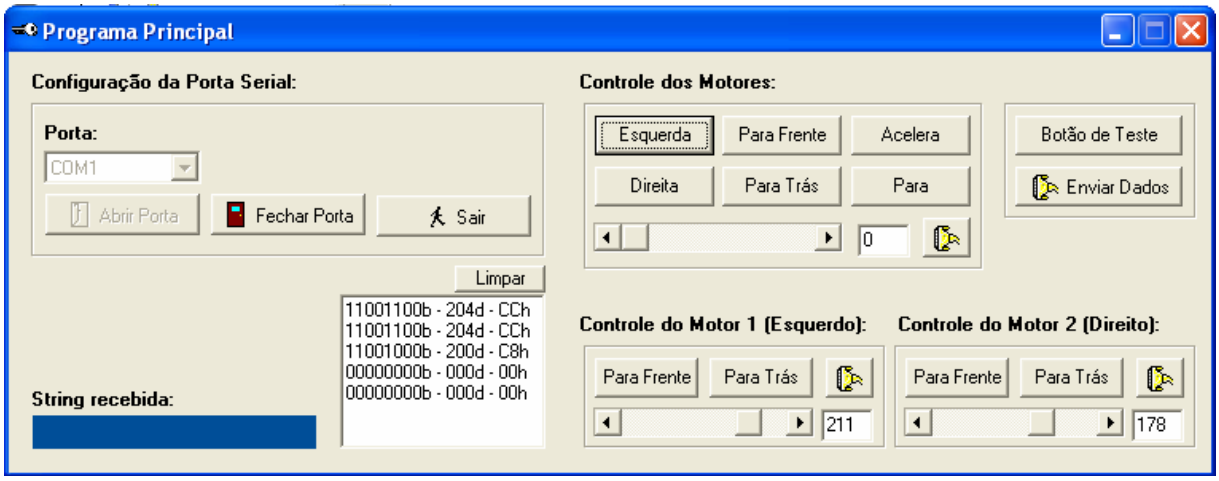


FIGURA 7.30 – TELA DO PROGRAMA DESENVOLVIDO EM EXECUÇÃO

Utilizando como exemplo, os comandos apresentados na Figura 7.30, o conceito da lógica de comunicação adotada, é ilustrado nas Figuras 7.31 e 7.32.

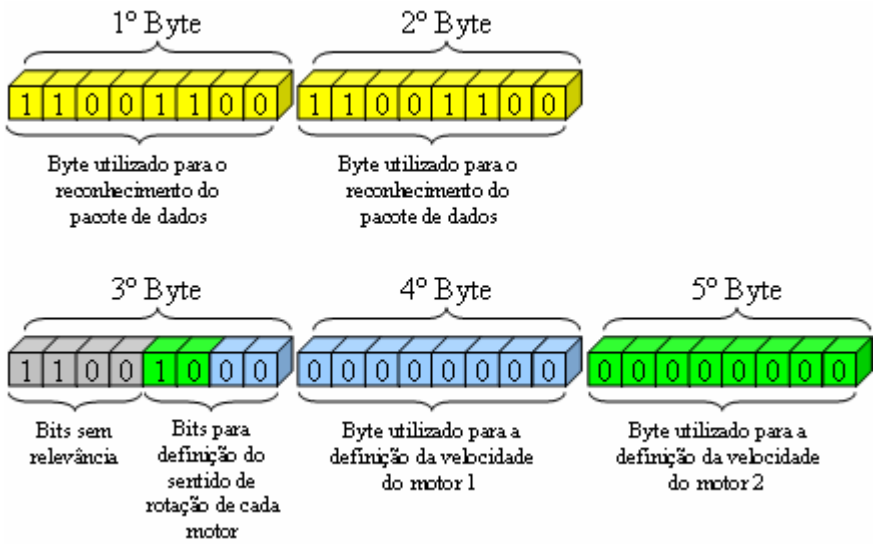


FIGURA 7.31 – PACOTE DE DADOS ENVIADO PELO PC AO CIRCUITO DE TRANSMISSÃO

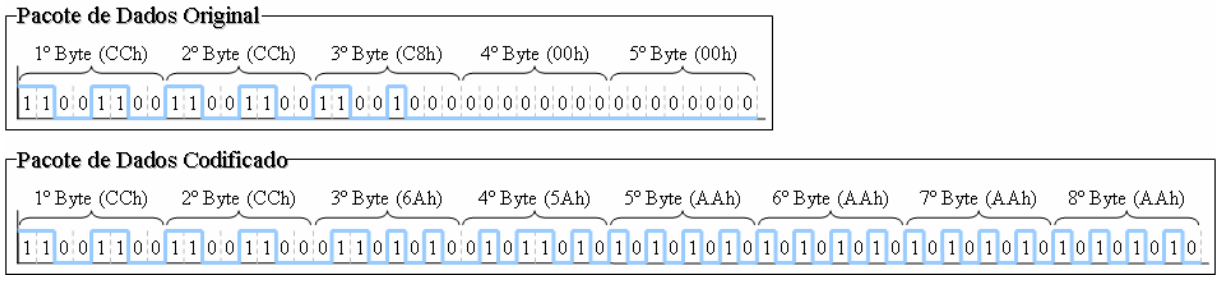


FIGURA 7.32 – PACOTE DE DADOS ORIGINAL E CODIFICADO

É possível observar que o pacote de dados original, enviado pelo PC ao circuito de transmissão, contempla primeiramente dois bytes de reconhecimento de envio dos dados, um terceiro byte que contém os comandos necessários que vão dar o sentido de rotação de cada motor, e por último dois bytes cada um responsável pela velocidade de rotação dos motores 1 e 2. Nessa fase, não é aplicada codificação para o envio dos dados.

Quanto ao envio dos dados do circuito de transmissão ao circuito de acionamento dos motores, os dois primeiros bytes de reconhecimento do pacote, não sofrem nenhum tipo de codificação (não existe a possibilidade de se ter um caractere codificado igual ao caractere CCh, esse é um dos motivos pelo qual esse caractere foi utilizado para o reconhecimento do pacote, conforme detalhes abordados anteriormente neste Capítulo); já os demais caracteres, 3º byte (C8h), 4º byte (00h) e 5º byte (00h), são codificados de acordo com a Tabela 7.3.

Por fim, para que esse teste funcionasse corretamente, foi necessário fazer a correlação entre os motores 1 e 2, esquerdo e direito respectivamente, com os comandos necessários para que o movimento do conjunto ocorresse da forma desejada.

A Tabela 7.4, apresenta essa correlação, semelhante ao já apresentado na Tabela 7.1.

TABELA 7.4 – RELAÇÃO ENTRE OS COMANDOS E OS MOVIMENTOS DOS MOTORES

BIN	HEX	DEC	Motor 1		Motor 2		Conjunto	
11000000	C0	192	Parado	✕	Parado	✕	Parado	✕
11000001	C1	193	Trás	↓	Parado	✕	Gira para esquerda	↺
11000010	C2	194	Frente	↑	Parado	✕	Gira para direita	↻
11000011	C3	195	Parado	✕	Parado	✕	Parado	✕
11000100	C4	196	Parado	✕	Frente	↑	Gira para esquerda	↺
11000101	C5	197	Trás	↓	Frente	↑	Gira para esquerda	↺
11000110	C6	198	Frente	↑	Frente	↑	Frente	↑↑
11000111	C7	199	Parado	✕	Frente	↑	Gira para esquerda	↺
11001000	C8	200	Parado	✕	Trás	↓	Gira para direita	↻
11001001	C9	201	Trás	↓	Trás	↓	Trás	↓↓
11001010	CA	202	Frente	↑	Trás	↓	Gira para direita	↻
11001011	CB	203	Parado	✕	Trás	↓	Gira para direita	↻
11001100	CC	204	Parado	✕	Parado	✕	Parado	✕
11001101	CD	205	Trás	↓	Parado	✕	Gira para esquerda	↺
11001110	CE	206	Frente	↑	Parado	✕	Gira para direita	↻
11001111	CF	207	Parado	✕	Parado	✕	Parado	✕

Após diversos testes e ajustes, conseguiu-se o desempenho desejado e o sistema passou a funcionar corretamente.

7.3.7 Acionamento dos Motores (Parte 2)

Esse teste é muito semelhante ao apresentado anteriormente, a diferença entre eles está no fato de que os comandos para o acionamento dos motores serão feitos por meio de radiofreqüência, ou seja, o objetivo desse teste é o de verificar o funcionamento dos motores, através de dados enviados a longa distância.

A Figura 7.33 apresenta o circuito elétrico dos dois módulos (receptor e transmissor) que correspondem ao teste realizado.

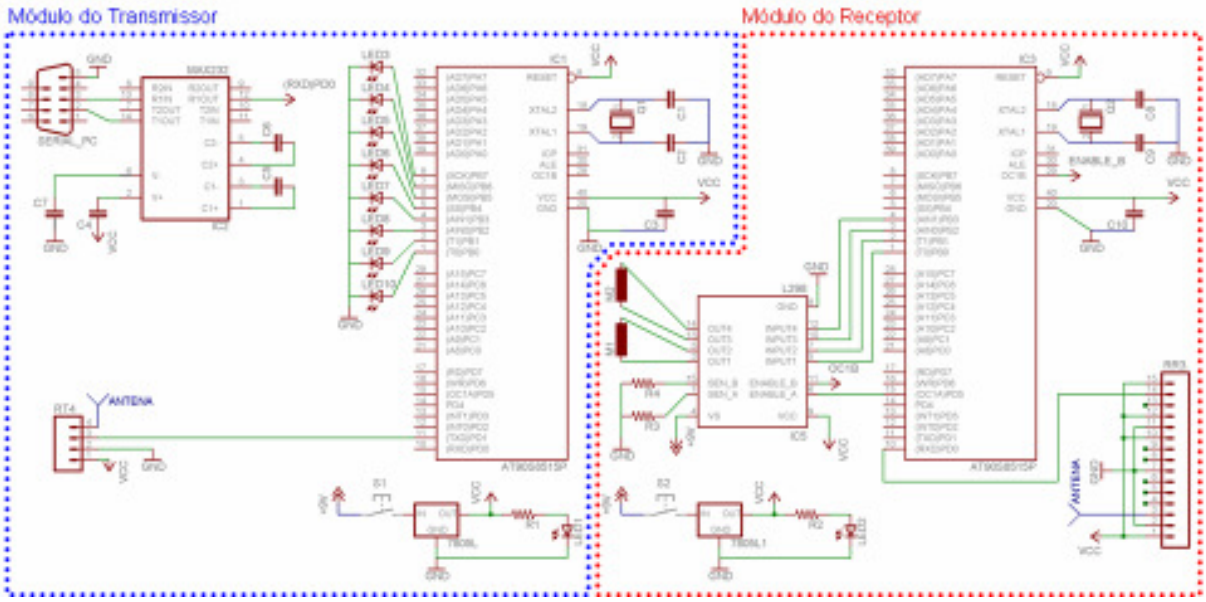


FIGURA 7.33 – DIAGRAMA ELÉTRICO DO CIRCUITO ACIONAMENTO DOS MOTORES (PARTE 2)

Nesse teste, não foi necessário nenhum tipo de desenvolvimento lógico, em nível de programação. O protocolo de comunicação utilizado no teste anterior funcionou perfeitamente na comunicação sem fio. Portanto, os programas utilizados nos microcontroladores, tanto no circuito de transmissão, quanto no circuito de acionamento dos motores, não sofreram nenhuma alteração em relação ao teste anterior.

O sistema apresentou um resultado muito semelhante ao anterior (comunicação direta entre os microcontroladores), exceto quando a carga da bateria de alimentação do circuito de acionamento dos motores estava baixa, onde os motores não respondiam a todos os comandos do circuito de transmissão.

7.3.8 Comunicação Bidirecional Via RF

Esse teste teve como objetivo testar o funcionamento da comunicação bidirecional por meio de RF entre o circuito de IHM e o circuito do robô.

Nesse teste, preocupou-se apenas em garantir que a informação recebida pelo circuito de recepção da IHM, provinda do robô e transmitida ao PC fosse correta.

A Figura 7.34 apresenta o circuito elétrico dos dois módulos (IHM e robô) que correspondem ao teste realizado.

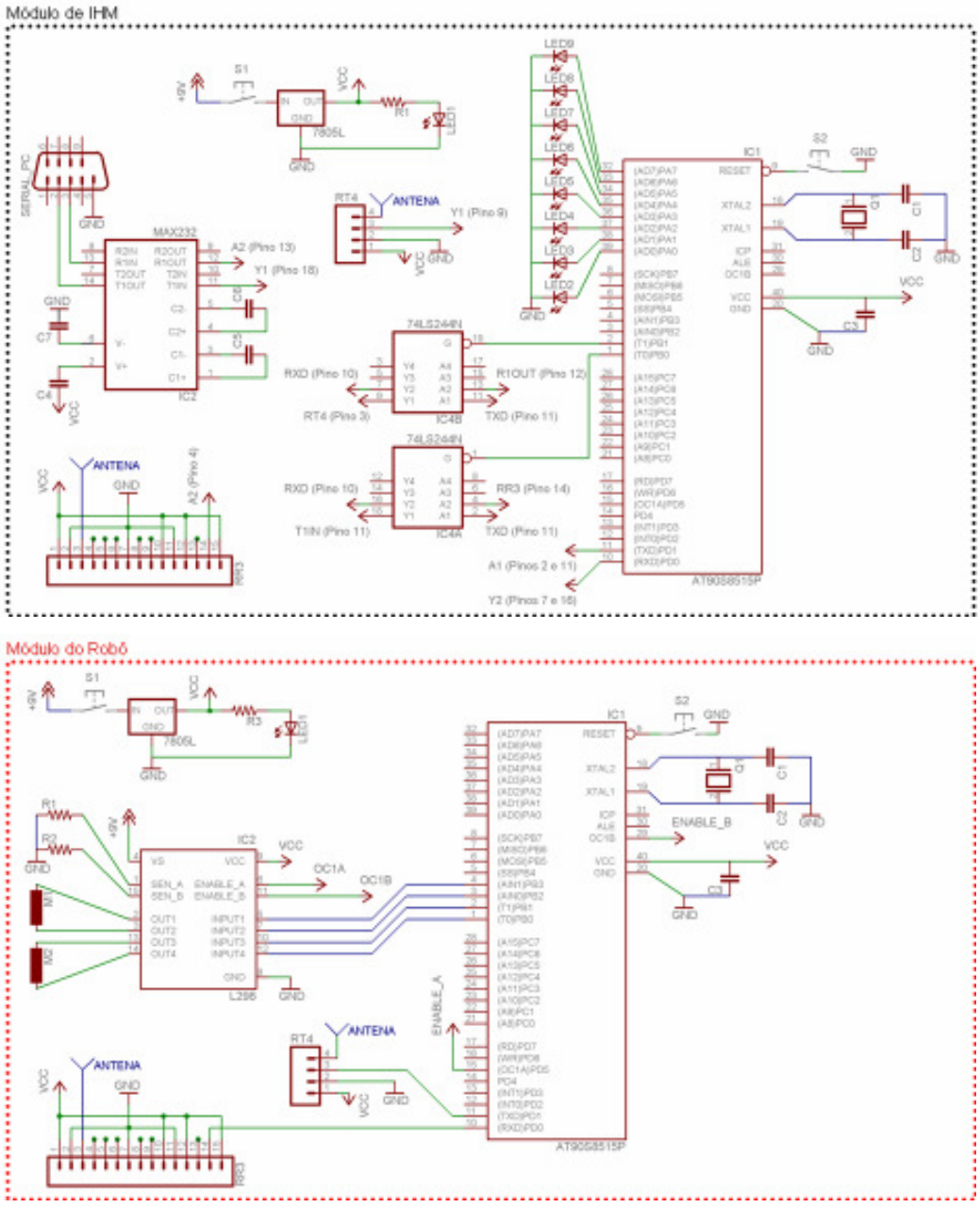


FIGURA 7.34 – DIAGRAMA ELÉTRICO DO CIRCUITO DE COMUNICAÇÃO BIDIRECIONAL VIA RF

A Figura 7.35 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador do módulo da IHM.

Através do fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.9.

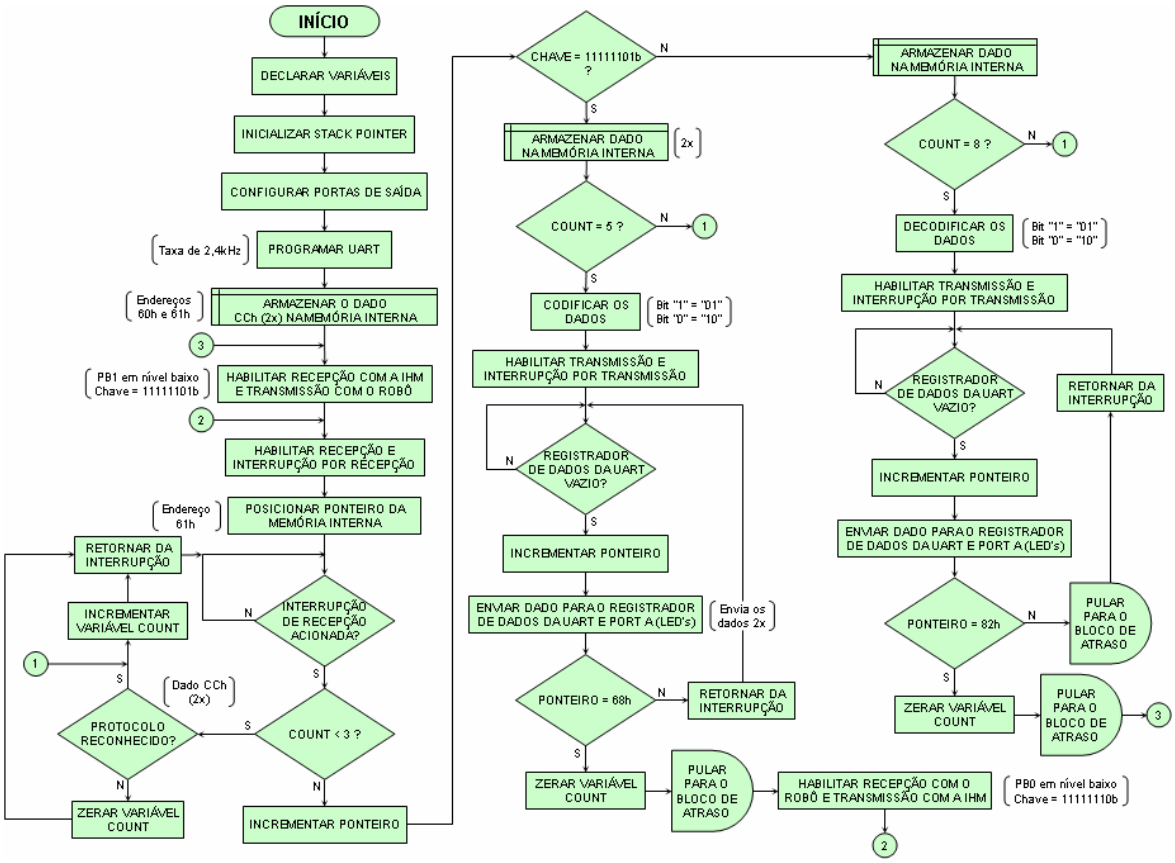


FIGURA 7.35 – FLUXOGRAMA DA LÓGICA UTILIZADA PARA O MÓDULO DA IHM

A Figura 7.36 apresenta o fluxograma correspondente à lógica de programação utilizada no microcontrolador do módulo do robô.

No fluxograma, é possível observar o conceito adotado na programação.

Para maiores detalhes, a listagem completa desse programa teste pode ser encontrada no Anexo 10.1.10.

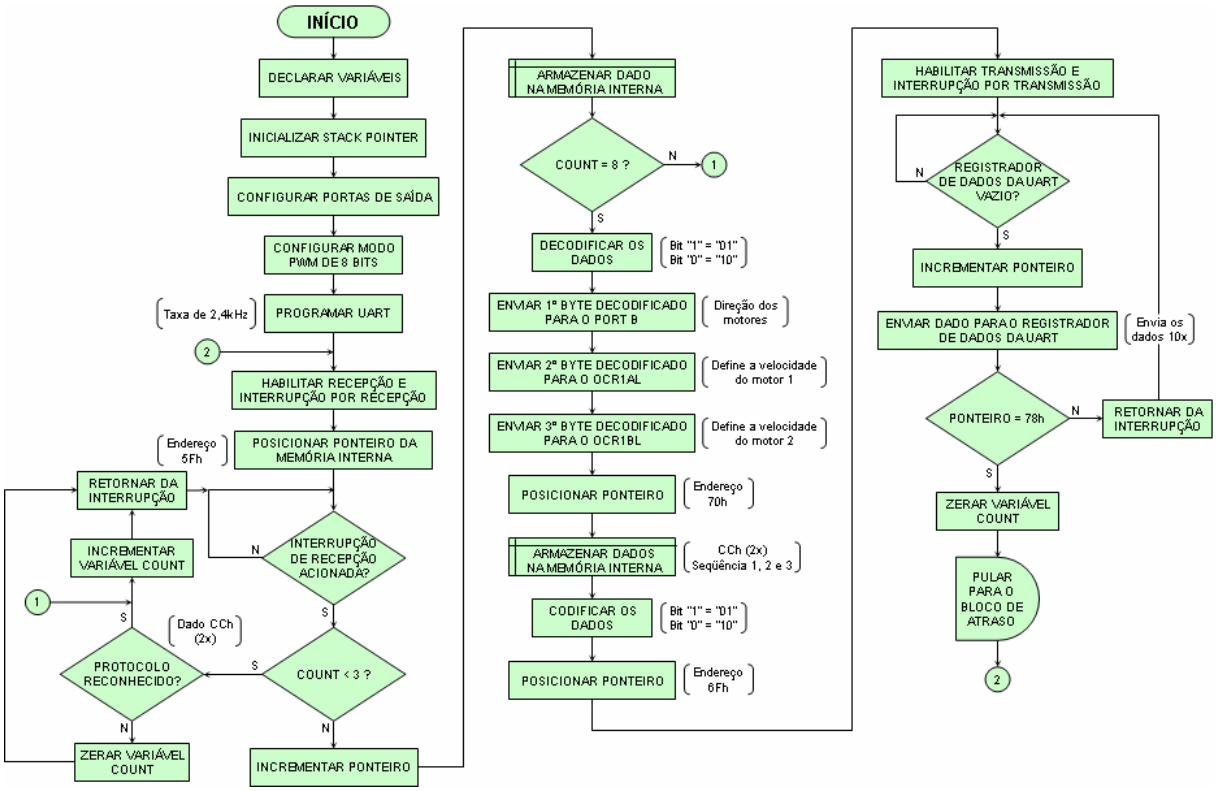


FIGURA 7.36 – FLUXOGRAMA DA LÓGICA UTILIZADA PARA O MÓDULO DO ROBÔ

A lógica de programação utilizada para a interface com o usuário, é a mesma da apresentada anteriormente (Acionamento dos Motores – Parte 1). A Figura 7.37 ilustra a tela do programa em execução.

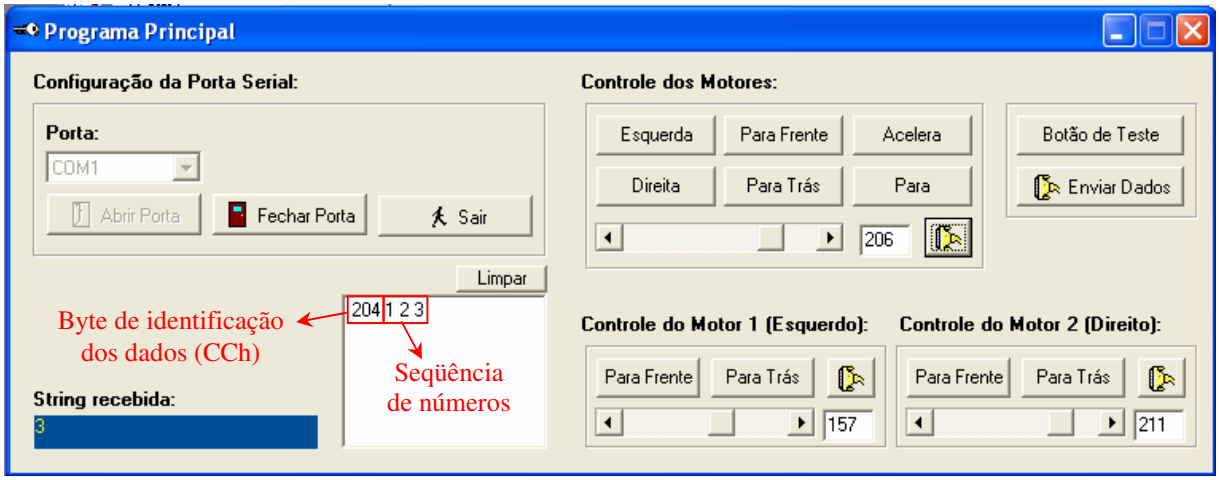


FIGURA 7.37 – TELA DO PROGRAMA DESENVOLVIDO EM EXECUÇÃO

É possível também observar na Figura 7.37 a seqüência de números recebidos pelo módulo de IHM, informação enviada pelo módulo de transmissão do robô.

O pacote de dados original, enviado pelo PC ao circuito de IHM, é o mesmo que o já demonstrado nas Figuras 7.31 e 7.32, assim como esses dados que são codificados e transmitidos ao módulo de recepção do robô.

Quando os dados são recebidos e reconhecidos pelo módulo do robô, o microcontrolador deste módulo armazena uma seqüência de 1 a 3 na memória, codifica esses dados e envia essa seqüência de números, juntamente com os dois primeiros bytes de identificação de início de pacote (CCh), através do módulo de transmissão do robô, para o módulo de recepção da IHM.

Quando os dados são recebidos e reconhecidos pelo módulo de IHM, esses dados são decodificados e enviados por meio de comunicação serial ao PC, para que o usuário possa visualizar os dados transmitidos pelo robô.

Para que esse teste funcionasse corretamente, no caso do módulo de IHM, que pode receber ou transmitir dados, tanto do PC, quanto dos módulos de comunicação RF, foi necessário estabelecer uma espécie de chave de seleção para uma comunicação isolada entre o microcontrolador e os módulos de comunicação RF e o PC. Para isso, foi utilizado um buffer de três estados, possibilitando dessa forma, o controle do fluxo de dados recebidos e transmitidos. A Figura 7.38, ilustra o conceito adotado.

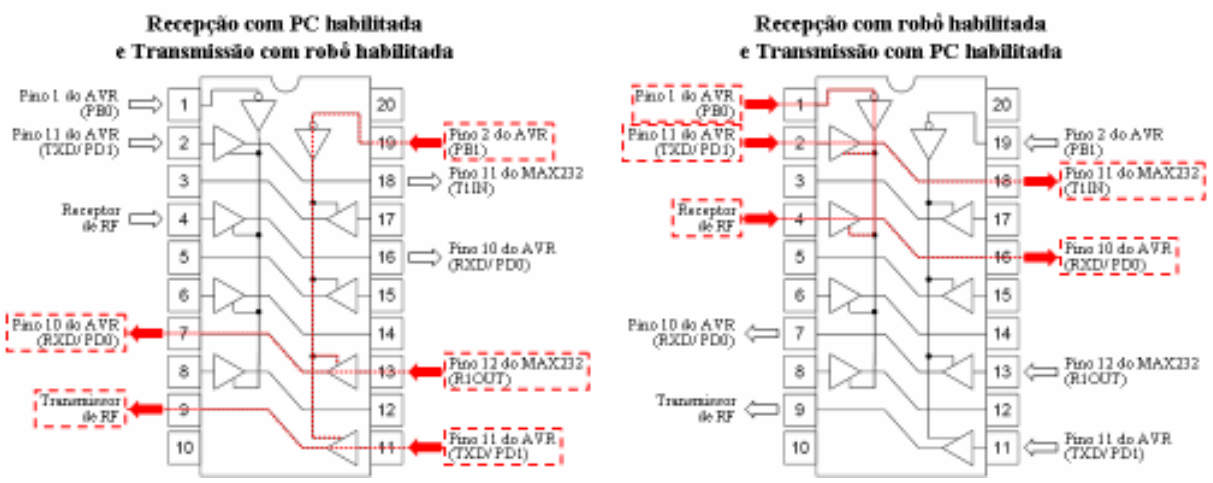


FIGURA 7.38 – CONCEITO SELETIVO ADOTADO PARA A COMUNICAÇÃO BIDIRECIONAL

É possível observar pela Figura 7.38, que os pinos 1 e 2 do microcontrolador é que definirão de qual módulo o sistema irá receber ou transmitir os dados. No caso, quando o pino 1 do AVR é colocado em nível baixo, a recepção com o robô e a transmissão com o PC são habilitadas, quando o pino 2 do AVR é colocado em nível baixo, a recepção com o PC e a transmissão com o robô são habilitadas.

Após adotar esse conceito de seleção chaveado, a comunicação bidirecional entre os módulos funcionou perfeitamente.

8 INFORMAÇÕES ADICIONAIS

8.1 CIRCUITO DE GRAVAÇÃO DO MICROCONTROLADOR

Inicialmente, juntamente com o circuito de interface com o usuário, encontrava-se o circuito utilizado para realizar a gravação dos dados no microcontrolador. A Figura 8.1 apresenta o diagrama elétrico desse circuito.

Essa prática ajudou muito no desenvolvimento do sistema, evitando gastos com circuitos disponíveis no mercado e também facilitando, nos diversos testes realizados que requeriam constantes gravações lógicas nos microcontroladores do sistema.

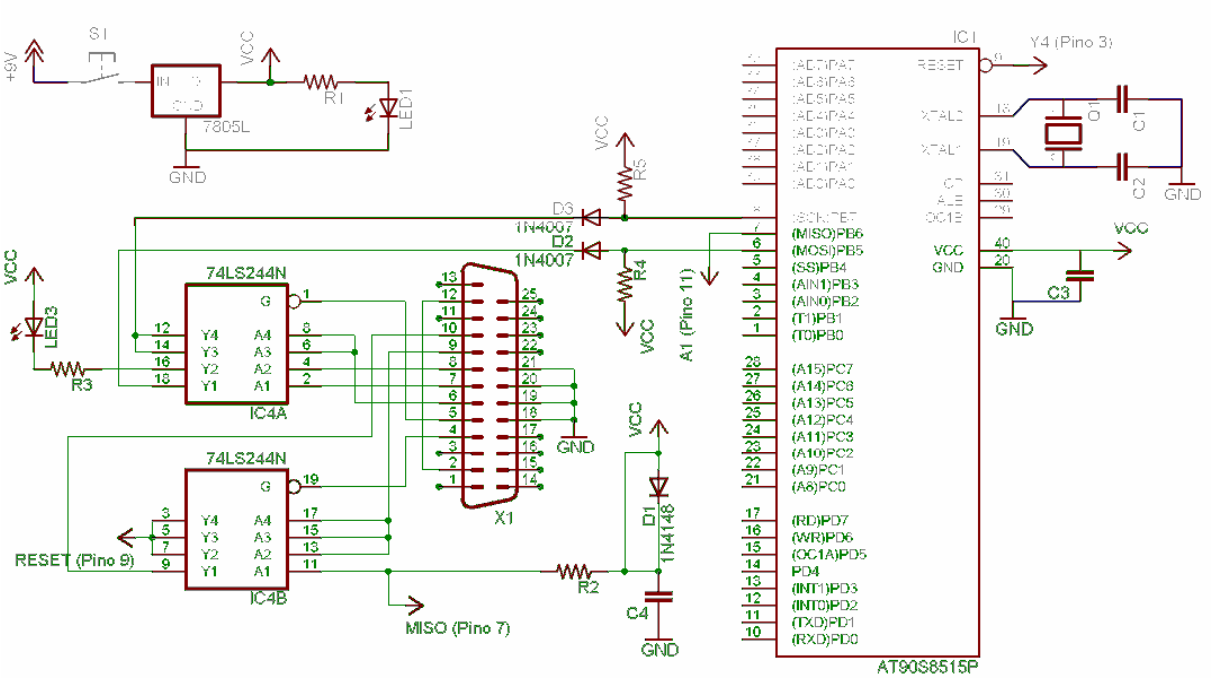


FIGURA 8.1 – DIAGRAMA ELÉTRICO DO CIRCUITO DE GRAVAÇÃO DO MICROCONTROLADOR

A Tabela 8.1 apresenta a lista de componentes utilizados para a realização da gravação da lógica de programação no microcontrolador, assim como o custo de mercado encontrado para cada um dos componentes.

TABELA 8.1 – LISTA DE COMPONENTES DO CIRCUITO DE GRAVAÇÃO (MICROCONTROLADOR)

Designação	Nomenclatura[46]	Fabricante	Valor/Unid.	Qtd	Custo[47] (R$)
Microcontrolador	AT90S8515	Atmel	-	1	15,00
Buffer de Três-Estados	HD74HC244	Hitachi	-	1	1,30
Cristal	Q1	KDS	4MHz	1	1,30
Capacitor	C1 e C2	Thonson	27pF	2	0,40
Capacitor	C3 e C4	Thonson	100nF	2	0,40
Resistor	R1 e R3	Acel	300Ω	2	0,10
Resistor	R4 e R5	Acel	1KΩ	2	0,10
Resistor	R2	Acel	100KΩ	1	0,05
Diodo	1N4007	Mic	1N4007	2	0,10
Diodo de Sinal	1N4148	Cons.	1N4148	1	0,05
Regulador de Tensão	7805L	STMicroelectronics	5V	1	1,00
LED	LED1	Cons.	Vermelho	1	1,20
Cabo Paralelo Adaptado	Paralelo	Leadership	DB25	1	3,50

8.2 ESTRUTURA MECÂNICA

As razões principais analisadas para a escolha da estrutura mecânica, em termos de tamanho e material foram:

- peso mínimo, na qual os motores de corrente contínua pudessem atuar de forma satisfatória;
- tamanho mínimo, mas que proporcionasse uma boa dinâmica ao robô e que o circuito de controle pudesse ser bem acondicionado e protegido;
- material fácil de ser retrabalhado e resistente à colisões.

Os principais tipos de matéria-primas analisadas foram: plásticos, isopor, ciba-tool e madeira balsa.

A Tabela 8.2, apresenta um pequeno sumário das principais características de cada um desses tipos de materiais.

[46] Nomenclatura baseada no circuito elétrico apresentado na Figura 8.1

[47] Valor médio aproximado encontrado na Região Sudeste (São Paulo, 2006)

TABELA 8.2 – ESTRUTURA MECÂNICA (TIPOS DE MATERIAIS AVALIADOS)

Características	**Unidade**	**Ciba-Tool**[48] (Renshape)		**Madeira Balsa**[49]	**Plásticos**[50] (Polipropileno)	**Isopor**[51] (Poliestireno Expandido)
		5030	**440**			
Aplicações	-	Modelos de estilo e testes de programas	Modelos de estilo, maquetes, testes de programas	Iscas de peixe, aeromodelismo, etc.	Brinquedos, recipientes para alimentos, ventiladores, etc.	Protetor de equipamentos, isolantes térmicos, pranchas para flutuação, etc.
Densidade	g/cm^3	0,30	0,52 - 0,57	0,35	0,89	0,015 - 0,030
Resistência à flexão	N/mm^2	6 - 8	15 - 17	9,50	34,47	0,06 - 0,48
Resistência à compressão	N/mm^2	6 - 8	9 - 10	11,64	24,13	0,06 - 0,25

Conforme já explicado, a estrutura do robô além de ser responsável pela rigidez, deve ter baixo peso, dimensões adequadas à aplicação do robô, deve ser resistente e deve ser fácil de ser manufaturado. Obviamente o projeto da estrutura mecânica deve estar associado ao ambiente onde o robô deverá se movimentar.

A proposta inicial adotada para este robô, conforme é apresentado na Figura 8.2, foi utilizando um material de composto plástico chamado ciba-tool, que após testes de resistência à tração e compressão, apresentou resultados muito bons, resistindo à quedas e colisões sem lascar. Sua estrutura um pouco esponjosa ajuda a absorver o impacto e possui um peso muito baixo.

A estrutura final utilizada, foi desenvolvida em madeira balsa, sendo que os dois principais motivos são: fácil de ser retrabalhado e fácil de ser encontrado no mercado.

Fazendo uma breve análise dos dados apresentados na Tabela 8.2, e utilizando a madeira balsa como referência, é possível observar que o ciba-tool (440), apresenta uma densidade maior em relação a madeira balsa, o que é um ponto negativo, pois significa que é um material mais pesado, a resistência à flexão é maior, porém a resistência à compressão é menor. Comparando a madeira balsa, com o tipo de plástico polipropileno, que são aqueles plásticos utilizados para fazer brinquedos, embalagens para alimentos e remédios, apresentam uma resistência à flexão e compressão bem superiores a da madeira, o que é um fator positivo,

[48] FONTE: *Huntsman Renshape Solutions Datasheet*

[49] FONTE: http://www.webcoreonline.com

[50] FONTE: http://www.owenscorning.com.br/interat.asp

[51] FONTE: http://www.acepe.pt/eps/eps_prop_tab.asp

entretanto a densidade é duas vezes e meia maior, o que é um fator bem desfavorável já que o peso é uma preocupação necessária devido a potência dos motores utilizados.

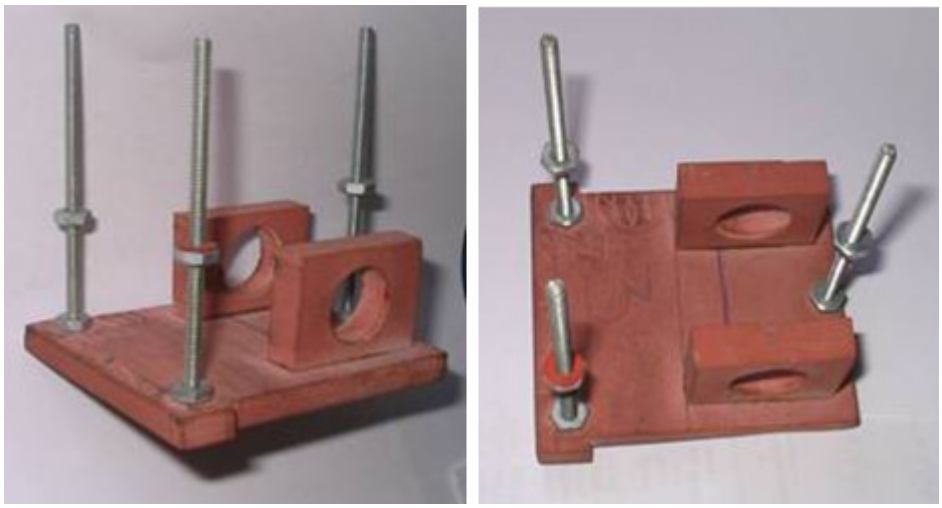


FIGURA 8.2 – ESTRUTURA INICIAL (CIBA-TOOL)

A Figura 8.3, apresenta a estrutura final em madeira balsa, projetada para o robô.

FIGURA 8.3 – ESTRUTURA FINAL (MADEIRA BALSA)

8.3 CUSTO TOTAL DO SISTEMA

Apesar de já apresentado nas Tabelas 4.1 e 4.2 o custo dos principais componentes utilizados, neste capítulo, pretende-se apresentar de uma maneira mais completa e detalhada, o custo total do sistema (incluindo também o circuito de gravação para o microcontrolador),

mesmo que em alguns casos os valores considerados sejam aproximados, pois alguns dos componentes não foram adquiridos comercialmente, mas sim reutilizados de projetos anteriores, ou até mesmo, doados de instituições/ pessoas físicas como incentivo.

Pretende-se nessa seção, listar não só os componentes eletrônicos, mas também os materiais utilizados para o completo desenvolvimento do projeto.

A Tabela 8.3, apresenta a lista completa de todos os componentes utilizados.

TABELA 8.3 – LISTA DE COMPONENTES/ MATERIAIS UTILIZADOS

Designação	**Descrição**	**Fabricante**	**Valor/Unid.**	**Qtd**	**Custo[52] (R$)**
Microcontrolador	AT90S8515	Atmel	-	2	30,00
RS232 Transceiver	MAX232	Maxim	-	1	2,50
Buffer de Três-Estados	HD74HC244	Hitachi	-	3	3,90
Receptor RF	RR3	Telecontrolli	433,92MHz	2	35,00
Transmissor RF	RT4	Telecontrolli	433,92MHz	2	35,00
Cristal	Xtal	KDS	4MHz	2	2,60
Ponte H (Driver)	L298N	STMicroelectronics	-	1	12,30
Capacitor	Cerâmico	Thonson	27pF	4	0,80
Capacitor	Cerâmico	Thonson	33pF	2	0,40
Capacitor	Cerâmico	Thonson	100nF	3	0,60
Capacitor	Eletrolítico	Thonson	1μF	4	0,80
Resistor	F. Carbono	Acel	10KΩ	2	0,10
Resistor	F. Carbono	Acel	2,7KΩ	2	0,10
Resistor	F. Carbono	Acel	300Ω	3	0,15
Resistor	F. Carbono	Acel	27Ω	2	0,10
Resistor	F. Carbono	Acel	1KΩ	2	0,10
Resistor	F. Carbono	Acel	100KΩ	1	0,05
Diodo	1N4007	Mic	1N4007	2	0,10
Diodo de Sinal	1N4148	Cons.	1N4148	1	0,05
Regulador de Tensão	7805L	STMicroelectronics	5V	2	2,00
LED	Redondo	Cons.	Vermelho	11	13,20
Motores CC	-	Faulhaber	1516E012S	2	340,00[53]
Redutor	-	Faulhaber	15/3K 41:1	2	510,00[53]

[52] Valor médio aproximado encontrado na Região Sudeste (São Paulo, 2006)

[53] FONTE: motores@martebal.com.br (Divisão Faulhaber do Brasil, Santa Rita do Sapucaí, Minas Gerais, 2006)

Designação	Descrição	Fabricante	Valor/Unid.	Qtd	Custo[52] (R$)
Sensor de Ultrasom	Sonar	Tato Eletrônicos	-	1	50,00[54]
Sensor de Velocidade	KMI18/2	NXP	2		30,00[55]
Ímãs	Neodímeo	Oximag	-	24	9,60[56]
Cabo Paralelo Adaptado	Paralelo	Leadership	DB25	1	3,50
Soquete	Torneado	-	40 pinos	2	4,00
Soquete	Torneado	-	20 pinos	2	2,60
Soquete	Torneado	-	16 pinos	1	0,80
Bateria	Recarregável	Panasonic	9V	2	40,00
Clip para Bateria	-	-	9V	2	2,00
Rodas	-	-	-	2	12,00
PCI	Universal	-	10x5cm	2	12,00
PCI	Universal	-	10x10cm	1	6,50
Cabo	Serial	-	DB09	1	5,00
Outros Materiais[57]	-	-	-	-	15,00
TOTAL	**-**	**-**	**-**	**-**	**1.182,85**

Com base nos valores apresentados acima, a Figura 8.4, procura ilustrar graficamente o custo total do projeto, dividido pelos seus principais componentes.

É possível observar que o custo do projeto é relativamente alto R$1.182,85, quando analisado o seu valor total de forma isolada.

Analisando mais detalhadamente, os custos dos motores e dos redutores, são praticamente 72% do valor total do projeto, desconsiderando esses valores, conforme apresentado na Figura 8.5, o valor de custo total do projeto cai para R$332,85. Fazendo uma pesquisa no mercado é possível avaliar a possibilidade provável de se encontrar motores que atendem os requisitos mínimos de torque e potência desse projeto com valores de custo bem mais acessíveis.

[54] FONTE: http://www.tato.ind.br/ (Tato Eletrônicos, São Paulo, 2006)

[55] FONTE: http://www.nxp.com/ (NXP Semiconducters, 2007)

[56] FONTE: http://www.oximag.com/principal.htm (Oximag)

[57] Outros materiais considerados: estanho, fita isolante, cola, fios de cobre (esmaltado e encapado), madeira (balsa e ciba-tool), chave liga/desliga, botão de reset, barras de terminais, conectores e jumpers

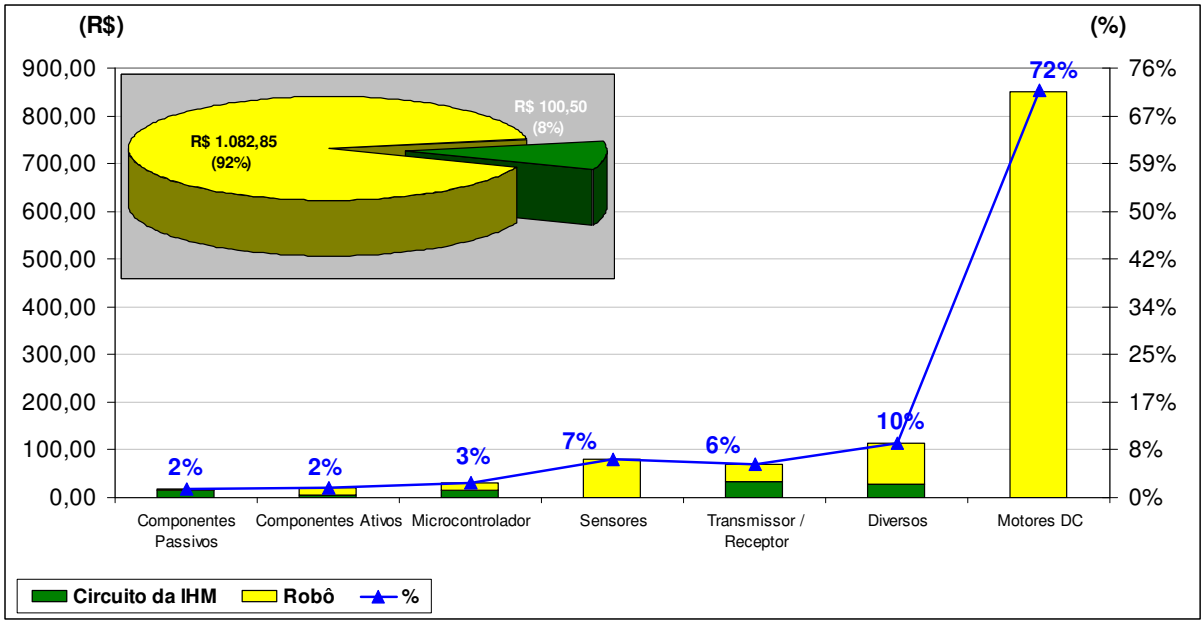


FIGURA 8.4 – GRÁFICO DO CUSTO TOTAL DO SISTEMA (CONSIDERANDO OS MOTORES)

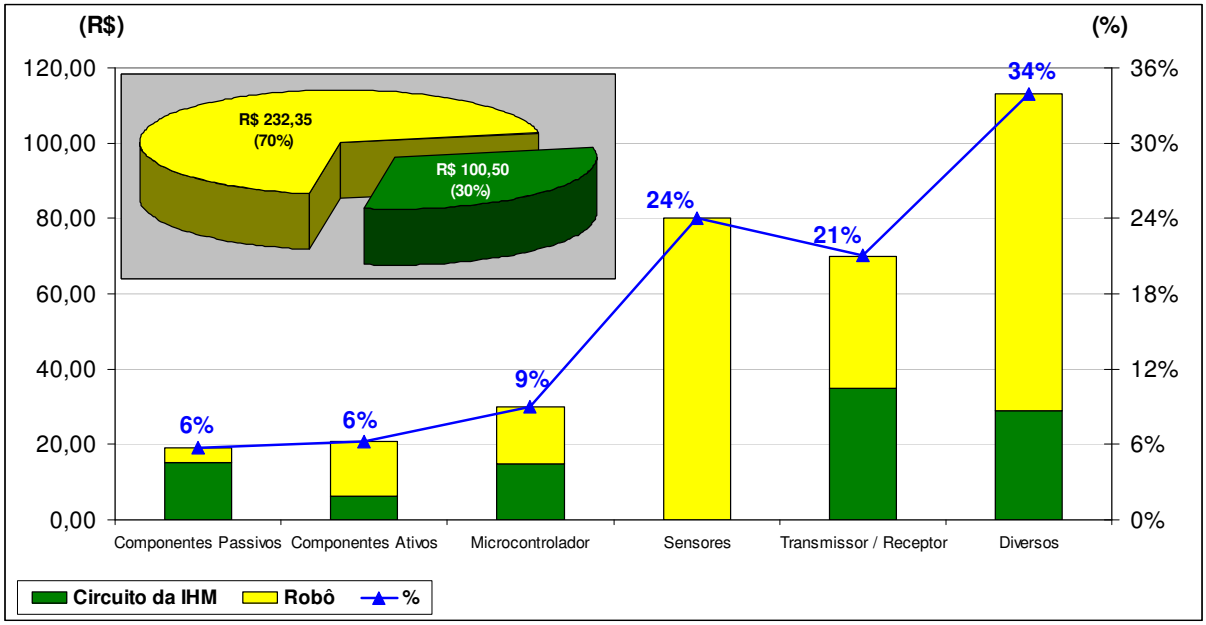


FIGURA 8.5 – GRÁFICO DO CUSTO TOTAL DO SISTEMA (DESCONSIDERANDO OS MOTORES)

9 CONCLUSÃO

Este trabalho teve como objetivo principal, o desenvolvimento de um robô móvel capaz de explorar ambientes hostis sem a intervenção humana, com a capacidade de detectar obstáculos e que apresentasse resultados confiáveis, com qualidade e baixo custo.

Grande parte do trabalho desenvolvido e dos conceitos adotados podem ser utilizados em diversas outras aplicações similares. Pode-se citar os seguintes exemplos:

- os conceitos de programação aplicados no controle e compensação da velocidade dos motores, utilizados para permitir que o caminho percorrido pelo robô seja o mais próximo possível de uma linha reta;
- a lógica implementada que permite definir o ângulo correto desejado que o robô percorra;
- o protocolo de comunicação criado para minimizar os erros de comunicação por radiofreqüência;
- entre outros.

Com relação à comunicação por meio de radiofreqüência, os módulos utilizados apresentaram bons resultados ao longo dos testes realizados quando o protocolo de comunicação e o sistema de codificação (similar ao Manchester) desenvolvido eram aplicados, caso contrário, a comunicação se tornava impraticável para o objetivo traçado. Como sugestão para simplificar o circuito elétrico e a lógica de programação, poderia ser utilizado um módulo de comunicação integrado, ou seja, com comunicação bidirecional, para que não fosse necessário utilizar dois módulos independentes, um para a transmissão e outro para a recepção.

No desenvolvimento do trabalho, procurou-se também apresentar a maioria dos testes realizados e de uma forma bem detalhada, além das lógicas de programação desenvolvidas, permitindo a sua perfeita reprodução.

É importante salientar, que no desenvolvimento do projeto, a estrutura física adotada para o robô, foi apenas um meio utilizado para se poder demonstrar o conceito de controle, mapeamento e comunicação implementado. Para que o robô possa ser utilizado em qualquer ambiente hostil, independente do caminho a ser percorrido, é necessário que sejam estudadas melhores formas estruturais. É importante a princípio pelo menos considerar que a estrutura seja capaz de proteger os circuitos eletrônicos, que dê uma boa durabilidade ao sistema, que

consiga absorver os impactos devido à movimentação em terrenos irregulares e que não seja afetado se estiver operando em ambientes úmidos, tóxicos, ou com temperaturas elevadas.

De uma maneira geral, através do resultado dos testes executados, o sistema implementado apresentou resultados satisfatórios, entretanto, sempre há espaço para melhorias quando se trata de conceitos envolvendo a eletrônica e a mecânica. Componentes mais baratos podem ser substituídos por componentes mais robustos e velozes, como por exemplo, os módulos de RF. Abaixo são listadas algumas sugestões adicionais para uma possível melhoria do sistema desenvolvido:

- Utilizar um único motor, mais potente, para que o movimento em linha reta seja preciso, não sofrendo interferência devido as diferentes características existentes entre os motores. As movimentações em curva podem ser feitas utilizando-se um servomotor na frente, com um ângulo de giro limitado.
- Adicionar ao sistema uma bússola digital para garantir a precisão na definição do ângulo de giro do robô.
- Considerar além do método de odometria utilizado, alguma medição adicional, como a inclusão de um GPS, para garantir que erros não sistemáticos, como escorregamento das rodas (solo escorregadio, aceleração acentuada, etc...), possam ser de alguma forma identificados.
- Aumentar a resolução do sistema de odometria, incluindo uma maior quantidade de ímãs, dessa forma a precisão e a medição do caminho a ser percorrido é melhorado de forma significativa.
- Incluir no robô um sistema de medição do nível de tensão da bateria, para que o sistema possa compensar automaticamente os ajustes de calibração.
- Adicionar outros sensores de ultrasom, ou um sistema de movimentação angular do sensor utilizado, para que o ambiente possa ser mapeado 360º, sem a necessidade de movimentação do robô.
- Adicionar um segundo microcontrolador, dedicado ao controle dos motores e leitura dos sensores de velocidade, permitindo um mapeamento constante do movimento do robô e não somente o resultado final do caminho percorrido pelo robô.
- Implementar na programação do software de controle, uma lógica que possibilite o mapeamento do ambiente sem a necessidade da intervenção humana, ou seja, aplicar conceitos que com base nos dados recebidos dos sensores do robô, este seja capaz de movimentar-se de forma autônoma.

Tentou-se, de uma forma geral, abranger todos os passos necessários para a construção de um robô móvel que pode ser facilmente adaptado para o desenvolvimento de trabalhos de pesquisa e acadêmicos.

Não existe limite para a aplicação e melhoria de qualquer sistema que esteja relacionado com a robótica. O limite está apenas na imaginação dos seres humanos.

10 ANEXOS

10.1 LISTAGEM DOS PROGRAMAS

10.1.1 Programa Comunicação Serial (Parte 2)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa Comunicação Serial (Parte 2)
Linguagem: Assembler
Arquivo: TRPC.asm
Objetivo: Testar a Comunicação Serial entre o Microcontrolador e o PC
Descrição: Uma seqüência de bytes de 0 a 255 é enviada pelo microcontrolador ao PC utilizando-se a UART.

```
.include "8515def.inc"

;** Declaração das Variáveis
.def    temp    = r16
.def    desloc  = r17
.def    desloc1 = r18

        rjmp    RESET                   ;Pula para o Programa Principal

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
        inc     temp                    ;Incrementa a variável temp
        out     UDR,temp                ;Envia os dados armazenados na variável temp para o
                                        ;registrador de dados da UART
        out     PORTB,temp              ;Envia a mesma informação para o Port B

;** Loop de Atraso
Atraso:
        ldi     desloc,255
        ldi     desloc1,255

Loop:
        dec     desloc
        brne    loop
        dec     desloc1
        brne    loop
        reti                            ;Retorna da Interrupção

RESET:

;** Programa Principal

;** Inicialização de Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp
        ldi     temp,0xFF

        out     DDRB,temp               ;Port B configurada como saída
        out     PORTB,temp              ;Garante 0 inicialmente na saída

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp               ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)
        ldi     temp,0b00101000
        out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
        clr     temp                    ;Limpa a variável temp
        sei                             ;Seta a Interrupção
```

```
Volta:
        rjmp    Volta                   ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.2 Programa Comunicação Serial (Parte 3)

PROGRAMA PARA COMPUTADOR PESSOAL

Título:	Programa Comunicação Serial (Parte 3)
Linguagem:	Delphi (Pascal)
Arquivo:	CSerial.pas
Objetivo:	Testar a Comunicação Serial entre o Microcontrolador e o PC
Descrição:	Uma seqüência de bytes de 0 a 255 é enviada pelo microcontrolador ao PC utilizando-se a UART e os dados são mostrados ao usuário.

```
unit CSerial;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Menus, ExtCtrls, FileCtrl, Buttons, jpeg, Registry;

type TMultlinha = class(TThread)
    private
    protected
    procedure Execute; override;
    procedure MostraString;    //Mostra dados na tela.
end;

type
  TForm1 = class(TForm)
    SpeedAbrirPorta: TSpeedButton;
    SpeedFecharPorta: TSpeedButton;
    SpeedSair: TSpeedButton;
    Label2: TLabel;
    Label41: TLabel;
    LabelStrRecebida: TLabel;
    Bevel1: TBevel;
    ComboBoxPorta: TComboBox;
    procedure FormCreate(Sender: TObject);
    procedure SpeedAbrirPortaClick(Sender: TObject);
    procedure SpeedSairClick(Sender: TObject);
    procedure SpeedFecharPortaClick(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure MostraPortasCom();    //Mostra portas COMx.
  private
    { Private declarations }
  public
    { Public declarations }
  end;

const
  LEN_BUFFER = 100;    //Tamanho dos Buffers de dados.

var
  Form1: TForm1;
  hCom: THANDLE;    //Handle da Porta Serial (API).
  dcb: TDCB;    //Estrutura DCB (API).
  CommTimeouts: TCOMMTIMEOUTS;    //Estrutura TCOMMTIMEOUTS (API).
  BytesLidos: DWORD;
  BufferRecebe : array [0..LEN_BUFFER] of Integer;        //Define o buffer.
  GLB_Conectado:boolean;    //Indica se a porta já está aberta.
```

```
  GLBEnviaDados:bool;    //Para habilitar/desabilitar o envio de dados pela serial.
  VRecebido: String;

implementation

{$R *.DFM}

//Abre a porta serial especificada.
function AbrirPorta(NomePorta:String):boolean;
begin
  hCom := CreateFile(
          PChar(NomePorta),
          GENERIC_READ or GENERIC_WRITE,
          0,    //Dispositivos COM abertos com acesso exclusivo.
          nil,    //Sem atributos de segurança.
          OPEN_EXISTING,    //deve usar OPEN_EXISTING.
          0,    //Entrada e saída sem overlap.
          0    //Deve ser NULL para COM.
        );
  if(hCom = INVALID_HANDLE_VALUE) then    //Se houve algum erro ao abrir a porta.
    result := false
  else
    result:= true;
end;

//Define Timeouts.
Function ConfiguraTimeOuts:boolean;
begin
  if not GetCommTimeouts(hCom, CommTimeouts) then result:= false;
  CommTimeouts.ReadIntervalTimeout := 2;
  CommTimeouts.ReadTotalTimeoutMultiplier := 0;
  CommTimeouts.ReadTotalTimeoutConstant := 2;
  CommTimeouts.WriteTotalTimeoutMultiplier := 5;
  CommTimeouts.WriteTotalTimeoutConstant := 5;
  if not SetCommTimeouts(hCom, CommTimeouts) then result:= false
  else
    result:= true;
end;

//Configura Porta Serial.
Function ConfiguraControle:boolean;
begin
  if not GetCommState(hCom, dcb) then
    result:= false;

  dcb.BaudRate := CBR_2400;    //define velocidade em bps.
  dcb.ByteSize := 8;    //define bits de dados.
  dcb.Parity := NOPARITY;    //define paridade.
  dcb.StopBits := ONESTOPBIT;    //define stop bit.

  if not SetCommState(hCom, dcb) then
    result:= false
  else
    result:= true;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 //Inicializa variáveis Globais.
 GLB_Conectado := false;    //Indica se a porta já está aberta.
 GLBEnviaDados := false;    //Para habilitar/desabilitar o envio de dados pela serial.
 MostraPortasCom;    //Carrega os nomes das porta COM num FilterComboBox.
end;

procedure TForm1.SpeedAbrirPortaClick(Sender: TObject);
var
 Sucesso:boolean;
begin
  if AbrirPorta(ComboBoxPorta.Text) = true then    //Abre a porta serial.
  begin
    GLB_Conectado := true;
    Sucesso := ConfiguraControle();
    Sucesso := ConfiguraTimeOuts();
    if Sucesso = false then
    begin
      GLB_Conectado := false;
```

```
        CloseHandle(hCom);    //Fecha a porta Serial.
        ShowMessage('Um outro programa já está usando a porta, ou a mesma não existe!');
      end
      else begin
        SpeedAbrirPorta.Enabled := false;
        SpeedFecharPorta.Enabled := true;
        ComboBoxPorta.Enabled := false;
        TMultlinha.Create(false);    //Inicia processo.
      end;
    end
    else begin
      GLB_Conectado := false;
      ShowMessage('Um outro programa já está usando a porta, ou a mesma não existe!');
    end;
end;

procedure TForm1.SpeedSairClick(Sender: TObject);
begin
 Close();
end;

procedure TForm1.SpeedFecharPortaClick(Sender: TObject);
begin
   SpeedFecharPorta.Enabled := false;
   SpeedAbrirPorta.Enabled := true;
   ComboBoxPorta.Enabled := true;
   GLB_Conectado := false;
   CloseHandle(hCom);    //Fecha a porta serial.
end;

//Mostra os nomes das portas "COM" instaladas no sistema num ComboBox.
procedure TForm1.MostraPortasCom();
var
  Registro: TRegistry;    //Para trabalhar com os Registros do windows.
  Lista: Tstrings;
  indice: Integer;    //Para incrementar.
begin
  Registro := TRegistry.Create;    //Cria e aloca espaço na memória para o objeto.
  try
    Registro.RootKey := HKEY_LOCAL_MACHINE;    //Define chave raiz.
    Registro.OpenKey('hardware\devicemap\serialcomm', False);    //Abre a chave.
    Lista := TstringList.Create;
    try
      //Obtém uma string contendo todos os nomes de valores associados com a chave atual.
      Registro.GetValueNames(Lista);
      //Pega os nomes das portas.
      for indice := 0 to Lista.Count - 1 do    //Count é a quantidade de portas existentes.
        ComboBoxPorta.Items.Add(Registro.ReadString( Lista.Strings[indice] ));

      //Adiciona os nomes das portas no ComboBox1.
      if ComboBoxPorta.Items.Count > 0 then
        ComboBoxPorta.ItemIndex := 0;    //Para exibir o nome da porta.
    finally
      Lista.Free;
    end;
    Registro.CloseKey;
  finally
    Registro.Free;
  end;
end;

//Mostra dados de forma sincronizada.
procedure TMultlinha.MostraString;
begin
    Form1.LabelStrRecebida.Caption := VRecebido;
end;

//Trata a Porta Serial.
procedure TMultlinha.Execute;
 var
   cont:DWORD;
   StrMens:AnsiString;
   tama:integer;
begin
  while not Terminated do    //loop infinito. Vida do programa.
    begin
```

```
    if GLB_Conectado = true then    //Se a porta serial foi aberta corretamente.
      begin
        BytesLidos := 0;
        cont := 0;
        if Readfile(hCom, BufferRecebe[cont], LEN_BUFFER, BytesLidos, nil) then    //Lê a porta Serial.
          VRecebido:=IntToStr(BufferRecebe[Cont]);
        if BytesLidos > 0 then    //Se algum caracter foi lido.
          begin
            while cont < BytesLidos do    //Enquanto há caracteres a serem lidos.
            begin
              Synchronize(MostraString);    //Mostra dados na tela.
              cont := cont + 1;
            end;    //Fim Do while 2.
          end;
      end
    else begin
      Sleep(1);    //Para não travar o processo.
  end;
 end;    //Fim Do while 1.
end;

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if(GLB_Conectado = true) then
    CloseHandle(hCom);
end;

end.

//FIM DO PROGRAMA
```

10.1.3Programa Comunicação Unidirecional Via RF (Transmissor)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título:	Programa Comunicação Unidirecional Via RF (Parte 2)
Módulo:	Transmissão
Linguagem:	Assembler
Arquivo:	CUniT_P2.asm
Objetivo:	Testar a Comunicação por meio de RF entre os Módulos de Comunicação
Descrição:	Uma seqüência de 9 bytes deve ser enviada através da UART sendo: Bytes 1 e 2: CCh (Identificação de Protocolo) Bytes 3 a 9: Dados Codificados (Seqüência numérica de 0 à 6) Lembrando que: 1 = 01 (Codificado) 0 = 10 (Codificado)

```
.include "8515def.inc"

;** Declaração das Variáveis
.def        temp      = r16
.def        desloc    = r17
.def        desloc1   = r18
.def        desloc2   = r19
.def        dado      = r20
.def        count     = r21
.def        temp1     = r22
.def        temp2     = r23
.def        temp3     = r24
.def        final     = r25

            rjmp      RESET                 ;Pula para o Programa Principal

.org UDREaddr                               ;UART Data Register Empty Interrupt Vector Address
            inc       XL                    ;Posiciona o ponteiro
```

```
            ld      temp,X              ;Carrega o dado na variável temp
            out     UDR,temp            ;Envia os dados armazenados na variável temp para o
                                        ;registrador de dados da UART
            out     PORTB,temp          ;Envia a mesma informação para o Port B
            cpi     XL,0x69             ;Compara para saber se o último caractere foi enviado
            breq    Atraso              ;Pula para o bloco de Atraso
            reti                        ;Retorna da Interrupção

Codificando:
            andi    temp,0x0F           ;Codifica apenas os quatro primeiros bits ("AND" com 00001111b)
            clr     final               ;Limpa a variável final
            mov     temp1,temp          ;Copia o valor da variável temp para a variável temp1
            com     temp                ;Faz o complemento da variável temp

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            mov     desloc,temp         ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1       ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x01        ;Codifica o primeiro bit ("AND" com 00000001b)
            andi    desloc,0x02         ;Codifica o primeiro bit ("AND" com 00000010b)
            add     desloc,desloc1      ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc        ;Adiciona o valor da variável desloc à variável final

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            lsl     temp1               ;Desloca os bits da variável temp1 para a esquerda
            mov     desloc,temp         ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1       ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x04        ;Codifica o segundo bit ("AND" com 00000100b)
            andi    desloc,0x08         ;Codifica o segundo bit ("AND" com 00001000b)
            add     desloc,desloc1      ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc        ;Adiciona o valor da variável desloc à variável final

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            lsl     temp1               ;Desloca os bits da variável temp1 para a esquerda
            mov     desloc,temp         ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1       ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x10        ;Codifica o terceiro bit ("AND" com 00010000b)
            andi    desloc,0x20         ;Codifica o terceiro bit ("AND" com 00100000b)
            add     desloc,desloc1      ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc        ;Adiciona o valor da variável desloc à variável final

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            lsl     temp1               ;Desloca os bits da variável temp1 para a esquerda
            mov     desloc,temp         ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1       ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x40        ;Codifica o quarto bit ("AND" com 01000000b)
            andi    desloc,0x80         ;Codifica o quarto bit ("AND" com 10000000b)
            add     desloc,desloc1      ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc        ;Adiciona o valor da variável desloc à variável final

            inc     XL                  ;Incrementa o ponteiro
            st      X,final             ;Armazena o dado no ponteiro

            cpi     dado,9              ;Compara para saber se todos os dados foram codificados
            breq    Transmitir          ;Caso positivo pula para o bloco Transmitir
            rjmp    MaisUm              ;Pula para o bloco MaisUm

;** Loop de Atraso
Atraso:
            cli                         ;Desabilita a Interrupção
            ldi     desloc,255
            ldi     desloc1,255
            ldi     desloc2,2
Loop:
            dec     desloc
            brne    loop
            dec     desloc1
            brne    loop
            dec     desloc2
            brne    loop

            rjmp    Transmitir          ;Pula para o bloco Transmitir

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
```

```
            ldi     temp,low(RAMEND)
            out     SPL,temp
            ldi     temp,high(RAMEND)
            out     SPH,temp
            ldi     temp,0xFF
            out     DDRB,temp           ;Port B configurada como saída
            ser     temp
            out     PORTB,temp          ;Garante 0 inicialmente na saída

;** Inicialização do Vetor de Dados
            ldi     XH,0x00

;** Programação da UART
            ldi     temp,103
            out     UBRR,temp           ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)
            ldi     temp,0b00101000
            out     UCR,temp            ;Habilita transmissão e interrupção por transmissão

;** Armazenamento das informações a serem transmitidas
            ldi     XL,0x60             ;Posiciona o ponteiro
            ldi     temp,0xCC           ;Armazena o valor 11001100b na variável temp
            st      X,temp              ;Armazena o mesmo valor no ponteiro
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o mesmo valor no ponteiro

MaisUm:
            inc     dado                ;Incrementa a variável dado
            mov     temp,dado           ;Copia o valor da variável dado para a variável temp
            mov     temp2,dado          ;Copia o valor da variável dado para a variável temp2
            rjmp    Codificando         ;Pula para o bloco Codificando

Transmitir:
            sei                         ;Habilita a Interrupção
            ldi     XL,0x5F             ;Posiciona o ponteiro

Volta:
            rjmp    Volta               ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.4Programa Comunicação Unidirecional Via RF (Receptor)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa Comunicação Unidirecional Via RF (Parte 2)
Módulo: Recepção
Linguagem: Assembler
Arquivo: CUniT_P2.asm
Objetivo: Testar a Comunicação por meio de RF entre os Módulos de Comunicação
Descrição: Uma seqüência de 9 bytes deve ser recebida através da UART sendo:
Bytes 1 e 2: CCh (Identificação de Protocolo)
Bytes 3 a 9: Dados Codificados (Seqüência numérica de 0 à 6)
Por fim, os dados decodificados são transmitidos ao PC utilizando-se a UART.
Lembrando que: 1 = 01 (Codificado)
0 = 10 (Codificado)

```
.include "8515def.inc"

;** Declaração das Variáveis
.def        temp    = r16
.def        temp1   = r17
.def        final   = r18
.def        count   = r19
.def        desloc  = r20
```

```
.def    desloc1   = r21
.def    desloc2   = r22
.def    chave     = r23

        rjmp    RESET                   ;Pula para o Programa Principal

.org URXCaddr                           ;UART Receive Complete Interrupt Vector Address
        rjmp    Receber

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
        rjmp    Convertendo

Receber:
        in      temp,UDR                ;Lê os dados recebidos
        inc     count                   ;Incrementa a variável count
        cpi     count,3                 ;Compara a variável count com 3
        brlo    Confirmar               ;Pula para o bloco Confirmar se count for menor que 3
        rjmp    Armazena                ;Pula para o bloco Armazena

;** Reconhecimento de pacote de dados válidos
Confirmar:
        cpi     temp,0xCC               ;Compara a variável temp com 11001100b
        breq    Confirmado              ;Pula para o bloco Confirmado se temp for igual a CCh
        clr     count                   ;Limpa a variável count
        reti                            ;Retorna da Interrupção

Confirmado:
        reti                            ;Retorna da Interrupção

;** Armazenamento dos dados recebidos após identificação do protocolo
Armazena:
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp                  ;Armazena o dado recebido
        cpi     count,9                 ;Compara count com 9 para armazenar até 7 caracteres recebidos
        breq    Converte                ;Pula para Decodificar os dados se count for igual à 9
        reti                            ;Retorna da Interrupção

Converte:
        cli                             ;Desabilita a Interrupção
        ldi     temp,0b00101000
        out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
        ldi     XL,0x5F                 ;Posiciona o ponteiro
        ldi     temp,0b11111111
        out     PORTB,temp              ;Desabilita comunicação com o Receptor
        sei                             ;Habilita a Interrupção
        rjmp    Volta                   ;Pula para o bloco Volta

Convertendo:
        inc     XL                      ;Incrementa o ponteiro
        ld      temp,X                  ;Carrega o dado codificado (recebido) na variável temp
        clr     final                   ;Limpa a variável final
        mov     temp1,temp              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x01               ;Verifica o valor do primeiro bit ("AND" com 00000001b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x02               ;Verifica o valor do segundo bit ("AND" com 00000010b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x04               ;Verifica o valor do terceiro bit ("AND" com 00000100b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x08               ;Verifica o valor do quarto bit ("AND" com 00001000b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        out     UDR,final               ;Envia os dados decodificados para o Registrador de Dados da UART
        out     PORTB,final             ;Envia a mesma informação para o Port B (mostra nos LED's)

;** Loop de Atraso
Atraso:
        ldi     desloc,255
        ldi     desloc1,255
        ldi     desloc2,4
Loop:
        dec     desloc
```

```
            brne    loop
            dec     desloc1
            brne    loop
            dec     desloc2
            brne    loop
            cpi     XL,0x66             ;Compara para saber se o último caractere foi enviado
            breq    DeNovo              ;Reinicia o processo se os 7 caracteres já tiverem sido enviados
            reti                        ;Retorna da Interrupção

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
            ldi     temp,low(RAMEND)
            out     SPL,temp
            ldi     temp,high(RAMEND)
            out     SPH,temp
            ldi     temp,0xFF
            out     DDRB,temp           ;Port B configurada como saída
            ser     temp
            out     PORTB,temp          ;Garante 0 inicialmente na saída

;** Inicialização do Vetor de Dados
            ldi     XH,0x00

;** Programação da UART
            ldi     temp,103
            out     UBRR,temp           ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)

DeNovo:
            cli                         ;Desabilita a Interrupção
            ldi     temp,0b11111110
            out     PORTB,temp          ;Habilita a comunicação com o Receptor
            ldi     temp,0b10010000
            out     UCR,temp            ;Habilita recepção e interrupção por recepção completa de dados
            clr     count               ;Limpa a variável count
            clr     temp                ;Limpa a variável temp
            ldi     XL,0x5F             ;Posiciona o ponteiro
            sei                         ;Habilita a Interrupção

Volta:
            rjmp    Volta               ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.5 Programa Teste do Funcionamento do Sensor de Ultrasom (Parte 2)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título:	Programa Teste do Funcionamento do Sensor (Sonar) (Parte 2)
Linguagem:	Assembler
Arquivo:	Sonar.asm
Objetivo:	Testar o funcionamento do sensor de Ultrasom, medindo a distância entre o Sensor e um obstáculo qualquer.
Descrição:	Através do Pino 0 da Porta B do microcontrolador, é medido o tempo em que a saída do Sensor fica em nível alto. Esse tempo é medido e enviado pela UART para o Computador Pessoal, o qual mostra na tela do Computador Pessoal o tempo medido em decimal.

```
.include "8515def.inc"

;** Declaração das Variáveis

.def        temp    = r16
.def        temp2   = r17
.def        desloc  = r18
```

```
.def    desloc1   = r19
.def    desloc2   = r20
.def    temp3     = r21

        rjmp    RESET                   ;Pula para o Programa Principal

;** Bloco para iniciar a transmissão dos dados ao Computador Pessoal

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
        rjmp    Enviar

Enviar:
        inc     XL                      ;Posiciona o ponteiro
        ld      temp,X                  ;Carrega o dado na variável temp
        out     UDR,temp                ;Envia os dados armazenados na variável temp para o
                                        ;registrador de dados da UART
        ldi     desloc2,2
        rcall   Atraso                  ;Chama o bloco Atraso
        cpi     XL,0x62                 ;Compara para saber se o último caractere foi enviado
        breq    DeNovo                  ;Pula para o bloco DeNovo (reiniciar o processo)
        reti                            ;Retorna da Interrupção

;** Bloco para parar a contagem do Timer
ParарCont:
        ldi     temp,(0<<CS11)|(0<<CS10)
        out     TCCR1B,temp             ;Para o Timer
        in      temp2,TCNT1L            ;Lê o valor do byte menos significativo do registrador de 16 bits
                                        ;e armazena na variável temp2
        in      temp3,TCNT1H            ;Lê o valor do byte mais significativo do registrador de 16 bits
                                        ;e armazena na variável temp3
;** Armazenamento das informações a serem transmitidas
        ldi     XL,0x60                 ;Posiciona o ponteiro
        ldi     temp,0xCC               ;Armazena o valor 11001100b na variável temp
        st      X,temp3                 ;Armazena o valor do byte mais significativo do registrador de 16 bits
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp2                 ;Armazena o valor do byte menos significativo do registrador de 16 bits
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp                  ;Armazena o valor 11001100b
        ldi     XL,0x5F                 ;Posiciona o ponteiro
        clr     temp                    ;Limpa a variável temp
        out     TCNT1H,temp             ;Limpa o registrador de 16 bits
        out     TCNT1L,temp
        rjmp    Transmitir              ;Pula para o bloco Transmitir

;** Loop de Atraso
Atraso:
        cli                             ;Desabilita a Interrupção
        ldi     desloc,255
        ldi     desloc1,255
Loop:
        dec     desloc
        brne    loop
        dec     desloc1
        brne    loop
        dec     desloc2
        brne    loop
        ret

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp

        ldi     temp,0b11111110
        out     DDRB,temp               ;Pino 0 do Port B configurado como entrada
                                        ;demais pinos configurados como saída
;** Inicialização do Vetor de Dados
        ldi     XH,0x00

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp               ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)
```

```
DeNovo:
        clr     temp
        out     UCR,temp            ;Desabilita transmissão e interrupção por transmissão
        ldi     desloc2,6
        rcall   Atraso              ;Chama o bloco de Atraso

;** Bloco de contagem do tempo em nível alto do pulso de saída do Sensor
Verifica1:
        sbis    PINB,0              ;Pula a próxima instrução se o pino 0 do Port B estiver em nível alto
        rjmp    Verifica1           ;Aguarda até que o pino 0 do Port B fique em nível alto
                                    ;(caso esteja em nível baixo, para iniciar a contagem do tempo)
        ldi     temp,(0<<CS11)|(1<<CS10)
        out     TCCR1B,temp         ;Ajuste do prescaler 1:1 (Tempo = Clock/1)

Verifica2:
        sbis    PINB,0              ;Pula a próxima instrução se o pino 0 do Port B estiver em nível alto
        rjmp    PararCont           ;Pula para o bloco PararCont assim que o pino 0 do Port B fique em
                                    ;nível baixo
        rjmp    Verifica2           ;Aguarda até que o pino 0 do Port B fique em nível baixo

;** Bloco para configurar o início da transmissão dos dados ao Computador Pessoal
Transmitir:
        ldi     temp,0b00101000
        out     UCR,temp            ;Habilita transmissão e interrupção por transmissão
        sei                         ;Habilitação geral das interrupções

Volta:
        rjmp    Volta               ;Pula para o bloco Volta até surgir uma interrupção

;** FIM DO PROGRAMA
```

10.1.6 Programa Teste para o Acionamento dos Motores (Transmissor)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa para o acionamento dos motores (Parte 1)
Módulo: Transmissão
Linguagem: Assembler
Arquivo: AcionaMT.asm
Objetivo: Testar o funcionamento dos motores de acordo com o comando enviado.
Descrição: Uma seqüência de 5 bytes deve ser recebida através da UART sendo:
- Bytes 1 e 2: CCh (Identificação de Protocolo)
- Byte 3: Direção dos Motores
- Bytes 4 a 5: Velocidade dos motores 1 e 2 respectivamente

Por fim, os dados são codificados e transmitidos ao módulo de recepção utilizando-se a comunicação UART.

```
.include "8515def.inc"

;** Declaração das Variáveis

.def    temp    = r16
.def    temp1   = r17
.def    temp2   = r18
.def    final   = r19
.def    count   = r20
.def    desloc  = r21
.def    desloc1 = r22
.def    chave   = r23

        rjmp    RESET               ;Pula para o Programa Principal

.org URXCaddr                       ;UART Receive Complete Interrupt Vector Address
```

```
            rjmp    Receber

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
            rjmp    Transmitindo

;** Bloco de transmissão dos dados codificados
Transmitindo:
            inc     XL                  ;Posiciona o ponteiro
            ld      temp,X              ;Carrega o dado armazenado na variável temp
            out     UDR,temp            ;Envia os dados armazenados na variável temp para o registrador de
                                        ;dados da UART
            out     PORTB,temp          ;Envia a mesma informação para o Port B
            cpi     XL,0x68             ;Compara para saber se o último caractere foi enviado
            breq    Atraso              ;Pula para o bloco Atraso se todos os dados foram enviados
            reti                        ;Retorna da Interrupção

;** Loop de Atraso
Atraso:
            ldi     desloc,255
            ldi     desloc1,255
Loop:
            dec     desloc
            brne    loop
            dec     desloc1
            brne    loop
            rjmp    DeNovo              ;Pula para o bloco DeNovo, para reiniciar o processo

;** Bloco que faz a leitura dos dados enviados pela serial do PC
Receber:
            in      temp,UDR            ;Lê os dados recebidos
            inc     count               ;Incrementa a variável count
            cpi     count,3             ;Compara a variável count com 3
            brlo    Confirmar           ;Pula para o bloco Confirmar se count for menor que 3
            rjmp    Armazena            ;Pula para o bloco Armazena se count for maior que 3

;** Reconhecimento de pacote de dados válidos
Confirmar:
            cpi     temp,0xCC           ;Compara a variável temp com 11001100b
            breq    Confirmado          ;Pula para o bloco Confirmado se temp for igual a CCh
            clr     count               ;Limpa a variável count se o pacote não for reconhecido
            reti                        ;Retorna da Interrupção

Confirmado:
            reti                        ;Retorna da Interrupção

;** Armazenamento dos dados recebidos após identificação do pacote de dados
Armazena:
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o dado recebido
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o dado recebido
            cpi     count,5             ;Compara count com 5 para armazenar até 3 caracteres recebidos
            breq    Posicionando        ;Pula para codificar os dados se count for igual à 5
            reti                        ;Retorna da Interrupção

Posicionando:
            cli                         ;Desabilita a Interrupção
            ldi     XL,0x61             ;Posiciona o ponteiro
            clr     count               ;Limpa a variável count
            inc     XL                  ;Incrementa o ponteiro
            ld      temp,X              ;Carrega o dado na variável temp

Codificando_P1:
            clr     temp2               ;Limpa a variável temp2
            inc     count               ;Incrementa a variável count
            cpi     count,4             ;Compara para saber se todos os dados foram codificados
            breq    Transmitir          ;Caso positivo pula para o bloco Transmitir
            andi    temp,0x0F           ;Codifica apenas os quatro primeiros bits ("AND" com 00001111b)

;** Bloco de codificação dos dados
Codificando_P2:
            clr     final               ;Limpa a variável final
            mov     temp1,temp          ;Copia o valor da variável temp para a variável temp1
            com     temp                ;Faz o complemento da variável temp

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
```

```
            mov     desloc,temp             ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1           ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x01            ;Codifica o primeiro bit ("AND" com 00000001b)
            andi    desloc,0x02             ;Codifica o primeiro bit ("AND" com 00000010b)
            add     desloc,desloc1          ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc            ;Adiciona o valor da variável desloc à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     desloc,temp             ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1           ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x04            ;Codifica o segundo bit ("AND" com 00000100b)
            andi    desloc,0x08             ;Codifica o segundo bit ("AND" com 00001000b)
            add     desloc,desloc1          ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc            ;Adiciona o valor da variável desloc à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     desloc,temp             ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1           ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x10            ;Codifica o terceiro bit ("AND" com 00010000b)
            andi    desloc,0x20             ;Codifica o terceiro bit ("AND" com 00100000b)
            add     desloc,desloc1          ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc            ;Adiciona o valor da variável desloc à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     desloc,temp             ;Copia o valor da variável temp para a variável desloc
            mov     desloc1,temp1           ;Copia o valor da variável temp1 para a variável desloc1
            andi    desloc1,0x40            ;Codifica o quarto bit ("AND" com 01000000b)
            andi    desloc,0x80             ;Codifica o quarto bit ("AND" com 10000000b)
            add     desloc,desloc1          ;Adiciona o valor da variável desloc1 à variável desloc
            add     final,desloc            ;Adiciona o valor da variável desloc à variável final

;** Bloco de verificação se o byte inteiro foi codificado
            st      X,final                 ;Armazena o dado no ponteiro
            inc     XL                      ;Incrementa o ponteiro
            ld      temp,X                  ;Lê os dados e armazena na variável temp
            cpi     temp2,0x01              ;Compara se a variável temp2 é igual a 1
            breq    Codificando_P1          ;Pula para o bloco Codificando_P1 se o byte inteiro foi codificado
            andi    temp,0xF0               ;Codifica apenas os quatro últimos bits ("AND" com 11110000b)
            swap    temp                    ;Troca de posição os quatro últimos bits pelos quatro primeiros
            ldi     temp2,0x01              ;Carrega 1 na variável temp2
            rjmp    Codificando_P2          ;Pula para o bloco que faz a codificação dos dados


;** Bloco de configuração para iniciar a transmissão dos dados codificados
Transmitir:
            ldi     XL,0x5E                 ;Posiciona o ponteiro
            ldi     temp,0b00101000
            out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
            sei                             ;Habilita a Interrupção
            rjmp    Volta                   ;Pula para o bloco Volta e fica aguardando a próxima Interrupção

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
            ldi     temp,low(RAMEND)
            out     SPL,temp
            ldi     temp,high(RAMEND)
            out     SPH,temp

            ldi     temp,0xFF
            out     DDRB,temp               ;Port B configurada como saída

            ser     temp
            out     PORTB,temp              ;Garante 0 inicialmente na saída

;** Inicialização do Vetor de Dados
            ldi     XH,0x00

;** Programação da UART
            ldi     temp,103
            out     UBRR,temp               ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)
```

```
;** Armazenamento dos caracteres de reconhecimento do pacote
        ldi     XL,0x60                 ;Posiciona o ponteiro
        ldi     temp,0xCC               ;Armazena o valor 11001100b na variável temp
        st      X,temp                  ;Armazena o mesmo valor na memória
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp                  ;Armazena o mesmo valor na memória

DeNovo:
        cli                             ;Desabilita a Interrupção
        ldi     temp,0b10010000
        out     UCR,temp                ;Habilita recepção e interrupção por recepção completa de dados
        clr     count                   ;Limpa a variável count
        clr     temp                    ;Limpa a variável temp
        ldi     XL,0x61                 ;Posiciona o ponteiro
        sei                             ;Habilita a Interrupção

Volta:
        rjmp    Volta                   ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.7 Programa Teste para o Acionamento dos Motores (Receptor)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa para o acionamento dos motores (Parte 1)
Módulo: Recepção
Linguagem: Assembler
Arquivo: AcionaMR.asm
Objetivo: Testar o funcionamento dos motores de acordo com o comando enviado.
Descrição: Uma seqüência de 5 bytes deve ser recebida através da UART sendo:
- Bytes 1 e 2: CCh (Identificação de Protocolo)
- Byte 3: Direção dos Motores
- Bytes 4 a 5: Velocidade dos motores 1 e 2 respectivamente

Por fim, os dados são recebidos deverão ser decodificados.

```
.include "8515def.inc"

;** Declaração das Variáveis
.def    temp    = r16
.def    temp1   = r17
.def    final   = r18
.def    vfinal  = r10
.def    count   = r20
.def    chave   = r21
.def    ponteiro1 = r22
.def    ponteiro2 = r23


        rjmp    RESET                   ;Pula para o Programa Principal

.org URXCaddr                           ;UART Receive Complete Interrupt Vector Address
        rjmp    Receber

Receber:
        in      temp,UDR                ;Lê os dados recebidos
        inc     count                   ;Incrementa a variável count
        cpi     count,3                 ;Compara a variável count com 3
        brlo    Confirmar               ;Pula para o bloco Confirmar se count for menor que 3
        rjmp    Armazena                ;Pula para o bloco Armazena se count for maior que 3

;** Reconhecimento de pacote de dados válidos
Confirmar:
        cpi     temp,0xCC               ;Compara a variável temp com 11001100b
```

```
        breq    Confirmado              ;Pula para o bloco Confirmado se temp for igual a CCh
        clr     count                   ;Limpa a variável count se não for reconhecido o pacote de dados
        reti                            ;Retorna da Interrupção

Confirmado:
        reti                            ;Retorna da Interrupção

;** Armazenamento dos dados recebidos após identificação do pacote
Armazena:
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp                  ;Armazena o dado recebido
        cpi     count,8                 ;Compara count com 8 para armazenar até 6 caracteres recebidos
        breq    Converte                ;Pula para decodificar os dados se count for igual à 8
        reti                            ;Retorna da Interrupção

Converte:
        cli                             ;Desabilita a Interrupção
        clr     count                   ;Limpa a variável count
        ldi     XL,0x5F                 ;Posiciona o ponteiro

Convertendo:
        inc     count                   ;Incrementa o contador
        inc     XL                      ;Incrementa o ponteiro
        ld      temp,X                  ;Carrega o dado codificado (recebido) na variável temp
        clr     final                   ;Limpa a variável final
        mov     temp1,temp              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x01               ;Verifica o valor do primeiro bit ("AND" com 00000001b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x02               ;Verifica o valor do segundo bit ("AND" com 00000010b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x04               ;Verifica o valor do terceiro bit ("AND" com 00000100b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x08               ;Verifica o valor do quarto bit ("AND" com 00001000b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final

        cpi     count,1                 ;Compara count com 1 para saber se apenas a primeira parte do byte
                                        ;foi decodificado
        breq    Parte1                  ;Pula para o bloco Parte 1 se verdadeiro
        rjmp    Parte2                  ;Pula para o bloco Parte 2 se já tiver sido decodificado o byte inteiro

Parte1:
        mov     vfinal,final            ;Armazena a primeira parte do byte decodificado na variável vfinal
        rjmp    Convertendo             ;Pula para o bloco Convertendo para continuar com a decodificação dos dados

Parte2:
        swap    final                   ;Inverte a posição dos últimos bits pelos primeiros
        add     vfinal,final            ;Adiciona a segunda parte do byte decodificado à primeira parte
        mov     ponteiro1,XL            ;Armazena a posição do ponteiro na variável ponteiro1
        cpi     ponteiro1,0x62          ;Verifica se o primeiro byte recebido já foi decodificado e
                                        ;armazenado
        brsh    Pula                    ;Pula para o bloco Pula se verdadeiro
        ldi     XL,0x80                 ;Posiciona o ponteiro
        mov     ponteiro2,XL            ;Armazena o valor do ponteiro na variável ponteiro2
        st      X,vfinal                ;Armazena o dado decodificado na memória
        clr     count                   ;Limpa a variável count
        clr     vfinal                  ;Limpa a variável vfinal
        mov     XL,ponteiro1            ;Posiciona o ponteiro
        rjmp    Convertendo             ;Pula para o bloco Convertendo

Pula:
        inc     ponteiro2               ;Posiciona o ponteiro
        mov     XL,ponteiro2            ;Armazena o valor do ponteiro na variável ponteiro2
        st      X,vfinal                ;Armazena o dado no ponteiro
        clr     count                   ;Limpa a variável count
        clr     vfinal                  ;Limpa a variável vfinal
        cpi     ponteiro2,0x82          ;Verifica se todos os dados recebidos foram decodificados e
                                        ;armazenados
        breq    Aciona                  ;Pula para o bloco Aciona se verdadeiro
        mov     XL,ponteiro1            ;Posiciona o ponteiro
        rjmp    Convertendo             ;Pula para o bloco Convertendo
```

```
;** Bloco de acionamento dos motores
Aciona:
        ldi     XL,0x80             ;Posiciona o ponteiro
        ld      temp,X              ;Copia o dado armazenado na variável temp
        out     PORTB,temp          ;Manda para o PortB o comando de direção dos motores
        inc     XL                  ;Incrementa o ponteiro
        ld      temp,X              ;Copia o dado armazenado na variável temp
        out     OCR1AL,temp         ;Configura a velocidade do PWM A de acordo com o valor recebido
        inc     XL                  ;Incrementa o ponteiro
        ld      temp,X              ;Copia o dado armazenado na variável temp
        out     OCR1BL,temp         ;Configura a velocidade do PWM B de acordo com o valor recebido
        rjmp    DeNovo              ;Pula para o bloco DeNovo para reiniciar o processo

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp
        ldi     temp,0b00100000
        out     DDRD,temp           ;Pino OC1A definido como saída demais bits do PortD configurados como entrada
        ldi     temp,0xFF
        out     DDRB,temp           ;Port B configurada como saída
        ser     temp
        out     PORTB,temp          ;Garante 0 inicialmente na saída

;** Inicialização do Vetor de Dados
        ldi     XH,0x00

;** Configuração do PWM (A e B)
        ldi     temp,0b11110001
        out     TCCR1A,temp         ;Configuração do modo PWM 8 bits e seleção da seguinte ação
                                    ;para os pinos OC1B e OC1A (PD5):
                                    ;Clear quando o valor de comparação atingido na contagem crescente
                                    ;Seta quando o valor de comparação atingido na contagem decrescente
                                    ;O pino OC1B é a saída PWM para o motor 2
                                    ;O pino OC1A (PD5) é a saída PWM para o motor 1
        ldi     temp,0b00000001
        out     TCCR1B,temp         ;Libera timer1, incrementado pelo clock sem prescaler
                                    ;Portanto freqüência do PWM = freqüência do clock
        ldi     temp,0              ;Zera o conteúdo dos registradores altos
        out     OCR1AH,temp         ;Referências para o PWM
        out     OCR1BH,temp
        ldi     temp,255            ;Força a velocidade inicial = 0
        out     OCR1AL,temp
        out     OCR1BL,temp

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp           ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)
        ldi     temp,0b10010000
        out     UCR,temp            ;Habilita recepção e interrupção por recepção completa de dados

DeNovo:
        clr     count               ;Limpa a variável count
        clr     temp                ;Limpa a variável temp
        ldi     XL,0x5F             ;Posiciona o ponteiro
        sei                         ;Habilita a Interrupção

Volta:
        rjmp    Volta               ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.8 Programa Teste para o Acionamento dos Motores

PROGRAMA PARA COMPUTADOR PESSOAL

Título:	Programa para o acionamento dos motores (Parte 1)
Linguagem:	Delphi (Pascal)
Arquivo:	Int_v01.pas
Objetivo:	Testar o funcionamento dos motores de acordo com o comando enviado.
Descrição:	Uma seqüência de 5 bytes é enviada através da comunicação serial do PC: Bytes 1 e 2: CCh (Identificação de Protocolo) Byte 3: Direção dos Motores Bytes 4 a 5: Velocidade dos motores 1 e 2 respectivamente.

```
unit Integ_v01;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Menus, ExtCtrls, FileCtrl, Buttons, jpeg, Registry;

type TMultlinha = class(TThread)
    private
    protected
    procedure Execute; override;
    procedure MostraString;    //Mostra dados na tela.
end;

type
  TForm1 = class(TForm)
    SpeedAbrirPorta: TSpeedButton;
    SpeedFecharPorta: TSpeedButton;
    SpeedSair: TSpeedButton;
    Label2: TLabel;
    Label41: TLabel;
    LabelStrRecebida: TLabel;
    Bevel1: TBevel;
    ComboBoxPorta: TComboBox;
    Button1: TButton;
    Label1: TLabel;
    BotaoEsquerdaM12: TButton;
    Bevel2: TBevel;
    Label3: TLabel;
    BotaoDireitaM12: TButton;
    BotaoPraTrasM12: TButton;
    BotaoFrenteM12: TButton;
    BotaodeTeste: TButton;
    BotaoParaM12: TButton;
    BotaoFrenteM1: TButton;
    Bevel3: TBevel;
    Label4: TLabel;
    BotaoPraTrasM1: TButton;
    Label5: TLabel;
    BotaoAceleraM12: TButton;
    ScrollBarM12: TScrollBar;
    EditM12: TEdit;
    Memo1: TMemo;
    ScrollBarM1: TScrollBar;
    EditM1: TEdit;
    Bevel5: TBevel;
    BotaoEnviaM1: TBitBtn;
    BotaoEnviaM12: TBitBtn;
    BotaoEnviaTeste: TBitBtn;
    Bevel4: TBevel;
    BotaoFrenteM2: TButton;
    BotaoPraTrasM2: TButton;
    ScrollBarM2: TScrollBar;
    EditM2: TEdit;
    BotaoEnviaM2: TBitBtn;
```

```
    procedure FormCreate(Sender: TObject);
    procedure SpeedAbrirPortaClick(Sender: TObject);
    procedure SpeedSairClick(Sender: TObject);
    procedure SpeedFecharPortaClick(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure MostraPortasCom();
    procedure Button1Click(Sender: TObject);
    procedure BotaodeTesteClick(Sender: TObject);
    procedure ScrollBarM2Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
    procedure BotaoEsquerdaM12Click(Sender: TObject);
    procedure BotaoEnviaM12Click(Sender: TObject);
    procedure BotaoEnviaTesteClick(Sender: TObject);
    procedure BotaoEnviaM1Click(Sender: TObject);
    procedure BotaoEnviaM2Click(Sender: TObject);
    procedure BotaoFrenteM12Click(Sender: TObject);
    procedure BotaoDireitaM12Click(Sender: TObject);
    procedure BotaoPraTrasM12Click(Sender: TObject);
    procedure BotaoParaM12Click(Sender: TObject);
    procedure BotaoFrenteM1Click(Sender: TObject);
    procedure BotaoPraTrasM1Click(Sender: TObject);
    procedure BotaoFrenteM2Click(Sender: TObject);
    procedure BotaoPraTrasM2Click(Sender: TObject);
    procedure ScrollBarM1Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
    procedure ScrollBarM12Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
    procedure BotaoAceleraM12Click(Sender: TObject);

  private
    { Private declarations }
  public
    { Public declarations }
  end;

const
  LEN_BUFFER = 100;   //Tamanho dos Buffers de dados.

var
  Form1: TForm1;
  hCom: THANDLE;   //Handle da Porta Serial (API).
  dcb: TDCB;   //Estrutura DCB (API).
  CommTimeouts: TCOMMTIMEOUTS;   //Estrutura TCOMMTIMEOUTS(API).
  BytesLidos: DWORD;
  BytesEscritos: DWORD;
  BufferRecebe : array [0..LEN_BUFFER] of Integer;   //Define o buffer.
  GLB_Conectado:boolean;   //Indica se a porta já está aberta.
  GLBEnviaDados:bool;   //Para habilitar/desabilitar o envio de dados pela serial.
  VRecebido: String;
  Acesso: Integer;
  D_Sent: array [0..90] of Byte;

Implementation

…(Conteúdo idêntico ao apresentado no Anexo 15.1.2)…

…continuação:

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
   if(GLB_Conectado = true) then
     CloseHandle(hCom);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    Form1.Memo1.Clear
end;

procedure TForm1.BotaodeTesteClick(Sender: TObject);
begin
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      if Acesso = 0 then
        begin
          Add('11000000b - 192d - C0h');
```

```
    D_Sent[3]:=192;
  end;
if Acesso = 1 then
  begin
    Add('11000001b - 193d - C1h');
    D_Sent[3]:=193;
  end;
if Acesso = 2 then
  begin
    Add('11000010b - 194d - C2h');
    D_Sent[3]:=194;
  end;
if Acesso = 3 then
  begin
    Add('11000011b - 195d - C3h');
    D_Sent[3]:=195;
  end;
if Acesso = 4 then
  begin
    Add('11000100b - 196d - C4h');
    D_Sent[3]:=196;
  end;
if Acesso = 5 then
  begin
    Add('11000101b - 197d - C5h');
    D_Sent[3]:=197;
  end;
if Acesso = 6 then
  begin
    Add('11000110b - 198d - C6h');
    D_Sent[3]:=198;
  end;
if Acesso = 7 then
  begin
    Add('11000111b - 199d - C7h');
    D_Sent[3]:=199;
  end;
if Acesso = 8 then
  begin
    Add('11001000b - 200d - C8h');
    D_Sent[3]:=200;
  end;
if Acesso = 9 then
  begin
    Add('11001001b - 201d - C9h');
    D_Sent[3]:=201;
  end;
if Acesso = 10 then
  begin
    Add('11001010b - 202d - CAh');
    D_Sent[3]:=202;
  end;
if Acesso = 11 then
  begin
    Add('11001011b - 203d - CBh');
    D_Sent[3]:=203;
  end;
if Acesso = 12 then
  begin
    Add('11001100b - 204d - CCh');
    D_Sent[3]:=204;
  end;
if Acesso = 13 then
  begin
    Add('11001101b - 205d - CDh');
    D_Sent[3]:=205;
end;
if Acesso = 14 then
  begin
    Add('11001110b - 206d - CEh');
    D_Sent[3]:=206;
  end;
if Acesso = 15 then
  begin
    Add('11001111b - 207d - CFh');
    D_Sent[3]:=207;
```

```
        Acesso:=-1;
       end;
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
      D_Sent[1]:=204;
      D_Sent[2]:=204;
      D_Sent[4]:=0;
      D_Sent[5]:=0;
      Acesso:=Acesso+1;
    End;
end;

procedure TForm1.BotaoEsquerdaM12Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=200;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM12.Position:=D_Sent[4];
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11001000b - 200d - C8h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.BotaoEnviaM12Click(Sender: TObject);
begin
  D_Sent[4]:=Form1.ScrollBarM12.Position;
  D_Sent[5]:=Form1.ScrollBarM12.Position;
  WriteFile( hCom,D_Sent,6, BytesEscritos, nil); //Envia string.
end;

procedure TForm1.BotaoEnviaTesteClick(Sender: TObject);
begin
  WriteFile( hCom,D_Sent,6, BytesEscritos, nil); //Envia string.
end;

procedure TForm1.BotaoEnviaM1Click(Sender: TObject);
begin
  D_Sent[4]:=Form1.ScrollBarM1.Position;
  WriteFile( hCom,D_Sent,6, BytesEscritos, nil); //Envia string.
end;

procedure TForm1.BotaoEnviaM2Click(Sender: TObject);
begin
  D_Sent[5]:=Form1.ScrollBarM2.Position;
  WriteFile( hCom,D_Sent,6, BytesEscritos, nil); //Envia string.
end;

procedure TForm1.BotaoFrenteM12Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=201;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM12.Position:=D_Sent[4];
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11001001b - 201d - C9h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;
```

```
procedure TForm1.BotaoDireitaM12Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=193;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM12.Position:=D_Sent[4];
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11000001b - 193d - C1h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.BotaoPraTrasM12Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=198;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM12.Position:=D_Sent[4];
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11000110b - 198d - C6h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.BotaoParaM12Click(Sender: TObject);
begin
  Memo1.Clear;
  Form1.ScrollBarM12.Position:=255;
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
end;

procedure TForm1.BotaoFrenteM1Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=193;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM1.Position:=D_Sent[4];
  Form1.EditM1.Text:=IntToStr(Form1.ScrollBarM1.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11000001b - 193d - C1h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.BotaoPraTrasM1Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=194;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM1.Position:=D_Sent[4];
  Form1.EditM1.Text:=IntToStr(Form1.ScrollBarM1.Position);
```

```
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11000010b - 194d - C2h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.BotaoFrenteM2Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=200;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM2.Position:=D_Sent[5];
  Form1.EditM2.Text:=IntToStr(Form1.ScrollBarM2.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11001000b - 200d - C8h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.BotaoPraTrasM2Click(Sender: TObject);
begin
  D_Sent[1]:=204;
  D_Sent[2]:=204;
  D_Sent[3]:=196;
  D_Sent[4]:=0;
  D_Sent[5]:=0;
  Form1.ScrollBarM2.Position:=D_Sent[5];
  Form1.EditM2.Text:=IntToStr(Form1.ScrollBarM2.Position);
  Memo1.Clear;
  with Memo1.Lines do
    begin
      Add('11001100b - 204d - CCh');
      Add('11001100b - 204d - CCh');
      Add('11000100b - 196d - C4h');
      Add('00000000b - 000d - 00h');
      Add('00000000b - 000d - 00h');
    end;
end;

procedure TForm1.ScrollBarM12Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
end;

procedure TForm1.ScrollBarM1Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  Form1.EditM1.Text:=IntToStr(Form1.ScrollBarM1.Position);
end;

procedure TForm1.ScrollBarM2Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  Form1.EditM2.Text:=IntToStr(Form1.ScrollBarM2.Position);
end;

procedure TForm1.BotaoAceleraM12Click(Sender: TObject);
begin
  Memo1.Clear;
  Form1.ScrollBarM12.Position:=Form1.ScrollBarM12.Position-1;
  Form1.EditM12.Text:=IntToStr(Form1.ScrollBarM12.Position);
  D_Sent[4]:=Form1.ScrollBarM12.Position;
  D_Sent[5]:=Form1.ScrollBarM12.Position;
end;

end.
```

//FIM DO PROGRAMA

10.1.9 Programa Teste da Comunicação Bidirecional (IHM)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa para o teste de comunicação bidirecional
Módulo: IHM
Linguagem: Assembler
Arquivo: CBiIHM.asm
Objetivo: Testar o funcionamento da comunicação bidirecional e do protocolo desenvolvido.
Descrição: Parte 1:
Uma seqüência de 5 bytes deve ser recebida através da UART (comunicação serial), sendo:
Bytes 1 e 2: CCh (identificação do pacote de dados)
Byte 3: Direção dos motores
Bytes 4 e 5: Velocidade dos motores 1 e 2 respectivamente
Os dados são codificados e transmitidos ao robô utilizando-se comunicação UART.

Parte 2:
Uma seqüência de 8 bytes deve ser recebida através da UART (módulo de recepção), sendo:
Bytes 1 e 2: CCh (identificação do pacote de dados)
Bytes 3 e 4: A9h e AAh (caractere decodificado = 1 em decimal)
Bytes 5 e 6: A6h e AAh (caractere decodificado = 2 em decimal)
Bytes 7 e 8: A5h e AAh (caractere decodificado = 3 em decimal)
Por fim, os dados são decodificados e transmitidos à IHM utilizando-se a comunicação UART.

```
.include "8515def.inc"

;** Declaração das Variáveis
.def    temp    = r16
.def    temp1   = r17
.def    temp2   = r18
.def    final   = r19
.def    vfinal  = r20
.def    count   = r21
.def    ponteiro1 = r22
.def    ponteiro2 = r23
.def    chave   = r24

        rjmp    RESET                   ;Pula para o Programa Principal

.org URXCaddr                           ;UART Receive Complete Interrupt Vector Address
        rjmp    Receber

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
        rjmp    Transmitindo

;** Bloco de transmissão dos dados
Transmitindo:
        inc     XL                      ;Posiciona o ponteiro
        ld      temp,X                  ;Carrega o dado armazenado na variável temp
        out     UDR,temp                ;Envia os dados armazenados na variável temp para o registrador de
                                        ;dados da UART
        out     PORTA,temp              ;Envia a mesma informação para o Port A
        cpi     chave,0b11111110        ;Verifica se está transmitindo para a IHM ou para o Robô
        breq    Transmitindo_IHM        ;Se for para a IHM pula para o bloco Transmitindo_IHM
        cpi     XL,0x68                 ;Compara para saber se o último caractere foi enviado
        breq    Atraso                  ;Pula para o bloco Atraso se todos os dados foram enviados
        reti                            ;Retorna da Interrupção

Atraso:
```

```
            rjmp    Atraso_P1

Transmitindo_IHM:
;** Bloco de atraso, para o envio dos dados ao PC
            ldi     ponteiro1,255
            ldi     ponteiro2,255
            ldi     temp,4
Loop2:
            dec     ponteiro1
            brne    loop2
            dec     ponteiro2
            brne    loop2
            dec     temp
            brne    loop2

            cpi     XL,0x82             ;Compara para saber se o último caractere foi enviado
            breq    Atraso              ;Pula para o bloco Atraso se todos os dados foram enviados
            reti                        ;Retorna da Interrupção

;** Bloco que faz a leitura dos dados recebidos
Receber:
            in      temp,UDR            ;Lê os dados recebidos
            inc     count               ;Incrementa a variável count
            cpi     count,3             ;Compara a variável count com 3
            brlo    Confirmar           ;Pula para o bloco Confirmar se count for menor que 3
            rjmp    Armazena            ;Pula para o bloco Armazena se count for maior que 3

;** Reconhecimento de pacote de dados válidos
Confirmar:
            cpi     temp,0xCC           ;Compara a variável temp com 11001100b
            breq    Retornando          ;Pula para o bloco Retornando se temp for igual a CCh
            clr     count               ;Limpa a variável count se o pacote não for reconhecido
            reti                        ;Retorna da Interrupção

Retornando:
            reti                        ;Retorna da interrupção

;** Armazenamento dos dados recebidos após identificação do pacote de dados
Armazena:
            cpi     chave,0b11111101    ;Verifica se está recebendo da IHM ou do Robô
            breq    Armazena_IHM        ;Se for da IHM pula para o bloco Armazena_IHM
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o dado recebido
            cpi     count,8             ;Compara count com 8 para armazenar até 6 caracteres recebidos
            breq    Converte            ;Pula para decodificar a informação recebida
            reti                        ;Retorna da Interrupção

Armazena_IHM:
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o dado recebido
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o dado recebido

            cpi     count,5             ;Compara count com 5 para armazenar até 3 caracteres recebidos
            breq    Codificar           ;Pula para codificar os dados se count for igual à 5
            reti                        ;Retorna da Interrupção

Codificar:
            rjmp    Posicionando

;** Bloco que faz a decodificação dos dados recebidos
Converte:
            cli                         ;Desabilita a Interrupção
            clr     count               ;Limpa a variável count
            ldi     XL,0x61             ;Posiciona o ponteiro

Convertendo:
            inc     count               ;Incrementa o contador
            inc     XL                  ;Incrementa o ponteiro
            ld      temp,X              ;Carrega o dado codificado (recebido) na variável temp
            clr     final               ;Limpa a variável final
            mov     temp1,temp          ;Copia o valor da variável temp para a variável temp1
            andi    temp,0x01           ;Verifica o valor do primeiro bit ("AND" com 00000001b)
            add     final,temp          ;Adiciona o valor da variável temp à variável final
            lsr     temp1               ;Desloca os bits da variável temp1 para a direita
            mov     temp,temp1          ;Copia o valor da variável temp para a variável temp1
```

```
            andi    temp,0x02           ;Verifica o valor do segundo bit ("AND" com 00000010b)
            add     final,temp          ;Adiciona o valor da variável temp à variável final
            lsr     temp1               ;Desloca os bits da variável temp1 para a direita
            mov     temp,temp1          ;Copia o valor da variável temp para a variável temp1
            andi    temp,0x04           ;Verifica o valor do terceiro bit ("AND" com 00000100b)
            add     final,temp          ;Adiciona o valor da variável temp à variável final
            lsr     temp1               ;Desloca os bits da variável temp1 para a direita
            mov     temp,temp1          ;Copia o valor da variável temp para a variável temp1
            andi    temp,0x08           ;Verifica o valor do quarto bit ("AND" com 00001000b)
            add     final,temp          ;Adiciona o valor da variável temp à variável final

            cpi     count,1             ;Compara count com 1 para saber se apenas a primeira parte do byte foi decodificado
            breq    Parte1              ;Pula para o bloco Parte 1 se verdadeiro
            rjmp    Parte2              ;Pula para o bloco Parte 2 se já tiver sido decodificado o byte inteiro

Parte1:
            mov     vfinal,final        ;Armazena a primeira parte do byte decodificado na variável vfinal
            rjmp    Convertendo         ;Pula para o bloco Convertendo para continuar com a decodificação dos dados

Parte2:
            swap    final               ;Inverte a posição dos últimos bits pelos primeiros
            add     vfinal,final        ;Adiciona a segunda parte do byte decodificado à primeira parte
            mov     ponteiro1,XL        ;Armazena a posição do ponteiro na variável ponteiro1
            cpi     ponteiro1,0x64      ;Verifica se o primeiro byte recebido já foi decodificado e
                                        ;armazenado
            brsh    Pula                ;Pula para o bloco Pula se verdadeiro

            ldi     XL,0x80             ;Posiciona o ponteiro
            mov     ponteiro2,XL        ;Armazena o valor do ponteiro na variável ponteiro2
            st      X,vfinal            ;Armazena o dado decodificado na memória
            clr     count               ;Limpa a variável count
            clr     vfinal              ;Limpa a variável vfinal
            mov     XL,ponteiro1        ;Posiciona o ponteiro
            rjmp    Convertendo         ;Pula para o bloco Convertendo

Pula:
            inc     ponteiro2           ;Posiciona o ponteiro
            mov     XL,ponteiro2        ;Armazena o valor do ponteiro na variável ponteiro2
            st      X,vfinal            ;Armazena o dado na memória
            clr     count               ;Limpa a variável count
            clr     vfinal              ;Limpa a variável vfinal
            cpi     ponteiro2,0x82      ;Verifica se todos os dados recebidos foram decodificados e
                                        ;armazenados
            breq    Posicionando        ;Pula para o bloco Posicionando se verdadeiro
            mov     XL,ponteiro1        ;Posiciona o ponteiro
            rjmp    Convertendo         ;Pula para o bloco Convertendo

Retornando2:
            reti                        ;Retorna da interrupção

;** Loop de Atraso
Atraso_P1:
            cpi     chave,0b11111110    ;Verifica se já transmitiu para a IHM o dado recebido pelo robô
            breq    Reinicio            ;Pula para o bloco Reinicio em caso positivo, para reiniciar todo o
                                        ;processo

            inc     count               ;Incrementa o valor da variável count
            ldi     XL,0x5F             ;Posiciona o ponteiro
            cpi     count,2             ;Compara para verificar se já enviou o mesmo dado 2 vezes
            brlo    Retornando2         ;Retorna da interrupção em caso negativo

;** Bloco de atraso, para o início do recebimento dos dados do robô
Atraso_P2:
            ldi     ponteiro1,255
            ldi     ponteiro2,1

Loop:
            dec     ponteiro1
            brne    loop
            dec     ponteiro2
            brne    loop

            ldi     chave,0b11111110
            out     PORTB,chave         ;Seleciona a recepção com o Robô (PB0 em nível baixo)
            rjmp    R_Robo              ;Pula para o bloco R_Robo, para receber os dados do robô
```

```
Reinicio:
        rjmp    R_IHM                   ;Pula para o bloco R_IHM, para reiniciar todo o processo

Posicionando:
        cli                             ;Desabilita a interrupção
        clr     count                   ;Limpa a variável count
        ldi     XL,0x62                 ;Posiciona o ponteiro
        ld      temp,X                  ;Carrega o dado na variável temp
        cpi     chave,0b11111101        ;Verifica se a informação deve ser codificada ou não
        brne    Transmitir              ;Se não precisar codificar, ou seja, informação para a IHM, pula
                                        ;para o bloco Transmitir
;** Bloco de codificação dos dados
Codificando_P1:
        clr     temp2                   ;Limpa a variável temp2
        inc     count                   ;Incrementa a variável count
        cpi     count,4                 ;Compara para saber se todos os dados foram codificados
        breq    Transmitir              ;Caso positivo pula para o bloco Transmitir
        andi    temp,0x0F               ;Codifica apenas os quatro primeiros bits ("AND" com 00001111b)

Codificando_P2:
        clr     final                   ;Limpa a variável final
        mov     temp1,temp              ;Copia o valor da variável temp para a variável temp1
        com     temp                    ;Faz o complemento da variável temp

        lsl     temp                    ;Desloca os bits da variável temp para a esquerda
        mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
        mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
        andi    ponteiro2,0x01          ;Codifica o primeiro bit ("AND" com 00000001b)
        andi    ponteiro1,0x02          ;Codifica o primeiro bit ("AND" com 00000010b)
        add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
        add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

        lsl     temp                    ;Desloca os bits da variável temp para a esquerda
        lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
        mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
        mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
        andi    ponteiro2,0x04          ;Codifica o segundo bit ("AND" com 00000100b)
        andi    ponteiro1,0x08          ;Codifica o segundo bit ("AND" com 00001000b)
        add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
        add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

        lsl     temp                    ;Desloca os bits da variável temp para a esquerda
        lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
        mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
        mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
        andi    ponteiro2,0x10          ;Codifica o terceiro bit ("AND" com 00010000b)
        andi    ponteiro1,0x20          ;Codifica o terceiro bit ("AND" com 00100000b)
        add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
        add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

        lsl     temp                    ;Desloca os bits da variável temp para a esquerda
        lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
        mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
        mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
        andi    ponteiro2,0x40          ;Codifica o quarto bit ("AND" com 01000000b)
        andi    ponteiro1,0x80          ;Codifica o quarto bit ("AND" com 10000000b)
        add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
        add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

;** Bloco de verificação se o byte inteiro foi codificado
        st      X,final                 ;Armazena o dado na memória
        inc     XL                      ;Incrementa o ponteiro
        ld      temp,X                  ;Lê os dados e armazena na variável temp
        cpi     temp2,0x01              ;Compara se a variável temp2 é igual a 1
        breq    Codificando_P1          ;Pula para o bloco Codificando_P1 se o byte inteiro foi codificado
        andi    temp,0xF0               ;Codifica apenas os quatro últimos bits ("AND" com 11110000b)
        swap    temp                    ;Troca de posição dos quatro últimos bits pelos quatro primeiros
        ldi     temp2,0x01              ;Carrega 1 na variável temp2
        rjmp    Codificando_P2          ;Pula para o bloco que faz a codificação dos dados

;** Bloco de configuração para iniciar a transmissão dos dados
Transmitir:
        clr     count
        ldi     temp,0b00101000
        out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
```

```
        cpi     chave,0b11111110        ;Verifica se vai transmitir para a IHM ou para o Robô
        breq    Transmitir_IHM          ;Se for para a IHM, pula para o bloco Transmitir_IHM
        ldi     XL,0x5D                 ;Posiciona o ponteiro
        sei                             ;Habilita a interrupção
        rjmp    Volta                   ;Pula para o bloco Volta e fica aguardando a próxima interrupção

Transmitir_IHM:
        ldi     XL,0x7F                 ;Posiciona o ponteiro
        ldi     temp,0xCC               ;Armazena o valor 11001100b na variável temp
        st      X,temp                  ;Armazena o mesmo valor na memória
        dec     XL                      ;Posiciona o ponteiro
        sei                             ;Habilita a interrupção
        rjmp    Volta                   ;Pula para o bloco Volta e fica aguardando a próxima interrupção

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp

        ldi     temp,0xFF
        out     DDRB,temp               ;Port B configurada como saída
        ldi     temp,0xFF
        out     DDRA,temp               ;Port A configurada como saída
        ser     temp
        out     PORTA,temp              ;Garante 0 inicialmente na saída (LED's)

;** Inicialização do Vetor de Dados
        ldi     XH,0x00

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp               ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)

;** Armazenamento dos caracteres de reconhecimento do pacote
        ldi     XL,0x60                 ;Posiciona o ponteiro
        ldi     temp,0xCC               ;Armazena o valor 11001100b na variável temp
        st      X,temp                  ;Armazena o mesmo valor na memória
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp                  ;Armazena o mesmo valor na memória

R_IHM:
        ldi     chave,0b11111101
        out     PORTB,chave             ;Seleciona a recepção com a IHM (PB1 em nível baixo)

R_Robo:
        cli                             ;Desabilita a interrupção
        ldi     XL,0x61                 ;Posiciona o ponteiro
        ldi     temp,0b10010000
        out     UCR,temp                ;Habilita recepção e interrupção por recepção completa de dados
        clr     count                   ;Limpa a variável count
        clr     temp                    ;Limpa a variável temp
        sei                             ;Habilita a interrupção

Volta:
        rjmp    Volta                   ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.10 Programa Teste da Comunicação Bidirecional (Robô)

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa para o teste de comunicação bidirecional
Módulo: Robô
Linguagem: Assembler
Arquivo: CBiRobo.asm
Objetivo: Testar o funcionamento da comunicação bidirecional e do protocolo desenvolvido.
Descrição: Parte 1:
Uma seqüência de 8 bytes codificados deve ser recebida através da UART (módulo de recepção), sendo:
Bytes 1 e 2: CCh (identificação do pacote de dados)
Bytes 3 e 4: Direção dos motores
Bytes 5 e 6: Velocidade do motor 1
Bytes 7 e 8: Velocidade do motor 2
Os dados são decodificados e os comandos executados pelo robô.

Parte 2:
Uma seqüência de 5 bytes deve ser transmitida ao módulo de recepção da IHM através da UART (módulo de transmissão), sendo:
Bytes 1 e 2: CCh (identificação do pacote de dados)
Bytes 3: 1 em decimal
Bytes 4: 2 em decimal
Bytes 5: 3 em decimal
Por fim, os dados são codificados e transmitidos à IHM utilizando-se o módulo de transmissão RF.

```
.include "8515def.inc"

;** Declaração das Variáveis
.def    temp    = r16
.def    temp1   = r17
.def    temp2   = r18
.def    final   = r19
.def    vfinal  = r20
.def    count   = r21
.def    chave   = r22
.def    ponteiro1 = r23
.def    ponteiro2 = r24


        rjmp    RESET                   ;Pula para o Programa Principal

.org URXCaddr                           ;UART Receive Complete Interrupt Vector Address
        rjmp    Receber

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
        rjmp    Transmitindo

;** Bloco de transmissão dos dados codificados
Transmitindo:
        inc     XL                      ;Posiciona o ponteiro
        ld      temp,X                  ;Carrega o dado armazenado na variável temp
        out     UDR,temp                ;Envia os dados armazenados na variável temp para o registrador de dados da UART
        cpi     XL,0x78                 ;Compara para saber se o último caractere foi enviado
        breq    Atraso                  ;Pula para o bloco Atraso se todos os dados foram enviados
        reti                            ;Retorna da interrupção

Atraso:
        rjmp    Atraso_P1

;** Bloco que faz a leitura dos dados recebidos
Receber:
        in      temp,UDR                ;Lê os dados recebidos
        inc     count                   ;Incrementa a variável count
```

```
        cpi     count,3                 ;Compara a variável count com 3
        brlo    Confirmar               ;Pula para o bloco Confirmar se count for menor que 3
        rjmp    Armazena                ;Pula para o bloco Armazena se count for maior que 3

;** Reconhecimento de pacote de dados válidos
Confirmar:
        cpi     temp,0xCC               ;Compara a variável temp com 11001100b
        breq    Retornando1             ;Pula para o bloco Retornando1 se temp for igual a CCh
        clr     count                   ;Limpa a variável count se não for reconhecido o pacote de dados
        reti                            ;Retorna da interrupção

Retornando1:
        reti                            ;Retorna da interrupção

;** Armazenamento dos dados recebidos após identificação do pacote
Armazena:
        inc     XL                      ;Incrementa o ponteiro
        st      X,temp                  ;Armazena o dado recebido
        cpi     count,8                 ;Compara count com 8 para armazenar até 6 caracteres recebidos
        breq    Converte                ;Pula para decodificar os dados se count for igual à 8
        reti                            ;Retorna da interrupção

;** Bloco que faz a decodificação dos dados recebidos
Converte:
        cli                             ;Desabilita a interrupção
        clr     count                   ;Limpa a variável count
        ldi     XL,0x5F                 ;Posiciona o ponteiro

Convertendo:
        inc     count                   ;Incrementa o contador
        inc     XL                      ;Incrementa o ponteiro
        ld      temp,X                  ;Carrega o dado codificado (recebido) na variável temp
        clr     final                   ;Limpa a variável final
        mov     temp1,temp              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x01               ;Verifica o valor do primeiro bit ("AND" com 00000001b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x02               ;Verifica o valor do segundo bit ("AND" com 00000010b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x04               ;Verifica o valor do terceiro bit ("AND" com 00000100b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final
        lsr     temp1                   ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1              ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x08               ;Verifica o valor do quarto bit ("AND" com 00001000b)
        add     final,temp              ;Adiciona o valor da variável temp à variável final

        cpi     count,1                 ;Compara count com 1 para saber se apenas a primeira parte do byte
                                        ;foi decodificado
        breq    Parte1                  ;Pula para o bloco Parte 1 se verdadeiro
        rjmp    Parte2                  ;Pula para o bloco Parte 2 se já tiver sido decodificado o byte inteiro

Parte1:
        mov     vfinal,final            ;Armazena a primeira parte do byte decodificado na variável vfinal
        rjmp    Convertendo             ;Pula para o bloco Convertendo para continuar com a decodificação
                                        ;dos dados

Parte2:
        swap    final                   ;Inverte a posição dos últimos bits pelos primeiros
        add     vfinal,final            ;Adiciona a segunda parte do byte decodificado à primeira parte
        mov     ponteiro1,XL            ;Armazena a posição do ponteiro na variável ponteiro1
        cpi     ponteiro1,0x62          ;Verifica se o primeiro byte recebido já foi decodificado e
                                        ;armazenado
        brsh    Pula                    ;Pula para o bloco Pula se verdadeiro

        ldi     XL,0x80                 ;Posiciona o ponteiro
        mov     ponteiro2,XL            ;Armazena o valor do ponteiro na variável ponteiro2
        st      X,vfinal                ;Armazena o dado decodificado na memória
        clr     count                   ;Limpa a variável count
        clr     vfinal                  ;Limpa a variável vfinal
        mov     XL,ponteiro1            ;Posiciona o ponteiro
        rjmp    Convertendo             ;Pula para o bloco Convertendo

Pula:
```

```
            inc     ponteiro2               ;Posiciona o ponteiro
            mov     XL,ponteiro2            ;Armazena o valor do ponteiro na variável ponteiro2
            st      X,vfinal                ;Armazena o dado na memória
            clr     count                   ;Limpa a variável count
            clr     vfinal                  ;Limpa a variável vfinal
            cpi     ponteiro2,0x82          ;Verifica se todos os dados recebidos foram decodificados e armazenados
            breq    Aciona                  ;Pula para o bloco Aciona se verdadeiro
            mov     XL,ponteiro1            ;Posiciona o ponteiro
            rjmp    Convertendo             ;Pula para o bloco Convertendo

;** Bloco de acionamento dos motores
Aciona:
            ldi     XL,0x80                 ;Posiciona o ponteiro
            ld      temp,X                  ;Copia o dado armazenado na variável temp
            out     PORTB,temp              ;Manda para o PortB o comando de direção dos motores
            inc     XL                      ;Incrementa o ponteiro
            ld      temp,X                  ;Copia o dado armazenado na variável temp
            out     OCR1AL,temp             ;Configura a velocidade do PWM A de acordo com o valor recebido
            inc     XL                      ;Incrementa o ponteiro
            ld      temp,X                  ;Copia o dado armazenado na variável temp
            out     OCR1BL,temp             ;Configura a velocidade do PWM B de acordo com o valor recebido
            rjmp    Leitura                 ;Pula para o bloco Leitura

Retornando2:
            reti                            ;Retorna da interrupção

;** Loop de Atraso
Atraso_P1:
            inc     count                   ;Incrementa o valor da variável count
            ldi     XL,0x6F                 ;Posiciona o ponteiro
            cpi     count,10                ;Compara para verificar se já enviou o mesmo dado 10 vezes
            brlo    Retornando2             ;Retorna da interrupção caso negativo

Atraso_P2:
            ldi     ponteiro1,255
            ldi     ponteiro2,1
Loop:
            dec     ponteiro1
            brne    loop
            dec     ponteiro2
            brne    loop

            rjmp    DeNovo                  ;Pula para o bloco DeNovo para reiniciar todo o processo

;** Bloco que armazena os caracteres 1,2 e 3 na memória para serem transmitidos à IHM
Leitura:
            cli                             ;Desabilita a Interrupção

            ldi     XL,0x70                 ;Posiciona o ponteiro
            ldi     temp,0xCC               ;Armazena o valor 11001100b na variável temp
            st      X,temp                  ;Armazena o mesmo valor na memória
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória

            ldi     temp,0x01               ;Armazena o valor 1 na variável temp
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória
            ldi     temp,0x00               ;Armazena o valor 0 na variável temp
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória
            ldi     temp,0x02               ;Armazena o valor 2 na variável temp
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória
            ldi     temp,0x00               ;Armazena o valor 0 na variável temp
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória
            ldi     temp,0x03               ;Armazena o valor 3 na variável temp
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória
            ldi     temp,0x00               ;Armazena o valor 0 na variável temp
            inc     XL                      ;Incrementa o ponteiro
            st      X,temp                  ;Armazena o mesmo valor na memória

Posicionando:
            clr     count                   ;Limpa a variável count
```

```
            ldi     XL,0x72                 ;Posiciona o ponteiro
            ld      temp,X                  ;Carrega o dado na variável temp

;** Bloco de codificação dos dados
Codificando_P1:
            clr     temp2                   ;Limpa a variável temp2
            inc     count                   ;Incrementa a variável count
            cpi     count,4                 ;Compara para saber se todos os dados foram codificados
            breq    Transmitir              ;Caso positivo pula para o bloco Transmitir
            andi    temp,0x0F               ;Codifica apenas os quatro primeiros bits ("AND" com 00001111b)

Codificando_P2:
            clr     final                   ;Limpa a variável final
            mov     temp1,temp              ;Copia o valor da variável temp para a variável temp1
            com     temp                    ;Faz o complemento da variável temp

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x01          ;Codifica o primeiro bit ("AND" com 00000001b)
            andi    ponteiro1,0x02          ;Codifica o primeiro bit ("AND" com 00000010b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x04          ;Codifica o segundo bit ("AND" com 00000100b)
            andi    ponteiro1,0x08          ;Codifica o segundo bit ("AND" com 00001000b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x10          ;Codifica o terceiro bit ("AND" com 00010000b)
            andi    ponteiro1,0x20          ;Codifica o terceiro bit ("AND" com 00100000b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x40          ;Codifica o quarto bit ("AND" com 01000000b)
            andi    ponteiro1,0x80          ;Codifica o quarto bit ("AND" com 10000000b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

;** Bloco de verificação se o byte inteiro foi codificado
            st      X,final                 ;Armazena o dado na memória
            inc     XL                      ;Incrementa o ponteiro
            ld      temp,X                  ;Lê os dados e armazena na variável temp
            cpi     temp2,0x01              ;Compara se a variável temp2 é igual a 1
            breq    Codificando_P1          ;Pula para o bloco Codificando_P1 se o byte inteiro foi codificado
            andi    temp,0xF0               ;Codifica apenas os quatro últimos bits ("AND" com 11110000b)
            swap    temp                    ;Troca de posição os quatro últimos bits pelos quatro primeiros
            ldi     temp2,0x01              ;Carrega 1 na variável temp2
            rjmp    Codificando_P2          ;Pula para o bloco que faz a codificação dos dados

;** Bloco de configuração para iniciar a transmissão dos dados codificados
Transmitir:
            clr     count                   ;Limpa a variável count
            ldi     XL,0x6D                 ;Posiciona o ponteiro
            ldi     temp,0b00101000
            out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
            sei                             ;Habilita a interrupção
            rjmp    Volta                   ;Pula para o bloco Volta e fica aguardando a próxima Interrupção


RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
```

```
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp

        ldi     temp,0b00100000
        out     DDRD,temp           ;Pino OC1A definido como saída demais bits do PortD
                                    ;configurados como entrada
        ldi     temp,0xFF
        out     DDRB,temp           ;Port B configurada como saída
        ser     temp
        out     PORTB,temp          ;Garante 0 inicialmente na saída

;** Inicialização do Vetor de Dados
        ldi     XH,0x00

;** Configuração do PWM (A e B)
        ldi     temp,0b11110001
        out     TCCR1A,temp         ;Configuração do modo PWM 8 bits e seleção da seguinte ação para
                                    ;os pinos OC1B e OC1A (PD5):
                                    ;Clear quando o valor de comparação atingido na contagem crescente e
                                    ;seta quando o valor de comparação atingido na contagem decrescente
                                    ;O pino OC1B é a saída PWM para o motor 2
                                    ;O pino OC1A (PD5) é a saída PWM para o motor 1
        ldi     temp,0b00000001
        out     TCCR1B,temp         ;Libera timer1, incrementado pelo clock sem prescaler
                                    ;Portanto freqüência do PWM = freqüência do clock
        ldi     temp,0              ;Zera o conteúdo dos registradores altos
        out     OCR1AH,temp         ;Referências para o PWM
        out     OCR1BH,temp
        ldi     temp,255            ;Força a velocidade inicial = 0
        out     OCR1AL,temp
        out     OCR1BL,temp

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp           ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)

DeNovo:
        cli                         ;Desabilita a interrupção
        ldi     temp,0b10010000
        out     UCR,temp            ;Habilita recepção e interrupção por recepção completa de dados
        clr     count               ;Limpa a variável count
        clr     temp                ;Limpa a variável temp
        ldi     XL,0x5F             ;Posiciona o ponteiro
        sei                         ;Habilita a interrupção

Volta:
        rjmp    Volta               ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.11 Programa de Teste do Sensor de Velocidade

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa de Teste do Sensor de Velocidade
Linguagem: Assembler
Arquivo: S_Odo.asm
Objetivo: Testar o funcionamento do Sensor de Velocidade
Descrição: Quando o sensor detecta a presença de um ímã próximo, um byte é acrescido ao contador e a soma resultante é enviada pelo microcontrolador ao PC utilizando-se a UART.

```
.include "8515def.inc"
```

```
;** Declaração das Variáveis
.def    temp    = r16
.def    desloc  = r17
.def    desloc1 = r18

        rjmp    RESET                   ;Pula para o Programa Principal

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address

Incremento:
        inc     temp                    ;Incrementa a variável temp
        out     UDR,temp                ;Envia os dados armazenados na variável temp para o registrador de dados da UART

;** Loop de Atraso
Atraso:
        ldi     desloc,255
        ldi     desloc1,255
Loop:
        dec     desloc
        brne    loop
        dec     desloc1
        brne    loop

Espera:
        sbis    PINB,0                  ;Ímã ainda sendo detectado?
        rjmp    Espera                  ;Continua verificando até que o ímã deixe de ser detectado
        rjmp    Volta

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp

        ldi     temp,0x00
        out     DDRB,temp               ;Port B configurada como entrada

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp               ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)

        ldi     temp,0b00001000
        out     UCR,temp                ;Habilita transmissão
        clr     temp

Volta:
        sbis    PINB,0                  ;Ímã detectado?
        rjmp    Incremento              ;Sim, então desvia para incremento
        rjmp    Volta                   ;Fica nesse loop até detectar um ímã

;** FIM DO PROGRAMA
```

10.1.12 Programa de Controle do Robô

PROGRAMA PARA COMPUTADOR PESSOAL

Título: Programa completo para o controle do robô (última versão)
Linguagem: Delphi (Pascal)
Arquivo: Int_v08.pas
Objetivo: Controlar o funcionamento do robô, através de uma comunicação serial, apresentando os dados medidos pelo robô e o caminho percorrido pelo mesmo.
Descrição: O caminho a ser percorrido pelo robô pode ser plotado e os respectivos dados para o correto controle do robô são calculados.
Uma seqüência de 7 bytes é enviada através da comunicação serial do PC com base nos resultados dos cálculos executados:

- Bytes 1 e 2: CCh (Identificação do Protocolo)
- Byte 3: Direção dos motores
- Bytes 4 e 5: Velocidade dos motores 1 e 2 respectivamente
- Bytes 6 e 7: Quantidade de detecções dos sensores de velocidade

Após o envio do último byte o software fica aguardando os dados a serem enviados pelo robô:

- Bytes 1 e 2: CCh (Identificação do Protocolo)
- Bytes 3, 5, 7 e 9: Números 1, 2, 3 e 4 respectivamente
- Bytes 4 e 6: Quantidade de detecções dos sensores de velocidade
- Bytes 8 e 10: Dado medido pelo microcontrolador com base na medição do sensor de ultrasom

Com base nos dados recebidos, novos cálculos são executados e os resultados obtidos são demonstrados de forma gráfica e numérica.

```
unit Integ_v08;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Menus, ExtCtrls, FileCtrl, Buttons, jpeg, Registry,
  Grids, TeeProcs, TeEngine, Chart, DBGrids, Math, Mask, COMobj, ComCtrls;

type TMultlinha = class(TThread)
    private
    protected
    procedure Execute; override;
    procedure MostraString;  //Mostra dados na tela.
end;

type
  TForm1 = class(TForm)
    SpeedAbrirPorta: TSpeedButton;
    SpeedFecharPorta: TSpeedButton;
    Label2: TLabel;
    Bevel1: TBevel;
    ComboBoxPorta: TComboBox;
    Label1: TLabel;
    Label3: TLabel;
    Memo1: TMemo;
    Button2: TButton;
    Image1: TImage;
    Shape1: TShape;
    Memo10: TMemo;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Memo11: TMemo;
    Bevel6: TBevel;
    SpeedLivre: TSpeedButton;
    SpeedElastica: TSpeedButton;
    SpeedButton2: TSpeedButton;
    Label10: TLabel;
```

```
Shape2: TShape;
Label6: TLabel;
StringGrid1: TStringGrid;
SpeedButton1: TSpeedButton;
SpeedButton3: TSpeedButton;
Edit1: TEdit;
ScrollBar1: TScrollBar;
Edit2: TEdit;
Bevel7: TBevel;
Label11: TLabel;
SpeedButton4: TSpeedButton;
SpeedButton5: TSpeedButton;
Edit3: TEdit;
ScrollBar2: TScrollBar;
Edit4: TEdit;
Label12: TLabel;
Label13: TLabel;
SpeedButton9: TSpeedButton;
Edit6: TEdit;
Label14: TLabel;
Label15: TLabel;
Edit7: TEdit;
Edit8: TEdit;
Edit9: TEdit;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Bevel8: TBevel;
StringGrid2: TStringGrid;
StringGrid3: TStringGrid;
StringGrid4: TStringGrid;
SpeedButton10: TSpeedButton;
Label19: TLabel;
Label20: TLabel;
Label21: TLabel;
Label22: TLabel;
SpeedButton11: TSpeedButton;
SpeedButton12: TSpeedButton;
SpeedButton13: TSpeedButton;
SpeedButton14: TSpeedButton;
SpeedButton15: TSpeedButton;
SpeedButton16: TSpeedButton;
SpeedButton17: TSpeedButton;
SpeedButton18: TSpeedButton;
SpeedButton19: TSpeedButton;
Edit10: TEdit;
Edit11: TEdit;
Edit12: TEdit;
Label23: TLabel;
Label24: TLabel;
Edit5: TEdit;
Edit13: TEdit;
Label4: TLabel;
Label5: TLabel;
Edit14: TEdit;
Label26: TLabel;
Edit15: TEdit;
Label27: TLabel;
Label28: TLabel;
Edit16: TEdit;
Label29: TLabel;
Edit17: TEdit;
Label30: TLabel;
Label31: TLabel;
Label32: TLabel;
Label33: TLabel;
Label34: TLabel;
Label35: TLabel;
SpeedButton8: TSpeedButton;
SpeedButton20: TSpeedButton;
Label36: TLabel;
Label37: TLabel;
Bevel2: TBevel;
Label38: TLabel;
SpeedButton21: TSpeedButton;
SpeedButton22: TSpeedButton;
```

```
SpeedButton23: TSpeedButton;
SpeedButton24: TSpeedButton;
SpeedButton25: TSpeedButton;
Bevel3: TBevel;
Label39: TLabel;
Label25: TLabel;
Edit18: TEdit;
Label40: TLabel;
Label41: TLabel;
Label42: TLabel;
Label43: TLabel;
Label44: TLabel;
SpeedButton26: TSpeedButton;
SpeedButton27: TSpeedButton;
SpeedButton28: TSpeedButton;
Bevel4: TBevel;
Bevel5: TBevel;
Bevel9: TBevel;
Image2: TImage;
Edit19: TEdit;
Edit20: TEdit;
Label45: TLabel;
Label46: TLabel;
Label47: TLabel;
Label48: TLabel;
Timer1: TTimer;
StatusBar1: TStatusBar;
SpeedButton6: TSpeedButton;
SpeedButton29: TSpeedButton;
SpeedButton30: TSpeedButton;
Edit21: TEdit;
Edit22: TEdit;
Edit23: TEdit;
Edit24: TEdit;
Edit25: TEdit;
Edit26: TEdit;
Label49: TLabel;
Label50: TLabel;
Label51: TLabel;
Label52: TLabel;
Label53: TLabel;
Bevel10: TBevel;
Label54: TLabel;
SpeedButton31: TSpeedButton;
Edit27: TEdit;
SpeedButton32: TSpeedButton;
Bevel11: TBevel;
Bevel12: TBevel;
Label55: TLabel;
SpeedButton33: TSpeedButton;
Bevel13: TBevel;
Label57: TLabel;
SpeedButton34: TSpeedButton;
SpeedButton35: TSpeedButton;
SpeedButton36: TSpeedButton;
SpeedButton37: TSpeedButton;
SpeedButton38: TSpeedButton;
SpeedButton39: TSpeedButton;
SpeedButton40: TSpeedButton;
Edit28: TEdit;
TrackBar1: TTrackBar;
Label59: TLabel;
TrackBar2: TTrackBar;
Label60: TLabel;
SpeedButton41: TSpeedButton;
SpeedButton42: TSpeedButton;
Edit29: TEdit;
Edit30: TEdit;
Label56: TLabel;
Label58: TLabel;
TrackBar3: TTrackBar;
TrackBar4: TTrackBar;
procedure FormCreate(Sender: TObject);
procedure SpeedAbrirPortaClick(Sender: TObject);
procedure SpeedFecharPortaClick(Sender: TObject);
procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
```

```
    procedure MostraPortasCom();
    procedure ScrollBar2Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
    procedure ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
    procedure Button2Click(Sender: TObject);
    procedure Image1MouseMove(Sender: TObject; Shift: TShiftState; X,Y: Integer);
    procedure Image1Click(Sender: TObject);
    procedure SpeedElasticaClick(Sender: TObject);
    procedure SpeedLivreClick(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton9Click(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
    procedure SpeedButton10Click(Sender: TObject);
    procedure SpeedButton11Click(Sender: TObject);
    procedure SpeedButton12Click(Sender: TObject);
    procedure SpeedButton13Click(Sender: TObject);
    procedure SpeedButton14Click(Sender: TObject);
    procedure SpeedButton15Click(Sender: TObject);
    procedure SpeedButton16Click(Sender: TObject);
    procedure SpeedButton17Click(Sender: TObject);
    procedure SpeedButton18Click(Sender: TObject);
    procedure SpeedButton19Click(Sender: TObject);
    procedure Edit6Change(Sender: TObject);
    procedure SpeedButton8Click(Sender: TObject);
    procedure SpeedButton20Click(Sender: TObject);
    procedure SpeedButton21Click(Sender: TObject);
    procedure SpeedButton22Click(Sender: TObject);
    procedure SpeedButton23Click(Sender: TObject);
    procedure SpeedButton24Click(Sender: TObject);
    procedure SpeedButton25Click(Sender: TObject);
    procedure SpeedButton26Click(Sender: TObject);
    procedure SpeedButton27Click(Sender: TObject);
    procedure SpeedButton28Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure SpeedButton6Click(Sender: TObject);
    procedure SpeedButton29Click(Sender: TObject);
    procedure SpeedButton30Click(Sender: TObject);
    procedure SpeedButton31Click(Sender: TObject);
    procedure SpeedButton32Click(Sender: TObject);
    procedure SpeedButton33Click(Sender: TObject);
    procedure SpeedButton34Click(Sender: TObject);
    procedure SpeedButton35Click(Sender: TObject);
    procedure SpeedButton36Click(Sender: TObject);
    procedure SpeedButton37Click(Sender: TObject);
    procedure SpeedButton38Click(Sender: TObject);
    procedure SpeedButton39Click(Sender: TObject);
    procedure SpeedButton40Click(Sender: TObject);
    procedure TrackBar1Change(Sender: TObject);
    procedure TrackBar2Change(Sender: TObject);
    procedure SpeedButton41Click(Sender: TObject);
    procedure SpeedButton42Click(Sender: TObject);
    procedure TrackBar3Change(Sender: TObject);
    procedure TrackBar4Change(Sender: TObject);

  private
    { Private declarations }
  public
    FlagLinhaLivre, FlagLinhaElastica: String; //Variável para detectar se um botão foi pressionado.
    Origem, MovePt: TPoint; //Variável para armazenar pontos.
  end;

const
  LEN_BUFFER = 100; //Tamanho dos Buffers de dados.
  Relogio = 4000000; //Relógio do sistema (4MHz).
  Pre_scaler = 1; //Ou seja, (1:1) acompanha o clock do sistema.

var
  Form1: TForm1;
  hCom: THANDLE; //Handle da Porta Serial (API).
  dcb: TDCB; //Estrutura DCB (API).
  CommTimeouts: TCOMMTIMEOUTS; //Estrutura TCOMMTIMEOUTS(API).
  BytesLidos,Contador,BytesEscritos: DWORD;
```

```
  Hora, Min, Sec, MSec : Word;
  GLB_Conectado: boolean; //Indica se a porta já está aberta.
  GLBEnviaDados: bool; //Para habilitar/desabilitar o envio de dados pela serial.
  VRecebido,Condicao,TempodeEnvio,Habilita: String;
  Origem_dado,Confirmar_Dados_Novamente,Solicitar_Dados_Novamente: string;
  BufferRecebe : array [0..LEN_BUFFER] of Integer; //Define o buffer.
  D_Sent: array [0..90] of Byte;
  VRecebido_Convertido: array [0..100000] of String;
  Enviado_Robo: array [0..10,0..2] of Integer;
  Porcentagem_Ajuste: array[1..10] of real;
  Nd, Ne, N1, Nd_Real, Ne_Real, Nd_Aproximado, Ne_Aproximado: Integer;
  Acesso,Leitura,Ponteiro,c,l,X_Anterior_Real2,Y_Anterior_Real2: Integer;
  X_Anterior_Real,Y_Anterior_Real,X_Anterior_Aproximado,Y_Anterior_Aproximado: Integer;
  Nres,Lr,Desenho,Sonar,Acumulador,X_Obstaculo,Y_Obstaculo: Integer;
  Teta_Possivel,Nd_Possivel,Ne_Possivel: array [0..100] of Real;
  rd, re, Teta_Final, Menor_Diferenca,Teta_Acumulado,Teta_Acumulado2: Real;
  Teta,Teta_Real,Teta_Anterior,Teta_Anterior_Real,Alfa,Alfa_Anterior,X: Real;
  X_Anterior,X_Anterior2,Y,Y_Anterior,Y_Anterior2,Hipotenusa,Distancia: Real;
  Teta_Aproximado,Teta_Anterior_Aproximado,Teta_Anterior_Real2,Distancia_Obstaculo: Real;
  X_Real,Y_Real,X_Aproximado,Y_Aproximado,Tvoo,Timer,Vel_Som,Temp_amb: Real;
  Ajuste_Nd_Teta,Ajuste_Ne_Teta,Teta_Ajuste_Real,Ajuste_Linha_Reta: Real;
  Tempo,Tempo_Inicio,Tempo_Fim: TDateTime;

implementation

{$R *.DFM}

//Abre a porta serial especificada.
function AbrirPorta(NomePorta:String):boolean;
begin
  hCom := CreateFile(
          PChar(NomePorta),
          GENERIC_READ or GENERIC_WRITE,
          0, //Dispositivos COM abertos com acesso exclusivo.
          nil, //Sem atributos de segurança.
          OPEN_EXISTING, //deve usar OPEN_EXISTING.
          0, //Entrada e saída sem overlap.
          0 //Deve ser NULL para COM.
        );
  if(hCom = INVALID_HANDLE_VALUE) then result := false //Se houve algum erro ao abrir a porta.
  else result:= true;
end;

//Define TIMEOUTs.
Function ConfiguraTimeOuts:boolean;
begin
  if not GetCommTimeouts(hCom, CommTimeouts) then result:= false;
  CommTimeouts.ReadIntervalTimeout := 2;
  CommTimeouts.ReadTotalTimeoutMultiplier := 0;
  CommTimeouts.ReadTotalTimeoutConstant := 2;
  CommTimeouts.WriteTotalTimeoutMultiplier := 5;
  CommTimeouts.WriteTotalTimeoutConstant := 5;
  if not SetCommTimeouts(hCom, CommTimeouts) then result:= false
  else result:= true;
end;

//Configura porta serial.
Function ConfiguraControle:boolean;
begin
   if not GetCommState(hCom, dcb) then result:= false;
   dcb.BaudRate := CBR_2400; //define velocidade em bps.
   dcb.ByteSize := 8; //define bits de dados.
   dcb.Parity := NOPARITY; //define paridade
   dcb.StopBits := ONESTOPBIT; //define stop bit.
   if not SetCommState(hCom, dcb) then result:= false
   else result:= true;
end;

//Função criada para converter um número hexadecimal para decimal.
Function HexToInt(const HexStr: string): longint;
var
  iNdx: integer;
  cTmp: char;
begin
  result := 0;
  for iNdx := 1 to Length(HexStr) do
```

```
    begin
      cTmp := HexStr[iNdx];
      case cTmp of
        '0'..'9': Result := 16 * Result + (Ord(cTmp) - $30);
        'A'..'F': Result := 16 * Result + (Ord(cTmp) - $37);
        'a'..'f': Result := 16 * Result + (Ord(cTmp) - $57);
      else
        raise EConvertError.Create('Caractere ilegal na string.');
      end;
    end;
end;

//Função para calcular o tempo de execução do programa e demora no tempo de resposta do robô.
Function fExecutando(N:TDateTime):String;
var
  Tempo_final: TDateTime;
begin
  Tempo_final := Time;
  if TempodeEnvio = 'True' then
    begin
      TempodeEnvio := 'False';
      DecodeTime(Tempo_final-N, Hora, Min, Sec, MSec);
      Result := Floattostr((Msec + (Sec * 1000) + (Min * 60000) + (Hora * 3600000))/1000);
    end
  Else
    Result := TimeToStr(Tempo_final-N);
end;

//Procedimento de inicialização do programa.
procedure TForm1.FormCreate(Sender: TObject);
var
  i,j: integer;
  temp1,temp2,temp3,temp4,temp5,temp6: real;
begin
  Tempo := Time; //Armazena a hora atual.
  //Inicializa variáveis Globais.
  GLB_Conectado := false; //Indica se a porta já está aberta.
  GLBEnviaDados := false; //Para habilitar/desabilitar o envio de dados pela porta serial.
  MostraPortasCom; //Carrega os nomes das portas COM num FilterComboBox.
  Form1.SpeedButton10.Enabled := False;
  Form1.SpeedButton11.Enabled := False;
  Form1.SpeedButton12.Enabled := False;
  Form1.SpeedButton13.Enabled := False;
  Form1.SpeedButton14.Enabled := False;
  Form1.SpeedButton15.Enabled := False;
  Form1.SpeedButton16.Enabled := False;
  Form1.SpeedButton17.Enabled := False;
  Form1.SpeedButton18.Enabled := False;
  Form1.SpeedButton19.Enabled := False;
  Form1.SpeedButton6.Enabled := False;
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Form1.SpeedElastica.Enabled := False;
  Form1.SpeedButton41.Click; //Chama procedimento para ajustar os valores com base nos parâmetros de ajuste do robô.
end;

//Procedimento de abertura da porta serial.
procedure TForm1.SpeedAbrirPortaClick(Sender: TObject);
var
  Sucesso: Boolean;
begin
  if AbrirPorta(ComboBoxPorta.Text) = true then //Abre a porta serial.
    begin
      GLB_Conectado := true;
      Sucesso := ConfiguraControle();
      Sucesso := ConfiguraTimeOuts();
      if Sucesso = false then
        begin
```

```
        GLB_Conectado := false;
        CloseHandle(hCom); //Fecha a porta Serial.
        ShowMessage('Um outro programa já está usando a porta, ou a mesma não existe!');
      end
     else
      begin
        Form1.Label2.Caption := '          Porta aberta:';
        SpeedAbrirPorta.Enabled := false;
        SpeedFecharPorta.Enabled := true;
        ComboBoxPorta.Enabled := false;
        TMultlinha.Create(false); //Inicia processo.
      end;
   end
  else
   begin
     GLB_Conectado := false;
     ShowMessage('Um outro programa já está usando a porta, ou a mesma não existe!');
   end;
end;

//Procedimento para fechar a porta serial.
procedure TForm1.SpeedFecharPortaClick(Sender: TObject);
begin
  Form1.Label2.Caption := 'Portas disponíveis:';
  SpeedFecharPorta.Enabled := false;
  SpeedAbrirPorta.Enabled := true;
  ComboBoxPorta.Enabled := true;
  GLB_Conectado := false;
  CloseHandle(hCom); //Fecha a porta serial.
end;

//Mostra os nomes das portas "COM" instaladas no sistema num ComboBox.
procedure TForm1.MostraPortasCom();
var
  Registro: TRegistry; //Para trabalhar com os Registros do windows.
  Lista: Tstrings;
  indice: Integer; //Para incrementar.
begin
  Registro := TRegistry.Create; //Cria e aloca espaço na memória para o objeto.
  try
    Registro.RootKey := HKEY_LOCAL_MACHINE; //Define chave raiz.
    Registro.OpenKey('hardware\devicemap\serialcomm', False); //Abre a chave.
    Lista := TstringList.Create;
    try
      //Obtém uma string contendo todos os nomes de valores associados com a chave atual.
      Registro.GetValueNames(Lista);
      //Pega nos nomes das portas.
      for indice := 0 to Lista.Count - 1 do //Count é a quantidade de portas existentes.
        ComboBoxPorta.Items.Add(Registro.ReadString( Lista.Strings[indice] ));
      //Adiciona os nomes das portas no ComboBox1.
      if ComboBoxPorta.Items.Count > 0 then
        ComboBoxPorta.ItemIndex := 0; //Para exibir o nome da porta.
    finally
      Lista.Free;
    end;
    Registro.CloseKey;
  finally
    Registro.Free;
  end;
end;

//Mostra dados de forma sincronizada.
procedure TMultlinha.MostraString;
var
  i,cont:integer;
begin
  Ponteiro := Ponteiro + 1;
  VRecebido_Convertido[Ponteiro]:= VRecebido;
  Form1.Memo1.Text := Form1.Memo1.Text + ' ' + VRecebido;
  //Seqüência lógica para mostrar automaticamente os dados recebidos
  if Ponteiro > 9 then
    begin
      If (((StrtoInt(VRecebido_Convertido[Ponteiro-3]) = 3) and (StrtoInt(VRecebido_Convertido[Ponteiro-1]) =
        4) and (StrtoInt(VRecebido_Convertido[Ponteiro-5]) = 2) and (StrtoInt(VRecebido_Convertido[Ponteiro
        7]) = 1) and (StrtoInt(VRecebido_Convertido[Ponteiro-8]) = 204)) or (StrtoInt(VRecebido_Convertido[Ponteiro-5]) -
        StrtoInt(VRecebido_Convertido[Ponteiro-7])*4 + StrtoInt(VRecebido_Convertido[Ponteiro-3]) +
```

```
StrtoInt(VRecebido_Convertido[Ponteiro-1]) + StrtoInt(VRecebido_Convertido[Ponteiro-8]) = 210) or
(StrtoInt(VRecebido_Convertido[Ponteiro-5]) - StrtoInt(VRecebido_Convertido[Ponteiro-7])*4 +
StrtoInt(VRecebido_Convertido[Ponteiro-3]) + StrtoInt(VRecebido_Convertido[Ponteiro-1]) +
StrtoInt(VRecebido_Convertido[Ponteiro-8]) = 209)) then
begin
 //Seqüência lógica para mostrar o tempo de envio/ recebimento dos dados do robô.
 TempodeEnvio := 'True';
 Form1.StatusBar1.Panels[2].Text:= 'Último dado recebido do robô em: '+fExecutando(Tempo_Inicio)+ 's';
 Form1.StatusBar1.Panels[1].Text := 'DADO RECEBIDO';
 //Seqüência para desabilitar o botão que enviou o dado.
 Case Desenho of
   1:  begin
         Form1.SpeedButton10.Enabled := False;
         Form1.SpeedButton16.Enabled := True;
       end;
   2:  begin
         Form1.SpeedButton16.Enabled := False;
         if l > 2 then Form1.SpeedButton11.Enabled := True;
         Habilita := 'False';
       end;
   3:  begin
         Form1.SpeedButton11.Enabled := False;
         Form1.SpeedButton17.Enabled := True;
       end;
   4:  begin
         Form1.SpeedButton17.Enabled := False;
         if l > 3 then Form1.SpeedButton12.Enabled := True;
         Habilita := 'False';
       end;
   5:  begin
         Form1.SpeedButton12.Enabled := False;
         Form1.SpeedButton18.Enabled := True;
       end;
   6:  begin
         Form1.SpeedButton18.Enabled := False;
         if l > 4 then Form1.SpeedButton13.Enabled := True;
         Habilita := 'False';
       end;
   7:  begin
         Form1.SpeedButton13.Enabled := False;
         Form1.SpeedButton19.Enabled := True;
       end;
   8:  begin
         Form1.SpeedButton19.Enabled := False;
         if l > 5 then Form1.SpeedButton14.Enabled := True;
         Habilita := 'False';
       end;
   9:  begin
         Form1.SpeedButton14.Enabled := False;
         Form1.SpeedButton15.Enabled := True;
       end;
  10: begin
         Form1.SpeedButton15.Enabled := False;
       end;
 end;
 //Verifica se está apenas recebendo os dados recebidos anteriormente.
 if Confirmar_Dados_Novamente = 'True' then
   begin
     X_Anterior_Real := X_Anterior_Real2;
     Y_Anterior_Real := Y_Anterior_Real2;
     Teta_Anterior_Real := Teta_Anterior_Real2;
     Form1.Image1.Picture.LoadFromFile('C:\General Files\Estudos\Em desenvolvimento\Imagem_Anterior.bmp');
     Confirmar_Dados_Novamente := 'False';
   end;
 if Solicitar_Dados_Novamente = 'True' then
   begin
     Form1.Image1.Picture.LoadFromFile('C:\General Files\Estudos\Em desenvolvimento\Imagem_Atual.bmp');
     Solicitar_Dados_Novamente := 'False';
   end;
 Form1.Image1.Picture.SaveToFile('C:\General Files\Estudos\Em desenvolvimento\Imagem_Anterior.bmp');
 Form1.Button2.Click;
 if (Desenho > 0) and (Sonar = 0) then
   begin
     Form1.StringGrid4.Cells[1,Desenho] := VRecebido_Convertido[Ponteiro-6];
     Form1.StringGrid4.Cells[2,Desenho] := VRecebido_Convertido[Ponteiro-4];
     Form1.Edit10.Text := VRecebido_Convertido[Ponteiro];
```

```
   Form1.Edit11.Text := VRecebido_Convertido[Ponteiro-2];
  end;
 with Form1.Image1 do
  begin
   if (Desenho > 0) and (Sonar = 0) and (Trunc(Desenho/2) <> Desenho/2) then
    begin
     Acumulador := Acumulador + 1;
     if StrtoInt(Form1.StringGrid1.Cells[5,Acumulador]) > 0 then
      Nd_Real := StrtoInt(Form1.StringGrid4.Cells[1,Desenho])
     else
      begin
       Nd_Real := -StrtoInt(Form1.StringGrid4.Cells[1,Desenho]);
       Form1.StringGrid4.Cells[1,Desenho] := '-' + Form1.StringGrid4.Cells[1,Desenho];
      end;
     if StrtoInt(Form1.StringGrid1.Cells[6,Acumulador]) > 0 then
      Ne_Real := StrtoInt(Form1.StringGrid4.Cells[2,Desenho])
     else
      begin
       Ne_Real := -StrtoInt(Form1.StringGrid4.Cells[2,Desenho]);
       Form1.StringGrid4.Cells[2,Desenho] := '-' + Form1.StringGrid4.Cells[2,Desenho];
      end;
    end;
   if (Desenho > 0) and (Sonar = 0) and (Trunc(Desenho/2) = Desenho/2) then
    begin
     Nd_Real := StrtoInt(Form1.StringGrid4.Cells[1,Desenho]);
     Ne_Real := Round(StrtoInt(Form1.StringGrid4.Cells[2,Desenho]));
    end;
   if (Desenho = 0) and (Sonar = 0) then
    begin
     Nd_Real := StrtoInt(VRecebido_Convertido[Ponteiro-6]);
     Ne_Real := StrtoInt(VRecebido_Convertido[Ponteiro-4]);
     Form1.Edit10.Text := VRecebido_Convertido[Ponteiro];
     Form1.Edit11.Text := VRecebido_Convertido[Ponteiro-2];
    end;
   if Origem_dado = 'Controle dos Motores' then
    begin
     if StrtoInt(Form1.StringGrid3.Cells[1,l-1]) < 0 then
      Nd_Real := -Nd_Real;
     if StrtoInt(Form1.StringGrid3.Cells[2,l-1]) < 0 then
      Ne_Real := -Ne_Real;
    end;
   if Sonar = 0 then
    begin
     Teta_Real := ((2*PI/(Lr*Nres))*(Nd_Real*rd-Ne_Real*re))*Teta_Ajuste_Real+Teta_Anterior_Real;
     if (Nd_Real > 0) and (Ne_Real > 0) then
      begin
       //O ângulo considerado é o anterior, pois só existe desvio no final, onde o ângulo adicional é considerado
       X_Real := (Nd_Real*rd+Ne_Real*re)*PI*Cos(Teta_Anterior_Real)/Nres+X_Anterior_Real;
       Y_Real := (Nd_Real*rd+Ne_Real*re)*PI*Sin(Teta_Anterior_Real)/Nres+Y_Anterior_Real;
      end
     else
      begin
       X_Real := (Nd_Real*rd+Ne_Real*re)*PI*Cos(Teta_Real)/Nres+X_Anterior_Real;
       Y_Real := (Nd_Real*rd+Ne_Real*re)*PI*Sin(Teta_Real)/Nres+Y_Anterior_Real;
      end;
     //Seqüência que calcula a distância do obstáculo detectado pelo robô.
     Form1.Edit19.Text := InttoHex(StrtoInt(Form1.Edit10.Text),2)+InttoHex(StrtoInt(Form1.Edit11.Text),2);
     Form1.Edit20.Text := InttoStr(HextoInt(InttoHex(StrtoInt(Form1.Edit10.Text),2)+InttoHex(StrtoInt(Form1.Edit11.Text),2)));
     Timer := HextoInt((InttoHex(StrtoInt(Form1.Edit10.Text),2)+InttoHex(StrtoInt(Form1.Edit11.Text),2)));
     Tvoo := Timer*Pre_scaler/Relogio; //Tempo de vôo em segundos.
     Form1.Edit12.Text := FloattoStr(Tvoo*1000);
     Distancia_Obstaculo := Vel_som*Tvoo*100/2; //Distância em centímetros.
     Form1.Edit18.Text := FloattoStr(Distancia_Obstaculo);
     //Seqüência que verifica em qual planilha deve ser mostrado o dado recebido.
     if Origem_dado = 'Controle dos Motores' then
      begin
       Form1.StringGrid3.Cells[1,l] := InttoStr(Nd_Real);
       Form1.StringGrid3.Cells[2,l] := InttoStr(Ne_Real);
       Form1.StringGrid3.Cells[3,l] := FloattoStr(Round(RadtoDeg(Teta_Real)));
       Form1.StringGrid3.Cells[4,l] := InttoStr(Round(Sqrt(Sqr(X_Real-X_Anterior_Real)+Sqr(Y_Real-Y_Anterior_Real))));
       Form1.StringGrid3.Cells[5,l] := InttoStr(Round(Distancia_Obstaculo));
       Form1.StringGrid3.Cells[6,l] := FloattoStr(Round(X_Real-X_Anterior_Real));
       Form1.StringGrid3.Cells[7,l] := FloattoStr(Round(Y_Real-Y_Anterior_Real));
       Form1.StringGrid3.Cells[8,l] := Condicao;
      end
     else
```

```
        begin
          Form1.StringGrid4.Cells[1,Desenho] := InttoStr(Nd_Real);
          Form1.StringGrid4.Cells[2,Desenho] := InttoStr(Ne_Real);
          Form1.StringGrid4.Cells[3,Desenho] := FloattoStr(Round(RadtoDeg(Teta_Real-Teta_Anterior_Real)));
          Form1.StringGrid4.Cells[4,Desenho] := FloattoStr(Round(Sqrt(Sqr(X_Real-X_Anterior_Real)+Sqr(Y_Real-
            Y_Anterior_Real))));
          Form1.StringGrid4.Cells[5,Desenho] := InttoStr(Round(Distancia_Obstaculo));
          Form1.StringGrid4.Cells[6,Desenho] := FloattoStr(Round(X_Real-X_Anterior_Real));
          Form1.StringGrid4.Cells[7,Desenho] := FloattoStr(Round(Y_Real-Y_Anterior_Real));
          Form1.StringGrid4.Cells[8,Desenho] := Condicao;
        end;
      //Plota o caminho percorrido pelo robô.
      Canvas.Pen.Width := 2;
      Canvas.Pen.Color := clRed;
      Canvas.MoveTo(X_Anterior_Real+250, -Y_Anterior_Real+500-250);
      Canvas.LineTo(Round(X_Real+250), Round(-Y_Real+500-250)); //desenha a nova linha
      X_Anterior_Real2 := X_Anterior_Real;
      Y_Anterior_Real2 := Y_Anterior_Real;
      Teta_Anterior_Real2 := Teta_Anterior_Real;
      X_Anterior_Real := Round(X_Real);
      Y_Anterior_Real := Round(Y_Real);
      Teta_Anterior_Real := Teta_Real;
      Ponteiro := 0;
      Origem_dado := '';
      //Cálculo da posição do obstáculo no espaço X e Y.
      if Form1.SpeedButton29.Enabled = False then
        begin
          Teta_Acumulado2 := RadtoDeg(Teta_Real-Teta_Anterior_Real2) + Teta_Acumulado2;
          Teta_Acumulado := Abs(Teta_Acumulado2);
          X_Obstaculo := Round(Sqrt(Sqr(Distancia_Obstaculo)/(1+Sqr(Tan(Teta_Real)))));
          Y_Obstaculo := Round(X_Obstaculo*Tan(Teta_Real));
          while Teta_Acumulado > 360 do
            Teta_Acumulado := Teta_Acumulado - 360;
          if (Teta_Acumulado > 180) and (Teta_Acumulado < 270) then
            begin
              X_Obstaculo := -X_Obstaculo;
              Y_Obstaculo := -Y_Obstaculo;
            end;
          if (Teta_Acumulado > 90) and (Teta_Acumulado < 180) then
            begin
              X_Obstaculo := -X_Obstaculo;
              Y_Obstaculo := -Y_Obstaculo;
            end;
          if (Teta_Acumulado > 0) and (Teta_Acumulado < 90) then
            begin
              X_Obstaculo := X_Obstaculo;
              Y_Obstaculo := Y_Obstaculo;
            end;
          if (Teta_Acumulado > 270) and (Teta_Acumulado < 360) then
            begin
              X_Obstaculo := X_Obstaculo;
              Y_Obstaculo := Y_Obstaculo;
            end;
          Form1.Image1.Canvas.MoveTo(Round(X_Real)+250+X_Obstaculo,-Round(Y_Real)+500-250-Y_Obstaculo);
          Form1.Image1.Canvas.Ellipse(Round(X_Real)+250+X_Obstaculo-2,-Round(Y_Real)+500-250-Y_Obstaculo-
            2,Round(X_Real)+250+X_Obstaculo+2,-Round(Y_Real)+500-250-Y_Obstaculo+2);
        end;
      Form1.Image1.Picture.SaveToFile('C:\General Files\Estudos\Em desenvolvimento\Imagem_Atual.bmp');
    end;
  end;
if Sonar = 1 then
  begin
    Sonar := 0;
    Form1.Edit10.Text := VRecebido_Convertido[Ponteiro];
    Form1.Edit11.Text := VRecebido_Convertido[Ponteiro-2];
    //Seqüência que calcula a distância do obstáculo detectado pelo robô.
    Form1.Edit19.Text := InttoHex(StrtoInt(Form1.Edit10.Text),2)+InttoHex(StrtoInt(Form1.Edit11.Text),2);
    Form1.Edit20.Text := InttoStr(HextoInt(InttoHex(StrtoInt(Form1.Edit10.Text),2)+InttoHex(StrtoInt(Form1.Edit11.Text),2)));
    Timer := HextoInt((InttoHex(StrtoInt(Form1.Edit10.Text),2)+InttoHex(StrtoInt(Form1.Edit11.Text),2)));
    Tvoo := Timer*Pre_scaler/Relogio; //Tempo de vôo em segundos.
    Form1.Edit12.Text := FloattoStr(Tvoo*1000);
    Distancia_Obstaculo := Vel_som*Tvoo*100/2; //Distância em centímetros.
    Form1.Edit18.Text := FloattoStr(Distancia_Obstaculo);
    //Cálculo da posição do obstáculo no espaço X e Y.
    if Form1.SpeedButton29.Enabled = False then
      begin
```

```
            X_Obstaculo := Round(Sqrt(Sqr(Distancia_Obstaculo)/(1+Sqr(Tan(Teta_Real)))));
            Y_Obstaculo := Round(X_Obstaculo*Tan(Teta_Real));
            while Teta_Acumulado > 360 do
              Teta_Acumulado := Teta_Acumulado - 360;
            if (Teta_Acumulado > 180) and (Teta_Acumulado < 270) then
              begin
                X_Obstaculo := -X_Obstaculo;
                Y_Obstaculo := -Y_Obstaculo;
              end;
            if (Teta_Acumulado > 90) and (Teta_Acumulado < 180) then
              begin
                X_Obstaculo := -X_Obstaculo;
                Y_Obstaculo := -Y_Obstaculo;
              end;
            if (Teta_Acumulado > 0) and (Teta_Acumulado < 90) then
              begin
                X_Obstaculo := X_Obstaculo;
                Y_Obstaculo := Y_Obstaculo;
              end;
            if (Teta_Acumulado > 270) and (Teta_Acumulado < 360) then
              begin
                X_Obstaculo := X_Obstaculo;
                Y_Obstaculo := Y_Obstaculo;
              end;
            Form1.Image1.Canvas.MoveTo(Round(X_Real)+250+X_Obstaculo,-Round(Y_Real)+500-250-Y_Obstaculo);
            Form1.Image1.Canvas.Ellipse(Round(X_Real)+250+X_Obstaculo-2,-Round(Y_Real)+500-250-Y_Obstaculo-
              2,Round(X_Real)+250+X_Obstaculo+2,-Round(Y_Real)+500-250-Y_Obstaculo+2);
          end;
        end;
      end;
    end;
end;

//Trata a porta serial.
procedure TMultlinha.Execute;
var
  cont:DWORD;
  StrMens:AnsiString;
  i, tama:integer;
begin
  while not terminated do //loop infinito.
    begin
      if GLB_Conectado = true then //Se a porta serial foi aberta corretamente.
        begin
          BytesLidos := 0;
          cont := 0;
          if Readfile(hCom, BufferRecebe[cont], LEN_BUFFER, BytesLidos, nil) then //Lê a porta Serial.
            VRecebido:=IntToStr(BufferRecebe[Cont]);
          if BytesLidos > 0 then //Se algum caracter foi lido.
            begin
              while cont < BytesLidos do //Enquanto há caracteres a serem lidos.
              begin
                Synchronize(MostraString); //Mostra dados na tela.
                contador := contador + 1;
                cont := cont + 1;
              end; //Fim do while 2
            end;
        end
      else Sleep(1); //Para não travar o processo.
    end;
end;

//Procedimento para fechar a comunicação com a porta serial.
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if(GLB_Conectado = true) then CloseHandle(hCom);
end;

//Procedimento para ajuste da velocidade no campo texto.
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
begin
  Form1.Edit2.Text:=IntToStr(Form1.ScrollBar1.Position);
end;

//Procedimento para ajuste da velocidade no campo texto.
procedure TForm1.ScrollBar2Scroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);
```

```
begin
  Form1.Edit4.Text:=IntToStr(Form1.ScrollBar2.Position);
end;

//Procedimento que faz a correção dos dados recebidos, caso apresentem algum erro.
procedure TForm1.Button2Click(Sender: TObject);
var
  i: Integer;
begin
  Form1.Memo1.Clear;
  Ponteiro := Ponteiro - 9;
  Condicao := '';
  if (VRecebido_Convertido[Ponteiro] = '0') or (VRecebido_Convertido[Ponteiro] = '204') then
    begin
      Condicao := 'CORRETO';
      Form1.Label6.Font.Color := clLime;
    end;
  if (VRecebido_Convertido[Ponteiro] <> '0') and (InttoStr(StrtoInt(VRecebido_Convertido[Ponteiro + 2]) -
    StrtoInt(VRecebido_Convertido[Ponteiro])) = '1') then
    begin
      Condicao := 'ERRO LEITURA';
      Form1.Label6.Font.Color := clRed;
    end;
  if InttoStr(StrtoInt(VRecebido_Convertido[Ponteiro + 2]) - StrtoInt(VRecebido_Convertido[Ponteiro])) = '0' then
    begin
      Condicao := 'ERRO ZERO';
      Form1.Label6.Font.Color := clRed;
    end;
  if InttoStr(StrtoInt(VRecebido_Convertido[Ponteiro + 2]) - StrtoInt(VRecebido_Convertido[Ponteiro])) = '-254' then
    begin
      Condicao := 'ERRO -254';
      Form1.Label6.Font.Color := clRed;
    end;
  if InttoStr(StrtoInt(VRecebido_Convertido[Ponteiro + 2]) - StrtoInt(VRecebido_Convertido[Ponteiro])) = '-203' then
    begin
      Condicao := 'ERRO -203';
      Form1.Label6.Font.Color := clRed;
    end;
  Form1.Label6.Caption := Condicao;
  VRecebido := VRecebido_Convertido[Ponteiro];
  for i:=1 to 10 do
    begin
      if Condicao = 'CORRETO' then
        Form1.Memo1.Text := Form1.Memo1.Text + ' ' + VRecebido_Convertido[Ponteiro];
      if Condicao = 'ERRO LEITURA' then
        begin
          Form1.Memo1.Text := Form1.Memo1.Text + ' ' + InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido));
          VRecebido_Convertido[Ponteiro] := InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido));
        end;
      if Condicao = 'ERRO ZERO' then
        begin
          Form1.Memo1.Text := Form1.Memo1.Text + ' ' + InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido) + 1);
          VRecebido_Convertido[Ponteiro] := InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido) + 1);
        end;
      if Condicao = 'ERRO -254' then
        begin
          Form1.Memo1.Text := Form1.Memo1.Text + ' ' + InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido) + 255);
          VRecebido_Convertido[Ponteiro] := InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido) + 255);
        end;
      if Condicao = 'ERRO -203' then
        begin
          Form1.Memo1.Text := Form1.Memo1.Text + ' ' + InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido) + 204);
          VRecebido_Convertido[Ponteiro] := InttoStr(Strtoint(VRecebido_Convertido[Ponteiro]) - Strtoint(VRecebido) + 204);
        end;
      Ponteiro := Ponteiro + 1;
    end;
  Ponteiro:= Ponteiro - 1;
end;

//Procedimento para desenhar o caminho na tela.
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState; X,Y: Integer);
begin
  if (FlagLinhaLivre = 'True') and (SpeedLivre.Enabled = False) then
    begin
      Canvas.MoveTo(Mouse.CursorPos.X, Mouse.CursorPos.Y-25);
      Canvas.LineTo(MovePt.X, MovePt.Y-25); //desenha a nova linha
```

```
    Form1.Memo10.Text := InttoStr(MovePt.X-286);
    Form1.Memo11.Text := InttoStr(500-MovePt.Y+279);
    MovePt := Point(Mouse.CursorPos.X, Mouse.CursorPos.Y); //memoriza as coordenadas para o próximo movimento
   end;
  if (FlagLinhaElastica = 'True') and (SpeedElastica.Enabled = False) then
   begin
    if l = 2 then
     begin
      with Form1.Image1 do
       begin
        Canvas.Pen.Width := 2;
        Canvas.Pen.Color := clBlue;
        Canvas.MoveTo(MovePt.X-286, MovePt.Y-279);
        Canvas.LineTo(0+250, 500-250); //desenha a nova linha
       end;
     end
    else
     begin
      with Form1.Image1 do
       begin
        Canvas.MoveTo(Origem.X-286, Origem.Y-279);
        Canvas.LineTo(MovePt.X-286, MovePt.Y-279); //desenha a nova linha
       end;
     end;
    Origem.X := MovePt.X;
    Origem.Y := MovePt.Y;
    Form1.Memo10.Text := InttoStr(MovePt.X-286);
    Form1.Memo11.Text := InttoStr(500-MovePt.Y+279);
   end;
end;

//Procedimento para desenhar o caminho na tela e calcular o dado a ser enviado ao robô.
procedure TForm1.Image1Click(Sender: TObject);
var
 i: integer;
 Temp: real;
begin
 if SpeedLivre.Enabled = False then
  begin
   if (FlagLinhaLivre = '') or (FlagLinhaLivre = 'False') then FlagLinhaLivre := 'True'
   else FlagLinhaLivre := 'False';
   MovePt := Point(Mouse.CursorPos.X, Mouse.CursorPos.Y); {memoriza as coordenadas para o próximo movimento}
  end
 else FlagLinhaLivre := 'False';
 if SpeedElastica.Enabled = False then
  begin
   if (FlagLinhaElastica = '') or (FlagLinhaElastica = 'False') then
    begin
     FlagLinhaElastica := 'True';
    end;
   MovePt := Point(Mouse.CursorPos.X, Mouse.CursorPos.Y); {memoriza as coordenadas para o próximo movimento}
   if Form1.SpeedButton30.Enabled = True then
    begin
     X := MovePt.X-286-250;
     Y := -MovePt.Y +500+279-250;
    end;
   if  l = 0 then
    begin
     Teta := radtodeg(arctan((Y)/(X)));
     Alfa := Teta;
     Distancia := Sqrt(Sqr(Y)+Sqr(X));
     l := l + 1;
    end;
   if l > 1 then
    begin
     if (X_Anterior - X_Anterior2 > 0) and (X - X_Anterior < 0) then
      begin
       if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior > 0) then
        begin
         Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
         Teta := 180 + Alfa - Alfa_anterior;
        end;
       if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior > 0) then
        begin
         Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
         Teta := -180 + Alfa - Alfa_anterior;
```

```
      end;
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior < 0) then
      begin
        Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
        Teta := 180 + Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior < 0) then
      begin
        Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
        Teta := -180 + Alfa - Alfa_anterior;
      end;
  end;
if (X_Anterior - X_Anterior2 < 0) and (X - X_Anterior > 0) then
  begin
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior > 0) then
      begin
        Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
        Teta := -180 + Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior > 0) then
      begin
        Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
        Teta := -180 + Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior < 0) then
      begin
        Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
        Teta := -180 + Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior < 0) then
      begin
        Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
        Teta := 180 + Alfa - Alfa_anterior;
      end;
  end;
if ((X_Anterior - X_Anterior2 < 0) and (X - X_Anterior < 0)) or ((X_Anterior - X_Anterior2 > 0) and (X - X_Anterior > 0)) then
  begin
    Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
    Teta := Alfa - Alfa_anterior;
  end;
if (X_Anterior - X_Anterior2 > 0) and (X - X_Anterior = 0) then
  begin
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior > 0) then
      begin
        Alfa := 90;
        Teta := Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior > 0) then
      begin
        Alfa := 90;
        Teta := Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior < 0) then
      begin
        Alfa := -90;
        Teta := Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior < 0) then
      begin
        Alfa := -90;
        Teta := Alfa - Alfa_anterior;
      end;
  end;
if (X_Anterior - X_Anterior2 < 0) and (X - X_Anterior = 0) then
  begin
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior > 0) then
      begin
        Alfa := -90;
        Teta := Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior > 0) then
      begin
        Alfa := -90;
        Teta := Alfa - Alfa_anterior;
      end;
    if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior < 0) then
```

```
          begin
            Alfa := 90;
            Teta := Alfa - Alfa_anterior;
          end;
        if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior < 0) then
          begin
            Alfa := 90;
            Teta := Alfa - Alfa_anterior;
          end;
      end;
    if (X_Anterior - X_Anterior2 = 0) and (X - X_Anterior < 0) then
      Begin
        if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior > 0) then
          begin
            Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
            Teta := Alfa + Alfa_anterior;
          end;
        if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior < 0) then
          begin
            Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
            Teta := Alfa + Alfa_anterior;
          end;
        if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior < 0) then
          begin
            Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
            Teta := 180 + Alfa - Alfa_anterior;
          end;
        if (Y_Anterior - Y_Anterior2 < 0) and (Y - Y_Anterior > 0) then
          begin
            Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
            Teta := -180 + Alfa - Alfa_anterior;
          end;
        if (Y_Anterior - Y_Anterior2 > 0) and (Y - Y_Anterior = 0) then
          begin
            Alfa := radtodeg(Arctan((Y-Y_Anterior)/(X-X_Anterior)));
            Teta := 90;
          end;
      end;
  end;
//Seqüência lógica para determinar o teta possível mais próximo com base no teta calculado.
Temp := 1000;
for i := 1 to 56 do
  begin
    Menor_Diferenca := Teta - Teta_Possivel[i];
    if Abs(Menor_Diferenca) < Temp then
      begin
        Temp := Menor_Diferenca;
        Teta_Final := Teta_Possivel[i];
        Nd := Round(Nd_Possivel[i]);
        Ne := Round(Ne_Possivel[i]);
      end;
  end;
c := 1;
with StringGrid1 Do
  begin
    Cells[c,l] := InttoStr(Round(X-X_Anterior));
    c := c + 1;
    Cells[c,l] := InttoStr(Round(Y-Y_Anterior));
    c := c + 1;
    Cells[c,l] := InttoStr(Round(Teta));
    c := c + 1;
    Cells[c,l] := FloattoStr(Teta_Final);
    c := c + 1;
    Cells[c,l] := FloattoStr(Nd);
    c := c + 1;
    Cells[c,l] := FloattoStr(Ne);
  end;
//Seqüência lógica para determinar o distância possível mais próxima com base no valor calculado.
Hipotenusa := Sqrt(Sqr(X-X_Anterior)+Sqr(Y-Y_Anterior));
Temp := 1000;
for i := 0 to 200 do
  begin
    Menor_Diferenca := Hipotenusa - (i/Nres)*2*Pi*rd*Ajuste_Linha_Reta;
    if Abs(Menor_Diferenca) < Temp then
      begin
        Temp := Menor_Diferenca;
```

```
            Distancia := (i/Nres)*2*Pi*rd*Ajuste_Linha_Reta;
            if i < 20 then
              begin
                Nd := Round(i*0.5);
                Ne := Round(i);
              end
            else
              begin
                Nd := Round(i-10);
                Ne := Round(i);
              end;
          end;
      end;
    c := 1;
    with StringGrid2 Do
      begin
        Cells[c,l] := InttoStr(0);
        c := c + 1;
        Cells[c,l] := InttoStr(Round(Hipotenusa));
        c := c + 1;
        Cells[c,l] := InttoStr(Round(Distancia));
        c := c + 1;
        Cells[c,l] := FloattoStr(Nd);
        c := c + 1;
        Cells[c,l] := FloattoStr(Ne);
      end;
    l := l + 1;
    Alfa_Anterior := Alfa;
    Teta_Anterior := Teta;
    X_Anterior2 := X_Anterior;
    Y_Anterior2 := Y_Anterior;
    X_Anterior := X;
    Y_Anterior := Y;
  end
 else FlagLinhaElastica := 'False';
 //Seqüência lógica para habilitar o botão de envio dos dados.
 if l = 2 then
  begin
    Form1.SpeedButton10.Enabled := True;
  end;
end;

//Procedimento para habilitar o desenho do caminho a ser percorrido pelo robô.
procedure TForm1.SpeedElasticaClick(Sender: TObject);
begin
 SpeedLivre.Enabled := True;
 SpeedElastica.Enabled := False;
 Form1.SpeedButton30.Enabled := True;
 Form1.SpeedButton31.Enabled := True;
 Form1.SpeedButton33.Enabled := True;
 Form1.SpeedButton34.Enabled := True;
 Form1.SpeedButton35.Enabled := True;
 Form1.SpeedButton36.Enabled := True;
 Form1.SpeedButton37.Enabled := True;
 Form1.SpeedButton38.Enabled := True;
 Form1.SpeedButton39.Enabled := True;
 Form1.SpeedButton40.Enabled := True;
 c := 1;
 l := 0;
 Image1.Canvas.Pen.Width := 1;
 Image1.Canvas.Pen.Color := clGray;
 Image1.Canvas.Pen.Style := psDash;
 Image1.Canvas.MoveTo(250, 250);
 Image1.Canvas.LineTo(250,0);
 Image1.Canvas.MoveTo(250, 250);
 Image1.Canvas.LineTo(500,250);
 Image1.Canvas.MoveTo(0, 0);
end;

//Procedimento que habilita o botão para desenhar um caminho livre (apenas como teste).
procedure TForm1.SpeedLivreClick(Sender: TObject);
begin
 SpeedElastica.Enabled := True;
 SpeedLivre.Enabled := False;
 FlagLinhaElastica := 'False';
end;
```

```
//Procedimento para limpar as variáveis utilizadas para o desenho gráfico.
procedure TForm1.SpeedButton2Click(Sender: TObject);
var
  i,j: Integer;
begin
  FlagLinhaElastica := 'False';
  SpeedLivre.Enabled := True;
  SpeedElastica.Enabled := True;
  c := 1;
  l := 0;
  Canvas.MoveTo(0, 0);
  Alfa_Anterior := 0;
  Teta_Anterior := 0;
  X_Anterior2 := 0;
  Y_Anterior2 := 0;
  X_Anterior := 0;
  Y_Anterior := 0;
  X_Real := 0;
  X_Anterior_Real := 0;
  X_Aproximado := 0;
  X_Anterior_Aproximado := 0;
  Y_Real := 0;
  Y_Anterior_Real := 0;
  Y_Aproximado := 0;
  Y_Anterior_Aproximado := 0;
  Acumulador := 0;
  Teta_Real := 0;
  Teta_Anterior_Real := 0;
  Teta_Aproximado := 0;
  Teta_Anterior_Aproximado := 0;
  Teta_Anterior_Real := 0;
  Teta_Anterior_Real2 := 0;
  Teta_Acumulado := 0;
  Teta_Acumulado2 := 0;
  Nd_Real := 0;
  Ne_Real := 0;
  Nd_Aproximado := 0;
  Ne_Aproximado := 0;
  Desenho := 0;
  Form1.Memo10.Text := '';
  Form1.Memo11.Text := '';
  Form1.SpeedButton10.Enabled := False;
  Form1.SpeedButton11.Enabled := False;
  Form1.SpeedButton12.Enabled := False;
  Form1.SpeedButton13.Enabled := False;
  Form1.SpeedButton14.Enabled := False;
  Form1.SpeedButton15.Enabled := False;
  Form1.SpeedButton16.Enabled := False;
  Form1.SpeedButton17.Enabled := False;
  Form1.SpeedButton18.Enabled := False;
  Form1.SpeedButton19.Enabled := False;
  //Seqüência lógica para limpar os dados que plotam o caminho aproximado.
  for i:= 1 to 10 do
    begin
      Enviado_Robo[Desenho,0] := 0;
      Enviado_Robo[Desenho,0] := 1;
    end;
  //Seqüência lógica para limpar a tela de desenho.
  Image1.Canvas.Brush.Color := clWhite;
  Image1.Canvas.Pen.Color := clWhite;
  Image1.Canvas.Pen.Width := 1;
  Image1.Canvas.Rectangle(0,0,Image1.Width,Image1.Height);
  //Seqüência lógica para limpar as planilhas onde se mostram os dados calculados.
  for i := 1 to 100 do
    begin
      for j := 1 to 8 do
        begin
          Form1.StringGrid1.Cells[j,i] := '';
          Form1.StringGrid2.Cells[j,i] := '';
          Form1.StringGrid3.Cells[j,i] := '';
          Form1.StringGrid4.Cells[j,i] := '';
        end;
    end;
end;
```

```
//Procedimento utilizado para preencher o cabeçalho das planilhas apresentadas.
procedure TForm1.FormActivate(Sender: TObject);
var
  i: Integer;
begin
  with StringGrid1 do
    begin
      for i := 1 to 100 do Cells[0,i] := InttoStr(i);
      Cells[1,0] := 'Delta X';
      Cells[2,0] := 'Delta. Y';
      Cells[3,0] := 'Ângulo C.';
      Cells[4,0] := 'Ângulo E.';
      Cells[5,0] := 'Volt. M1';
      Cells[6,0] := 'Volt. M2';
      Cells[7,0] := '';
      Cells[8,0] := 'STATUS';
    end;
  with StringGrid2 do
    begin
      for i := 1 to 100 do Cells[0,i] := InttoStr(i);
      Cells[1,0] := 'Ângulo';
      Cells[2,0] := 'Dist. C.';
      Cells[3,0] := 'Dist. E.';
      Cells[4,0] := 'Volt. M1';
      Cells[5,0] := 'Volt. M2';
      Cells[6,0] := '';
      Cells[7,0] := '';
      Cells[8,0] := 'STATUS';
    end;
  with StringGrid4 do
    begin
      for i := 1 to 100 do Cells[0,i] := InttoStr(i);
      Cells[1,0] := 'Volt. M1';
      Cells[2,0] := 'Volt. M2';
      Cells[3,0] := 'Ângulo C.';
      Cells[4,0] := 'Dist. C.';
      Cells[5,0] := 'Dist. Ob.';
      Cells[6,0] := 'Delta X';
      Cells[7,0] := 'Delta Y';
      Cells[8,0] := 'STATUS';
    end;
  with StringGrid3 do
    begin
      for i := 1 to 100 do Cells[0,i] := InttoStr(i);
      Cells[1,0] := 'Volt. M1';
      Cells[2,0] := 'Volt. M2';
      Cells[3,0] := 'Ângulo C.';
      Cells[4,0] := 'Dist. C.';
      Cells[5,0] := 'Dist. Ob.';
      Cells[6,0] := 'Delta X';
      Cells[7,0] := 'Delta Y';
      Cells[8,0] := 'STATUS';
    end;
end;

//Procedimento para incrementar o valor de Nd.
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  Nd := StrtoInt(Form1.Edit1.Text);
  Nd := Nd + 1;
  Form1.Edit1.Text:=IntToStr(Nd);
end;

//Procedimento para reduzir o valor de Nd.
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
  Nd := StrtoInt(Form1.Edit1.Text);
  Nd := Nd - 1;
  Form1.Edit1.Text:=IntToStr(Nd);
end;

//Procedimento para incrementar o valor de Ne.
procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
  Ne := StrtoInt(Form1.Edit3.Text);
  Ne := Ne + 1;
```

```
  Form1.Edit3.Text:=IntToStr(Ne);
end;

//Procedimento para reduzir o valor de Ne.
procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
  Ne := StrtoInt(Form1.Edit3.Text);
  Ne := Ne - 1;
  Form1.Edit3.Text:=IntToStr(Ne);
end;

//Procedimento utilizado para enviar comandos independentes para cada motor.
procedure TForm1.SpeedButton9Click(Sender: TObject);
var
  temp1,temp2: integer;
begin
  Origem_dado := 'Controle dos Motores';
  temp1 := StrtoInt(Form1.Edit1.Text);
  temp2 := StrtoInt(Form1.Edit3.Text);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (temp1 > 0) and (temp2 > 0) then D_Sent[3] := 198;
  if (temp1 > 0) and (temp2 < 0) then D_Sent[3] := 202;
  if (temp1 < 0) and (temp2 > 0) then D_Sent[3] := 197;
  if (temp1 < 0) and (temp2 < 0) then D_Sent[3] := 201;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[4] := Form1.ScrollBar1.Position;
  D_Sent[5] := Form1.ScrollBar2.Position;
  D_Sent[6] := Abs(temp1);
  D_Sent[7] := Abs(temp2);
  Nd := temp1;
  Ne := temp2;
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Mostra na planilha os dados enviados e calculados.
  Teta := ((2*PI)/(Lr*Nres))*(Nd*rd-Ne*re)+Teta_Anterior;
  X := (Nd*rd+Ne*re)*PI*Cos(Teta)/Nres+X_Anterior;
  Y := (Nd*rd+Ne*re)*PI*Sin(Teta)/Nres+Y_Anterior;
  l := l + 1;
  with StringGrid3 do
    begin
      Cells[1,l] := Inttostr(Nd);
      Cells[2,l] := Inttostr(Ne);
      Cells[3,l] := Inttostr(Round(RadtoDeg(Teta-Teta_Anterior)));
      Cells[4,l] := Inttostr(Round(Sqrt(Sqr(X-X_Anterior)+Sqr(Y-Y_Anterior)))); //Distância percorrida em cm.
      Cells[5,l] := 'NA';
      Cells[6,l] := InttoStr(Round(X-X_Anterior));
      Cells[7,l] := InttoStr(Round(Y-Y_Anterior));
      l := l + 1;
    end;
  X_Anterior := X;
  Y_Anterior := Y;
  Teta_Anterior := Teta;
end;

//Procedimento para sair do programa caso a tecla esc seja pressionada.
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if key = VK_Escape then Close();
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton10Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 1;
  Temp1 := StrtoInt(Form1.StringGrid1.Cells[5,1]);
  Temp2 := StrtoInt(Form1.StringGrid1.Cells[6,1]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
```

```
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton11Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 3;
  Temp1 := StrtoInt(Form1.StringGrid1.Cells[5,2]);
  Temp2 := StrtoInt(Form1.StringGrid1.Cells[6,2]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
```

```
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton12Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 5;
  Temp1 := StrtoInt(Form1.StringGrid1.Cells[5,3]);
  Temp2 := StrtoInt(Form1.StringGrid1.Cells[6,3]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton13Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 7;
  Temp1 := StrtoInt(Form1.StringGrid1.Cells[5,4]);
  Temp2 := StrtoInt(Form1.StringGrid1.Cells[6,4]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
```

```
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton14Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 9;
  Temp1 := StrtoInt(Form1.StringGrid1.Cells[5,5]);
  Temp2 := StrtoInt(Form1.StringGrid1.Cells[6,5]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton15Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 10;
  Temp1 := StrtoInt(Form1.StringGrid2.Cells[4,5]);
  Temp2 := StrtoInt(Form1.StringGrid2.Cells[5,5]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
```

```
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  //Habilita o botão para plotar o caminho aproximado.
  Form1.SpeedButton8.Glyph.LoadFromFile('Line Graph 16 h t.bmp');
  Form1.SpeedButton8.Flat := False;
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton16Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 2;
  Temp1 := StrtoInt(Form1.StringGrid2.Cells[4,1]);
  Temp2 := StrtoInt(Form1.StringGrid2.Cells[5,1]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
```

```
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  if l > 2 then Habilita := 'True'
  else
    begin
      //Habilita o botão para plotar o caminho aproximado.
      Form1.SpeedButton8.Glyph.LoadFromFile('Line Graph 16 h t.bmp');
      Form1.SpeedButton8.Flat := False;
    end;
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton17Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 4;
  Temp1 := StrtoInt(Form1.StringGrid2.Cells[4,2]);
  Temp2 := StrtoInt(Form1.StringGrid2.Cells[5,2]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  if l > 3 then Habilita := 'True'
  else
    begin
      //Habilita o botão para plotar o caminho aproximado.
      Form1.SpeedButton8.Glyph.LoadFromFile('Line Graph 16 h t.bmp');
      Form1.SpeedButton8.Flat := False;
    end;
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton18Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 6;
  Temp1 := StrtoInt(Form1.StringGrid2.Cells[4,3]);
  Temp2 := StrtoInt(Form1.StringGrid2.Cells[5,3]);
```

```
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  if l > 4 then Habilita := 'True'
  else
    begin
      //Habilita o botão para plotar o caminho aproximado.
      Form1.SpeedButton8.Glyph.LoadFromFile('Line Graph 16 h t.bmp');
      Form1.SpeedButton8.Flat := False;
    end;
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento de envio de dados para o Robô (caminho desenhado aproximado).
procedure TForm1.SpeedButton19Click(Sender: TObject);
var
  Temp1,Temp2: Integer;
begin
  Desenho := 8;
  Temp1 := StrtoInt(Form1.StringGrid2.Cells[4,4]);
  Temp2 := StrtoInt(Form1.StringGrid2.Cells[5,4]);
  //Verifica a direção dos motores com base no sinal do Nd e Ne (negativo sentido anti-horário e positivo sentido horário).
  if (Temp1 > 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 198;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  if (Temp1 > 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 202;
      D_Sent[4] := StrtoInt(Form1.Edit22.Text);
      D_Sent[5] := StrtoInt(Form1.Edit21.Text);
    end;
  if (Temp1 < 0) and (Temp2 > 0) then
    begin
      D_Sent[3] := 197;
      D_Sent[4] := StrtoInt(Form1.Edit24.Text);
      D_Sent[5] := StrtoInt(Form1.Edit23.Text);
    end;
  if (Temp1 < 0) and (Temp2 < 0) then
    begin
      D_Sent[3] := 201;
      D_Sent[4] := StrtoInt(Form1.Edit26.Text);
```

```
      D_Sent[5] := StrtoInt(Form1.Edit25.Text);
    end;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[6] := Abs(Temp1);
  D_Sent[7] := Abs(Temp2);
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  if l > 5 then Habilita := 'True'
  else
    begin
      //Habilita o botão para plotar o caminho aproximado.
      Form1.SpeedButton8.Glyph.LoadFromFile('Line Graph 16 h t.bmp');
      Form1.SpeedButton8.Flat := False;
    end;
  //Armazena os valores de Nd e Ne, para plotar o caminho aproximado.
  Enviado_Robo[Desenho,0] := Temp1;
  Enviado_Robo[Desenho,1] := Temp2;
end;

//Procedimento para atualizar a imagem do botão de sincronização dos dados.
procedure TForm1.Edit6Change(Sender: TObject);
begin
  Form1.SpeedButton41.Glyph.LoadFromFile('disconnect16_h.bmp');
  Form1.SpeedButton41.Flat := False;
  Form1.SpeedElastica.Enabled := False;
end;

//Procedimento utilizado para plotar o caminho aproximado realizado pelo Robô.
procedure TForm1.SpeedButton8Click(Sender: TObject);
var
  i: integer;
begin
  Acumulador := 0;
  for i := 1 to Desenho do
    begin
      if (Trunc(i/2) <> i/2) then
        begin
          Nd_Aproximado := Round((Enviado_Robo[i,0])*Ajuste_Nd_Teta);
          Ne_Aproximado := Round((Enviado_Robo[i,1])*Ajuste_Ne_Teta);
        end
      else
        begin
          Nd_Aproximado := Round(Enviado_Robo[i,1]*Ajuste_Linha_Reta);
          Ne_Aproximado := Round(Enviado_Robo[i,1]*Ajuste_Linha_Reta);
        end;
      with Form1.Image1 do
        begin
          Teta_Aproximado := ((2*PI/(Lr*Nres))*(Nd_Aproximado*rd-Ne_Aproximado*re))+Teta_Anterior_Aproximado;
          X_Aproximado := (Nd_Aproximado*rd+Ne_Aproximado*re)*PI*Cos(Teta_Aproximado)/Nres+X_Anterior_Aproximado;
          Y_Aproximado := (Nd_Aproximado*rd+Ne_Aproximado*re)*PI*Sin(Teta_Aproximado)/Nres+Y_Anterior_Aproximado;
          Canvas.Pen.Width := 2;
          Canvas.Pen.Color := clGreen;
          Canvas.MoveTo(X_Anterior_Aproximado+250,-Y_Anterior_Aproximado+500-250);
          Canvas.LineTo(Round(X_Aproximado+250), Round(-Y_Aproximado+500-250)); {desenha a nova linha}
          X_Anterior_Aproximado := Round(X_Aproximado);
          Y_Anterior_Aproximado := Round(Y_Aproximado);
          Teta_Anterior_Aproximado := Teta_Aproximado;
        end;
    end;
  Form1.SpeedButton8.Glyph.LoadFromFile('Line Graph 16 d t.bmp');
  Form1.SpeedButton8.Flat := True;
  //Zerando as variáveis acumulativas.
  Teta_Anterior_Aproximado := 0;
  X_Anterior_Aproximado := 0;
  Y_Anterior_Aproximado := 0;
end;

//Procedimento para plotagem do caminho percorrido pelo Robô (entrada manual).
procedure TForm1.SpeedButton20Click(Sender: TObject);
begin
  try //Exceção criada para evitar erro de conversão de caracteres.
    begin
      Desenho := Desenho + 1;
```

```
      with Form1.Image1 do
       begin
         Nd_Real := StrtoInt(Form1.Edit5.Text);
         Ne_Real := StrtoInt(Form1.Edit13.Text);
         Form1.StringGrid3.Cells[3,Desenho] := InttoStr(Nd_Real);
         Form1.StringGrid3.Cells[4,Desenho] := InttoStr(Ne_Real);
         Teta_Real := (2*PI/(Lr*Nres)*(Nd_Real*rd-Ne_Real*re))+Teta_Anterior_Real;
         Form1.StringGrid3.Cells[5,Desenho] := FloattoStr(Round(RadtoDeg(Teta_Real)));
         X_Real := (Nd_Real*rd+Ne_Real*re)*PI*Cos(Teta_Real)/Nres+X_Anterior_Real;
         Form1.StringGrid3.Cells[6,Desenho] := FloattoStr(Round(X_Real));
         Y_Real := (Nd_Real*rd+Ne_Real*re)*PI*Sin(Teta_Real)/Nres+Y_Anterior_Real;
         Form1.StringGrid3.Cells[7,Desenho] := FloattoStr(Round(Y_Real));
         Canvas.Pen.Width := 1;
         Canvas.Pen.Color := clGray;
         Canvas.Pen.Style := psDash;
         Canvas.MoveTo(250, 250);
         Canvas.LineTo(250,0);
         Canvas.MoveTo(250, 250);
         Canvas.LineTo(500,250);
         Canvas.MoveTo(0, 0);
         Canvas.Pen.Width := 2;
         Canvas.Pen.Color := clBlack;
         Canvas.MoveTo(X_Anterior_Real+250,-Y_Anterior_Real+500-250);
         Canvas.LineTo(Round(X_Real+250),Round(-Y_Real+500-250));
         X_Anterior_Real := Round(X_Real);
         Y_Anterior_Real := Round(Y_Real);
         Teta_Anterior_Real := Teta_Real;
       end;
    end;
  except
    on EConvertError do
      begin
        MessageDlg('Não é um caractere válido, favor digitar apenas números inteiros positivos ou negativos.', mtError, [mbOK], 0);
      end;
  end;
end;

//Procedimento utilizado para transferir dados para o Microsoft Excel.
procedure TForm1.SpeedButton21Click(Sender: TObject);
var
  Excel :variant;
  i: integer;
begin
  try
    Excel := CreateOleObject('excel.application\');
  if not Excel.Application.Visible then
    begin
      Excel.Application.Visible := true;
      Excel.WorkBooks.Add;
      Excel.Cells[1,1] := 'Teta';
      Excel.Cells[1,2] := 'Nd';
      Excel.Cells[1,3] := 'Ne';
      for i := 1 to 74 do
        begin
          Excel.Cells[i+1,1] := Teta_Possivel[i];
          Excel.Cells[i+1,2] := Nd_Possivel[i];
          Excel.Cells[i+1,3] := Ne_Possivel[i];
        end;
    end;
  except
    Showmessage('Ocorreu erro ao executar a transferência.');
  end;
end;

//Procedimento utilizado para transferir dados para o Microsoft Excel.
procedure TForm1.SpeedButton22Click(Sender: TObject);
var
  Excel :variant;
  i,j: integer;
begin
  try
    Excel := CreateOleObject('excel.application\');
  if not Excel.Application.Visible then
    begin
      Excel.Application.Visible := true;
      Excel.WorkBooks.Add;
```

```
    for i := 0 to 5 do
      begin
        for j := 0 to 8 do
          Excel.Cells[i+1,j+1] := Form1.StringGrid1.Cells[j,i];
      end;
  end;
 except
  Showmessage('Ocorreu erro ao executar a transferência.');
 end;
end;

//Procedimento utilizado para transferir dados para o Microsoft Excel.
procedure TForm1.SpeedButton23Click(Sender: TObject);
var
 Excel :variant;
 i,j: integer;
begin
 try
  Excel := CreateOleObject('excel.application\');
 if not Excel.Application.Visible then
  begin
    Excel.Application.Visible := true;
    Excel.WorkBooks.Add;
    for i := 0 to 5 do
      begin
        for j := 0 to 8 do
          Excel.Cells[i+1,j+1] := Form1.StringGrid2.Cells[j,i];
      end;
  end;
 except
  Showmessage('Ocorreu erro ao executar a transferência.');
 end;
end;

//Procedimento utilizado para transferir dados para o Microsoft Excel.
procedure TForm1.SpeedButton24Click(Sender: TObject);
var
 Excel :variant;
 i,j: integer;
begin
 try
  Excel := CreateOleObject('excel.application\');
 if not Excel.Application.Visible then
  begin
    Excel.Application.Visible := true;
    Excel.WorkBooks.Add;
    for i := 0 to 10 do
      begin
        for j := 0 to 8 do
          Excel.Cells[i+1,j+1] := Form1.StringGrid4.Cells[j,i];
      end;
  end;
 except
  Showmessage('Ocorreu erro ao executar a transferência.');
 end;
end;

//Procedimento utilizado para transferir dados para o Microsoft Excel.
procedure TForm1.SpeedButton25Click(Sender: TObject);
var
 Excel :variant;
 i,j: integer;
begin
 try
  Excel := CreateOleObject('excel.application\');
 if not Excel.Application.Visible then
  begin
    Excel.Application.Visible := true;
    Excel.WorkBooks.Add;
    for i := 0 to 99 do
      begin
        for j := 0 to 8 do
          Excel.Cells[i+1,j+1] := Form1.StringGrid3.Cells[j,i];
      end;
  end;
 except
```

```
    Showmessage('Ocorreu erro ao executar a transferência.');
  end;
end;

//Procedimento de solicitação de leitura do sonar novamente.
procedure TForm1.SpeedButton26Click(Sender: TObject);
begin
  Sonar := 1;
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[3] := 255;
  D_Sent[4] := 255;
  D_Sent[5] := 255;
  D_Sent[6] := 255;
  D_Sent[7] := 255;
  WriteFile(hCom,D_Sent,8,BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
end;

//Procedimento utilizado para re-enviar o último pacote de dados enviado ao robô.
procedure TForm1.SpeedButton27Click(Sender: TObject);
begin
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  Confirmar_Dados_Novamente := 'True';
end;

//Procedimento utilizado para solicitar ao robô o re-envio do último pacote de dados já enviado.
procedure TForm1.SpeedButton28Click(Sender: TObject);
begin
  D_Sent[1] := 204;
  D_Sent[2] := 204;
  D_Sent[3] := 0;
  D_Sent[4] := 0;
  D_Sent[5] := 0;
  D_Sent[6] := 0;
  D_Sent[7] := 0;
  WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
  //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
  Tempo_Inicio := Time;
  Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
  Confirmar_Dados_Novamente := 'True';
end;

//Procedimento para mostrar o tempo de execução do programa.
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  StatusBar1.Panels[0].Text:= ' Tempo de Execução do Programa: ' + fExecutando(Tempo);
end;

//Procedimento para não plotar a leitura do sonar.
procedure TForm1.SpeedButton6Click(Sender: TObject);
begin
  Form1.SpeedButton29.Enabled := True;
  Form1.SpeedButton6.Enabled := False;
end;

//Procedimento para plotar a leitura do sonar.
procedure TForm1.SpeedButton29Click(Sender: TObject);
begin
  Form1.SpeedButton6.Enabled := True;
  Form1.SpeedButton29.Enabled := False;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton30Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
```

```
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 5 do
   begin
     case j of
       1:  begin
             X := 10;
             Y := 40;
           end;
       2:  begin
             X := 30;
             Y := 50;
           end;
       3:  begin
             X := 45;
             Y := 40;
           end;
       4:  begin
             X := 30;
             Y := 10;
           end;
       5:  begin
             X := 0;
             Y := 0;
           end;
     end;
     Form1.Image1Click(Sender);
     Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
     Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
   end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton31Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 5 do
   begin
     case j of
       1:  begin
             X := 45;
             Y := 10;
           end;
       2:  begin
             X := 25;
             Y := 20;
           end;
       3:  begin
             X := 45;
             Y := 35;
           end;
       4:  begin
             X := 15;
             Y := 45;
           end;
```

```
      5:  begin
            X := 0;
            Y := 0;
          end;
    end;
    Form1.Image1Click(Sender);
    Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
    Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
  end;
 Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para solicitar os dados novamente ao robô.
procedure TForm1.SpeedButton32Click(Sender: TObject);
begin
 D_Sent[1] := 204;
 D_Sent[2] := 204;
 D_Sent[3] := 0;
 D_Sent[4] := 0;
 D_Sent[5] := 0;
 D_Sent[6] := 0;
 D_Sent[7] := 0;
 WriteFile(hCom,D_Sent,8, BytesEscritos, nil); //Envia os dados.
 //Inicia a contagem do tempo de envio / recebimento dos dados do robô.
 Tempo_Inicio := Time;
 Form1.StatusBar1.Panels[1].Text := 'DADO ENVIADO';
 Solicitar_Dados_Novamente := 'True';
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton33Click(Sender: TObject);
var
 j: integer;
begin
 Form1.SpeedButton30.Enabled := False;
 Form1.SpeedButton31.Enabled := False;
 Form1.SpeedButton33.Enabled := False;
 Form1.SpeedButton34.Enabled := False;
 Form1.SpeedButton35.Enabled := False;
 Form1.SpeedButton36.Enabled := False;
 Form1.SpeedButton37.Enabled := False;
 Form1.SpeedButton38.Enabled := False;
 Form1.SpeedButton39.Enabled := False;
 Form1.SpeedButton40.Enabled := False;
 Image1.Canvas.Pen.Width := 2;
 Image1.Canvas.Pen.Color := clGray;
 for j := 1 to 3 do
  begin
   case j of
    1:  begin
          X := 28;
          Y := 30;
        end;
    2:  begin
          X := 50;
          Y := 15;
        end;
    3:  begin
          X := 83;
          Y := 41;
        end;
   end;
   Form1.Image1Click(Sender);
   Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
   Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
  end;
 Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton34Click(Sender: TObject);
var
 j: integer;
begin
 Form1.SpeedButton30.Enabled := False;
 Form1.SpeedButton31.Enabled := False;
```

```
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 3 do
    begin
      case j of
        1:  begin
              X := 55;
              Y := 20;
            end;
        2:  begin
              X := 30;
              Y := 45;
            end;
        3:  begin
              X := 0;
              Y := 0;
            end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton35Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 4 do
    begin
      case j of
        1:  begin
              X := 50;
              Y := 20;
            end;
        2:  begin
              X := 40;
              Y := 60;
            end;
        3:  begin
              X := -10;
              Y := 40;
            end;
        4:  begin
              X := 0;
              Y := 0;
            end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;
```

```
//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton36Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 5 do
    begin
      case j of
        1:  begin
              X := 10;
              Y := 25;
            end;
        2:  begin
              X := 30;
              Y := 35;
            end;
        3:  begin
              X := 40;
              Y := 20;
            end;
        4:  begin
              X := 20;
              Y := 15;
            end;
        5:  begin
              X := 25;
              Y := 0;
            end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton37Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 5 do
    begin
      case j of
        1:  begin
              X := 40;
              Y := 10;
            end;
        2:  begin
              X := 45;
              Y := 30;
```

```
          end;
        3:  begin
              X := 25;
              Y := 40;
            end;
        4:  begin
              X := 45;
              Y := 55;
            end;
        5:  begin
              X := 75;
              Y := 45;
            end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton38Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 5 do
    begin
      case j of
        1:  begin
              X := 10;
              Y := 35;
            end;
        2:  begin
              X := 25;
              Y := 25;
            end;
        3:  begin
              X := 35;
              Y := 45;
            end;
        4:  begin
              X := 50;
              Y := 30;
            end;
        5:  begin
              X := 40;
              Y := 5;
            end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton39Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
```

```
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 5 do
    begin
      case j of
        1:  begin
              X := 20;
              Y := 20;
            end;
        2:  begin
              X := 40;
              Y := 0;
            end;
        3:  begin
              X := 40;
              Y := 40;
            end;
        4:  begin
              X := 0;
              Y := 40;
            end;
        5:  begin
              X := 0;
              Y := 0;
            end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento para desenhar um caminho já conhecido.
procedure TForm1.SpeedButton40Click(Sender: TObject);
var
  j: integer;
begin
  Form1.SpeedButton30.Enabled := False;
  Form1.SpeedButton31.Enabled := False;
  Form1.SpeedButton33.Enabled := False;
  Form1.SpeedButton34.Enabled := False;
  Form1.SpeedButton35.Enabled := False;
  Form1.SpeedButton36.Enabled := False;
  Form1.SpeedButton37.Enabled := False;
  Form1.SpeedButton38.Enabled := False;
  Form1.SpeedButton39.Enabled := False;
  Form1.SpeedButton40.Enabled := False;
  Image1.Canvas.Pen.Width := 2;
  Image1.Canvas.Pen.Color := clGray;
  for j := 1 to 4 do
    begin
      case j of
        1:  begin
              X := 40;
              Y := 15;
            end;
        2:  begin
              X := 25;
              Y := 30;
            end;
        3:  begin
              X := 15;
              Y := -15;
            end;
        4:  begin
              X := 0;
              Y := 0;
```

```
          end;
      end;
      Form1.Image1Click(Sender);
      Form1.Image1.Canvas.MoveTo(Round(250+X_Anterior2),Round(250-Y_Anterior2));
      Form1.Image1.Canvas.LineTo(Round(250+X),Round(250-Y));
    end;
  Form1.SpeedElastica.Enabled := True;
end;

//Procedimento de ajuste do valor de Nd.
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  Form1.Edit27.Text := FloattoStr((Form1.TrackBar1.Position)/100);
  Ajuste_Nd_Teta := StrtoFloat(Form1.Edit27.Text);
  Form1.SpeedButton41.Glyph.LoadFromFile('disconnect16_h.bmp');
  Form1.SpeedButton41.Flat := False;
  Form1.SpeedElastica.Enabled := False;
end;

//Procedimento de ajuste do valor de Ne.
procedure TForm1.TrackBar2Change(Sender: TObject);
begin
  Form1.Edit28.Text := FloattoStr((Form1.TrackBar2.Position)/100);
  Ajuste_Ne_Teta := StrtoFloat(Form1.Edit28.Text);
  Form1.SpeedButton41.Glyph.LoadFromFile('disconnect16_h.bmp');
  Form1.SpeedButton41.Flat := False;
  Form1.SpeedElastica.Enabled := False;
end;

//Procedimento utilizado para atualizar os valores com base nos parâmetros de ajuste do robô.
procedure TForm1.SpeedButton41Click(Sender: TObject);
var
  i,j,k: integer;
  temp1,temp2,temp3,temp4,temp5,temp6: real;
begin
  try //Exceção criada para evitar erro de conversão de caracteres.
    begin
      Nres := StrtoInt(Form1.Edit6.Text); //Resolução dos sensores, que é o número de pulsos lidos em uma revolução das rodas.
      rd := StrtoFloat(Form1.Edit7.Text); //Raio da roda direita.
      re := StrtoFloat(Form1.Edit8.Text); //Raio da roda esquerda.
      Lr := StrtoInt(Form1.Edit9.Text); //Comprimento do eixo do robô.
      Temp_amb := (StrtoInt(Form1.Edit17.Text)) + 273.15; //Temperatura ambiente em graus Kelvin.
      Vel_Som := 331.4*Sqrt(Temp_amb/273);
      Form1.Edit16.Text := InttoStr(Round(Vel_Som));
      Ajuste_Nd_Teta := StrtoFloat(Form1.Edit27.Text);
      Ajuste_Ne_Teta := StrtoFloat(Form1.Edit28.Text);
      Teta_Ajuste_Real := StrtoFloat(Form1.Edit29.Text);
      Ajuste_Linha_Reta := StrtoFloat(Form1.Edit30.Text);
      //Armazena os valores de ângulos possíveis com base no Nres.
      for i := 1 to 28 do
        begin
          Nd := i;
          Ne := -1;
          Teta_Possivel[i] := Round((RadtoDeg((2*PI/(Lr*Nres))*(Nd*rd-Ne*re)))*Ajuste_Nd_Teta);
          if Abs(Teta_Possivel[i]) <= 180 then
            begin
              Nd_Possivel[i] := i;
              Ne_Possivel[i] := -1;
            end
          else
            begin
              Nd_Possivel[i] := 0;
              Ne_Possivel[i] := 0;
              Teta_Possivel[i] := 0;
            end;
          //Seqüência lógica necessária, caso os ângulos calculados, sejam maiores do que 360 graus.
          //while Teta_Possivel[i] > 360 do Teta_Possivel[i] := Teta_Possivel[i] - 360;
        end;
      //Armazena os valores de ângulos possíveis com base no Nres.
      for i := 29 to 56 do
        begin
          Nd := -1;
          Ne := i-28;
          Teta_Possivel[i] := Round((RadtoDeg((2*PI/(Lr*Nres))*(Nd*rd-Ne*re)))*Ajuste_Ne_Teta);
          if Abs(Teta_Possivel[i]) <= 180 then
            begin
```

```
            Nd_Possivel[i] := -1;
            Ne_Possivel[i] := i-28;
           end
          else
           begin
            Nd_Possivel[i] := 0;
            Ne_Possivel[i] := 0;
            Teta_Possivel[i] := 0;
           end;
          //Seqüência lógica necessária, caso os ângulos calculados, sejam maiores do que 360 graus.
          //while Teta_Possivel[i] > 360 do Teta_Possivel[i] := Teta_Possivel[i] - 360;
        end;
      //Ordena os ângulos possíveis de forma ascendente.
      for i := 1 to 56 do
        begin
          for j := i+1 to 56 do
           begin
            if (Teta_Possivel[i] > Teta_Possivel[j]) and (Teta_Possivel[i] <> Temp2) then
             begin
              temp1 := Teta_Possivel[i];
              temp3 := Nd_Possivel[i];
              temp5 := Ne_Possivel[i];
              Teta_Possivel[i] := Teta_Possivel[j];
              Nd_Possivel[i] := Nd_Possivel[j];
              Ne_Possivel[i] := Ne_Possivel[j];
              Teta_Possivel[j] := temp1;
              Nd_Possivel[j] := temp3;
              Ne_Possivel[j] := temp5;
             end;
           end;
          temp2 := Teta_Possivel[i];
          temp4 := Nd_Possivel[i];
          temp6 := Ne_Possivel[i];
        end;
      k := 0;
      for i := 1 to 56 do
        begin
          if Teta_Possivel[i] <> 0 then
           begin
            k := k + 1;
            Teta_Possivel[k] := Teta_Possivel[i];
            Nd_Possivel[k] := Nd_Possivel[i];
            Ne_Possivel[k] := Ne_Possivel[i];
           end;
        end;
      //Zera a variável VRecebido_Convertido.
      for i := 1 to 100000 do VRecebido_Convertido[i] := '0';
      //Atualiza o padrão do botão de atualização dos parâmetros do Robô.
      Form1.SpeedButton41.Glyph.LoadFromFile('connect16_d.bmp');
      Form1.SpeedButton41.Flat := True;
      Form1.SpeedElastica.Enabled := True;
    end;
  except
    on EConvertError do
      begin
        MessageDlg('Não é um caractere válido, favor digitar apenas números positivos.', mtError, [mbOK], 0);
      end;
  end;
end;

//Seqüência lógica utilizada para listar os pulsos vs as distâncias equivalentes.
procedure TForm1.SpeedButton42Click(Sender: TObject);
var
  i: integer;
begin
  for i := 0 to 15 do
    begin
      Distancia := (i/Nres)*2*Pi*rd*Ajuste_Linha_Reta;
      if i < 20 then
        begin
          Nd := Round(i*0.5);
          Ne := Round(i);
        end
      else
        begin
          Nd := Round(i-10);
```

```
        Ne := Round(i);
       end;
     Form1.StringGrid3.Cells[1,i+1] := InttoStr(Round(Distancia));
     Form1.StringGrid3.Cells[2,i+1] := InttoStr(Round(Nd));
     Form1.StringGrid3.Cells[3,i+1] := InttoStr(Round(Ne));
   end;
end;

//Procedimento para ajuste do valor de teta real.
procedure TForm1.TrackBar3Change(Sender: TObject);
begin
  Form1.Edit29.Text := FloattoStr((Form1.TrackBar3.Position)/100);
  Teta_Ajuste_Real := StrtoFloat(Form1.Edit29.Text);
  Form1.SpeedButton41.Glyph.LoadFromFile('disconnect16_h.bmp');
  Form1.SpeedButton41.Flat := False;
  Form1.SpeedElastica.Enabled := False;
end;

//Procedimento para ajuste do cálculo da linha reta.
procedure TForm1.TrackBar4Change(Sender: TObject);
begin
  Form1.Edit30.Text := FloattoStr((Form1.TrackBar4.Position)/100);
  Ajuste_Linha_Reta := StrtoFloat(Form1.Edit30.Text);
  Form1.SpeedButton41.Glyph.LoadFromFile('disconnect16_h.bmp');
  Form1.SpeedButton41.Flat := False;
  Form1.SpeedElastica.Enabled := False;
end;

end.

//FIM DO PROGRAMA
```

10.1.13 Programa Completo do Circuito de IHM

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa completo do circuito de IHM (última versão)
Módulo: IHM
Linguagem: Assembler
Arquivo: FIHM_v04.asm
Objetivo: Testar o funcionamento completo do circuito de IHM.
Descrição: Parte 1 (Recepção por meio de comunicação serial):
Uma seqüência de 7 bytes deve ser recebida do PC através da UART, sendo:
- Bytes 1 e 2: CCh (identificação do pacote de dados)
- Byte 3: Direção dos motores
- Byte 4: Velocidade do motor 2 (esquerdo)
- Byte 5: Velocidade do motor 1 (direito)
- Byte 6: Número de pulsos detectados pelo sensor do motor 1
- Byte 7: Número de pulsos detectados pelo sensor do motor 2

Os dados são codificados e enviados para o robô por meio de RF.

Parte 2 (Recepção por meio de RF):
Uma seqüência de 10 bytes deve ser transmitida pelo robô por meio de RF e recebida pelo módulo de IHM através da UART, sendo:
- Bytes 1 e 2: CCh (identificação do pacote de dados)
- Bytes 3 e 4: Quantidade de rotações do Motor 1
- Bytes 5 e 6: Quantidade de rotações do Motor 2
- Bytes 7 e 8: Tempo de vôo que a onda leva para retornar ao sensor, registrador de 16 bits (TCNT1H)
- Bytes 9 e 10: Tempo de vôo que a onda leva para retornar ao sensor, registrador de 16 bits (TCNT1L)

Por fim, os dados são decodificados e transmitidos ao PC utilizando-se a comunicação serial.

```
.include "8515def.inc"

;** Declaração das Variáveis
.def    temp      = r16
.def    temp1     = r17
.def    temp2     = r18
.def    final     = r19
.def    vfinal    = r20
.def    count     = r21
.def    ponteiro1 = r22
.def    ponteiro2 = r23
.def    chave     = r24
.def    count2    = r25

        rjmp    RESET                   ;Pula para o Programa Principal

.org URXCaddr                           ;UART Receive Complete Interrupt Vector Address
        rjmp    Receber

.org UDREaddr                           ;UART Data Register Empty Interrupt Vector Address
        rjmp    Transmitindo

;** Bloco de transmissão dos dados
Transmitindo:
        inc     XL                      ;Posiciona o ponteiro
        ld      temp,X                  ;Carrega o dado armazenado na variável temp
        out     UDR,temp                ;Envia os dados armazenados na variável temp para o registrador de dados da UART
        out     PORTA,temp              ;Envia a mesma informação para o Port A
        cpi     chave,0b11111110        ;Verifica se está transmitindo para a IHM ou para o Robô
        breq    Transmitindo_IHM        ;Se for para a IHM pula para o bloco Transmitindo_IHM
        cpi     XL,0x6D                 ;Compara para saber se o último caractere foi enviado
        breq    Atraso                  ;Pula para o bloco Atraso se todos os dados foram enviados
        reti                            ;Retorna da Interrupção
```

```
Atraso:
        rjmp    Atraso_P1           ;Pula para o bloco Atraso_P1

Transmitindo_IHM:
;** Bloco de atraso, para o envio dos dados ao PC
        ldi     ponteiro1,255       ;Carrega o valor 255 na variável ponteiro1
        ldi     ponteiro2,255       ;Carrega o valor 255 na variável ponteiro2
        ldi     temp,2              ;Carrega o valor 2 na variável temp
Loop2:
        dec     ponteiro1           ;Decrementa a variável ponteiro1
        brne    Loop2               ;Pula para Loop2 se ponteiro1 for diferente de zero
        dec     ponteiro2           ;Decrementa a variável ponteiro2
        brne    Loop2               ;Pula para Loop2 se ponteiro2 for diferente de zero
        dec     temp                ;Decrementa a variável temp
        brne    Loop2               ;Pula para Loop2 se temp for diferente de zero
        cpi     XL,0x87             ;Compara para saber se o último caractere foi enviado
        breq    Atraso              ;Pula para o bloco Atraso se todos os dados foram enviados
        reti                        ;Retorna da Interrupção

;** Bloco que faz a leitura dos dados recebidos
Receber:
        in      temp,UDR            ;Lê os dados recebidos
        inc     count               ;Incrementa a variável count
        cpi     count,3             ;Compara a variável count com 3
        brlo    Confirmar           ;Pula para o bloco Confirmar se count for menor que 3
        rjmp    Armazena            ;Pula para o bloco Armazena se count for maior que 3

;** Reconhecimento de pacote de dados válidos
Confirmar:
        cpi     temp,0xCC           ;Compara a variável temp com 11001100b
        breq    Retornando          ;Pula para o bloco Retornando se temp for igual a CCh
        clr     count               ;Limpa a variável count se o pacote não for reconhecido
        reti                        ;Retorna da interrupção

Retornando:
        reti                        ;Retorna da interrupção

;** Armazenamento dos dados recebidos após identificação do pacote de dados
Armazena:
        cpi     chave,0b11111101    ;Verifica se está recebendo da IHM ou do Robô
        breq    Armazena_IHM        ;Se for da IHM pula para o bloco Armazena_IHM
        inc     XL                  ;Incrementa o ponteiro
        st      X,temp              ;Armazena o dado recebido
        cpi     count,0x0A          ;Compara count com 10 para armazenar até 8 caracteres recebidos
        breq    Converte            ;Pula para decodificar a informação recebida
        reti                        ;Retorna da Interrupção

Armazena_IHM:
        inc     XL                  ;Incrementa o ponteiro
        st      X,temp              ;Armazena o dado recebido
        inc     XL                  ;Incrementa o ponteiro
        st      X,temp              ;Armazena o dado recebido
        cpi     count,7             ;Compara count com 7 para armazenar até 5 caracteres recebidos
        breq    Codificar           ;Pula para codificar os dados se count for igual à 7
        reti                        ;Retorna da Interrupção

Codificar:
        rjmp    Posicionando

;** Bloco que faz a decodificação dos dados recebidos
Converte:
        cli                         ;Desabilita a Interrupção
        clr     count               ;Limpa a variável count
        ldi     XL,0x61             ;Posiciona o ponteiro

Convertendo:
        inc     count               ;Incrementa o contador
        inc     XL                  ;Incrementa o ponteiro
        ld      temp,X              ;Carrega o dado codificado (recebido) na variável temp
        clr     final               ;Limpa a variável final
        mov     temp1,temp          ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x01           ;Verifica o valor do primeiro bit ("AND" com 00000001b)
        add     final,temp          ;Adiciona o valor da variável temp à variável final
        lsr     temp1               ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1          ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x02           ;Verifica o valor do segundo bit ("AND" com 00000010b)
```

```
            add     final,temp          ;Adiciona o valor da variável temp à variável final
            lsr     temp1               ;Desloca os bits da variável temp1 para a direita
            mov     temp,temp1          ;Copia o valor da variável temp para a variável temp1
            andi    temp,0x04           ;Verifica o valor do terceiro bit ("AND" com 00000100b)
            add     final,temp          ;Adiciona o valor da variável temp à variável final
            lsr     temp1               ;Desloca os bits da variável temp1 para a direita
            mov     temp,temp1          ;Copia o valor da variável temp para a variável temp1
            andi    temp,0x08           ;Verifica o valor do quarto bit ("AND" com 00001000b)
            add     final,temp          ;Adiciona o valor da variável temp à variável final
            cpi     count,1             ;Compara count com 1 para saber se apenas a primeira parte do byte foi decodificado
            breq    Parte1              ;Pula para o bloco Parte 1 se verdadeiro
            rjmp    Parte2              ;Pula para o bloco Parte 2 se já tiver sido decodificado o byte inteiro

Parte1:
            mov     vfinal,final        ;Armazena a primeira parte do byte decodificado na variável vfinal
            rjmp    Convertendo         ;Pula para o bloco Convertendo para continuar com a decodificação dos dados

Parte2:
            swap    final               ;Inverte a posição dos últimos bits pelos primeiros
            add     vfinal,final        ;Adiciona a segunda parte do byte decodificado à primeira parte
            mov     ponteiro1,XL        ;Armazena a posição do ponteiro na variável ponteiro1
            cpi     ponteiro1,0x64      ;Verifica se o primeiro byte recebido já foi decodificado e armazenado
            brsh    Pula                ;Pula para o bloco Pula se verdadeiro
            ldi     XL,0x80             ;Posiciona o ponteiro
            mov     ponteiro2,XL        ;Armazena o valor do ponteiro na variável ponteiro2
            inc     count2
            st      X,count2            ;Armazena 1,2,3 ou 4
            inc     XL                  ;Incrementa o ponteiro
            mov     ponteiro2,XL        ;Armazena o valor do ponteiro na variável ponteiro2
            st      X,vfinal            ;Armazena o dado decodificado na memória
            clr     count               ;Limpa a variável count
            clr     vfinal              ;Limpa a variável vfinal
            mov     XL,ponteiro1        ;Posiciona o ponteiro
            rjmp    Convertendo         ;Pula para o bloco Convertendo

Pula:
            inc     ponteiro2           ;Posiciona o ponteiro
            mov     XL,ponteiro2        ;Armazena o valor do ponteiro na variável ponteiro2
            inc     count2
            st      X,count2            ;Armazena 1,2,3 ou 4
            inc     ponteiro2           ;Posiciona o ponteiro
            mov     XL,ponteiro2        ;Armazena o valor do ponteiro na variável ponteiro2
            st      X,vfinal            ;Armazena o dado na memória
            clr     count               ;Limpa a variável count
            clr     vfinal              ;Limpa a variável vfinal
            cpi     ponteiro2,0x87      ;Verifica se todos os dados recebidos foram decodificados e armazenados
            breq    Posicionando        ;Pula para o bloco Posicionando se verdadeiro
            mov     XL,ponteiro1        ;Posiciona o ponteiro
            rjmp    Convertendo         ;Pula para o bloco Convertendo

Retornando2:
            reti                        ;Retorna da interrupção

;** Loop de Atraso
Atraso_P1:
            cpi     chave,0b11111110    ;Verifica se já transmitiu para a IHM o dado recebido pelo robô
            breq    Reinicio            ;Pula para o bloco Reinicio em caso positivo, para reiniciar todo o processo
            inc     count               ;Incrementa o valor da variável count
            ldi     XL,0x5F             ;Posiciona o ponteiro
            cpi     count,10            ;Compara para verificar se já enviou o mesmo dado 10 vezes
            brlo    Retornando2         ;Retorna da interrupção em caso negativo

;** Bloco de atraso, para o início do recebimento dos dados do robô
Atraso_P2:
            ldi     ponteiro1,255       ;Carrega 255 na variável ponteiro1
            ldi     ponteiro2,1         ;Carrega 1 na variável ponteiro2
Loop:
            dec     ponteiro1           ;Decrementa a variável ponteiro1
            brne    Loop                ;Pula para Loop se ponteiro1 for diferente de zero
            dec     ponteiro2           ;Decrementa a variável ponteiro2
            brne    Loop                ;Pula para Loop se ponteiro2 for diferente de zero
            ldi     chave,0b11111110
            out     PORTB,chave         ;Seleciona a recepção com o Robô (PB0 em nível baixo)
            rjmp    R_Robo              ;Pula para o bloco R_Robo, para receber os dados do robô

Reinicio:
```

```
            rjmp    R_IHM                   ;Pula para o bloco R_IHM, para reiniciar todo o processo

Posicionando:
            cli                             ;Desabilita a interrupção
            clr     count                   ;Limpa a variável count
            ldi     XL,0x62                 ;Posiciona o ponteiro
            ld      temp,X                  ;Carrega o dado na variável temp
            cpi     chave,0b11111101        ;Verifica se a informação deve ser codificada ou não
            brne    Transmitir              ;Se não precisar codificar, ou seja, informação para a IHM,
                                            ;pula para o bloco Transmitir
;** Bloco de codificação dos dados
Codificando_P1:
            clr     temp2                   ;Limpa a variável temp2
            inc     count                   ;Incrementa a variável count
            cpi     count,6                 ;Compara para saber se todos os dados foram codificados
            breq    Transmitir              ;Caso positivo pula para o bloco Transmitir
            andi    temp,0x0F               ;Codifica apenas os quatro primeiros bits ("AND" com 00001111b)

Codificando_P2:
            clr     final                   ;Limpa a variável final
            mov     temp1,temp              ;Copia o valor da variável temp para a variável temp1
            com     temp                    ;Faz o complemento da variável temp

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x01          ;Codifica o primeiro bit ("AND" com 00000001b)
            andi    ponteiro1,0x02          ;Codifica o primeiro bit ("AND" com 00000010b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x04          ;Codifica o segundo bit ("AND" com 00000100b)
            andi    ponteiro1,0x08          ;Codifica o segundo bit ("AND" com 00001000b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x10          ;Codifica o terceiro bit ("AND" com 00010000b)
            andi    ponteiro1,0x20          ;Codifica o terceiro bit ("AND" com 00100000b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                    ;Desloca os bits da variável temp para a esquerda
            lsl     temp1                   ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp          ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1         ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x40          ;Codifica o quarto bit ("AND" com 01000000b)
            andi    ponteiro1,0x80          ;Codifica o quarto bit ("AND" com 10000000b)
            add     ponteiro1,ponteiro2     ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1         ;Adiciona o valor da variável ponteiro1 à variável final

;** Bloco de verificação se o byte inteiro foi codificado
            st      X,final                 ;Armazena o dado na memória
            inc     XL                      ;Incrementa o ponteiro
            ld      temp,X                  ;Lê os dados e armazena na variável temp
            cpi     temp2,0x01              ;Compara se a variável temp2 é igual a 1
            breq    Codificando_P1          ;Pula para o bloco Codificando_P1 se o byte inteiro foi codificado
            andi    temp,0xF0               ;Codifica apenas os quatro últimos bits ("AND" com 11110000b)
            swap    temp                    ;Troca de posição dos quatro últimos bits pelos quatro primeiros
            ldi     temp2,0x01              ;Carrega 1 na variável temp2
            rjmp    Codificando_P2          ;Pula para o bloco que faz a codificação dos dados

;** Bloco de configuração para iniciar a transmissão dos dados
Transmitir:
            clr     count                   ;Limpa a variável count
            ldi     temp,0b00101000
            out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
            cpi     chave,0b11111110        ;Verifica se vai transmitir para a IHM ou para o Robô
            breq    Transmitir_IHM          ;Se for para a IHM, pula para o bloco Transmitir_IHM
```

```
            ldi     XL,0x5D             ;Posiciona o ponteiro
            sei                         ;Habilita a interrupção
            rjmp    Volta               ;Pula para o bloco Volta e fica aguardando a próxima interrupção

Transmitir_IHM:
            ldi     XL,0x7F             ;Posiciona o ponteiro
            ldi     temp,0xCC           ;Armazena o valor 11001100b na variável temp
            st      X,temp              ;Armazena o mesmo valor na memória
            dec     XL                  ;Posiciona o ponteiro
            sei                         ;Habilita a interrupção
            rjmp    Volta               ;Pula para o bloco Volta e fica aguardando a próxima interrupção

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
            ldi     temp,low(RAMEND)
            out     SPL,temp
            ldi     temp,high(RAMEND)
            out     SPH,temp
            ldi     temp,0xFF
            out     DDRB,temp           ;Porta B configurada como saída
            ldi     temp,0xFF
            out     DDRA,temp           ;Porta A configurada como saída
            ser     temp
            out     PORTA,temp          ;Garante 0 inicialmente na saída (LED's)

;** Inicialização do Vetor de Dados
            ldi     XH,0x00

;** Programação da UART
            ldi     temp,103
            out     UBRR,temp           ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)

;** Armazenamento dos caracteres de reconhecimento do pacote
            ldi     XL,0x60             ;Posiciona o ponteiro
            ldi     temp,0xCC           ;Armazena o valor 11001100b na variável temp
            st      X,temp              ;Armazena o mesmo valor na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o mesmo valor na memória

R_IHM:
            ldi     chave,0b11111101
            out     PORTB,chave         ;Seleciona a recepção com a IHM (PB1 em nível baixo)

R_Robo:
            cli                         ;Desabilita a interrupção
            ldi     XL,0x61             ;Posiciona o ponteiro
            ldi     temp,0b10010000
            out     UCR,temp            ;Habilita recepção e interrupção por recepção completa de dados
            clr     count               ;Limpa a variável count
            clr     count2              ;Limpa a variável count2
            clr     temp                ;Limpa a variável temp
            sei                         ;Habilita a interrupção

Volta:
            rjmp    Volta               ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

10.1.14 Programa Completo do Circuito do Robô

PROGRAMA PARA MICROCONTROLADOR A T 9 0 S 8 5 1 5

Título: Programa completo do circuito do Robô (última versão)
Módulo: Robô
Linguagem: Assembler
Arquivo: FRob_v09.asm
Objetivo: Testar o funcionamento completo do robô.
Descrição: Parte 1 (Recepção por meio de RF):
Uma seqüência de 12 bytes codificados deve ser recebida através da UART (módulo de recepção), sendo:

- Bytes 1 e 2: CCh (identificação do pacote de dados)
- Bytes 3 e 4: Direção dos motores
- Bytes 5 e 6: Velocidade do motor 2 (esquerdo)
- Bytes 7 e 8: Velocidade do motor 1 (direito)
- Bytes 9 e 10: Número de pulsos detectados pelo sensor do motor 1
- Bytes 11 e 12: Número de pulsos detectados pelo sensor do motor 2

Os dados são decodificados e os comandos executados pelo robô.

Sensores de velocidade:
Após a leitura dos pulsos detectados pelos sensores de velocidade, os motores devem ser desligados de forma independente, de acordo com a leitura dos pulsos, após a parada total dos motores, os sensores de velocidade são desligados.

Obstáculo:
O sensor de ultrasom é acionado.
Através do pino 4 da Porta B do microcontrolador, é medido o tempo em que a saída do sensor fica em nível alto. Esse tempo é armazenado na memória o qual será enviado ao PC em decimal o tempo medido.

Transmissão por meio de RF:
Uma seqüência de 6 bytes deve ser transmitida ao módulo de recepção da IHM através da UART (módulo de transmissão), sendo:

- Bytes 1 e 2: CCh (identificação do pacote de dados)
- Bytes 3: Quantidade de rotações do Motor 1
- Bytes 4: Quantidade de rotações do Motor 2
- Bytes 5: Tempo de vôo que a onda leva para retornar ao sensor registrador de 16 bits (TCNT1H)
- Bytes 6: Tempo de vôo que a onda leva para retornar ao sensor registrador de 16 bits (TCNT1L)

Por fim, os dados são codificados e transmitidos à IHM utilizando-se o módulo de transmissão RF.

Parte 2 (Transmitir novamente):
Se os bytes recebidos após a identificação do pacote de dados for igual a AA em hexadecimal, isso significa que o dado anteriormente enviado, deve ser enviado novamente.

Parte 3 (Leitura do sonar):
Se os bytes recebidos após a identificação do pacote de dados for igual a 55 em hexadecimal, isso significa que o sensor de ultrasom precisa fazer uma nova leitura do ambiente e esse dado deve ser enviado ao módulo de recepção.

```
.include "8515def.inc"

;** Declaração das Variáveis
.def    temp    = r16
.def    temp1   = r17
.def    temp2   = r18
```

```
.def    temp3    = r19
.def    final    = r20
.def    vfinal   = r21
.def    count    = r22
.def    chave    = r23
.def    ponteiro1 = r24
.def    ponteiro2 = r25

        rjmp     RESET                  ;Pula para o Programa Principal

.org URXCaddr                          ;UART Receive Complete Interrupt Vector Address
        rjmp     Receber

.org UDREaddr                          ;UART Data Register Empty Interrupt Vector Address
        rjmp     Transmitindo

;** Bloco de transmissão dos dados codificados
Transmitindo:
        inc      XL                    ;Posiciona o ponteiro
        ld       temp,X                ;Carrega o dado armazenado na variável temp
        out      UDR,temp              ;Envia os dados armazenados na variável temp para o registrador de dados da UART
        cpi      XL,0x7A               ;Compara para saber se o último caractere foi enviado
        breq     Atraso                ;Pula para o bloco Atraso se todos os dados foram enviados
        reti                           ;Retorna da interrupção

Atraso:
        rjmp     Atraso_P1             ;Comando intermediário de pulo, pois a função breq estava fora de alcance

;** Bloco que faz a leitura dos dados recebidos
Receber:
        in       temp,UDR              ;Lê os dados recebidos
        inc      count                 ;Incrementa a variável count
        cpi      count,3               ;Compara a variável count com 3
        brlo     Confirmar             ;Pula para o bloco Confirmar se count for menor que 3
        rjmp     Armazena              ;Pula para o bloco Armazena se count for maior que 3

;** Reconhecimento de pacote de dados válidos
Confirmar:
        cpi      temp,0xCC             ;Compara a variável temp com 11001100b
        breq     Retornando1           ;Pula para o bloco Retornando1 se temp for igual a CCh
        clr      count                 ;Limpa a variável count se não for reconhecido o pacote de dados
        reti                           ;Retorna da interrupção

Retornando1:
        reti                           ;Retorna da interrupção

;** Armazenamento dos dados recebidos após identificação do pacote
Armazena:
        inc      XL                    ;Incrementa o ponteiro
        st       X,temp                ;Armazena o dado recebido
        cpi      count,12              ;Compara count com 12 para armazenar até 10 caracteres recebidos
        breq     Converte              ;Pula para decodificar os dados se count for igual à 12
        reti                           ;Retorna da interrupção

;** Bloco Checksum, utilizado para confirmar se o dado recebido anteriormente está correto
ReceberDenovo:
        ldi      temp3,2               ;Carrega 2 na variável temp3
        clr      count                 ;Limpa a variável count
        ldi      XL,0x7F               ;Posiciona o ponteiro
        reti                           ;Retorna da interrupção

Checksum1:
        ldi      XL,0x5F               ;Posiciona o ponteiro
        clr      temp1                 ;Limpa a variável temp1

Checksum2:
        inc      XL                    ;Incrementa o ponteiro
        ld       temp,X                ;Carrega o dado codificado (recebido) na variável temp
        add      temp1,temp            ;Adiciona o valor da variável temp com a variável temp1 e
                                       ;armazena o resultado na variável temp1
        cpi      XL,0x69               ;Compara para saber se todos os valores foram somados
        brne     Checksum2             ;Pula para o bloco Checksum2 caso negativo

Checksum3:
        ldi      XL,0x7F               ;Posiciona o ponteiro
        clr      temp2                 ;Limpa a variável temp2
```

```
Checksum4:
        inc     XL              ;Incrementa o ponteiro
        ld      temp,X          ;Carrega o dado codificado (recebido) na variável temp
        add     temp2,temp      ;Adiciona o valor da variável temp com a variável temp2 e
                                ;armazena o resultado na variável temp2
        cpi     XL,0x89         ;Compara para saber se todos os valores foram somados
        brne    Checksum4       ;Pula para o bloco Checksum4 caso negativo

        ldi     temp3,3         ;Carrega 3 na variável temp3
        cp      temp1,temp2     ;Compara para saber se os dados recebidos são iguais
        breq    Converte        ;Pula para o bloco Converte em caso positivo
        rjmp    Denovo          ;Se os dados forem diferentes pula para o bloco Denovo para reiniciar o processo

;** Bloco que faz a transmissão do dado já enviado anteriormente
TransmitirDeNovo:
        ldi     temp3,4         ;Carrega 4 na variável temp
        ldi     XL,0x60         ;Posiciona o ponteiro
        ld      temp,X          ;Carrega o dado codificado (recebido) na variável temp
        cpi     temp,0xAA       ;Compara com o valor AA em hexadecimal
        brne    Converte        ;Caso não seja igual a AA, pula para o bloco Converte
        rjmp    Transmitir      ;Caso seja igual a AA, pula para o bloco Transmitir, pois significa que o dado já foi
                                ;enviado anteriormente e dessa forma já está codificado para o envio

;** Bloco que faz a leitura do sensor de ultrasom
SonarDeNovo:
        clr     temp3           ;Limpa a variável temp3
        ldi     XL,0x60         ;Posiciona o ponteiro
        ld      temp,X          ;Carrega o dado codificado (recebido) na variável temp
        cpi     temp,0x55       ;Compara com o valor 55 em hexadecimal
        brne    Converte        ;Caso não seja igual a 55, pula para o bloco Converte
        ldi     XL,0x75         ;Caso seja igual a 55, posiciona o ponteiro
        rjmp    Sonar           ;Pula para o bloco Sonar

;** Bloco que faz a decodificação dos dados recebidos
Converte:
        cpi     temp3,1
        breq    ReceberDenovo   ;Pula para o bloco que verifica se os dados já foram recebidos duas vezes
        cpi     temp3,2
        breq    Checksum1       ;Pula para o bloco que faz a confirmação dos dados recebidos
        cpi     temp3,3
        breq    TransmitirDeNovo ;Pula para o bloco que verifica se os dados enviados
                                ;anteriormente, devem ser enviados de novo
        cpi     temp3,4
        breq    SonarDeNovo     ;Pula para o bloco que verifica se foi solicitado uma nova
                                ;leitura dos dados do sensor de ultrasom
        cli                     ;Desabilita a interrupção
        clr     count           ;Limpa a variável count
        ldi     XL,0x5F         ;Posiciona o ponteiro

;** Bloco que inicia a decodificação dos dados recebidos
Convertendo:
        inc     count           ;Incrementa o contador
        inc     XL              ;Incrementa o ponteiro
        ld      temp,X          ;Carrega o dado codificado (recebido) na variável temp
        clr     final           ;Limpa a variável final
        mov     temp1,temp      ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x01       ;Verifica o valor do primeiro bit ("AND" com 00000001b)
        add     final,temp      ;Adiciona o valor da variável temp à variável final
        lsr     temp1           ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1      ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x02       ;Verifica o valor do segundo bit ("AND" com 00000010b)
        add     final,temp      ;Adiciona o valor da variável temp à variável final
        lsr     temp1           ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1      ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x04       ;Verifica o valor do terceiro bit ("AND" com 00000100b)
        add     final,temp      ;Adiciona o valor da variável temp à variável final
        lsr     temp1           ;Desloca os bits da variável temp1 para a direita
        mov     temp,temp1      ;Copia o valor da variável temp para a variável temp1
        andi    temp,0x08       ;Verifica o valor do quarto bit ("AND" com 00001000b)
        add     final,temp      ;Adiciona o valor da variável temp à variável final
        cpi     count,1         ;Compara count com 1 para saber se apenas a primeira parte do byte foi decodificado
        breq    Parte1          ;Pula para o bloco Parte 1 se verdadeiro
        rjmp    Parte2          ;Pula para o bloco Parte 2 se já tiver sido decodificado o byte inteiro

Parte1:
```

```
        mov     vfinal,final            ;Armazena a primeira parte do byte decodificado na variável vfinal
        rjmp    Convertendo             ;Pula para o bloco Convertendo para continuar com a decodificação dos dados

Parte2:
        swap    final                   ;Inverte a posição dos últimos bits pelos primeiros
        add     vfinal,final            ;Adiciona a segunda parte do byte decodificado à primeira parte
        mov     ponteiro1,XL            ;Armazena a posição do ponteiro na variável ponteiro1
        cpi     ponteiro1,0x62          ;Verifica se o primeiro byte recebido já foi decodificado e armazenado
        brsh    Pula                    ;Pula para o bloco Pula se verdadeiro
        ldi     XL,0x80                 ;Posiciona o ponteiro
        mov     ponteiro2,XL            ;Armazena o valor do ponteiro na variável ponteiro2
        st      X,vfinal                ;Armazena o dado decodificado na memória
        clr     count                   ;Limpa a variável count
        clr     vfinal                  ;Limpa a variável vfinal
        mov     XL,ponteiro1            ;Posiciona o ponteiro
        rjmp    Convertendo             ;Pula para o bloco Convertendo

Pula:
        inc     ponteiro2               ;Posiciona o ponteiro
        mov     XL,ponteiro2            ;Armazena o valor do ponteiro na variável ponteiro2
        st      X,vfinal                ;Armazena o dado na memória
        clr     count                   ;Limpa a variável count
        clr     vfinal                  ;Limpa a variável vfinal
        cpi     ponteiro2,0x84          ;Verifica se todos os dados recebidos foram decodificados e armazenados
        breq    Aciona                  ;Pula para o bloco Aciona se verdadeiro
        mov     XL,ponteiro1            ;Posiciona o ponteiro
        rjmp    Convertendo             ;Pula para o bloco Convertendo

;** Bloco de acionamento dos motores
Aciona:
        ldi     XL,0x80                 ;Posiciona o ponteiro
        ld      temp,X                  ;Copia o dado armazenado na variável temp
        out     PORTB,temp              ;Manda para o PortB o comando de direção dos motores
        inc     XL                      ;Incrementa o ponteiro
        ld      temp,X                  ;Copia o dado armazenado na variável temp
        out     OCR1AL,temp             ;Configura a velocidade do PWM A de acordo com o valor recebido
        inc     XL                      ;Incrementa o ponteiro
        ld      temp,X                  ;Copia o dado armazenado na variável temp
        out     OCR1BL,temp             ;Configura a velocidade do PWM B de acordo com o valor recebido
        rjmp    Leitura                 ;Pula para o bloco Leitura

Retornando2:
        reti                            ;Retorna da interrupção

;** Loop de Atraso
Atraso_P1:
        inc     count                   ;Incrementa o valor da variável count
        ldi     XL,0x6F                 ;Posiciona o ponteiro
        cpi     count,20                ;Compara para verificar se já enviou o mesmo dado 20 vezes
        brlo    Retornando2             ;Retorna da interrupção caso negativo

;** Inicia o loop de atraso
Atraso_P2:
        ldi     ponteiro1,255           ;Carrega 255 na variável ponteiro1
        ldi     ponteiro2,2             ;Carrega 2 na variável ponteiro2

Loop:
        dec     ponteiro1               ;Decrementa a variável ponteiro1
        brne    loop                    ;Pula para Loop caso não esteja zerado a variável ponteiro1
        dec     ponteiro2               ;Decrementa a variável ponteiro2
        brne    loop                    ;Pula para Loop caso não esteja zerado a variável ponteiro2
        cli                             ;Desabilita a interrupção
        rjmp    RESET                   ;Pula para o bloco RESET para reiniciar todo o processo

;** Bloco que armazena os valores de velocidade e direção dos motores
Leitura:
        cli                             ;Desabilita a interrupção
        ldi     temp,0x00               ;Copia o valor 00h para a variável temp
        out     PORTC,temp              ;Liga os sensores de velocidade
        clr     final                   ;Limpa a variável final
        clr     temp                    ;Limpa a variável temp
        clr     temp1                   ;Limpa avariável temp1
        clr     temp2                   ;Limpa a variável temp2
        clr     temp3                   ;Limpa a variável temp3
        ldi     XL,0x83                 ;Posiciona o ponteiro
        ld      temp,X                  ;Copia o dado armazenado na variável temp (motor direito)
```

```
            ldi     XL,0x84             ;Posiciona o ponteiro
            ld      temp1,X             ;Copia o dado armazenado na variável temp1 (motor esquerdo)
            ldi     XL,0x81             ;Posiciona o ponteiro
            ld      r24,X               ;Armazena a velocidade do PWM A (Motor esquerdo) no registrador r24
            inc     XL                  ;Incrementa o ponteiro
            ld      r25,X               ;Armazena a velocidade do PWM B (Motor direito) no registrador r25
            ldi     r22,0xFE            ;Copia o valor FEh para o registrador r22
            rjmp    Espera_2b           ;Pula para o bloco Espera_2b

;Variáveis relacionadas com cada um dos motores:
            ;Motor direito:  temp,  t2, r25 e OCR1BL
            ;Motor esquerdo: temp1, t3, r24 e OCR1AL

;** Bloco que compara a leitura atual dos pulsos de cada um dos sensores de velocidade
Compara_t2_t3:
            cp      temp2,temp3
            brlo    t2_menor_t3         ;Pula para o bloco t2_menor_t3, caso o sensor de velocidade
                                        ;do lado direito tenha detectado menos pulsos que o esquerdo
            cp      temp3,temp2
            brlo    t3_menor_t2         ;Pula para o bloco t3_menor_t2, caso o sensor de velocidade
                                        ;do lado esquerdo tenha detectado menos pulsos que o direito
            cpi     r21,1
            breq    Espera_1aa          ;Pula para a posição de leitura dos sensores
            cpi     r21,2
            breq    Espera_2aa          ;Pula para a posição de leitura dos sensores
            cpi     r21,3
            breq    Espera_1bb          ;Pula para a posição de leitura dos sensores
            cpi     r21,4
            breq    Espera_2bb          ;Pula para a posição de leitura dos sensores

t2_menor_t3:
            cpi     r22,0xFF
            breq    Pula1               ;Caso o motor esquerdo já esteja desligado, pula para o bloco Pula1 e não faz nada
            cpi     r25,30
            brsh    Mdireito_maior_30   ;Caso o motor direito não esteja com velocidade máxima, pula
                                        ;para o bloco que aumenta a velocidade em aproximadamente 10%
            cpi     r24,200
            brsh    Pula1               ;Caso o motor esquerdo já esteja com uma velocidade baixa, cerca
                                        ;de 20% do total, pula para o bloco Pula1 e não faz nada
            ldi     r20,30              ;Carrega 30 no registrador r20
            add     r24,r20             ;Adiciona 30 ao registrador r24
            out     OCR1AL,r24          ;Reduz a velocidade do motor esquerdo em aproximadamente 10%

Pula1:
            cpi     r21,1
            breq    Espera_1a           ;Pula para a posição de leitura dos sensores
            cpi     r21,2
            breq    Espera_2a           ;Pula para a posição de leitura dos sensores
            cpi     r21,3
            breq    Espera_1bb          ;Pula para a posição de leitura dos sensores
            cpi     r21,4
            breq    Espera_2bb          ;Pula para a posição de leitura dos sensores

Mdireito_maior_30:
            subi    r25,30              ;Subtrai 30 do registrador r25
            out     OCR1BL,r25          ;Aumenta a velocidade do motor direito em aproximadamente 10%
            cpi     r21,1
            breq    Espera_1a           ;Pula para a posição de leitura dos sensores
            cpi     r21,2
            breq    Espera_2a           ;Pula para a posição de leitura dos sensores
            cpi     r21,3
            breq    Espera_1b           ;Pula para a posição de leitura dos sensores
            cpi     r21,4
            breq    Espera_2bb          ;Pula para a posição de leitura dos sensores

t3_menor_t2:
            cpi     r22,0xFF
            breq    Pula2               ;Caso o motor direito já esteja desligado, pula para o bloco Pula2 e não faz nada
            cpi     r24,30
            brsh    Mesquerdo_maior_30  ;Caso o motor esquerdo não esteja com velocidade máxima, pula
                                        ;para o bloco que aumenta a velocidade em aproximadamente 10%
            cpi     r25,200
            brsh    Pula2               ;Caso o motor direito já esteja com uma velocidade baixa, cerca
                                        ;de 20% do total, pula para o bloco Pula2 e não faz nada
            ldi     r20,30              ;Carrega 30 no registrador r20
            add     r25,r20             ;Adiciona 30 ao registrador r25
```

```
            out     OCR1BL,r25          ;Reduz a velocidade do motor direito em aproximadamente 10%

Pula2:
            cpi     r21,1
            breq    Espera_1a           ;Pula para a posição de leitura dos sensores
            cpi     r21,2
            breq    Espera_2a           ;Pula para a posição de leitura dos sensores
            cpi     r21,3
            breq    Espera_1b           ;Pula para a posição de leitura dos sensores
            cpi     r21,4
            breq    Espera_2b           ;Pula para a posição de leitura dos sensores

Mesquerdo_maior_30:
            subi    r24,30              ;Subtrai 30 do registrador r24
            out     OCR1AL,r24          ;Aumenta a velocidade do motor esquerdo em aproximadamente 10%
            cpi     r21,1
            breq    Espera_1a           ;Pula para a posição de leitura dos sensores
            cpi     r21,2
            breq    Espera_2a           ;Pula para a posição de leitura dos sensores
            cpi     r21,3
            breq    Espera_1b           ;Pula para a posição de leitura dos sensores
            cpi     r21,4
            breq    Espera_2b           ;Pula para a posição de leitura dos sensores

Espera_1aa:
            rjmp    Espera_1a           ;Pula para a posição de leitura dos sensores
Espera_2aa:
            rjmp    Espera_2a           ;Pula para a posição de leitura dos sensores
Espera_1bb:
            rjmp    Espera_1b           ;Pula para a posição de leitura dos sensores
Espera_2bb:
            rjmp    Espera_2b           ;Pula para a posição de leitura dos sensores

;** Bloco de leitura dos sensores de velocidade
;** Situação inicial
Detecta:
            sbis    PINB,7              ;Ímã detectado?
            rjmp    Incremento_1a       ;Sim, então desvia para incremento
            sbis    PINB,6              ;Ímã detectado?
            rjmp    Incremento_2a       ;Sim, então desvia para incremento
            rjmp    Detecta             ;Fica nesse loop até detectar um ímã

;** Situação: (7 é setado, 6 não é setado)
Incremento_1a:
            inc     temp2               ;Incrementa a variável temp2
            ldi     r21,1               ;Carrega 1 no registrador r21
            rjmp    Compara_t2_t3       ;Pula para o bloco que faz a compensação das velocidades
Espera_1a:
            cp      temp,temp2          ;Compara para ver se a quantidade de pulsos desejado já foi atingida
            breq    Desabilita_1a       ;Pula para o bloco Desabilita_1a em caso positivo
            sbis    PINB,6              ;Ímã detectado?
            rjmp    Incremento_2b       ;Sim, então desvia para incremento
            sbis    PINB,7              ;Ímã ainda sendo detectado?
            rjmp    Espera_1a           ;Continua verificando até que o ímã do lado direito deixe de ser detectado
            rjmp    Detecta             ;Pula para o bloco Detecta em caso positivo

;** Situação: (7 não é setado, 6 é setado)
Incremento_2a:
            inc     temp3               ;Incrementa a variável temp3
            ldi     r21,2               ;Carrega 2 no registrador r21
            rjmp    Compara_t2_t3       ;Pula para o bloco que faz a compensação das velocidades
Espera_2a:
            cp      temp1,temp3         ;Compara para ver se a quantidade de pulsos desejado já foi atingida
            breq    Desabilita_2a       ;Pula para o bloco Desabilita_2a em caso positivo
            sbis    PINB,7              ;Ímã detectado?
            rjmp    Incremento_1b       ;Sim, então desvia para incremento
            sbis    PINB,6              ;Ímã ainda sendo detectado?
            rjmp    Espera_2a           ;Continua verificando até que o ímã do lado esquerdo deixe de ser detectado
            rjmp    Detecta             ;Pula para o bloco Detecta em caso positivo

;** Situação: (7 é setado, 6 está setado)
Incremento_1b:
            inc     temp2               ;Incrementa a variável temp2
            ldi     r21,3               ;Carrega 3 no registrador r21
            rjmp    Compara_t2_t3       ;Pula para o bloco que faz a compensação das velocidades
Espera_1b:
```

```
            cp      temp,temp2          ;Compara para ver se a quantidade de pulsos desejado já foi atingida
            breq    Desabilita_1b       ;Pula para o bloco Desabilita_1b em caso positivo
            sbic    PINB,6              ;Ímã ainda sendo detectado?
            rjmp    Espera_1a           ;Sim, então desvia para Espera_1a
            sbis    PINB,7              ;Ímã ainda sendo detectado?
            rjmp    Espera_1b           ;Continua verificando até que o ímã do lado direito deixe de ser detectado
            rjmp    Espera_2a           ;Pula para o bloco Espera_2b em caso positivo

;** Situação: (7 está setado, 6 é setado)
Incremento_2b:
            inc     temp3               ;Incrementa a variável temp3
            ldi     r21,4               ;Carrega 4 no registrador r21
            rjmp    Compara_t2_t3       ;Pula para o bloco que faz a compensação das velocidades
Espera_2b:
            cp      temp1,temp3         ;Compara para ver se a quantidade de pulsos desejado já foi atingida
            breq    Desabilita_2b       ;Pula para o bloco Desabilita_2b em caso positivo
            sbic    PINB,7              ;Ímã ainda sendo detectado?
            rjmp    Espera_2a           ;Sim, então desvia para Espera_2a
            sbis    PINB,6              ;Ímã ainda sendo detectado?
            rjmp    Espera_2b           ;Continua verificando até que o ímã do lado esquerdo deixe de ser detectado
            rjmp    Espera_1a           ;Pula para o bloco Espera_1a em caso positivo

;** Bloco para zerar a velocidade dos motores
Desabilita_1a:
            out     OCR1BL,r22          ;Configura a velocidade do PWM B para zero
            cp      temp3,temp1
            brsh    Desabilita_1_dec    ;Pula para o bloco que desabilita os dois motores, caso ambos
                                        ;tenham atingido a quantidade de pulsos desejada
            inc     temp2               ;Incrementa a variável temp2
            inc     r22                 ;Incrementa o registrador r22
            ldi     r25,0xFF            ;Armazena FF no registrador r25
            rjmp    Espera_1a           ;Pula para a posição de leitura dos sensores

Desabilita_1b:
            out     OCR1BL,r22          ;Configura a velocidade do PWM B para zero
            cp      temp3,temp1
            brsh    Desabilita_1_dec    ;Pula para o bloco que desabilita os dois motores, caso ambos
                                        ;tenham atingido a quantidade de pulsos desejada
            inc     temp2               ;Incrementa a variável temp2
            inc     r22                 ;Incrementa o registrador r22
            ldi     r25,0xFF            ;Armazena FF no registrador r25
            rjmp    Espera_1b           ;Pula para a posição de leitura dos sensores

Desabilita_2a:
            out     OCR1AL,r22          ;Configura a velocidade do PWM A para zero
            cp      temp2,temp
            brsh    Desabilita_2_dec    ;Pula para o bloco que desabilita os dois motores, caso ambos
                                        ;tenham atingido a quantidade de pulsos desejada
            inc     temp3               ;Incrementa a variável temp3
            inc     r22                 ;Incrementa o registrador r22
            ldi     r24,0xFF            ;Armazena FF no registrador r24
            rjmp    Espera_2a           ;Pula para a posição de leitura dos sensores

Desabilita_2b:
            out     OCR1AL,r22          ;Configura a velocidade do PWM A para zero
            cp      temp2,temp
            brsh    Desabilita_2_dec    ;Pula para o bloco que desabilita os dois motores, caso ambos
                                        ;tenham atingido a quantidade de pulsos desejada
            inc     temp3               ;Incrementa a variável temp3
            inc     r22                 ;Incrementa o registrador r22
            ldi     r24,0xFF            ;Armazena FF no registrador r24
            rjmp    Espera_2b           ;Pula para a posição de leitura dos sensores

Desabilita_1_dec:
            cpi     r22,0xFE
            breq    Desabilita          ;Verifica se o sensor esquerdo já havia detectado a quantidade de pulsos desejada
            dec     temp3               ;Decrementa a variável temp3
            rjmp    Desabilita          ;Pula para o bloco que vai desabilitar o funcionamento dos motores, mas continuará
                                        ;fazendo a leitura dos sensores de velocidade até a parada completa dos motores
Desabilita_2_dec:
            cpi     r22,0xFE
            breq    Desabilita          ;Verifica se o sensor direito já havia detectado a quantidade de pulsos desejada
            dec     temp2               ;Decrementa a variável temp2
            rjmp    Desabilita          ;Pula para o bloco que vai desabilitar o funcionamento dos motores, mas continuará
                                        ;fazendo a leitura dos sensores de velocidade até a parada completa dos motores
```

```
;** Bloco para desabilitar o funcionamento dos motores
Desabilita:
        ldi     temp,0x00
        out     PORTB,temp              ;Manda para o PortB o comando de direção dos motores

;** Bloco que continua fazendo a detecção dos ímãs
        ldi     temp,255                ;Carrega 255 na variável temp
        ldi     ponteiro1,0             ;Zera a variável ponteiro1
        ldi     ponteiro2,0             ;Zera a variável ponteiro2
        ldi     final,0                 ;Zera a variável final

;** Situação inicial
Detecta_continua:
        inc     ponteiro1               ;Incrementa a variável ponteiro1
        cpi     ponteiro1,255           ;Compara ponteiro1 com 255
        breq    D_cont_ponteiro         ;Pula para o bloco que verifica se já passou tempo suficiente
                                        ;para que os motores tenham parado, caso ponteiro1 seja igual a 255
        sbic    PINB,7                  ;Ímã detectado?
        rjmp    Inc_1a_continua         ;Sim, então desvia para incremento
        sbic    PINB,6                  ;Ímã detectado?
        rjmp    Inc_2a_continua         ;Sim, então desvia para incremento
        rjmp    Detecta_continua        ;Fica nesse loop até detectar um ímã ou o tempo estourar

;** Bloco que verifica se o tempo de espera de parada dos motores já foi atingido
D_cont_ponteiro:
        inc     ponteiro2               ;Incrementa a variável ponteiro2
        cpi     ponteiro2,255           ;Compara ponteiro2 com 255
        breq    D_cont_ponteiro2        ;Pula para o bloco que verifica se já passou tempo suficiente
                                        ;para que os motores tenham parado, caso ponteiro2 seja igual a 255
        rjmp    Detecta_continua        ;Pula para o bloco Detecta_continua, caso o tempo não tenha estourado

D_cont_ponteiro2:
        inc     final                   ;Incrementa a variável final
        cpi     final,10                ;Compara final com 10
        breq    Guardar_fim_pulo        ;Pula para o bloco que desabilita os sensores de velocidade
                                        ;pois o tempo de parada dos motores já foi alcançado (final = 10)
        rjmp    Detecta_continua        ;Pula para o bloco Detecta_continua, caso o tempo não tenha estourado

Guardar_fim_pulo:
        rjmp    Guardar_fim

;** Situação: (7 é setado, 6 não é setado)
Inc_1a_continua:
        inc     temp2                   ;Incrementa a variável temp2
        ldi     ponteiro1,0             ;Zera a variável ponteiro1
        ldi     ponteiro2,0             ;Zera a variável ponteiro2
        ldi     final,0                 ;Zera a variável final
E_1a_continua:
        inc     ponteiro1               ;Incrementa a variável ponteiro1
        cpi     ponteiro1,255           ;Compara ponteiro1 com 255
        breq    E_1a_cont_ponteiro      ;Pula para o bloco que verifica se já passou tempo suficiente
                                        ;para que os motores tenham parado, caso ponteiro1 seja igual a 255
        sbic    PINB,6                  ;Ímã detectado?
        rjmp    Inc_2b_continua         ;Sim, então desvia para incremento
        sbic    PINB,7                  ;Ímã ainda sendo detectado?
        rjmp    E_1a_continua           ;Continua verificando até que o ímã deixe de ser detectado, ou o tempo estoure
        ldi     ponteiro1,0             ;Zera a variável ponteiro1
        ldi     ponteiro2,0             ;Zera a variável ponteiro2
        ldi     final,0                 ;Zera a variável final
        rjmp    Detecta_continua        ;Pula para o bloco Detecta_continua, caso o tempo não tenha estourado

;** Situação: (7 não é setado, 6 é setado)
Inc_2a_continua:
        inc     temp3                   ;Incrementa a variável temp3
        ldi     ponteiro1,0             ;Zera a variável ponteiro1
        ldi     ponteiro2,0             ;Zera a variável ponteiro2
        ldi     final,0                 ;Zera a variável final
E_2a_continua:
        inc     ponteiro1               ;Incrementa a variável ponteiro1
        cpi     ponteiro1,255           ;Compara ponteiro1 com 255
        breq    E_2a_cont_ponteiro      ;Pula para o bloco que verifica se já passou tempo suficiente
                                        ;para que os motores tenham parado, caso ponteiro1 seja igual a 255
        sbic    PINB,7                  ;Ímã detectado?
        rjmp    Inc_1b_continua         ;Sim, então desvia para incremento
        sbic    PINB,6                  ;Ímã ainda sendo detectado?
        rjmp    E_2a_continua           ;Continua verificando até que o ímã deixe de ser detectado, ou o tempo estoure
```

```
            ldi         ponteiro1,0             ;Zera a variável ponteiro1
            ldi         ponteiro2,0             ;Zera a variável ponteiro2
            ldi         final,0                 ;Zera a variável final
            rjmp        Detecta_continua        ;Pula para o bloco Detecta_continua, caso o tempo não tenha estourado

;** Situação: (7 é setado, 6 está setado)
Inc_1b_continua:
            inc         temp2                   ;Incrementa a variável temp2
            ldi         ponteiro1,0             ;Zera a variável ponteiro1
            ldi         ponteiro2,0             ;Zera a variável ponteiro2
            ldi         final,0                 ;Zera a variável final
E_1b_continua:
            inc         ponteiro1               ;Incrementa a variável ponteiro1
            cpi         ponteiro1,255           ;Compara ponteiro1 com 255
            breq        E_1b_cont_ponteiro      ;Pula para o bloco que verifica se já passou tempo suficiente
                                                ;para que os motores tenham parado, caso ponteiro1 seja igual a 255
            sbis        PINB,6                  ;Ímã ainda sendo detectado?
            rjmp        E_1a_continua           ;Sim, então desvia para o bloco E_1a_continua
            sbic        PINB,7                  ;Ímã ainda sendo detectado?
            rjmp        E_1b_continua           ;Continua verificando até que o ímã deixe de ser detectado, ou o tempo estoure
            ldi         ponteiro1,0             ;Zera a variável ponteiro1
            ldi         ponteiro2,0             ;Zera a variável ponteiro2
            ldi         final,0                 ;Zera a variável final
            rjmp        E_2a_continua           ;Pula para o bloco E_2a_continua, caso o tempo não tenha
                                                ;estourado e o ímã não esteja mais sendo detectado
;** Situação: (7 está setado, 6 é setado)
Inc_2b_continua:
            inc         temp3                   ;Incrementa a variável temp
            ldi         ponteiro1,0             ;Zera a variável ponteiro1
            ldi         ponteiro2,0             ;Zera a variável ponteiro2
            ldi         final,0                 ;Zera a variável final
E_2b_continua:
            inc         ponteiro1               ;Incrementa a variável ponteiro1
            cpi         ponteiro1,255           ;Compara ponteiro1 com 255
            breq        E_2b_cont_ponteiro      ;Pula para o bloco que verifica se já passou tempo suficiente
                                                ;para que os motores tenham parado, caso ponteiro1 seja igual a 255
            sbis        PINB,7                  ;Ímã ainda sendo detectado?
            rjmp        E_2a_continua           ;Sim, então desvia para o bloco E_2a_continua
            sbic        PINB,6                  ;Ímã ainda sendo detectado?
            rjmp        E_2b_continua           ;Continua verificando até que o ímã deixe de ser detectado, ou o tempo estoure
            ldi         ponteiro1,0             ;Zera a variável ponteiro1
            ldi         ponteiro2,0             ;Zera a variável ponteiro2
            ldi         final,0                 ;Zera a variável final
            rjmp        E_1a_continua           ;Pula para o bloco E_1a_continua, caso o tempo não tenha
                                                ;estourado e o ímã não esteja mais sendo detectado
;** Bloco que verifica se o tempo de espera de parada dos motores já foi atingido
E_1a_cont_ponteiro:
            inc         ponteiro2               ;Incrementa a variável ponteiro2
            cpi         ponteiro2,255           ;Compara a variável ponteiro2 com 255
            breq        E_1a_cont_ponteiro2     ;Pula para o bloco que verifica se já passou tempo suficiente
                                                ;para que os motores tenham parado, caso ponteiro2 seja igual a 255
            rjmp        E_1a_continua           ;Pula para o bloco E_1a_continua, caso o tempo não tenha estourado

E_1a_cont_ponteiro2:
            inc         final                   ;Incrementa a variável final
            cpi         final,10                ;Compara a variável final com 10
            breq        Guardar_fim             ;Pula para o bloco que desabilita os sensores de velocidade
                                                ;pois o tempo de parada dos motores já foi alcançado (final = 10)
            rjmp        E_1a_continua           ;Pula para o bloco E_1a_continua, caso o tempo não tenha estourado

E_1b_cont_ponteiro:
            inc         ponteiro2               ;Incrementa a variável ponteiro2
            cpi         ponteiro2,255           ;Compara a variável ponteiro2 com 255
            breq        E_1b_cont_ponteiro2     ;Pula para o bloco que verifica se já passou tempo suficiente
                                                ;para que os motores tenham parado, caso ponteiro2 seja igual a 255
            rjmp        E_1b_continua           ;Pula para o bloco E_1b_continua, caso o tempo não tenha estourado

E_1b_cont_ponteiro2:
            inc         final                   ;Incrementa a variável final
            cpi         final,10                ;Compara a variável final com 10
            breq        Guardar_fim             ;Pula para o bloco que desabilita os sensores de velocidade
                                                ;pois o tempo de parada dos motores já foi alcançado (final = 10)
            rjmp        E_1b_continua           ;Pula para o bloco E_1b_continua, caso o tempo não tenha estourado

E_2a_cont_ponteiro:
            inc         ponteiro2               ;Incrementa a variável ponteiro2
```

```
            cpi     ponteiro2,255       ;Compara a variável ponteiro2 com 255
            breq    E_2a_cont_ponteiro2 ;Pula para o bloco que verifica se já passou tempo suficiente
                                        ;para que os motores tenham parado, caso ponteiro2 seja igual a 255
            rjmp    E_2a_continua       ;Pula para o bloco E_2a_continua, caso o tempo não tenha estourado\

E_2a_cont_ponteiro2:
            inc     final               ;Incrementa a variável final
            cpi     final,10            ;Compara a variável final com 10
            breq    Guardar_fim         ;Pula para o bloco que desabilita os sensores de velocidade
                                        ;pois o tempo de parada dos motores já foi alcançado (final = 10)
            rjmp    E_2a_continua       ;Pula para o bloco E_2a_continua, caso o tempo não tenha estourado

E_2b_cont_ponteiro:
            inc     ponteiro2           ;Incrementa a variável ponteiro2
            cpi     ponteiro2,255       ;Compara a variável ponteiro2 com 255
            breq    E_2b_cont_ponteiro2 ;Pula para o bloco que verifica se já passou tempo suficiente
                                        ;para que os motores tenham parado, caso ponteiro2 seja igual a 255
            rjmp    E_2b_continua       ;Pula para o bloco E_2b_continua, caso o tempo não tenha estourado

E_2b_cont_ponteiro2:
            inc     final               ;Incrementa a variável final
            cpi     final,10            ;Compara a variável final com 10
            breq    Guardar_fim         ;Pula para o bloco que desabilita os sensores de velocidade
                                        ;pois o tempo de parada dos motores já foi alcançado (final = 10)
            rjmp    E_2b_continua       ;Pula para o bloco E_2b_continua, caso o tempo não tenha estourado

;** Bloco que desabilita os sensores de velocidade
Guardar_fim:
            ldi     temp,0xFF           ;Copia o valor FFh para a variável temp
            out     PORTC,temp          ;Corta a alimentação dos sensores de velocidade
            ldi     XL,0x70             ;Posiciona o ponteiro
            ldi     temp,0xCC           ;Armazena o valor 11001100b na variável temp
            st      X,temp              ;Armazena o mesmo valor na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp              ;Armazena o mesmo valor na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp2             ;Armazena o valor da variável temp2 na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp2             ;Armazena o mesmo valor na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp3             ;Armazena o valor da variável temp3 na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp3             ;Armazena o mesmo valor na memória

;** Bloco de contagem do tempo em nível alto do pulso de saída do Sensor
Sonar:
            clr     temp                ;Limpa a variável temp
            out     TCCR1A,temp         ;Zera a contagem do tempo
            out     TCCR1B,temp         ;Zera a contagem do tempo
            out     TCNT1H,temp         ;Limpa o registrador de 16 bits
            out     TCNT1L,temp         ;Limpa o registrador de 16 bits

Verifica1:
            sbis    PINB,4              ;Pula a próxima instrução se o pino 4 do Port B estiver em nível alto
            rjmp    Verifica1           ;Aguarda até que o pino 4 do Port B fique em nível alto
                                        ;(caso esteja em nível baixo, para iniciar a contagem do tempo)
            ldi     r16,(0<<CS11)|(1<<CS10)
            out     TCCR1B,r16          ;Ajuste do prescaler 1:1 (Tempo = Clock/1)

Verifica2:
            sbis    PINB,4              ;Pula a próxima instrução se o pino 4 do Port B estiver em nível alto
            rjmp    PararCont           ;Pula para o bloco PararCont assim que o pino 4 do Port B fique em nível baixo
            rjmp    Verifica2           ;Aguarda até que o pino 4 do Port B fique em nível baixo

;** Bloco para parar a contagem do Timer
PararCont:
            ldi     r16,(0<<CS11)|(0<<CS10)
            out     TCCR1B,r16          ;Para o Timer
            in      temp1,TCNT1L        ;Lê o valor do byte menos significativo do registrador de 16 bits
                                        ;e armazena na variável temp1
            in      temp2,TCNT1H        ;Lê o valor do byte mais significativo do registrador de 16 bits
                                        ;e armazena na variável temp2
;** Bloco de armazenamento das informações a serem transmitidas
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp1             ;Armazena o valor da variável temp1 na memória
            inc     XL                  ;Incrementa o ponteiro
```

```
            st      X,temp1             ;Armazena o mesmo valor na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp2             ;Armazena o valor da variável temp2 na memória
            inc     XL                  ;Incrementa o ponteiro
            st      X,temp2             ;Armazena o mesmo valor na memória
            clr     temp                ;Limpa a variável temp
            out     TCNT1H,temp         ;Limpa o registrador de 16 bits
            out     TCNT1L,temp         ;Limpa o registrador de 16 bits

Posicionando:
            clr     count               ;Limpa a variável count
            ldi     XL,0x72             ;Posiciona o ponteiro
            ld      temp,X              ;Carrega o dado na variável temp

;** Bloco de codificação dos dados
Codificando_P1:
            clr     temp2               ;Limpa a variável temp2
            inc     count               ;Incrementa a variável count
            cpi     count,6             ;Compara para saber se todos os dados foram codificados
            breq    Transmitir          ;Caso positivo pula para o bloco Transmitir
            andi    temp,0x0F           ;Codifica apenas os quatro primeiros bits ("AND" com 00001111b)

Codificando_P2:
            clr     final               ;Limpa a variável final
            mov     temp1,temp          ;Copia o valor da variável temp para a variável temp1
            com     temp                ;Faz o complemento da variável temp

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            mov     ponteiro1,temp      ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1     ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x01      ;Codifica o primeiro bit ("AND" com 00000001b)
            andi    ponteiro1,0x02      ;Codifica o primeiro bit ("AND" com 00000010b)
            add     ponteiro1,ponteiro2 ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1     ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            lsl     temp1               ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp      ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1     ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x04      ;Codifica o segundo bit ("AND" com 00000100b)
            andi    ponteiro1,0x08      ;Codifica o segundo bit ("AND" com 00001000b)
            add     ponteiro1,ponteiro2 ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1     ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            lsl     temp1               ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp      ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1     ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x10      ;Codifica o terceiro bit ("AND" com 00010000b)
            andi    ponteiro1,0x20      ;Codifica o terceiro bit ("AND" com 00100000b)
            add     ponteiro1,ponteiro2 ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1     ;Adiciona o valor da variável ponteiro1 à variável final

            lsl     temp                ;Desloca os bits da variável temp para a esquerda
            lsl     temp1               ;Desloca os bits da variável temp1 para a esquerda
            mov     ponteiro1,temp      ;Copia o valor da variável temp para a variável ponteiro1
            mov     ponteiro2,temp1     ;Copia o valor da variável temp1 para a variável ponteiro2
            andi    ponteiro2,0x40      ;Codifica o quarto bit ("AND" com 01000000b)
            andi    ponteiro1,0x80      ;Codifica o quarto bit ("AND" com 10000000b)
            add     ponteiro1,ponteiro2 ;Adiciona o valor da variável ponteiro2 à variável ponteiro1
            add     final,ponteiro1     ;Adiciona o valor da variável ponteiro1 à variável final

;** Bloco de verificação se o byte inteiro foi codificado
            st      X,final             ;Armazena o dado na memória
            inc     XL                  ;Incrementa o ponteiro
            ld      temp,X              ;Lê os dados e armazena na variável temp
            cpi     temp2,0x01          ;Compara se a variável temp2 é igual a 1
            breq    Codificando_P1      ;Pula para o bloco Codificando_P1 se o byte inteiro foi codificado
            andi    temp,0xF0           ;Codifica apenas os quatro últimos bits ("AND" com 11110000b)
            swap    temp                ;Troca de posição os quatro últimos bits pelos quatro primeiros
            ldi     temp2,0x01          ;Carrega 1 na variável temp2
            rjmp    Codificando_P2      ;Pula para o bloco que faz a codificação dos dados

;** Bloco de configuração para iniciar a transmissão dos dados codificados
Transmitir:
            clr     count               ;Limpa a variável count
```

```
        ldi     XL,0x6D                 ;Posiciona o ponteiro
        ldi     temp,0b00101000
        out     UCR,temp                ;Habilita transmissão e interrupção por transmissão
        sei                             ;Habilita a interrupção
        rjmp    Volta                   ;Pula para o bloco Volta e fica aguardando a próxima Interrupção

RESET:
;** Programa Principal

;** Inicialização de Stack Pointer
        ldi     temp,low(RAMEND)
        out     SPL,temp
        ldi     temp,high(RAMEND)
        out     SPH,temp
        ldi     temp,0b00100000
        out     DDRD,temp               ;Pino OC1A definido como saída demais bits do PortD configurados como entrada
        ldi     temp,0b00101111
        out     DDRB,temp               ;Pinos 0 à 3 definidos como saída para controle dos motores
                                        ;Pino 5 definido como saída para controle do sensor de ultrasom
                                        ;Pino 4 definido como entrada para recebimento do sinal do sensor de ultrasom
                                        ;Pinos 6 e 7 definidos como entrada para recebimento do sinal
                                        ;dos sensores de velocidade
        ldi     temp,0b00000011
        out     DDRC,temp               ;Pinos 0 e 1 definidos como saída para controle dos sensores de
                                        ;velocidade, demais pinos configurados como entrada
;** Inicialização do Vetor de Dados
        ldi     XH,0x00

;** Configuração do PWM (A e B)
        ldi     temp,0b11110001
        out     TCCR1A,temp             ;Configuração do modo PWM 8 bits e seleção da seguinte ação para os pinos OC1B
                                        ;e OC1A (PD5):
                                        ;Clear quando o valor de comparação atingido na contagem crescente, seta quando o
                                        ;valor de comparação atingido na contagem decrescente
                                        ;O pino OC1B é a saída PWM para o motor 2
                                        ;O pino OC1A (PD5) é a saída PWM para o motor 1
        ldi     temp,0b00000001
        out     TCCR1B,temp             ;Libera timer1, incrementado pelo clock sem prescaler
                                        ;Portanto freqüência do PWM = freqüência do clock
        ldi     temp,0                  ;Zera o conteúdo dos registradores altos
        out     OCR1AH,temp             ;Referências para o PWM
        out     OCR1BH,temp
        ldi     temp,255                ;Força a velocidade inicial = 0
        out     OCR1AL,temp
        out     OCR1BL,temp

;** Programação da UART
        ldi     temp,103
        out     UBRR,temp               ;Ajuste da freqüência Baud Rate = 2400Hz (Tabela)

DeNovo:
        cli                             ;Desabilita a interrupção
        ldi     temp,0b10010000
        out     UCR,temp                ;Habilita recepção e interrupção por recepção completa de dados
        ldi     temp3,1                 ;Carrega 1 na variável temp3
        clr     count                   ;Limpa a variável count
        clr     temp                    ;Limpa a variável temp
        ldi     XL,0x5F                 ;Posiciona o ponteiro
        sei                             ;Habilita a interrupção

Volta:
        rjmp    Volta                   ;Fica nele mesmo até receber uma interrupção

;** FIM DO PROGRAMA
```

11 REFERÊNCIAS

[1] HAYES, A. T., MARTINOLI, A. e GOODMAN, R. M., **"Distributed Odor Source Localization",** IEEE Sensors Journal, Vol. 2, No. 3, Junho, 2002.

[2] KAWAGUCHI, Y., YOSHIDA, I., KURUMATANI, H., KIKUTA, T. e YAMADA, Y., **"Internal Pipe Inspection Robot"**, Proceedings of IEEE International Conferences on Robotics and Automation, Nagoya, Japan, Vol. 1, p. 857 – 862, 1995.

[3] DUCKETT, T., AXELSSON, M., SAFFIOTTI, A., **"Learning to Locate an Odour Source with a Mobile Robot"**, Proceedings of International Conferences on Robotics and Automation, Vol. 4, p. 4017 – 4022, 2001.

[4] LILIENTHAL, A., DUCKETT, T., **"Gas Source Localisation by Constructing Concentration Gridmaps with a Mobile Robot"**, Proceedings of the European Conference on Mobile Robots, 2003.

[5] MONKMAN, G. J. e TAYLOR, P. M., **"Thermal Tactile Sensing",** IEEE Transactions on Robotics and Automation, Vol. 9, No. 3, Junho, 1993.

[6] RUSSEL, R. A., **"Heat Trails as Short-Lived Navigational Markers for Mobile Robots",** Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, Abril, 1997.

[7] WAGNER, I. A., LINDENBAUM, M. e BRUCKSTEIN, A. M., **"Distributed Covering by Ant-Robots Using Evaporating Traces"**, IEEE Transactions on Robotics and Automation, Vol. 15, No. 5, Outubro, 1999.

[8] COSTA, A. H. R. e PEGORARO, R., **"Construindo Robôs Autônomos para Partidas de Futebol: O Time Guaraná"**, SBA Controle & Automação, Vol. 11, No. 03, Set., Out., Nov., Dezembro de 2000.

[9] GOMES, M. M., KAWAMURA, J., LEONARDI, F., PARRO, V. C. e SEIFER, P. G., **"Aplicação do Sistema de Futebol de Robôs como Ferramenta Didática"**, XXIX COBENGE – Congresso Brasileiro de Ensino de Engenharia, Porto Alegre, RS, Setembro, 2001.

[10] KITANO, H., ASADA, M., NODA, I. e MATSUBARA, H., **"RoboCup: Robot World Cup"**, IEEE Robotics & Automation Magazine, Setembro, 1998.

[11] VIEIRA, F. C., ALSINA, P. J. e MEDEIROS, A. A. D., **"Micro-Robot Soccer Team – Mechanical and Hardware Implementation"**, Proceedings of COBEM, Robotics and Control – XVI Congresso Brasileiro de Engenharia Mecânica, Vol. 15, p. 534 – 540, Novembro, 2001.

[12] HWANG, S. e KINTIGH, B. P., **"Implementation of An Intelligent Roving Robot Using Multiple Sensors"**, Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI' 94), Las Vegas, NV, Outubro, 1994.

[13] DINNOUTI, L. S., VICTORINO, A. C. e SILVEIRA, G. F., **"Simultaneous Localization and Map Building by a Mobile Robot Using Sonar Sensors"** Proceedings of the 17th International Congress of Mechanical Engineering – COBEM 2003, São Paulo, Novembro, 2003.

[14] AIRES, K. R. T., ALSINA, P. J. e MEDEIROS, A. A. D., **"A Global Vision System for Mobile Mini-Robots"** Proceedings of XVI COBEM – Congresso Brasileiro de Engenharia Mecânica, Vol. 15, p. 240 – 247, Novembro, 2001.

[15] AMBROSE, R., ASKEW, R. S., **"An Experimental Investigation of Actuators for Space Robots"** IEEE International Conference on Robotics and Automation, p. 2625 – 2630, 1995.

[16] ASSIS, W. O., COELHO, A. D., GOMES, M. M., LABATE, C. G., CALASSO, D. F., CONDE FILHO, J. C. G., GUIMARÃES, F. A. "Construção de robôs jogadores de futebol", Revista Mecatrônica Fácil, nº 26, São Paulo, Janeiro/ Fevereiro, 2006.

[17] ADÔRNO, B. V. “Controles de velocidade e trajetória para o robô Sonic Shark e sensoriamento por ultra-som”, Curso de Engenharia Mecatrônica, Faculdade de Tecnologia, Brasília, Junho, 2004.

[18] OTTONI, G. L., LAGES, W. F. “Navegação de Robôs Móveis em Ambientes Desconhecidos utilizando Sonares de Ultra-som”, Revista Controle & Automação, Vol. 14, n. 4, Outubro/ Novembro/ Dezembro, 2003.

[19] BIANCHI, R. A. C., SIMÕES, A. S., COSTA, A. R. “Comportamentos reativos para seguir pistas em um robô móvel guiado por visão”, Simpósio Brasileiro de Automação Inteligente, 2001.

[20] SANDI, F. A., HEMERLY, E. M., LAGES, W. F. “Sistema para navegação e guiagem de robôs móveis autônomos”, SBA Controle & Automação, Vol. 9, nº 3, Setembro/ Outubro/ Novembro/ Dezembro, 1998.

[21] MAXIM, INTEGRATED PRODUCTS., ***“+/- 15kV ESD-Protected, +5V RS-232 Transceivers”***, 1996. Disponível em: <http://www.maxim-ic.com>. Acesso em: Jan/2004.

[22] HITACHI, Ltd., ***“HD74HC244 – Octal Buffers/ Line Drivers/ Line Receivers (with noninverted 3-state outputs)”,*** 1999. Disponível em: <http://www.has.hitachi.com.sg>. Acesso em: Jan/2004.

[23] BRAGA, N. C., **“Controle Remoto de 4 Canais”,** Revista Saber Eletrônica, Editora Saber Ltda., São Paulo, Ano 34, No. 316, páginas 8 a 12, Maio, 1999.

[24] SOARES, M. J., **“Robô RF”,** Revista Mecatrônica Fácil, Editora Saber Ltda., São Paulo, Ano 3, No. 15, páginas 38 a 45, Março, 2004.

[25] ATMEL, CORPORATION., ***“8 Bit AVR Microcontroller with 8K Bytes In-System Programmable Flash AT90S8515”***, 2001. Disponível em: <http://www.atmel.com>. Acesso em: Jan/2004.

[26] STMICROELECTRONICS., **"L298 – *Dual Full Bridge Driver*"**, 2000. Disponível em: <http://www.st.com>. Acesso em: Jan/2004.

[27] PIERI, E. R., **"Curso de Robótica Móvel",** Universidade Federal de Santa Catarina, Programa de Pós-graduação em Engenharia Elétrica, Florianópolis, Março, 2002.

[28] FENG, L., BORENSTEIN, J. and EVERETT, H. R., **"Where am I? - Sensors and Methods for Autonomous Mobile Robot Positioning",** The University of Michigan, Vol. 3, December, 1994.

[29] BORENSTEIN, J., EVERETT, H. R. e FENG, L., **"Where am I? - Sensors and Methods for Mobile Robot Positioning",** The University of Michigan, Abril, 1996.

[30] PHILIPS SEMICONDUCTORS., **"KMI18/2 Integrated rotational speed sensor"**, Setembro, 2000. Disponível em: <http://www.nxp.com>. Acesso em: Abr/2007.

[31] BEZERRA, C. G., **"Localização de um Robô Móvel usando Odometria e Marcos Naturais",** Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Março, 2004.

[32] RIBEIRO, M. I., **"Localização em Robótica Móvel - Odometria",** Instituto Superior Técnico, Instituto de Sistemas e Robótica, Portugal, Outubro, 1999.

[33] SOUZA, G. T., **"Controle e Automação Industrial",** Escola Técnica Estadual Pedro Ferreira Alves, Curso de Mecatrônica, Mogi Mirim, Julho, 2004. Disponível em: <http:// www.juliostampa.com.br/~junior/robotica/cai.pdf>. Acesso em: Jul/2007.

[34] MARCO, CANTÙ., **"Dominando o Delphi 7: a Bíblia"**. 1. ed. São Paulo: Makron Books, 2003. 896p.

[35] ASSIS, W. O., GENOVA, W. J., GOMES, M. M., COELHO, A. D. **"Aplicação de Tecnologias de Controle de Processos Contínuos e Processamento de Imagens no Futebol de Robôs"**. Anais da Competição Brasileira de Robótica, Florianópolis – SC, 2007.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo