UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CINTURÃO DE ULTRA-SOM PARA O ROBÔ OMNI

ALEXANDRE SIMÕES MARTINS
DIOGO ANDRADE

ORIENTADOR: GEOVANY ARAÚJO BORGES

RELATÓRIO DE PROJETO FINAL DE CURSO EM ENGENHARIA MECATRÔNICA (CONTROLE E AUTOMAÇÃO)

BRASÍLIA/DF: FEVEREIRO – 2005

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CINTURÃO DE ULTRA-SOM PARA O ROBÔ OMNI

ALEXANDRE SIMÕES MARTINS

DIOGO ANDRADE

MONOGRAFIA APRESENTADA AO CURSO DE ENGENHARIA MECATRÔNICA DA UNIVERSIDADE DE BRASÍLIA COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO TÍTULO DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

APROVADA POR:

_____

Prof. Geovany Araújo Borges, Docteur ès Sciences (ENE-UnB)

(Orientador)

_____

Prof. Victor Hugo Casanova Alcalde, PhD (ENE-UnB)

(Examinador Interno)

_____

Prof. Lélio Soares Ribeiro Jr., M.Sc (ENE-UnB)

(Examinador Interno)

BRASÍLIA/DF, 02 DE FEVEREIRO DE 2005

AGRADECIMENTOS

À Deus, pela vida e pelas pessoas que colocou em meu caminho.

Ao meu orientador Geovany pelos ensinamentos, dedicação, paciência e esforço para realização deste projeto.

À minha família pelo apoio, compreensão e paciência durante todos esses anos de vida acadêmica.

A todos os professores, amigos e colegas que partilharam conhecimentos e momentos de estudo, lazer, descontração e desespero.

Aos amigos do LCVC, em especial ao Flávio, que sempre parou o que estava fazendo para me ajudar, ao Diogo, meu parceiro de projeto, ao Roberto, pela companhia neste semestre e todos os demais, Zé, Cíntia, Leandro, Gustavo, Cascão, Marcio, Renato, Carla, Heiji, Leandro Silva, Antônio, Maurílio, Lucas, pela amizade e convivência.

A todos técnicos do DTL-ENE, em particular ao Célio e ao Cícero que sempre ajudaram em nossas necessidades.

À Finatec, pelo apoio financeiro.

Alexandre Simões Martins

AGRADECIMENTOS

Em primeiro lugar, quero agradecer á DEUS, por ter me proporcionado todos os momentos vividos nesta Universidade nestes inesquecíveis cinco anos.

Aos meus pais, Hélio e Rosa, que sempre tiveram fé em mim, por todo suporte dado em casa, confortando nos momentos tristes, repreendendo nos momentos de preguiça, e sempre dando total apoio psicológico nas vezes em que pensei em largar tudo.
Aos colegas da turma, que por mais que nossas escolhas acadêmicas tenham nos levado a rumos distintos, a união sempre se fez presente, seja nas viradas de noite na biblioteca ou nas festas de fim de semana.

Aos companheiros da equipe de guerra de Robôs UnBots, pelos momentos duros passados na construção de nossas máquinas, pelos momentos de alegria quando víamos nossos robôs funcionando, pelo apoio e união quando os robôs nos deixavam na mão.

Aos amigos do LCVC: Alexandre Martins, José Cezário, Cintia Bizarria, Lenadro Cotta, Bruno Roque, Márcio Rodrigo, Renato Sampaio, Flávio Vidal, Lucas Romano, Heiji Inuzuka, Antônio Padilha, Leandro Silva e Bruno Vilhena. Principalmente pela companhia, ajuda prestada sempre com muita boa vontade.

À minha namorada Lícia, que esteve ao meu lado este cinco anos, sempre me apoiando, nos momentos felizes ou tristes. E pela sua compreensividade nas horas em que meu curso me tomava todo o tempo livre.

Aos amigos da 216/416 sul, que desde os tempos de infância me proporcionaram quase toda a diversão que já tive em minha vida.

A todos os professores, que não apenas nos ensinaram engenharia, mas nos fez crescer como pessoas, em especial meu orientador Geovany Borges, apesar de nos conhecermos a pouco tempo, sempre teve muita paciência e vontade de ensinar tornando-se um verdadeiro amigo.

Diogo Andrade

RESUMO

CINTURÃO DE ULTRA-SOM PARA O ROBÔ OMNI

Autores: Alexandre Simões Martins, Diogo Andrade
Orientador: Geovany Araújo Borges
Trabalho de Graduação em Engenharia Mecatrônica
Brasília, fevereiro de 2005.

Na busca do projeto de robôs cada vez mais autônomos, a necessidade de sistemas de instrumentação cada vez mais robustos, confiáveis e precisos cresce com a complexidade que estes robôs são idealizados.

Este projeto engloba a concepção de um cinturão de sensores ultra-sônicos para medição de distâncias e detecção de obstáculos em robôs móveis. O projeto consiste no desenvolvimento de uma arquitetura microcontrolada, modularizada, de fácil conectividade entre os dispositivos e de boa capacidade de expansão, inclusive podendo-se adicionar ou remover dispositivos dinamicamente, durante o funcionamento do cinturão sem comprometer o funcionamento dos outros dispositivos. Outra característica fundamental do cinturão é a integração com o computador embarcado no robô via a plataforma RTX, possibilitando controle total do hardware do computador, dando-nos plenas condições de desenvolver um protocolo de comunicações eficiente de tempo real, bem como desenvolver as estratégias necessárias para uma aquisição de dados otimizada.

ABSTRACT

ULTRASSONIC SENSORS NETWORK FOR THE OMNI ROBOT

Author: Alexandre Simões Martins, Diogo Andrade
Supervisor: Geovany Araújo Borges
Undergraduate project: Mechatronics Engineering
Brasília, January 2005

In development of autonomous robots, the need of reliable and robust sensory systems grows with the complexity in which these robots are designed. This project involves the creation of a ultrasonic sensor network for distance measurement and obstacle detection in mobile robots. It consists in the develepmentof a microcontrolled, flexible and modular architeture, allowing dynamic addition or removal of the sensors online, without disturbing the other sensors.

Another key feature of the network is the interface between the robot's embedded computer through the RTX plataform, allowing total hardware control, giving full power to develop an efficient real-time communications protocol, as well as developing the most appropriate strategy for optimized obstacle detection.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

ρ – densidade

κ – índice de calor específico

λ_L – comprimento de onda

α_R – coeficiente de reflexão

α_T – coeficiente de refração

A/D – Analógico para Digital

c_L – velocidade longitudinal de propagação da onda

CTS – Clear to Send

DSR – Data Set Ready

DTR – Data Terminal Ready

dy – deslocamento de uma camada *y* de partículas

f_L – freqüência de oscilação da onda

GND – Signal Ground

HAL – Hardware Abstraction Layer

Hz – Hertz

J – Joule

K – Kelvin

kg – kilograma

l – distância

m – metro

ºC – grau Celsius

Oy – eixo cartesiano das ordenadas

p – pressão

R – constante universal dos gases

R_1 – resistência acústica do meio 1

R_2 – resistência acústica do meio 2

RI – Ring Indicator

RTS – Request to Send

Rx – Receiver Data

s – segundo

T – temperatura

t – tempo

T_T – tempo de vôo

TTL – transistor-transistor logic

Tx – Transmitted Data

UART – Universal Asynchronous serial Receiver and Transmitter

USART - Universal Synchronous and Asynchronous serial Receiver and Transmitter

V – Volt

W_I – energia incidente

W_R – energia refletida

W_T – energia transmitida

y – posição no eixo Oy

Z_A – impedância acústica

1. INTRODUÇÃO

1.1. ESTADO TECNOLÓGICO

Uma das primeiras aplicações de ultra-som foi feita em 1883 quando Galton projetou um apito para medir o limite de resposta do ouvido humano. Apesar do efeito piezoelétrico utilizado nos sensores de hoje já ser conhecido naquela época, somente depois de avanços na área da eletrônica que ele passou a ser utilizado [3] e [6].

O primeiro grande uso dos ultra-sons ocorreu durante a guerra de 1914-18 no desenvolvimento de sonares subaquáticos. Desde então, contínuos progressos foram feitos progressos em medições de constantes de propagação. Após então a guerra de 1939-45, surgiram então os maiores avanços como a técnica de pulso advinda da descoberta do radar e sua aplicação para teste não destrutivos de materiais e diagnósticos médicos, bem como, surgimento de novos materiais e transdutores [6].

Recentemente, a concepção de veículos robóticos autônomos tem exigido dispositivos sensores para sua própria localização, modelamento do ambiente, evitamento de obstáculos, enfim, para interagir com mundo externo. A dificuldade envolvida nisso é que todo sensor tem suas limitações em extrair do meio as informações desejadas, seja por restrição do campo de atuação, baixa resolução, influências externas, ou até mesmo pelo processamento de sinal requerido. Assim, o uso de múltiplos sensores e o emprego de técnicas de fusão sensorial tem sido cada vez mais difundido.

Sensores ultra-sônicos são frequentemente usados para mapeamento de ambiente e desvios de obstáculos em robôs móveis. Devido às características do ultra-som de direcionabilidade, propagação no ar e de refletir em praticamente qualquer obstáculo, ele é comumente usado para medir distâncias e posições angulares não só em robótica móvel como também na indústria. Atualmente, ele é largamente utilizado em sistemas de alarmes, principalmente os automotivos. Sistemas mais avançados têm empregado combinações de ultra-som com infra-vermelho e também com câmera para medições de objetos em três dimensões, eliminando a necessidade de uma segunda câmera e tratamento de múltiplas

imagens. Por ser também de baixo custo, a utilização de vários sensores é viável, podendo, entretanto, aumentar o esforço computacional do robô.

1.2. PROPOSTA DE TRABALHO

Este trabalho engloba o projeto e implementação de uma rede de sensores ultra-sônicos para medição de distância a obstáculos em robôs móveis, sendo parte complementar de um projeto maior que visa desvio de obstáculos, navegação e fusão-sensorial.

Visando tal objetivo, o projeto consiste no desenvolvimento de uma arquitetura com microcontroladores que fosse robusta, altamente modularizada e de instalação dinâmica, significando que sensores podem ser adicionados e retirados da rede e o sistema será atualizado de acordo com a montagem atual sem comprometer o funcionamento da rede como um todo, caracterizando um funcionamento *Plug-and-Play*.

O trabalho inclui concepção do *hardware* eletrônico, como circuito transceptor de ultra-som, bem como, estrutura mecânica, como caixas e conectores. A utilização de microcontroladores dever tratar dos cálculos da distância e sua compensação em função da temperatura e, claro, da comunicação com o sistema do robô. A topologia desta rede deve favorecer uma comunicação robusta e que facilite sua expansão, conseqüentemente, o protocolo de troca de mensagens deve ser eficiente para diminuir a latência de troca de mensagens, baseado no fato de que o meio físico da rede tem uma alta imunidade a erros de comunicações. Este protocolo também deve ter características dinâmicas que estejam constantemente atualizando o estado da rede, de tal forma que a característica *"Plug-and-Play"* seja satisfeita. Como a troca de mensagens entre o computador embarcado e a rede de sensores deve ser bem temporizada, os sistemas operacionais comerciais comuns não satisfazem esta condição, por não serem determinísticos e não fornecerem ao usuário controle total sobre o hardware. Portanto, os algoritmos de controle da rede de sensores devem ser desenvolvidos em uma plataforma de tempo real.

.

1.3. APRESENTAÇÃO DO MANUSCRITO

Neste relatório, o Capítulo 2 apresentará todo fundamento teórico a respeito de ondas de ultra-som, comunicação serial entre dispositivos, descrevendo os padrões RS232 e RS485, as características do microcontrolador utilizado, e a descrição da utilidade da plataforma RTX para a presente aplicação.

No capítulo 3 são mostradas todas as implementações práticas, como os circuitos que geram e recebem os sinais de ultra-som, os algoritmos do microcontrolador que controla os sensores e do computador embarcado que controla toda a rede, além do procedimento prático para a correta calibração dos sensores.

O capítulo 4 é destinado à discussão dos resultados obtidos, e por fim o capítulo 5 apresenta as conclusões gerais e recomendações para trabalhos futuros.

2. FUNDAMENTOS TEÓRICOS

2.1. ULTRA-SOM

O termo ultra-som refere-se às ondas sonoras com freqüências acima da audibilidade humana, isto é, maiores que mais ou menos 18kHz. As principais aplicações de ultra-som podem ser divididas em duas seções, uma relacionada às vibrações de pequena amplitude e a outra com potência elevada [4]. Esta última envolve alterações permanentes no meio de propagação das ondas ultra-sônicas e suas aplicações incluem limpeza, perfuração, processos químicos, produção de emulsões, e fogem do assunto deste trabalho.

Por outro lado, a preocupação na propagação de ondas de pequena amplitude está no efeito que meio tem sobre as ondas, como reflexão, refração, velocidade de propagação ou atenuação. As técnicas de medição destes efeitos são utilizadas nas mais variadas aplicações como determinação de módulos de elasticidade em metais, sonares, fluxímetros, espessímetros, obtenção de imagens de partes do corpo humano, entre outras.

As ondas ultra-sônicas obedecem às mesmas leis básicas da acústica, entretanto têm algumas vantagens em relação às ondas sonoras de baixa freqüência:

- Ondas de alta freqüência têm curto comprimento de onda, o que permite uma redução da difração e dispersão por um dado obstáculo, facilitando também o direcionamento de uma emissão de ultra-som.

- Ondas ultra-sônicas podem passar facilmente por paredes de metal de tubos e navios, e até mesmo em tecidos biológicos permitindo sistemas de medição não invasivos.

2.1.1. Princípio básico da transmissão de ultra-som

Os dispositivos mais comumente usados para emissão e recepção de ultra-sons são os cristais piezoelétricos, que têm capacidade de converter energia mecânica em energia

elétrica e vice-versa. Os transmissores ultra-sônicos utilizam o efeito piezoelétrico inverso, isto é, se uma tensão senoidal é aplicada no cristal transmissor este produzirá uma deformação senoidal correspondente *x* (Figura 1). Esta vibração é transmitida para as partículas no início do meio de transmissão num movimento senoidal, causando a vibração das outras partículas até que a perturbação é transmitida ao final do meio. Esta então é detectada pelo receptor ultra-sônico que utiliza o efeito piezoelétrico direto, onde a força gerada pelas ondas de pressão do meio sobre a área do cristal produz uma variação correspondente na carga *q* e corrente *i*. Esta corrente, sobre a carga Z_L gera a tensão de saída V_{OUT}.

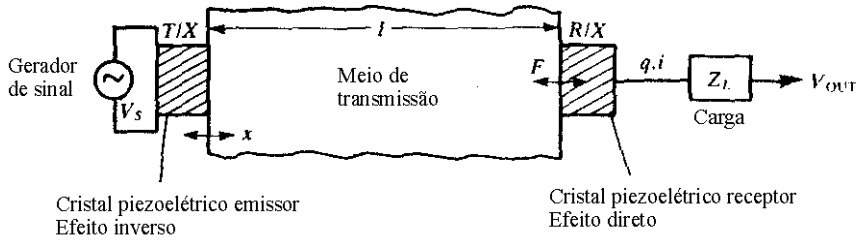


Figura 1 - Princípio da transmissão do ultra-som
Fonte: Adaptado de BENTLEY, John P., Principles of measurement systems. 1988

2.1.2. Propagação de ondas ultra-sônicas

A propagação de ondas ultra-sônicas, como foi dito antes, obedece aos mesmos princípios das ondas acústicas de baixa freqüência. Como os gases suportam apenas tensões de tração e compressão, mas não tensões cisalhantes, apenas ondas longitudinais passam através dos gases. Em ondas longitudinais o deslocamento *dy* das partículas é paralelo à direção de propagação da onda, ou seja, paralelo ao eixo *Oy*, e descrito pela seguinte equação diferencial parcial

$$\frac{\partial^2 dy}{\partial y^2} = \frac{1}{c_L}\frac{\partial^2 dy}{\partial t^2}, \qquad (1)$$

na qual *dy* é o deslocamento de uma camada *y* de partículas do meio no tempo *t* e c_L é a velocidade de propagação da onda longitudinal. Para uma onda se propagando ao longo do sentido positivo do eixo *y*, a solução para (1) é

$$dy = \hat{d}ysen\left(2\pi f_L\left(t - \frac{y}{c_L}\right)\right), \tag{2}$$

em que $\hat{d}y$ é a amplitude do deslocamento. E para propagação no sentido negativo de y

$$dy = \hat{d}ysen\left(2\pi f_L\left(t - \frac{y}{c_L}\right)\right). \tag{3}$$

A distância entre duas camadas adjacentes, que têm o mesmo deslocamento, é definida comprimento de onda λ_L. E a partir das equações acima podemos chegar à seguinte relação:

$$c_L = f_L \lambda_L . \tag{4}$$

Em geral a velocidade do som nos gases é dada por

$$c_L = \sqrt{\kappa \frac{p}{\rho}} \tag{5}$$

na qual κ é o índice adiabático ou índice de calor específico do gás (relação entre calor específico a pressão constante e calor específico a volume constante), p é a pressão e ρ sua densidade. Utilizando a lei dos gases ideais, a equação acima fica idêntica à

$$c_L = \sqrt{\kappa R T} , \tag{6}$$

em que R é constante universal dos gases dividida pela massa molar do gás e T é a temperatura absoluta em Kelvin. A equação acima mostra que velocidade do som depende apenas do meio e da temperatura, e não da pressão. A umidade tem também uma pequena influência na velocidade já que altera massa molar do gás.

Para o ar seco, encontra-se na literatura R = 287.05 J/kgK e $\kappa \approx 1{,}4$[1] e assim a equação reduz-se a:

$$c_L \cong 20.04\sqrt{T} \text{ m/s}. \tag{7}$$

[1] O índice de calor específico tem uma influência muito pequena da temperatura, vide anexo a.

2.1.3. Impedância acústica

Impedância acústica é definida como:

$$Z_A = \frac{p}{u}, \tag{8}$$

na qual $u=\partial d/\partial t$ e p é a pressão. Em geral Z_A é um número complexo da forma R_A+jX_A, entretanto para ondas planas progressivas, a parte imaginária é zero, e pode ser mostrado que R_A é o produto da densidade ρ e da velocidade do som c_L, i.é.,

$$R_A = \rho \cdot c_L. \tag{9}$$

2.1.4. Atenuação de uma onda no meio

Na prática, uma onda sonora nunca atravessa um meio com amplitude de deslocamento da partícula $\hat{d}y$ e amplitude da variação da pressão $\hat{p}$ constantes. A onda vai sendo atenuada à medida que percorre o meio, e as amplitudes caem exponencialmente com a distância y percorrida.

2.1.5. Reflexão e refração na fronteira entre dois materiais

Quando uma onda percorrendo um material de determinada impedância incide sobre a fronteira entre este e um segundo material de impedância diferente, parte da energia segue adiante, como uma onda percorrendo o segundo meio, enquanto que parte é refletida de volta no primeiro meio.

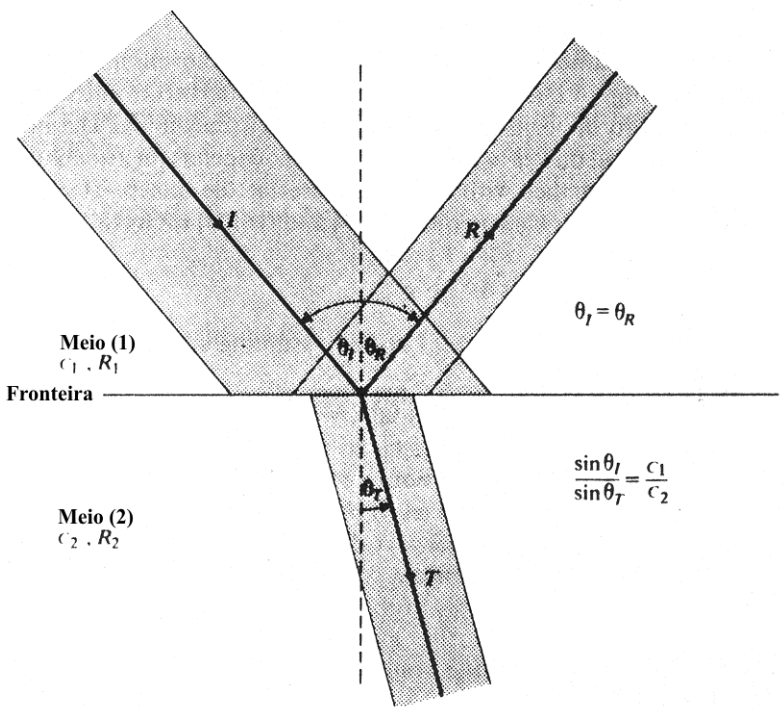


Figura 2 – Reflexão e Refração do som na fronteira entre dois materiais
Fonte: Adaptado de BENTLEY, John P., Principles of measurement systems. 1988

A relação entre a energia refletida W_R e a energia incidente W_I é chamada coeficiente de reflexão α_R, enquanto que o coeficiente de refração α_T é definido como a relação entre a energia transmitida W_T sobre a incidente W_I.

$$\alpha_R = \frac{W_R}{W_I}, \qquad \alpha_T = \frac{W_T}{W_I} \tag{10}$$

Para ângulos de incidência normais ou quase normais, ou seja, para $\theta_I \approx \theta_R \approx \theta_T \approx 0$, pode ser mostrado que

$$\alpha_R = \frac{(R_2 - R_1)^2}{(R_2 + R_1)^2}, \qquad \text{e} \qquad \alpha_R = \frac{4R_1R_2}{(R_2 + R_1)^2}, \tag{11}$$

em que R_1 e R_2 são as impedâncias do meio 1 e 2 respectivamente. Note que $\alpha_R + \alpha_T = 1$, como conseqüência da conservação de energia.

Como a impedância do ar é da ordem de 1000 a 10000 vezes maior que a maioria dos líquidos e sólidos o coeficiente de reflexão é praticamente igual a 1 para a grande parte das fronteiras ar/líquido e ar/sólido. Essa característica faz do ultra-som uma ótima aplicação para detecção de obstáculos.

2.1.6. Medição de distância por emissão de pulso e detecção de eco

O principio de medição utilizado por este método consiste basicamente na emissão de um pulso modulado por ultra-som, isto é, um trem finito de pulsos na freqüência ultra-sônica em um determinado meio. Se um obstáculo de dimensões maiores que o comprimento de onda e com impedância consideravelmente diferente da impedância do meio, então a maior parte da onda é refletida de volta. O pulso refletido é então detectado por um cristal receptor e o tempo de vôo t_T desde a emissão do pulso até a detecção de seu eco é mensurado. Se a velocidade do som no meio é conhecida, a distância l entre o sensor e o obstáculo pode ser obtida por:

$$l = \frac{t_T c_L}{2}. \tag{12}$$

Algumas precauções devem ser feitas para este tipo de medição. Primeiro, por que múltiplos de ecos são gerados, pois parte do pulso é refletida no sensor receptor e novamente refletida no obstáculo, podendo isso ocorrer repetidas vezes até que se extingam pela atenuação do meio. Assim, a medição deve levar em conta apenas o primeiro eco, e uma segunda medição deve ser feita apenas depois que meio estiver limpo de ecos advindos da medição anterior.

Outras questões a serem observadas dizem respeito à distância do obstáculo e à largura do pulso. O pulso deve ser largo o suficiente comparado com o período da onda ultra-sônica para garantir que o pulso tenha vários ciclos e energia necessária para ir e voltar. Entretanto, a distância mínima ao obstáculo deve ser grande o bastante em comparação com a largura do pulso para evitar interferência entre o pulso emitido e o pulso refletido próximo aos transdutores, já que o receptor não distinguiria um pulso do outro.

2.2. OS TRANSDUTORES MA40B8R/S DA MURATA

Neste trabalho utiliza-se um par de transdutores, sendo um para emissão (MA40B8S) e outro para detecção (MA40B8S) já que, além de facilitar, separando os circuitos de transmissão da recepção, cada transdutor é otimizado de acordo com sua função.

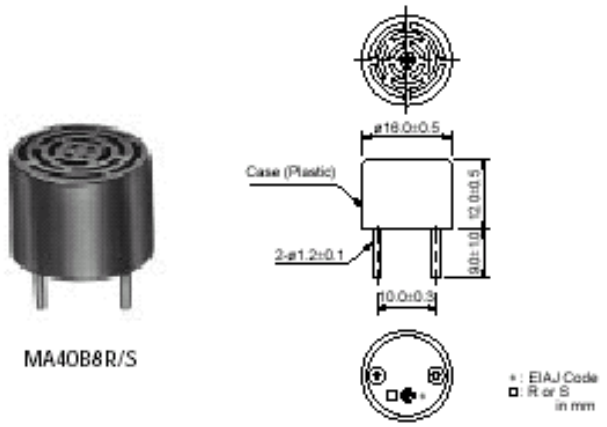
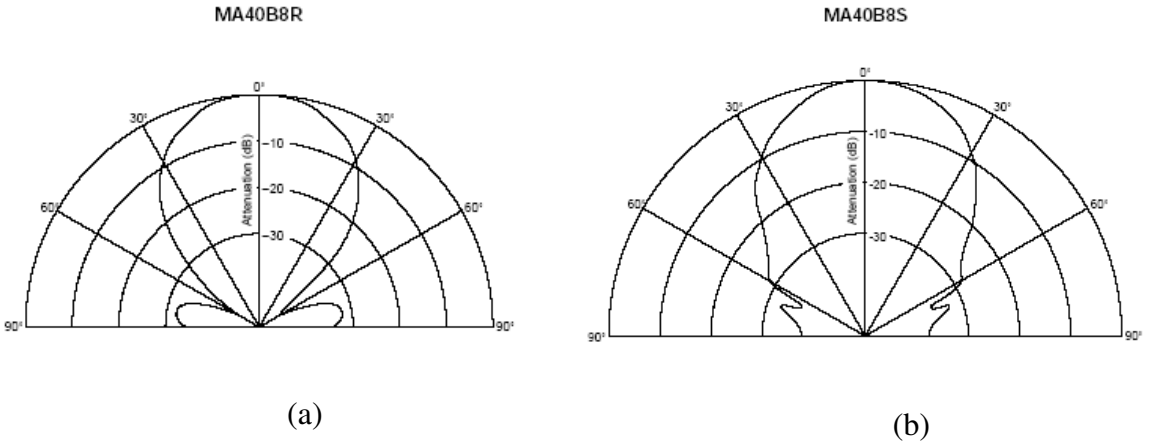


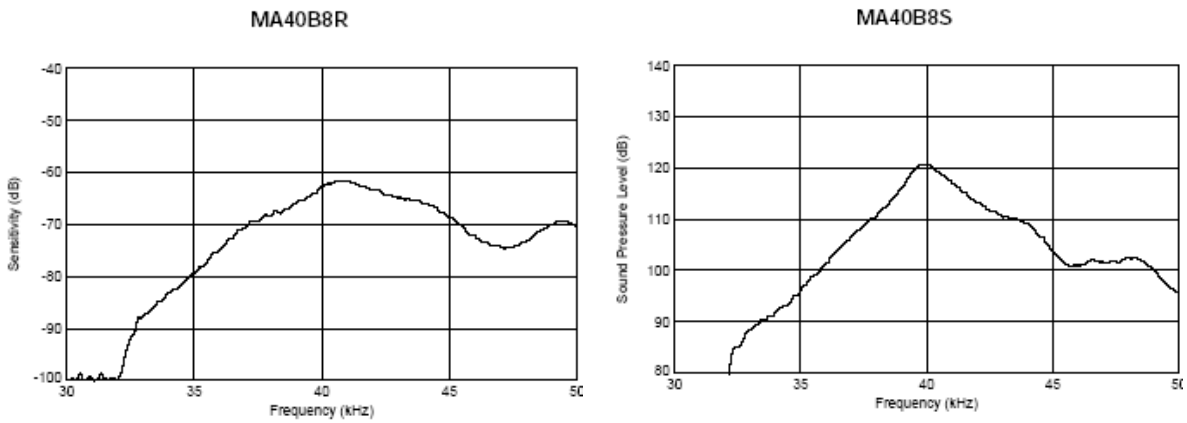
Figura 3 – Os transutores murata MA40B8R/S
Fonte: Data sheet Ultrasonic Sensors. Murata Manufacturing Co., Ltd. Disponível em: <http://www.murata.com>. Acesso em: 10 ago. 2004.

Os elementos sensores MA40B8R/S são de cristais cerâmicos piezoelétricos encapsulados em cilindros plásticos de ϕ16x12mm. Possuem freqüência nominal de 40kHz, ângulo de abertura de 50º, capacitância de 2000pF, faixa de temperatura de operação de -30 a 85 ºC e tensão máxima de 40V pico-a-pico para o emissor. A Figura 4 mostra a característica do ângulo de abertura, tanto do emissor quanto do receptor, já a Figura 5 mostra as características de freqüência de cada um.



(a) (b)

Figura 4 - Ângulo de abertura (a) do receptor e (b) do emissor
Fonte: Data sheet Ultrasonic Sensors. Murata Manufacturing Co., Ltd. Disponível em: <http://www.murata.com>. Acesso em: 10 ago. 2004.



(a) (b)

Figura 5 – Características de freqüência. (a) receptor e (b) emissor
Fonte: Data sheet Ultrasonic Sensors. Murata Manufacturing Co., Ltd. Disponível em: <http://www.murata.com>. Acesso em: 10 ago. 2004.

2.3. COMUNICAÇÃO SERIAL

2.3.1 Interface RS232

A interface RS232 foi concebida como um meio de comunicação *full-duplex* ponto-a-ponto entre PC e um terminal qualquer. Este terminal pode ser um Modem, um microcontrolador, um CLP. Enfim, uma infinidade de periféricos externos.

Inicialmente, esta interface era composta de inúmeras vias de controle, e utilizava um conector padrão DB25. Atualmente, as portas seriais dos computadores modernos utilizam apenas 9 pinos, ficando o padrão DB9. O significado de cada pino pode ser visto na Figura 6.

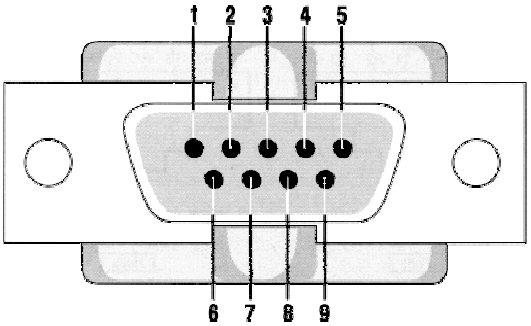


Figura 6 – Pinagem da porta serial RS232
Fonte: Disponível em: <http://www.arcelect.com/rs232.htm> Acesso em: 21 jan. 2005.

1 – Data Carrier Detect – (DCD)

Este sinal é usado para detecção de um sinal de tom de um modem, identificando que ele está conectado na linha telefônica. Este sinal é geralmente ignorado na comunicação de outros dispositivos.

2 – Receiver Data (RX)

Neste pino são recebidos os bits transmitidos pelo dispositivo terminal.

3 – Transmitted Data (TX)

Neste pino são enviados os bits para o dispositivo terminal.

4 – Data Terminal Ready (DTR)

Este pino sinaliza para o dispositivo terminal que o PC está pronto para se comunicar.

5 – Signal Ground (GND)

É a referencia das tensões de transmissão e recepção, para que o PC e o dispositivo terminal tenham o mesmo nível de referencia de 0 volts.

6 – Data Set Ready (DSR)

Este pino sinaliza para o PC que o dispositivo terminal está pronto para se comunicar.

7 – Request To send (RTS)

Este pino sinaliza do PC para o dispositivo terminal que o PC quer transmitir.

8 – Clear to send (CTS)

Este pino sinaliza do dispositivo terminal para o PC que está pronto para receber uma transmissão.

9 – Ring indicator (RI)

Usado apenas quando o dispositivo terminal é um modem, indica que uma ligação chegou pela linha telefônica.

O padrão RS232 trabalha com tensões elevadas de forma a aumentar a capacidade da distancia de transmissão. O nível lógico “1” é simbolizado por uma tensão entre -3 e -15 volts, já o nível lógico “0” é interpretado como uma tensão entre +3 e +15V, essas tensões com referência ao terra comum.

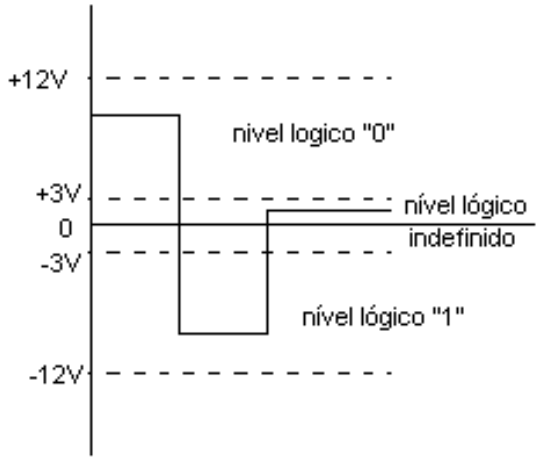


Figura 7 – Tensões do RS232

O formato de dados utilizado geralmente é o caractere ASCII. A temporização da transmissão é feita segundo o formato UART, que é um protocolo assíncrono, onde o canal de comunicação é mantido em nível lógico “1”, quando é feita uma transmissão, o canal desce para o nível lógico “0”, em seguida os 8 bits são transmitidos, seguidos de um bit de paridade (opcional), e de dois bits “1” de parada, conforme a Figura 8.

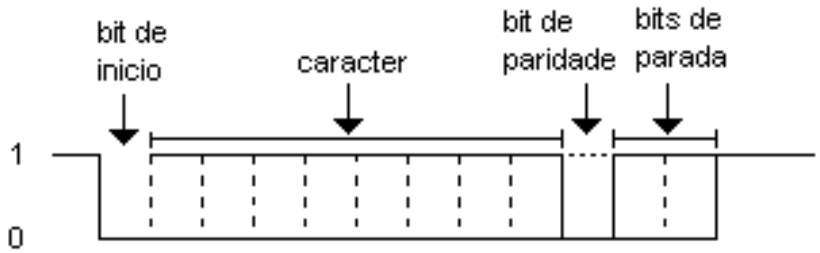


Figura 8 – Formato de dados UART

A velocidade de comunicação neste protocolo é dada em Kbps (Kilobits por segundo), e os valores vão de desde 2400 a 512000 bps. Nos micros IBM-PC, a velocidade máxima é de 115200 bps.

2.3.2 Interface RS485

O padrão RS485 foi proposto para contornar diversos problemas de comunicação entre dispositivos normalmente encontrados nos meios onde são operados como: ruído induzido, diferenças de tensão entre os aterramentos dos terminais de comunicação, assim como distancias elevadas entre os mesmos. É amplamente utilizado em aplicações industriais onde geralmente tem-se um ambiente bastante ruidoso, e onde também o canal de comunicação deve percorrer longas distâncias cobrindo a maioria dos equipamentos do chão de fábrica. Este padrão também é bastante utilizado em configurações onde se tem um dispositivo mestre que comanda vários dispositivos escravos.

A interface RS485 consiste em uma topologia de rede de comunicações do tipo barramento. Este barramento consiste em um par trançado, onde as informações são ttransmitidas de forma *half-duplex* e são interpretadas através da tensão diferencial entre eles. Desta forma, utilizar este tipo de interface para comunicação entre dispositivos, requer bastante cuidado quanto à sincronização dos dados, visto que apenas um dispositivo por vez pode transmitir pelo barramento. Se dois dispositivos usarem o barramento ao mesmo tempo, haverá erro ou perda de informação. Muitos dispositivos de interface RS485 suportam até 32 transceptores conectados a ele sem auxilio de repetidores. Um deles é o circuito integrado DS485, um transceptor RS485, cujo diagrama interno é mostrado na Figura 9:

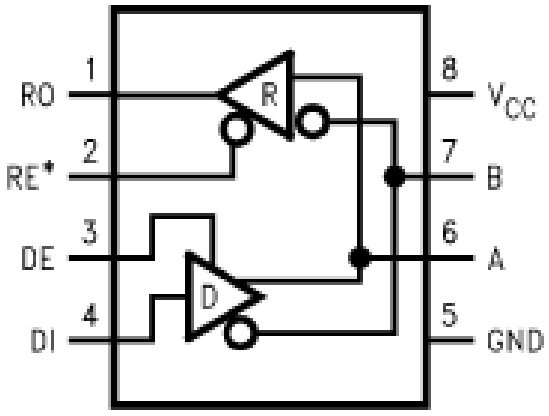


Figura 9 – Pinagem do DS485
Fonte: Data sheet DS485, National Semiconductor Corporation. Disponível em <http://www.national.com>. Acesso em 22 jan. 2005

Os transceptores de RS485 normalmente consistem em um *driver* de transmissão **D** e um de recepção **R** conectados juntos, tendo seus respectivos pinos de habilitação **RE*** (ativo 0) e **DE** (ativo 1). Normalmente estes pinos são conectados juntos, de forma que o transceptor esteja apenas recebendo ou transmitindo, nunca habilitando os dois drivers juntos,evitando inconsistência nas comunicações. Os pinos **RO** e **DI** representam *saída da recepção* e *driver de entrada* respectivamente, e eles trabalham com níveis lógico TTL (0 a 5V). Já as saídas para o baramento, **A** e **B,** operam com tensão diferencial entre seus terminais. O transceptor normalmente é alimentado com 5 Volts DC, e consome geralmente menos de 0,5 mA sem carga, e suporta uma taxa de transmissão superior a 10Mbps.

Desta forma, para que um dispositivo transmita um dado pelo barramento, é necessário que ele eleve para 5Volts o pino **DE**, fazendo com que **RE*** seja desabilitado, para então transmitir a informação necessária pelo pino **DI**, e ao fim da transmissão, desabilitar **DE** reabilitando **RE***, de forma que o transceptor volte ao modo de recepção. Esta habilitação pode ser feita via software, controlando pinos de um microcontrolador ou de uma porta de comunicações de um microcomputador, ou então através de um hardware construído especialmente para detectar inicio de transmissão para o formato de dados utilizado (*e.g., USART)*, o que é útil no caso de precisar adaptar uma aplicação já existente escrita para uma interface como RS 232 para RS 485.

Outra característica importante destes transceptores é que eles suportam uma diferença entre as tensões de aterramento de 2 transceptores conectados de até ± 7Volts (modo comum), possibilitando a comunicação entre dispositivos presentes em locais distantes e com aterramentos diferenciados.

A transmissão por tensão diferenciada é feita da seguinte forma: se quisermos transmitir o nível lógico 0, o pino de saída **A** é setado em 0Volts, e o pino **B** é setado em 5Volts, e a tensão diferencial **A**–**B**=-5Volts. De modo análogo, se quisermos transmitir o nível lógico 1, o pino **A** é setado em 5Volts e o pino **B** é setado em 0 Volts e **A**–**B**=+5Volts, conforme na Figura 10.

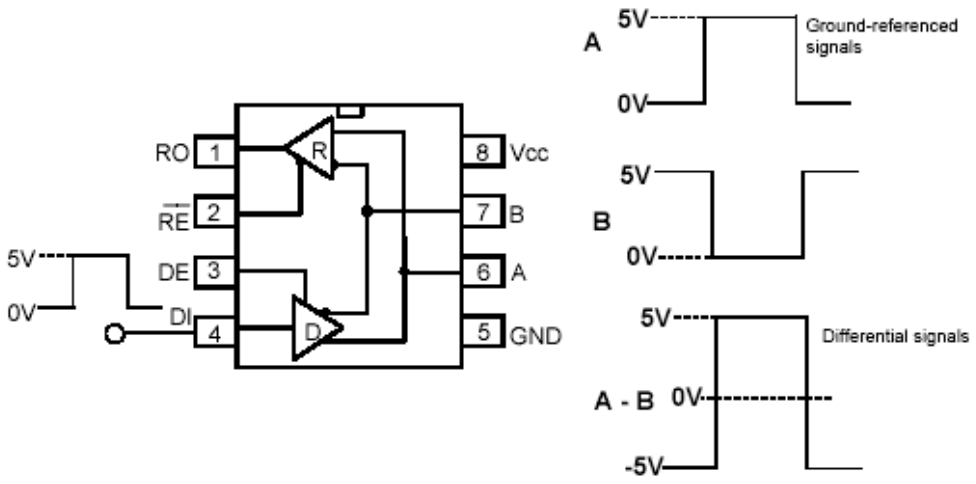


Figura 10 – Sinais Diferencias

Fonte: Application Notes ST485, STMicroelectronics. Disponível em <http://www.st.com>. Acesso em 23 jan. 2005

Como trabalhamos com uma tensão diferencial, a escolha do par trançado elimina grande parte do ruído eletromagnético induzido pelo meio, que pode provir desde de campos magnéticos de outros aparelhos de telecomunicação até harmônicos gerados por máquinas de grande potência na rede elétrica, sobretudo em ambiente industrial. Este efeito de cancelamento de ruído é baseado no fato de que um ruído somado ao sinal de um dos fios também será somado ao sinal de seu par trançado, e quando é feita a tensão diferencial, este ruído é cancelado, como pode ser percebido na Figura 11.

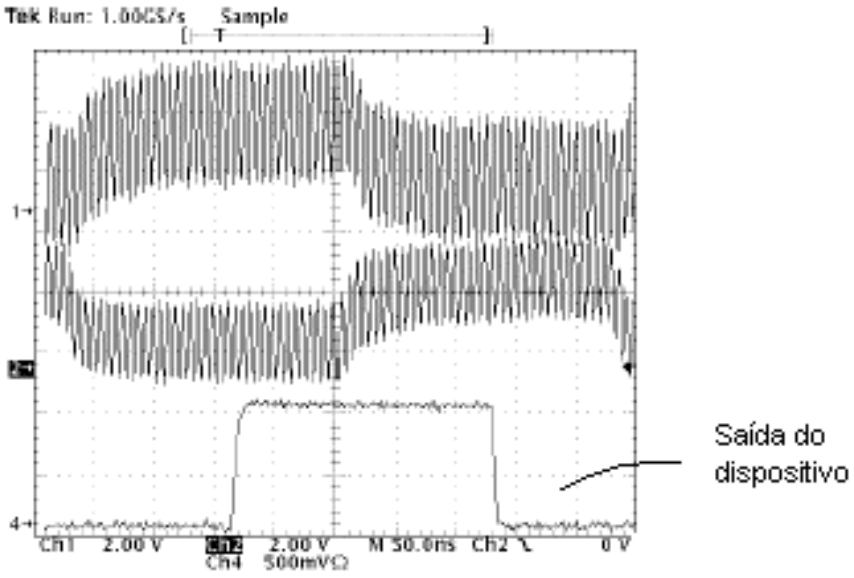


Figura 11 – Ruído induzido no par trançado

Fonte: Application Notes ST485, STMicroelectronics. Disponível em <http://www.st.com>. Acesso em 23 jan. 2005

Quando utilizamos meios guiados para comunicações, devemos conhecer a impedância característica do cabo, que é a composição de sua resistência e reatância, quando ser comprimento tende a infinito. Se o cabo for utilizado para transmissões a altas freqüência podem ocorrer reflexões do sinal em sua extremidade provocando inconsistência nos dados transmitidos. Para que isso não ocorra, basta adicionar um resistor de valor igual à impedância característica do cabo para que ele se comporte como um cabo infinito.

Outra consideração a ser feita é quando todos os nós estão em modo de recepção, o nível lógico do barramento pode ficar indefinido. Para garantir que o barramento fique sempre em nível lógico 1, devemos adicionar um resitor *pull-up* no pino A e um de *pull-down* no pino B. Esse resistores devem ter igual valor para não alterar o balancemento da linha de transmissão.

2.4. O MICROCONTROLADOR ATMEL AVR ATMEGA8

Microcontroladores são dispositivos que contêm em um único encapsulamento um microprocessador, memória e diversos periféricos como *timers*, entradas e saídas digitais ou analógicas, interface serial, conversores A/D, etc., que auxiliam na interface e controle de sistemas. Devido ao seu tamanho reduzido, e baixos consumo e custo, os microcontroladores são largamente usados em projetos de controle e automação.

O ATmega8 é um microcontrolador CMOS de alta performance com conjunto de instruções reduzido (RISC) com 130 instruções e empregando arquitetura Harvard com barramentos e espaços de endereçamento separados para memória de dados e de programa. Dentre suas diversas características, destacam-se:

- Instruções fixas de 16 bits;
- Barramento de dados da memória de programa de 16 bits para acesso em um único ciclo;
- Pipeline de 2 estágios, enquanto uma instrução é executada, outra é carregada da memória de programa;

- Conjunto de instruções que facilitam programação em linguagens de alto nível contendo 130 instruções otimizadas, sendo que aproximadamente 80% delas são executadas em apenas 1 ciclo;
- 32 registradores de uso geral que podem ser acessados simultaneamente pela CPU permitindo códigos de programa menores;
- Faixa de operação de 4,5 a 5,5V;
- Memória Flash de 8K *bytes* programável em circuito sem necessidade de tensão adicional, podendo inclusive ser autoprogramável;
- Três registradores de contagem sendo dois de 8 bits e um de 16 bits;
- Três canais PWM;
- Conversor de A/D de 10 bits com 6 entradas multiplexadas;
- USART serial programável;
- *Wacthdog* timer programável;
- Comparador analógico;
- 23 pinos de I/O configuráveis;
- Oscilador interno calibrável;

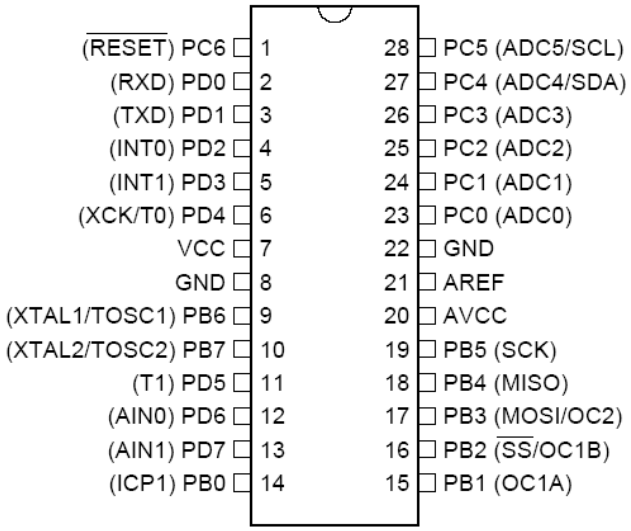


Figura 12 – Pinagem do ATmega8

Fonte: Data sheet AVR ATmega8. Atmel Corporation. Disponível em: <http://www.atmel.com>. Acesso em: 31 mar. 2004.

A Figura 12, mostra o diagrama dos pinos. O desenvolvimento do software foi feito em linguagem C utilizando o ambiente WinAVR de código-aberto que vem com o conhecido compilador gcc. Depois de compilado o código binário é transferido ao microcontrolador utilizando a porta paralela do PC.

2.5. O ROBÔ OMNI

O robô OMNI é uma plataforma ominidirecional com três rodas orientáveis e tracionáveis. Essas três rodas são inteiramente motorizadas, cada uma delas é equipada com dois motores, um que comanda seu eixo de orientação e outro ligado ao seu eixo de tração. Dessa forma, o robô tem plena mobilidade no plano.

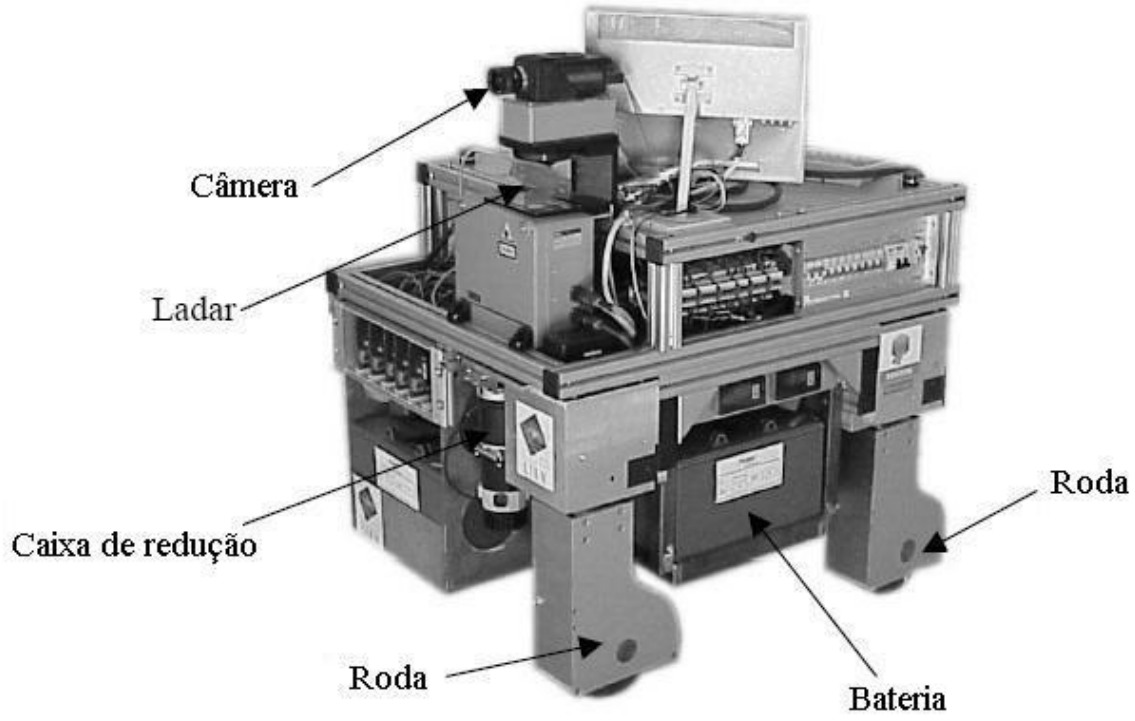


Figura 13 – Robô OMNI

Fonte: Adaptado de ARAGONES, Julien. Commande et evaluation des performances d'un robot omnidirectionnel à roues. 2002

Pela Figura 13 é possível ter uma visão da geometria do robô. Suas dimensões são 860mm de comprimento, 725mm de largura e 760mm de altura, pesando cerca de 325Kg. É equipado com um PC com *Windows NT* e plataforma de tempo real RTX. Além disso, possui vários *encoders* para medir seu deslocamento, um radar a laser (LADAR) e uma câmera utilizados para localização abslouta. Para melhorar ainda mais a precisão da estimativa de deslocação, há também um girômetro óptico que mede diretamente as mudanças de orientação ou a orientação relativa do robô.

Para a detecção de obstáculos, o OMNI não possui sensores apropriados. Por exemplo, o LADAR só é capaz de fazer uma varredura num plano horizontal um pouco acima da altura do robô. Obstáculos abaixo ou acima desse plano não são detectados. A câmera de vídeo possui campo de visão limitado. Por estes motivos faz-se necessário construir uma rede de sensores apropriada para a detecção de obstáculos ao redor do OMNI, uma vez que este pode realizar movimentos em todas as direções.

2.6. A PLATAFORMA RTX

A plataforma RTX 4.3 é uma extensão ao sistema operacional Windows NT 4.0, que possibilita a utilização de um PC de forma determinística, eliminando a necessidade de um processador de sinais auxiliar para aplicações em tempo real. A plataforma está integrada ao kernel do Windows NT, utilizando uma versão determinística da API do Windows para utilizar as funções e outras facilidades do RTX.

Além de eliminar a necessidade de hardware adicional, O RTX apresenta outros benefícios, tais como:

- Redução da complexidade do sistema, por utilizar apenas um sistema operacional. O RTX é uma extensão do Windows NT, proporcionando ao mesmo determinismo na execução e chaveamento de tarefas, e não um sistema operacional diferente.
- Redução dos Custos de Desenvolvimento, por compartilhar as mesmas ferramentas de desenvolvimento do Windows NT.
- Garante a Compatibilidade dos aplicativos RTX e Win32, pela presença de uma biblioteca de funções RTAPI (real time API) comum a ambos os ambientes. Como o RTX não altera o kernel do Windows NT, a portabilidade das aplicações criadas para futuras versões do Windows NT está grantida.

O principio de funcionamento do RTX está em colocar entre o Kernel do Windows NT e o hardware, uma HAL (Hardware Abstraction Layer), que continua servindo ao kernel do Windows NT sem alterar seu funcionamento e proporciona ao RTX acesso direto ao hardware, interrupções, relógios e memória sem a necessidade da interface com um driver de dispositivo, proporcionando a utilização do PC como um hardware com características deterministicas de tempo real. Por exemplo, se o Windows NT gera um erro fatal, ocasionando a famosa “tela azul”, A HAL de tempo real continua servindo os processos RTSS até que a falha do Windows NT possa ser recuperada. A organização do funcionamento do RTX, bem como sua interface com o hardware e Windows NT estão esquematizados na Figura 14.

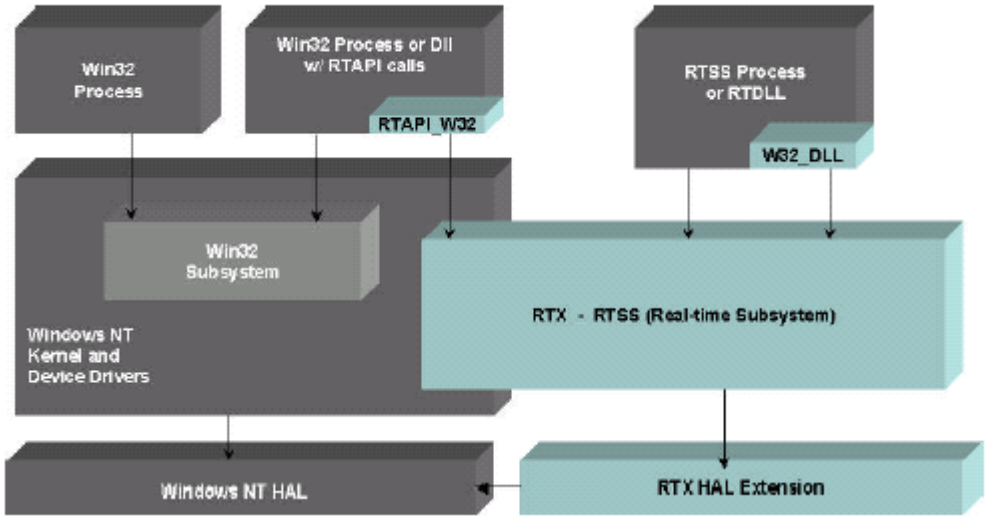


Figura 14 – Estrutura do RTX
Fonte: RTX User Guide

Acima da HAL, e funcionando como um kernel, temos o RTSS (Real time sub system), fornecendo as funções e gerencia de recursos do RTX. É implementado como um driver de dispositivo do Windows NT, utilizando de serviços tanto do Windows NT quanto da HAL. A partir da HAL, o RTSS fornece todas as funções de relógio e temporização e também os serviços de interrupções. A partir do kernel do Windows NT, o RTSS fornece a gerencia de memória. O RTSS também dispõe de um gerenciador de processos, e um escalonador de prioridades fixas, assim como uma interface entre aplicações RTSS e Win32 por meio de serviços de comunicação entre processos (IPC) de tempo real.

A interface de programação de tempo real (RTAPI), é uma biblioteca que proporciona extensões de tempo real essenciais á API Win32 do Windows NT. Esta biblioteca pode ser usada tanto com processos RTX quanto processos Win32, estando presente em ambos ambientes de execução. Assim, podemos construir aplicativos que se subdividem em seções criticas de tempo real que executam no ambiente RTX e seções não criticas que executam no ambiente Windows NT. De forma análoga, podemos também construir aplicativos Windows NT e aplicativos RTX que se comunicam em tempo real tocando informações, como por exemplo, um aplicativo de interface gráfica mostrando dados coletados por um aplicativo RTX.

Resumindo, as principais facilidades fornecidas pelo RTX são:

- Gerencia de processos e threads eficiente;
- Escalonador com prioridades fixas;

- Uma gerencia de memória eficiente, de tal forma a garantir que segmentos de memória sejam reservados para um dado processo, eliminando a possibilidade de paginação deste segmento em disco;
- Comunicação entre processos, com semáforos, mensagens e memória compartilhada;
- Relógios e temporizadores de alta velocidade e precisão;
- Acesso direto a dispositivos de Entrada e Saída, eliminando a latência gerada por um driver de dispositivo;
- Mapeamento de memória física, possibilitando o mapeamento de um endereço físico no espaço de endereçamento virtual;
- Gerencia de interrupções, possibilitando criar rotinas que são associadas a determinadas interrupções. Esta característica junto com a característica de entrada e saída e de mapeamento da memória proporciona o controle total de um dispositivo em nível de usuário sem a necessidade de um *driver* de dispositivo para ele.

3. DESENVOLVIMENTO

3.1. A ESTRUTURA DO CINTURÃO

A estrutura do cinturão desenvolvida foi idealizada após os primeiros testes do transceptor de ultra-som para medição de distância já utilizando o microcontrolador ATmega8. Assim, teve-se um maior conhecimento das capacidades e limitações do microcontrolador. Além disso, foi levado em conta o número de componentes, a necessidade do robô em fazer medições simultâneas, a forma de comunicação com o robô e a padronização de cada dispositivo do cinturão.

A Figura 15 ilustra a composição do sistema e sua interface com o robô. Foram definidos dois módulos distintos que serão detalhados nas próximas seções: um módulo sensor com os transdutores e o circuito para emissão e recepção de ultra-som, e um módulo com o microcontrolador responsável principalmente pelo controle dos sensores a ele conectados, cálculo da distância com compensação de temperatura e pela comunicação com o robô.

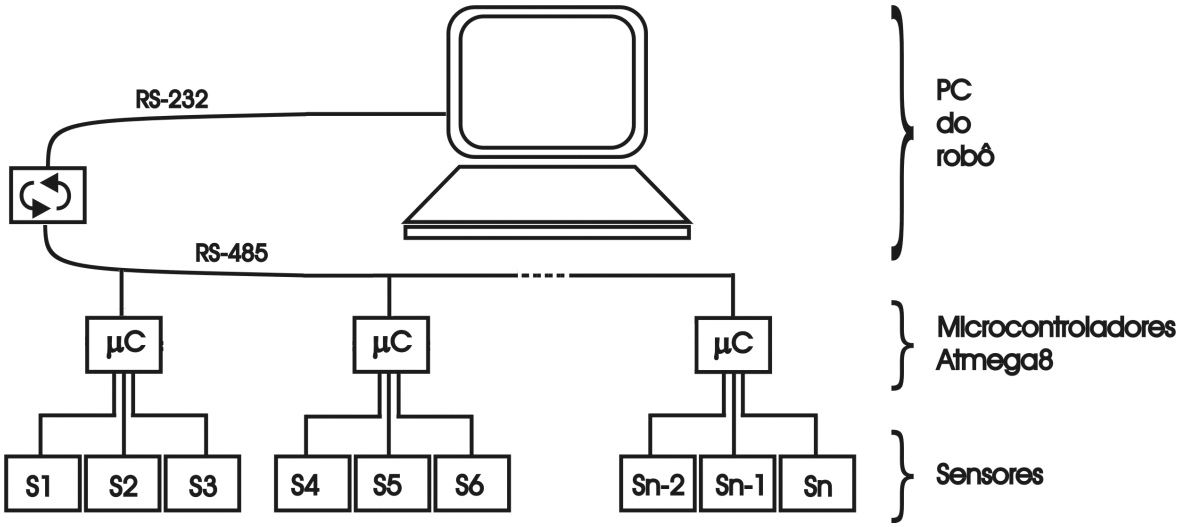


Figura 15 – Estrutura do cinturão

Cada módulo de controle pode suportar até três módulos sensores, representando já uma grande economia de microcontroladores. A limitação a três sensores se deve principalmente à quantidade de portas de I/O do AVR. Por outro lado, a existência de mais de um módulo controlador permite que se façam aquisições de distância ao mesmo tempo,

com cada microcontrolador utilizando apenas um sensor. Isso pode ser útil em alguns casos desde que o campo de atuação de um sensor não interfira no de outro.

Pelo fato da interface RD485 suportar até 32 dispositivos em um único canal, a transferência de dados se dá por um barramento RS485 de dois fios de modo *half-duplex*, onde o PC do robô é o mestre e os módulos de controle são escravos, isto é, os microcontroladores só respondem à requisições do PC. Um pequeno pseudoconversor de RS232 para RS485 também foi desenvolvido para interligar o PC do robô ao barramento.

A seção 3.2 mostra o desenvolvimento dos módulos sensores, os circuitos utilizados e a montagem mecânica. As interfaces com o microcontrolador, suas conexões e utilização é descrita na seção 3.3. A seção 3.4 mostra o projeto do conversor de RS232 para RS485. O protocolo de comunicação é descrito na seção 3.5, e as seções 3.6 e 3.7 explicam o *software* implementado no ATmega8 e no PC do robô respectivamente.

3.2. O MÓDULO SENSOR

Composto pelos circuitos de transmissão, recepção e os transdutores, o módulo sensor é responsável pela emissão do ultra-som através da excitação do cristal piezoelétrico, e pela detecção de ecos através do sinal gerado pelo cristal receptor, no qual é amplificado, filtrado e condicionado para o microcontrolador.

3.2.1. Circuito de transmissão

A Figura 16 ilustra o circuito de emissão do ultra-som. De fato, o pulso de excitação é gerado pelo microcontrolador e o circuito apenas eleva a tensão para aumentar a amplitude do ultra-som emitido, permitindo um maior tempo de vida do pulso durante sua propagação no meio. O responsável por essa conversão de tensão é o circuito integrado MAX232 que possui dois *drivers* elevadores de +5V para ±10V, quando sem carga.

O sinal de excitação nada mais é do que um trem finito de pulsos na freqüência de emissão do transdutor, no presente caso, foram utilizado 15 pulsos. O transistor da figura apenas inverte esse sinal na entrada do segundo *driver* do MAX232. Dessa forma, ter-se-ia entre os terminais do cristal ora uma tensão de +20V, ora uma tensão de -20V que equivaleria a uma tensão de 40V pico-a-pico, valor máximo especificado pelo fabricante.

Na prática, devido ao efeito de carregamento, foi verificado que a tensão de saída dos *drivers* caía para cerca ±8V, resultando cerca de 32Vpp no transdutor.

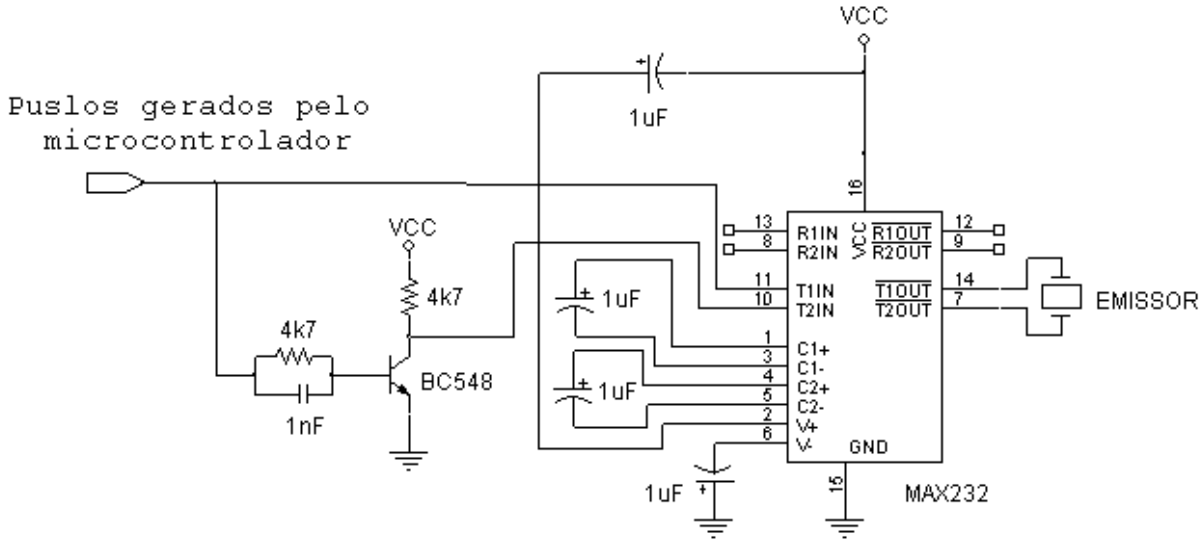


Figura 16 – Circuito emissor

3.2.2. Circuito de recepção

O esquemático do circuito de recepção é mostrado na Figura 17. O cristal receptor é ligado em paralelo a uma resistência com um dos terminais aterrados. Isso garante que a tensão no transdutor seja zero quando nenhum sinal estiver sendo detectado. O sinal gerado pelo cristal passa então por um capacitor de desacoplamento e é em seguida amplificado. O circuito integrado utilizado foi o TL074 que contém quatro amplificadores operacionais. A taxa de *slew rate* é de 13V/μs. O primeiro, seguindo o esquemático da figura, amplifica em cerca de 56 vezes o sinal recebido utilizando configuração inversora. Como o produto ganho-banda passante desse amplificador é de apenas 3MHz, o ganho para uma freqüência de corte a 40kHz não poderia ser maior que 75. Outra questão é que o capacitor de desacoplamento DC implementa um circuito passa-altas, na qual para os valores de resistência e capacitância utilizados, tem uma freqüência de corte de 12.6kHz, que é abaixo da freqüência do ultra-som.

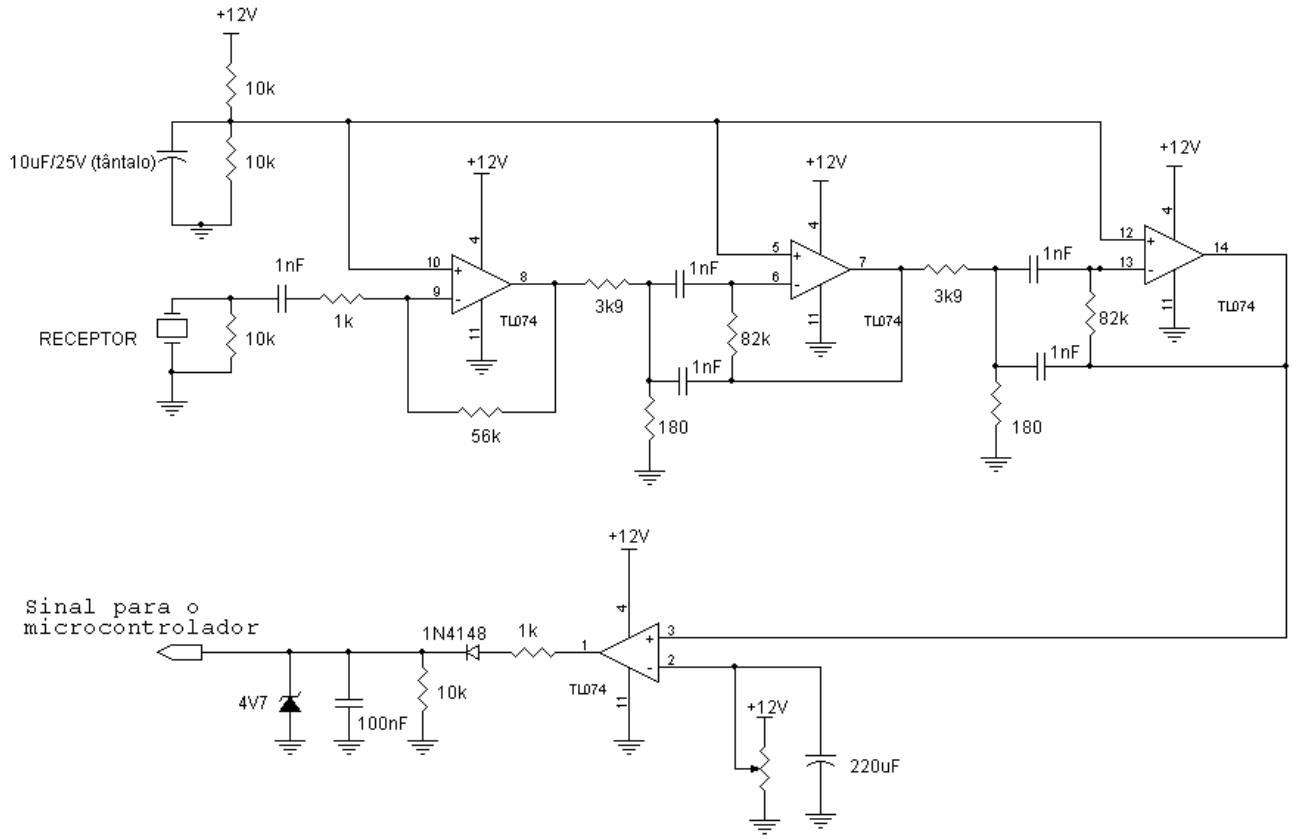


Figura 17 – Circuito de recepção

Os amplificadores seguintes são dois filtros passa-faixa de segunda ordem ligados em cascata para aumentar o fator de qualidade. Possuem freqüência central de 40kHz, ganho de 10 e fator de qualidade 10, minimizando componentes de freqüências fora da faixa de 40kHz. O último amplificador operacional trabalha como um comparador saturando o sinal em cerca de 1V e 11V, devido à alimentação assimétrica do amplificador operacional.

O restante do circuito nada mais é do que um detector de envoltória, formado pelo diodo, resistor e capacitor para restaurar o pulso que foi modulado em ultra-som, e um "ceifador" constituído pelo zenner de 4V7 para adequar o sinal à entrada do microcontrolador. Apesar da freqüência de corte 1/RC do detector de envoltória ser de apenas 10kHz, ele apresenta uma desvantagem por atrasar o sinal, tornando as transições lentas e fazendo o sinal perder sua característica digital que tinha na saída do comparador.

Por outro lado, há uma grande vantagem advinda do detector de envoltória no que diz respeito à aplicação do cinturão na detecção de obstáculos em robôs móveis. Como foi

visto na seção 2.1.6, para medição de distância, há uma distância mínima entre o obstáculo e o sensor, dependente da largura do pulso emitido. Se o obstáculo estiver a uma distância menor que a distância mínima, o sinal pode não ter sido completamente emitido e já estar sendo detectado ocorrendo ondas de choques destrutivas, além do mais, inúmeras reflexões podem ocorrer entre o obstáculo e o sensor sem espaçamentos entre eles, fazendo com que qualquer medição nessas condições seja errada. Já com o detector de envoltória também não se consegue medir, porém nessa situação a saída se mantém em nível alto, sendo fácil com o *software* verificar se há um objeto a uma distância menor que a mínima, apenas verificando o nível do sinal logo após o término do envio dos pulsos de excitação.

Outra questão em relação ao sinal de saída é que seu valor mínimo é por volta de 1V por causa da alimentação assimétrica, e o microcontrolador já toma como nível alto valores acima de 0,6V. Para resolver esse problema e também o da transição lenta foi utilizado o comparador analógico do ATmega8 com referência externa como será mostrado na seção 3.3.

3.2.3. Montagem mecânica

Um ponto importante a ser ressaltado é a montagem mecânica do circuito e transdutores. Devido a restrições de espaço no robô e à sua grande dimensão, os módulos, além de pequenos, não devem extrapolar as dimensões do robô, devendo ficar internos ao mesmo. Outro problema é a fixação dos transdutores, pois, por acoplamento mecânico, o cristal receptor pode também vibrar durante a emissão, atrapalhando a medição.

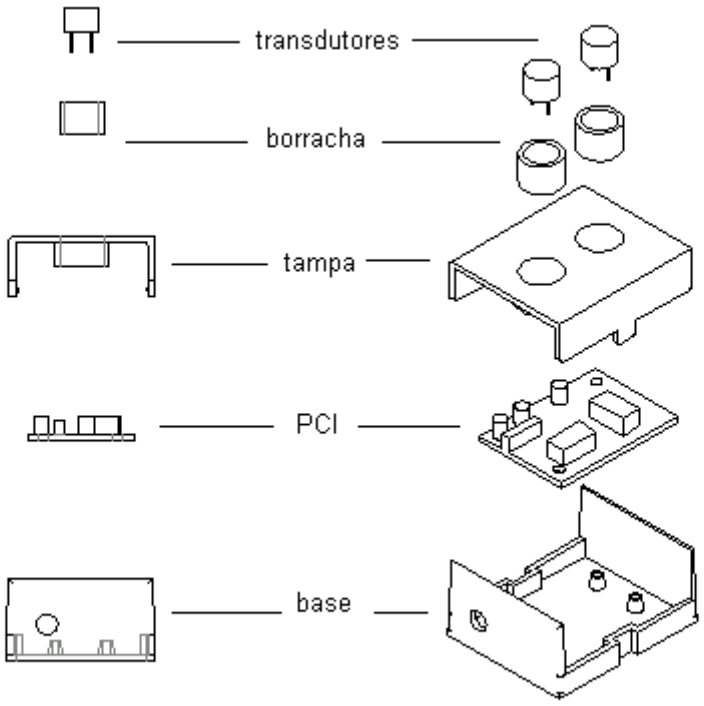


Figura 18 – Montagem mecânica

A Figura 18 abaixo ilustra a caixa utilizada para fixação dos sensores e da placa de circuito impresso. A caixa possui dimensões de 40 x 70 x 85 mm e os transdutores são fixados em tubos de ¾'' revestidos com borracha EVA de 5mm. A conexão dos transdutores à placa de circuito impresso é feita com cabos blindados. Um pequeno furo na lateral da base da caixa permite acesso para ajuste do *trimpot* da entrada do comparador.

3.3. O MÓDULO CONTROLADOR E COMUNICADOR

O coração deste módulo é o microcontrolador ATmega8, responsável por realizar a medição da distância em si, desde a emissão do pulso, detecção de eco e cálculo da distância com compensação de temperatura, bem como responder prontamente a comandos do mestre, o robô.

O circuito integrado DS485 é apenas um conversor de níveis de tensão TTL para tensões diferenciais balanceadas, que são as utilizadas no barramento RS485. Como a comunicação no barramento é *half-duplex*, o CI permite seleção entre modo de escrita no

barramento ou leitura do mesmo, escolha esta que é feita através de um pino de i/o do microcontrolador.

Há três conectores do tipo RJ45 fêmeos para conexão dos módulos sensores. Cada um desses conectores possui oito contatos, sendo dois de alimentação, um para emissão dos pulsos de ultra-som e outro para recepção. Os quatro contatos restantes foram utilizados para detecção de conexão do módulo sensor quando um curto é verificado entre eles, pois é gerada uma mudança de nível lógico em uma das entradas do ATmega8.

O endereçamento dos módulos é feito externamente por três pequenos conectores que, da mesma forma que o conector dos sensores, tem o nível lógico de um de seus pinos lido pelo microcontrolador, obtendo-se, dessa forma, uma faixa de oito endereços e garantindo a característica modular do cinturão. Já os módulos sensores são endereçados indiretamente a partir do endereço base do microcontrolador e do conector ao qual está interligado.

Outro componente de importante valia é o sensor de temperatura LM35, responsável por medir a temperatura do ambiente necessária para obtenção da velocidade do som. Com apenas três pinos, dois de alimentação e um de saída, o LM35 retorna uma tensão proporcional à temperatura. A relação é praticamente linear sendo de $10mV.ºC^{-1}$ e sem deslocamento, isto é, uma tensão de 0V equivale a uma temperatura de 0 ºC. A saída é ligada diretamente a uma das entradas analógicas do ATmega8.

Três saídas digitais geram os pulsos de excitação, uma para cada sensor, e outras três entradas analógicas recebe o pulso de eco. A Tabela 1 mostra a utilização dos pinos do ATmega8. O módulo também possui uma interface para programação do microcontrolador no próprio circuito.

Tabela 1 – Utilização dos pinos do microcontrolador ATmega8

Utilização	Pino	Descrição
Alimentação	07 VCC	+5V
	08 GND	0V
Cristal de oscilação	09 XTAL1	Entrada de *clock*
	10 XTAL2	Saída de *clock*
Comparador Analógico	12 AIN0	Entrada negativa do comparador
Comunicação RS485	02 RX	Canal de recebimento de dados seriais
	03 TX	Canal de envio de dados seriais

	04 PD4	Seleção do modo leitura/escrita do DS485
Endereçamento	27 PC4	Bit 0 (LSB)
	28 PC5	Bit 1
	13 PD7	Bit 2 (MSB)
Sensor de temperatura LM35	26 ADC3	Entrada analógica para conversor A/D
	20 AVCC	AVCC – Alimentação do conversor A/D
Programador	01 RESET	Pára ou inicia execução do programa
	17 MOSI	Entrada serial de dados
	18 MISO	Saída serial de dados
	19 SCK	*Clock* serial de gravação
Sensor 1	14 PB0	Saída do pulso de excitação para sensor 1
	23 ADC0	Entrada analógica do pulso de eco do sensor 1
	05 PD3	Estado da conexão do sensor 1
Sensor 2	15 PB1	Saída do pulso de excitação para sensor 2
	24 ADC1	Entrada analógica do pulso de eco do sensor 2
	06 PD4	Estado da conexão do sensor 2
Sensor 3	16 PB2	Saída do pulso de excitação para sensor 3
	25 ADC2	Entrada analógica do pulso de eco do sensor 3
	11 PD5	Estado da conexão do sensor 3

A freqüência de trabalho do microcontrolador é dada por um cristal de oscilação externo de 11,0592MHz. A escolha pelo uso de um cristal foi devida principalmente pela precisão, já que o oscilador interno de 8MHz possuía uma grande variação com a temperatura. Já a opção por este valor de freqüência em específico foi por permitir comunicação serial a taxas padrões de até 230kbps sem erros de divisão de freqüência.

3.4. O MÓDULO DE INTERFACE COM O PC

O módulo de interface com o PC é composto de dois blocos: O primeiro faz a conversão dos níveis lógicos RS232 para níveis TTL, e o segundo bloco é um transceptor RS485 para a comunicação com o barramento. Como a porta Serial foi usada para comunicar-se com um barramento RS485, muitos dos sinais de sincronização presentes no padrão RS232 foram ignorados, assumindo que sempre há um dispositivo presente na outra ponta, simplificando a comunicação. Apenas o pino RTS foi utilizado para o controle de acesso ao baramento via software. Assim, os únicos pinos utilizados foram TX, RX, RTS e GND. Esta escolha demandou a criação de uma biblioteca especifica para nossa aplicação, já que as funções típicas da API do Windows NT não dão suporte ao controle do pino RTS.

3.5. O PROTOCOLO DE COMUNICAÇÃO

O protocolo de comunicações entre PC e dispositivos foi concebido levando em conta a topologia da rede e a característica da aplicação. Como o PC apenas faz requisições e os AVR's apenas respondem quando solicitados, utilizamos controle centralizado, onde o PC é o mestre e todos outros dispositivos são escravos. Esta estratégia de controle de acesso mostrou-se mais eficaz para nossa aplicação por eliminar algoritmos de detecção de colisão e algoritmos de acesso ao meio complicados presentes nos algoritmos de controle distribuído. A multiplexação da utilização do barramento foi feita na base do tempo, de forma assíncrona, visto que multiplexação em freqüência exigiria construção de filtros especiais para cada nó de comunicação, complicando enormemente a implementação. A mutliplexação por divisão no tempo síncrona poderia também ser uma solução que diminuiria o numero de dados trafegando no barramento, porém, teríamos que usar datagramas maiores contendo as infromações de sincronização, além do que os AVR's precisariam utilizar mais um timer para controlar sua transmissão, o que seria inviável.

Como o trafego de informações deve ser em tempo real, nunca poderíamos basear nossa camada de enlace em um reconhecimento com retransmissão. Então a única coisa que fazemos quando o PC aguarda uma resposta de um AVR, e esta resposta não chega em um dado prazo, ou chega uma resposta inesperada, simplesmente reconhecemos este AVR como desativado.

Assim sendo, os mecanismos de controle ao meio de nosso protocolo pode ser descrito como: Controle centralizado, multiplexado por divisão no tempo assíncrona, com tolerância à falhas.

As mensagens trocadas neste protocolo são bastante simples, onde temos apenas um byte de cabeçalho e dois bytes de conteúdo. O PC envia apenas o cabeçalho, já os AVR's podem mandar apenas o cabeçalho ou o cabeçalho seguido de dois bytes de conteúdo, o que torna o protocolo não uniforme. O byte de cabeçalho contém 4 informações importantes, listados abaixo (do bit mais significativo para o menos significativo), conforme na Figura 19:

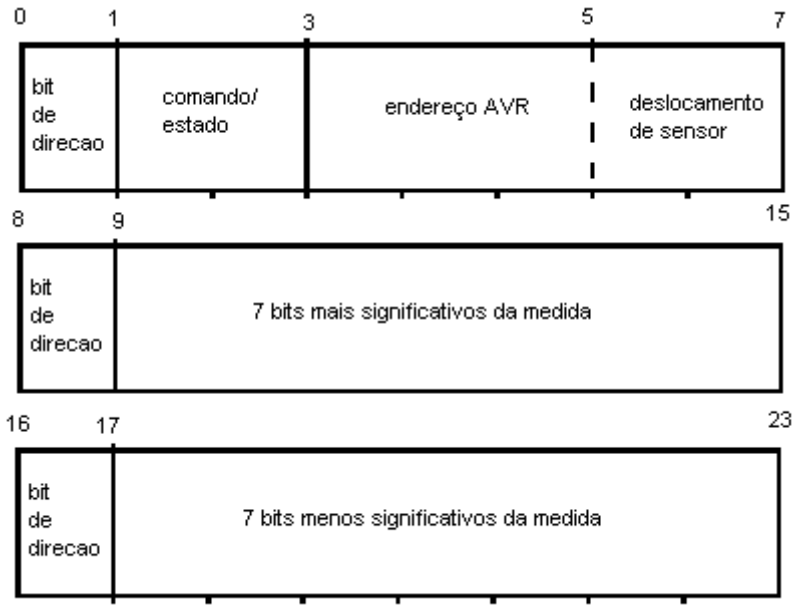


Figura 19 – Cabeçalho do protocolo

- Bit 0 – direção: se o valor estiver setado, é por que o PC enviou o byte. Caso esteja zerado, significa que algum AVR o enviou. Isto elimina a possibilidade de algum dado enviado por um AVR ser confondido com um comando vindo do PC.

- Bits 1 e 2 – comando e resposta: Este campo indica, em um byte enviado pelo PC, a requisição feita ao AVR, que pode ser um pedido de Status, medição ou de retorno de dados. Já quando enviado pelo AVR, este campo pode significar que o dispositivo está pronto, efetuando uma medição, ou que o sensor requisitado está inativo.

- Bits 3 a 7 – endereço: o campo de endereço ainda pode ser quebrado em dois níveis: bits de 3 a 5 representam o endereço do AVR, e os bits 6 e 7 indicam qual dos 3 sensores o byte se refere.

Desta forma podemos construir nossa aplicação transparente em relação aos AVR's, ou seja, a aplicação de tempo real irá endereçar sensores distintos, dando ao protocolo uma forte característica modular.

As mensagens enviadas do PC para os ATMega8 são construídas setando o bit de direção, os bits de comando pertinentes e o endereço do ATMega8 junto com o deslocamento de sensor. Quando a mensagem deve ser respondida, o ATMega8 zera o bit de direção de todos os bytes, repete seu endereço junto com o deslocamento do sensor em questão no campo de endereço e no campo comando ou status ele diz ao PC que tipo de resposta está enviando. A mensagem de resposta do ATMega8 pode ter 1 ou 3 bytes.

A mensagem do tipo comando status é utilizada para termos conhecimento se um dado sensor de um dado ATMega8 do barramento está conectado ou desconectado. Se o sensor pesquisado estiver conectado, o ATMega8 responderá com apenas o byte de cabeçalho, com o campo de status setado como "pronto". Caso contrário, retornará com status inativo. A mensagem do tipo "comando medir" é utilizada para disparar um ciclo de medida do sensor seleconado. Este comando não causa uma resposta por parte do ATMega8. Por último, a mensagem do tipo "retorno de dados" faz com que o ATMega8 sempre responda com 3 bytes de dados, sendo um de cabeçalho e os outros dois contendo, cada um, os 7 bits mais significativos e os 7 bits menos significativos do valor inteiro da distância medida expressa em milímetros. Como o eco de ultra-som leva um certo intervalo para ser processado dependendo da distância do objeto, pode acontecer de que o PC execute um comando "retorno de dados" antes que esta medida esteja completa. Neste caso, o ATMega8 responderá prontamente com o campo de status de seu byte de cabeçalho setado como "medindo", e os 2 bytes de conteúdo zerados.

Sempre que uma resposta é esperada e ela não chega em um determinado tempo esperado, o ATMega8 que falhou é considerado como desligado, e não será interrogado novamente até que se faça uma nova pesquisa de status e ele esteja ligado.

3.6. O *SOFTWARE* DO MICROCONTROLADOR ATMEGA8

Como visto na seção 3.3, o microcontrolador é responsável por diversas funções que podem ser relacionadas a três processos distintos: comunicação, medição e verificação das conexões e estados dos sensores.

As funções relacionadas à comunicação englobam receber e interpretar comandos recebidos do mestre e responder imediatamente a eles. Já as funções de medição consistem em emitir o pulso de excitação, cronometrar o tempo, receber o pulso ecoado dentro de uma janela de tempo, medir a temperatura, e calcular velocidade do som e a distância ao obstáculo, enquanto que as funções restantes dizem respeito ao estado de conexão dos sensores, isto é, desconectados ou não.

Estes três processos ocorrem de maneira concorrente no microcontrolador, uma vez que ele deve estar sempre apto a realizar uma medição, a tratar a comunicação, além de estar continuamente averiguando o estado de cada sensor. De fato, por estar-se usando um sistema de tempo real determinístico no robô, o processo de comunicação torna-se prioritário. Dessa forma, todo o seu controle é feito por interrupção de hardware, o que permite uma resposta mais imediata possível.

Quanto ao processo de medição, também grande parte de suas funções são feitas por interrupção de hardware, como geração dos pulsos a 40kHz, recebimento do eco e o anúncio de fim da janela de tempo para recebimento do eco. A tarefa mais crítica, que seria a cronometragem do tempo, é feita de forma paralela por um contador que é incrementado pelo *clock*. Assim, a maior parte do processamento no processo de medição é gasta na conversão analógico-digital da temperatura e nos cálculos de velocidade e distância, que não chega a ser nada crucial em termos de custo de execução. Todo o tempo de processamento restante é então utilizado para realizar uma varredura nos pinos que indicam a conexão de cada um dos três sensores, sendo o processo de menor prioridade. A Figura 20 ilustra o diagrama simplificado do programa principal. As subseções seguintes detalham o funcionamento de cada processo.

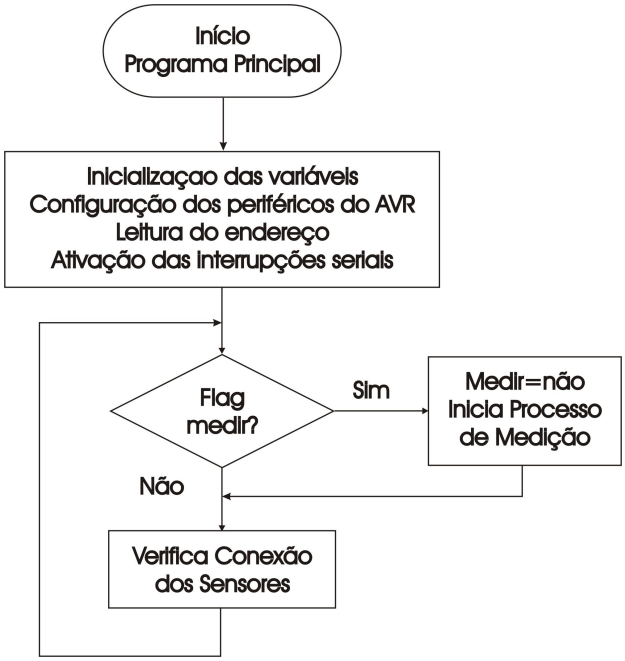


Figura 20 – Diagrama do programa principal do ATmega8

3.6.1. Processo de comunicação

Há basicamente uma função de transmissão e outras duas rotinas de interrupções utilizadas para comunicação serial. Uma das interrupções sinaliza o recebimento de um byte, empregada para tratar o protocolo, enquanto que a outra indica o fim de uma transmissão, isto é, quando o buffer de transmissão se esvazia, que é usada para voltar o DS485 para o modo de leitura do barramento. As figuras abaixo ilustram o decorrer dessas rotinas de interrupção e da função utilizada.

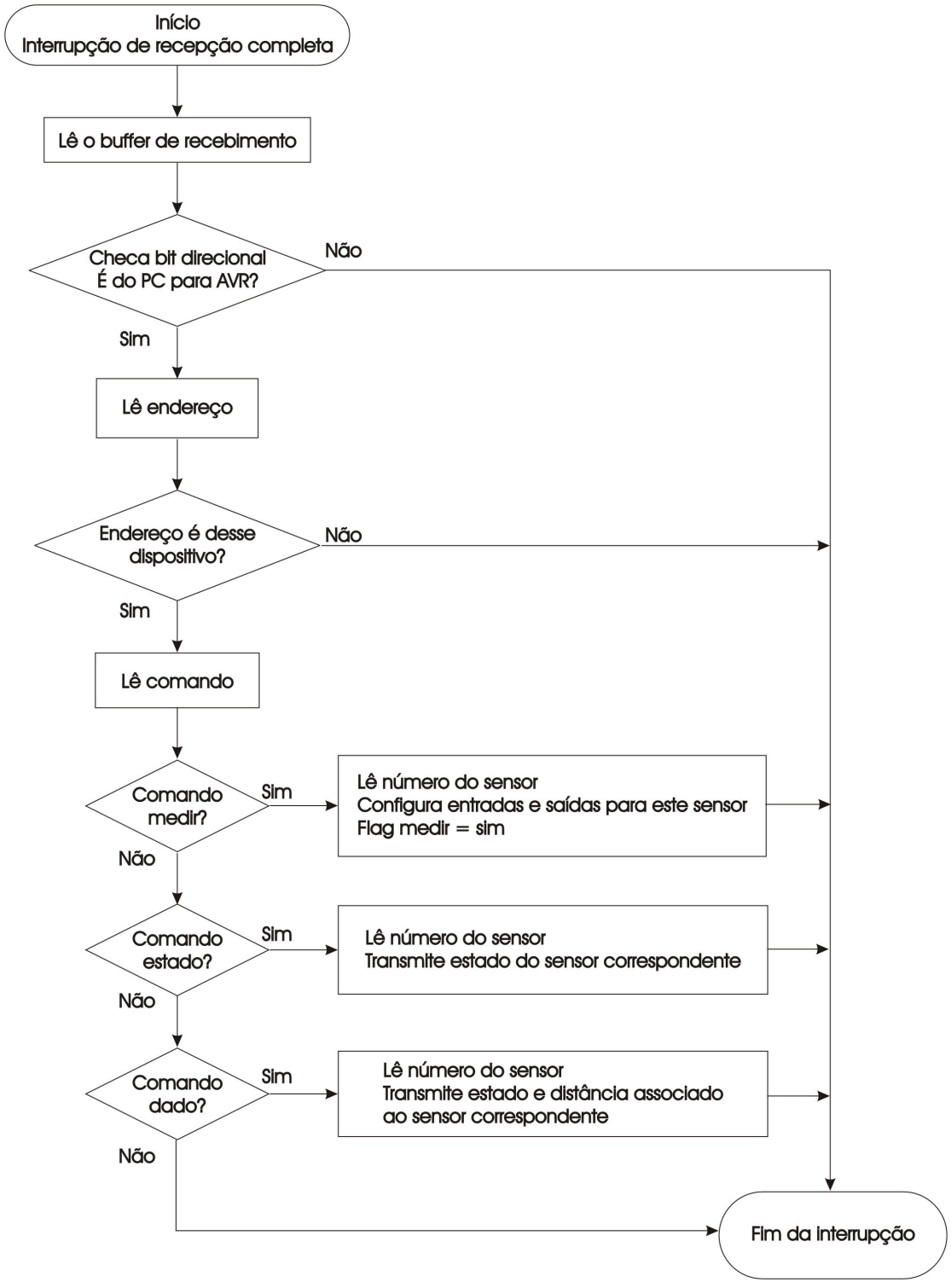


Figura 21 – Diagrama da rotina de interrupção de recepção completa

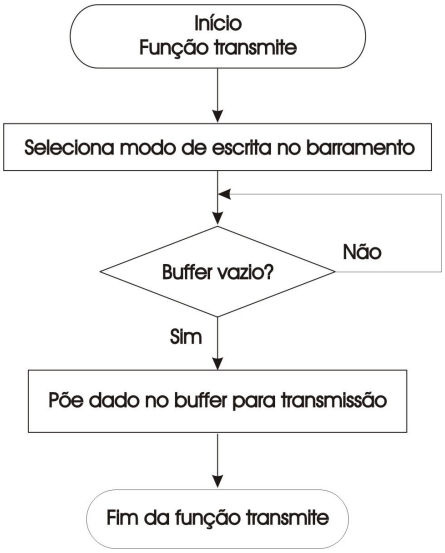


Figura 22 – Diagrama da rotina da função de transmissão

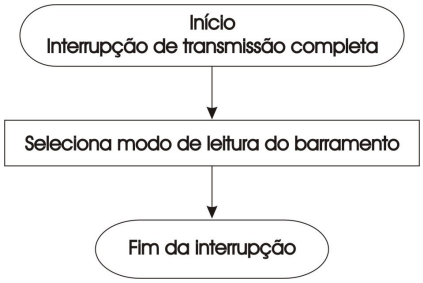


Figura 23 – Diagrama da rotina de interrupção de transmissão completa

3.6.2. Processo de medição

O processo de medição é constituído por uma função e três interrupções. A função dá início ao processo configurando o hardware e habilitando algumas interrupções que dão continuidade ao processo.

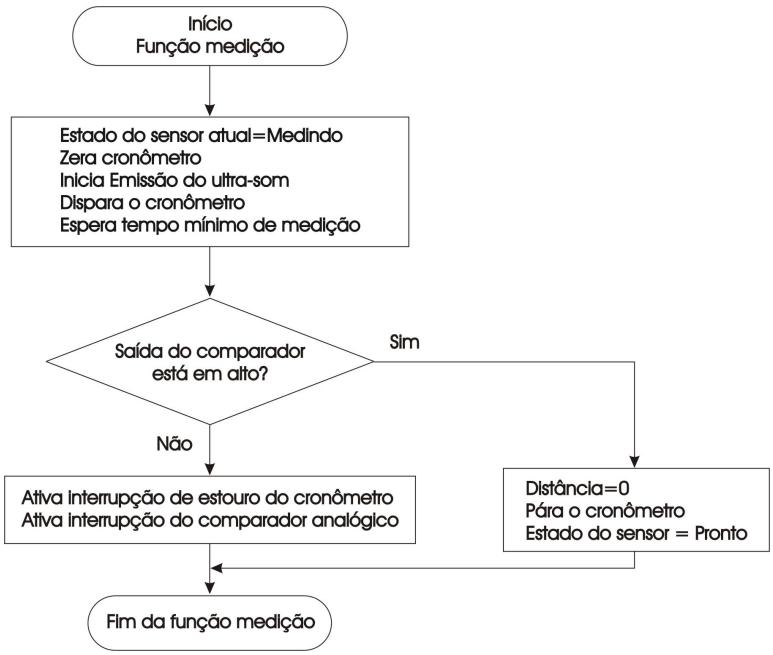


Figura 24 – Diagrama da função que dá início ao processo de medição

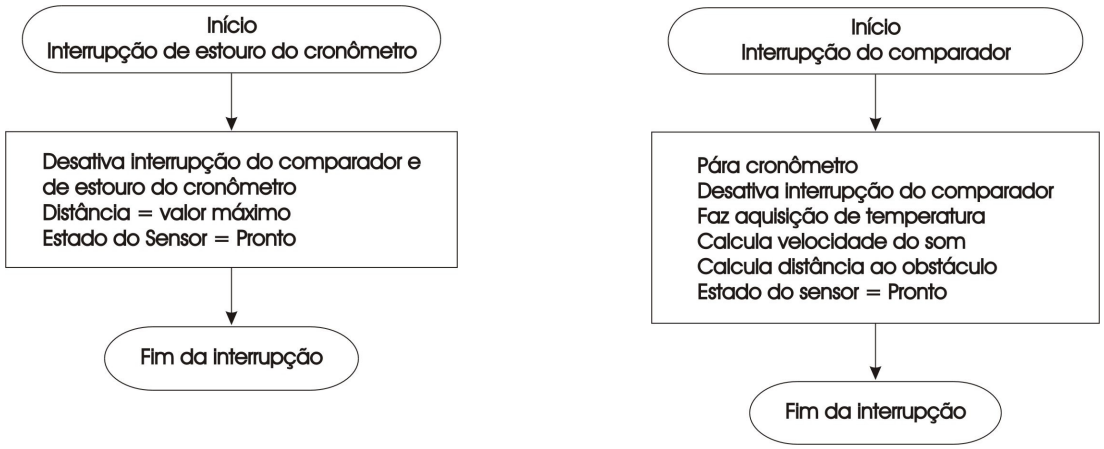
As três interrupções utilizadas são:

1. Estouro do timer0 – trata-se de um contador de 8 bits utilizado para gerar pulsos à freqüência de 40kHz. A rotina é configurada para que gere uma interrupção a cada 12,5μs;

2. Estouro do timer1 – o timer1 é contador de 16 bits utilizado como cronômetro. É incrementado na freqüência do *clock* dividido por 8, isto é, a 1,2834MHz ou a cada aproximadamente 0,72μs. Seu estouro indica tempo máximo de espera pelo eco, que, no caso, é de 47,4ms equivalendo a uma distância máxima de cerca de 8,12 metros a 20 ºC;

3. Borda de subida do comparador analógico – esta interrupção indica chegada de um pulso de eco. Em sua rotina o cronômetro é parado, sua interrupção é desativada para evitar detecção de múltiplos ecos, é feita uma aquisição de temperatura e é então calculada a distância ao obstáculo.

Cabe ressaltar que as indicações para obstáculo a uma distância menor que limite e para não recebimento de eco são distância igual a zero e distância igual ao valor máximo respectivamente.



(a) (b)

Figura 25 – Diagrama de interrupção (a) do estouro da janela de recepção de eco e (b) do comparador analógico

3.6.3. Verificação do estado de conexão dos sensores

Além dos estados medindo e pronto, os sensores podem assumir o estado inativo quando não conectados ao microcontrolador. A rotina que faz essa verificação ocorre no laço principal do programa como mostrado na Figura 20. Sua descrição é mostrada no diagrama abaixo.

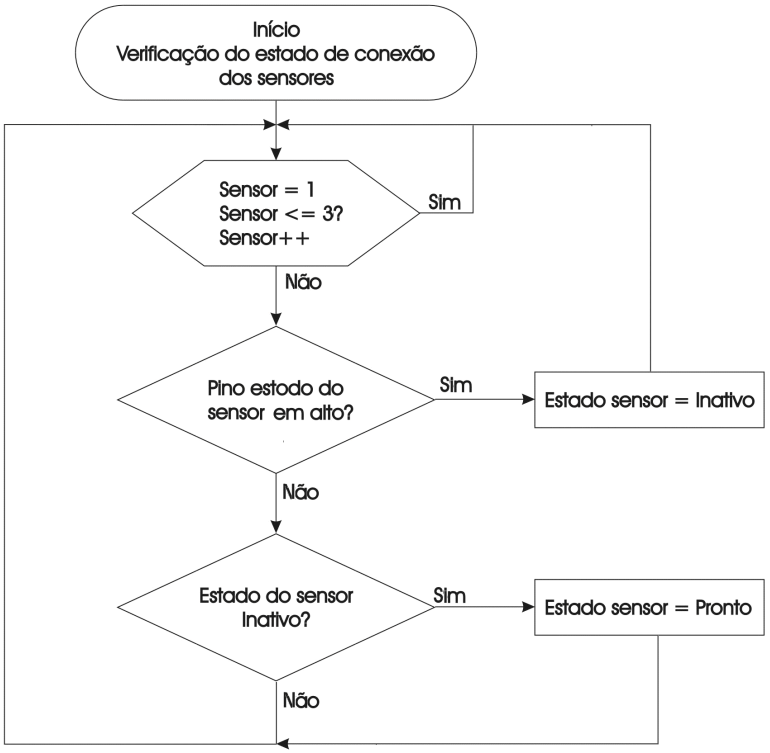


Figura 26 – Diagrama do processo de verificação

3.7. O *SOFTWARE* DO ROBÔ

O Robô comunica-se com o barramento RS485 requisitando e coletando as medidas dos ATMega8 e seus sensores através de uma rotina em tempo real desenvolvida na plataforma RTX. Isto é feito sob a forma de uma tarefa periódica, onde em cada disparo se faz uma varredura de todos os sensores do cinturão, coletando suas medidas e identificando sensores desconectados. O intervalo de disparo desta tarefa deve ser escolhido cuidadosamente para que seja extenso o suficiente de modo que seja possível fazer varredura completa de todos os sensores considerando que todos demorem seu tempo máximo de medição, i.é., ocorra o estouro da janela de tempo para recebimento do eco (e.g. uma situação onde o omni está navegando em um espaço vazio sem obstáculos), ou então a característica determinística do RTX pode ser perdida (i.e. uma tarefa lançada antes do término de outra em execução). O algoritmo desta rotina encontra-se no fluxografo da figura 27.

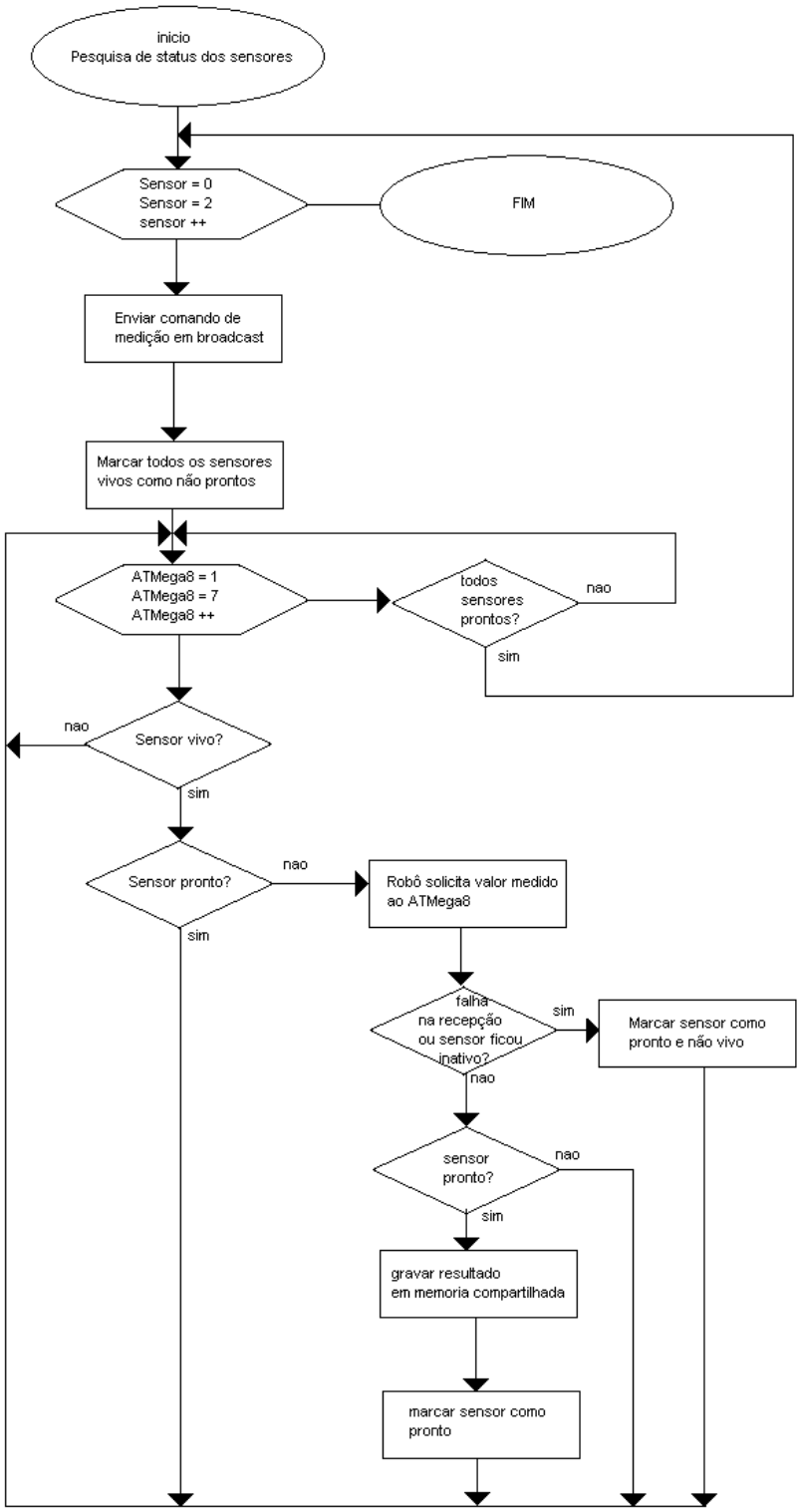


Figura 27 – Fluxograma do programa no PC

Os resultados medidos são gravados em uma memória compartilhada, e a partir daí, os dados coletados podem ser utilizados por qualquer aplicação desenvolvida pelo

programador do robô, que pode ir de um simples monitoramento a um elaborado algoritmo de desvio de obstáculos.

4. AVALIAÇÃO EXPERIMENTAL

4.1. MÓDULO SENSOR

4.1.1. – Circuito de transmissão

Foi verificado que, na prática, devido ao efeito de carregamento, as tensões de saída dos *drivers* do circuito integrado MAX232 caíam para cerca de ±8V durante a transmissão do pulso, resultando cerca de 32Vpp no transdutor, ao invés dos 40Vpp originalmente previsto.

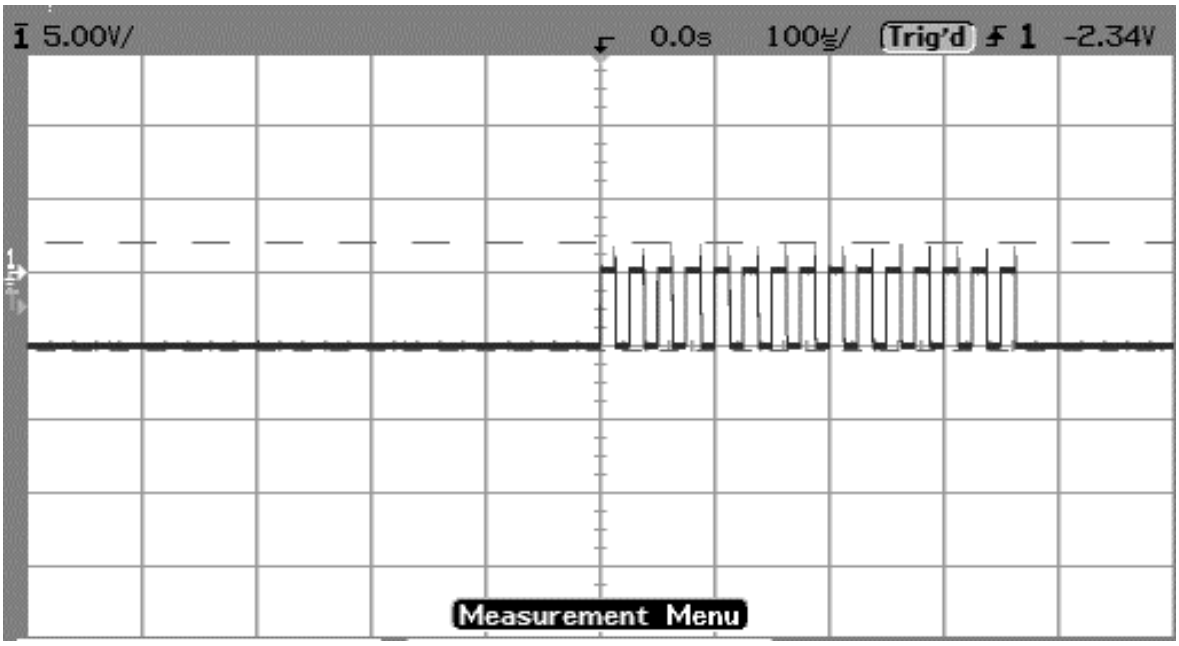


Figura 28 – Sinal de excitação gerado pelo AVR

Além disso, notou-se que apesar de excitarmos o cristal com apenas 15 pulsos, isto é, uma excitação de 375µs de duração a freqüência de 40kHz, o cristal, por inércia, continuava oscilando por um tempo até atenuar-se por completo.

4.1.2 – Circuito de recepção

Para o circuito de recepção, foi-se conferindo cada estágio do mesmo com o osciloscópio. A Figura 29 abaixo ilustra o sinal na saída do primeiro amplificador que dá um ganho de 56, juntamente com o sinal de excitação do transmissor.

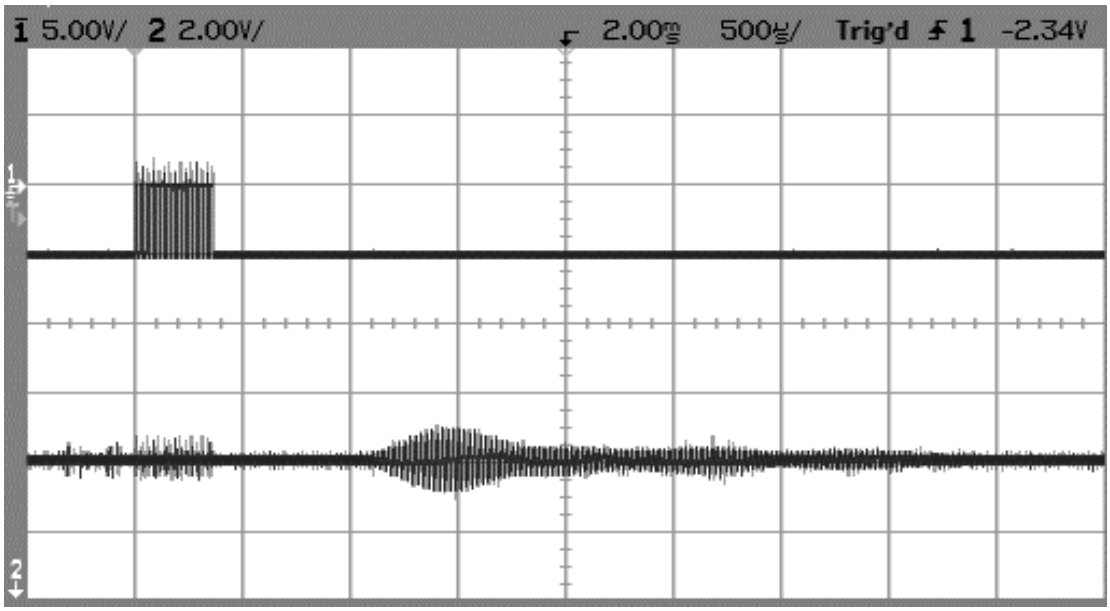


Figura 29 - Sinal após o primeiro amplificador

O ruído presente no sinal de recepção durante a transmissão dos pulsos pelo microcontrolador, que pode ser visto nas figuras, é proveniente de interferência elétrica, uma vez que este foi notado mesmo na ausência dos transdutores. Entretanto, ele não apresenta qualquer problema, visto que durante este período o sinal de recepção é ignorado. Na figura abaixo temos o sinal após os dois filtros em cascata. Note como o ganho melhora consideravelmente.

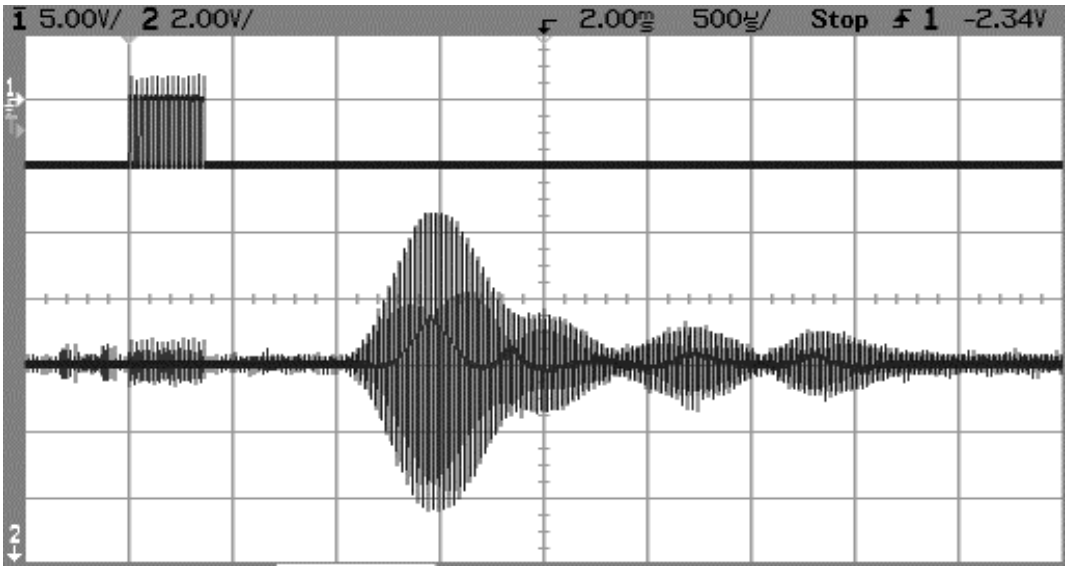


Figura 30 – Sinal após filtros em cascata

Os sinais de menor amplitude após o primeiro pulso são ecos gerados pelo primeiro. A proximidade entre eles é devido à pequena distância entre o sensor e o obstáculo. Depois o sinal é passado por um comparador onde é saturado. A saída é mostrada naFigura 31.

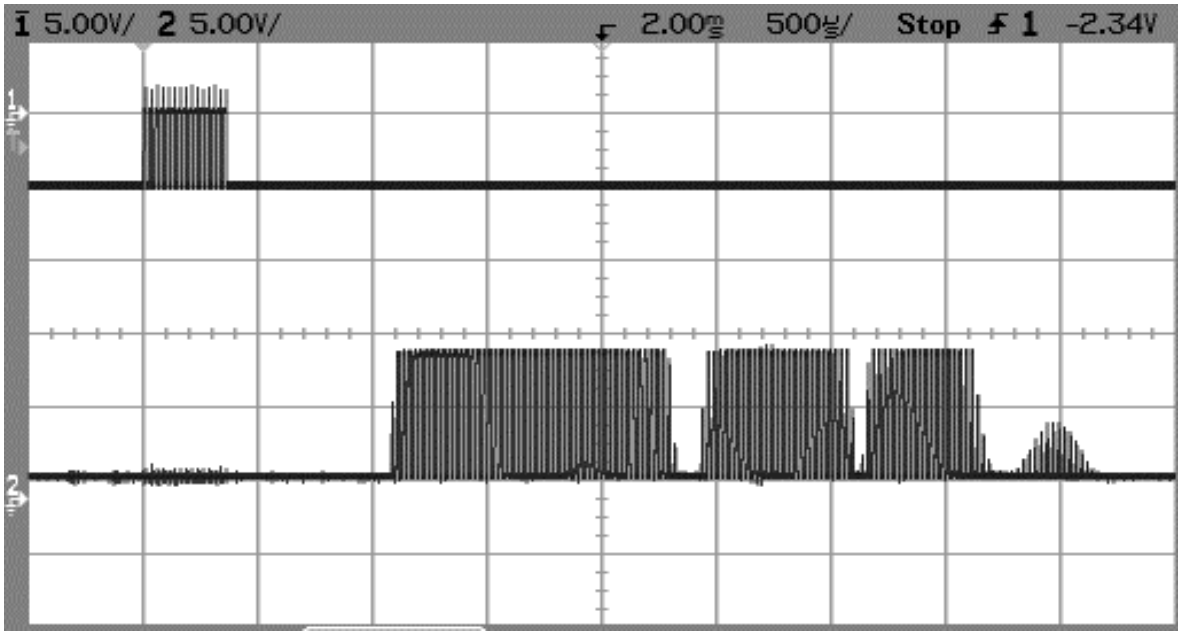


Figura 31 – Sinal após passar pelo comparador

Por fim, antes de ir para o microcontrolador, o sinal passa pelo detector de envoltória e pelo zenner ceifador resultando na forma de onda mostrada na Figura 32.

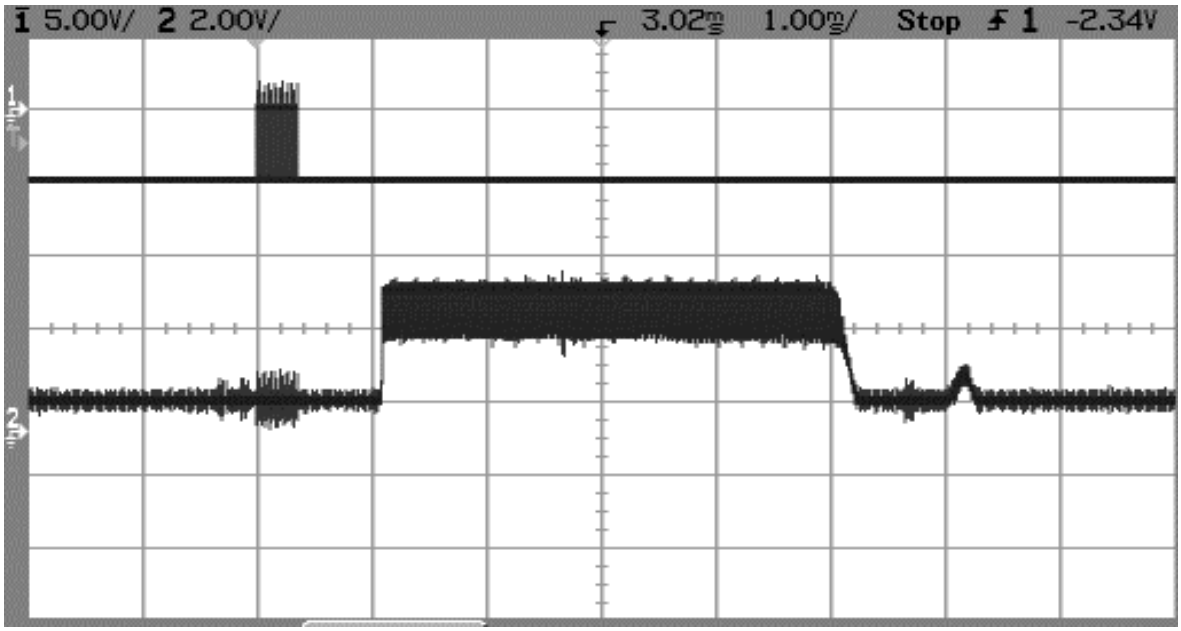


Figura 32 – Sinal de retorno para o microcontrolador

As figuras seguintes mostram os sinais retornados para obstáculos mais distantes. No primeiro deles é possível ver alguns pulsos, já que o obstáculo está mais próximo.

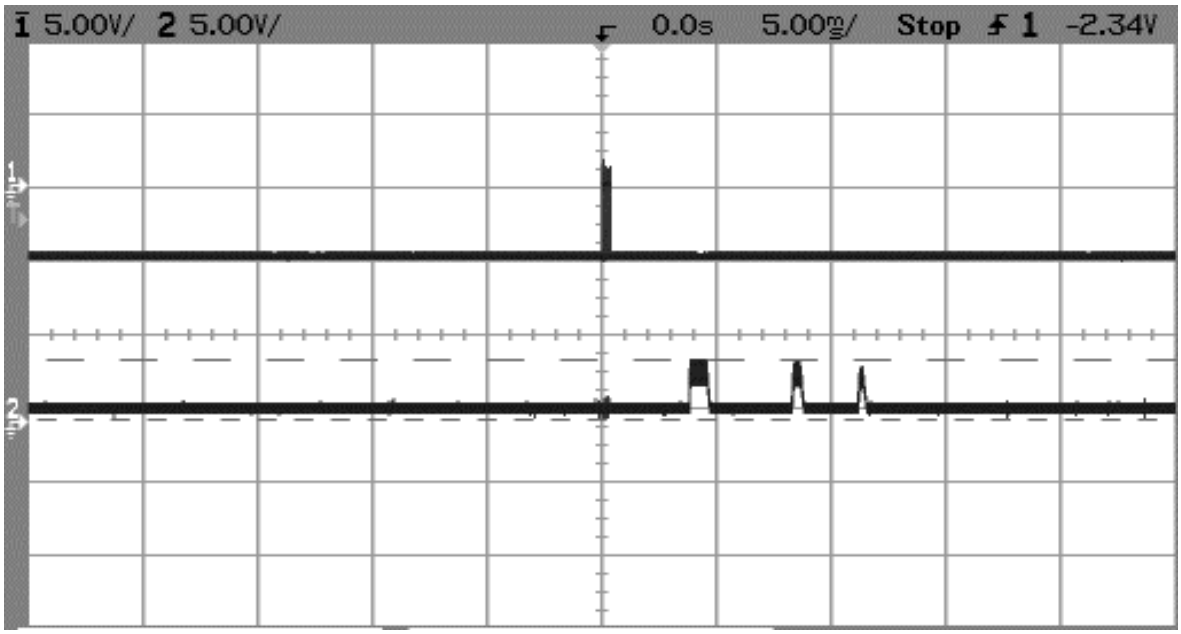


Figura 33 – Ecos de ultra-som

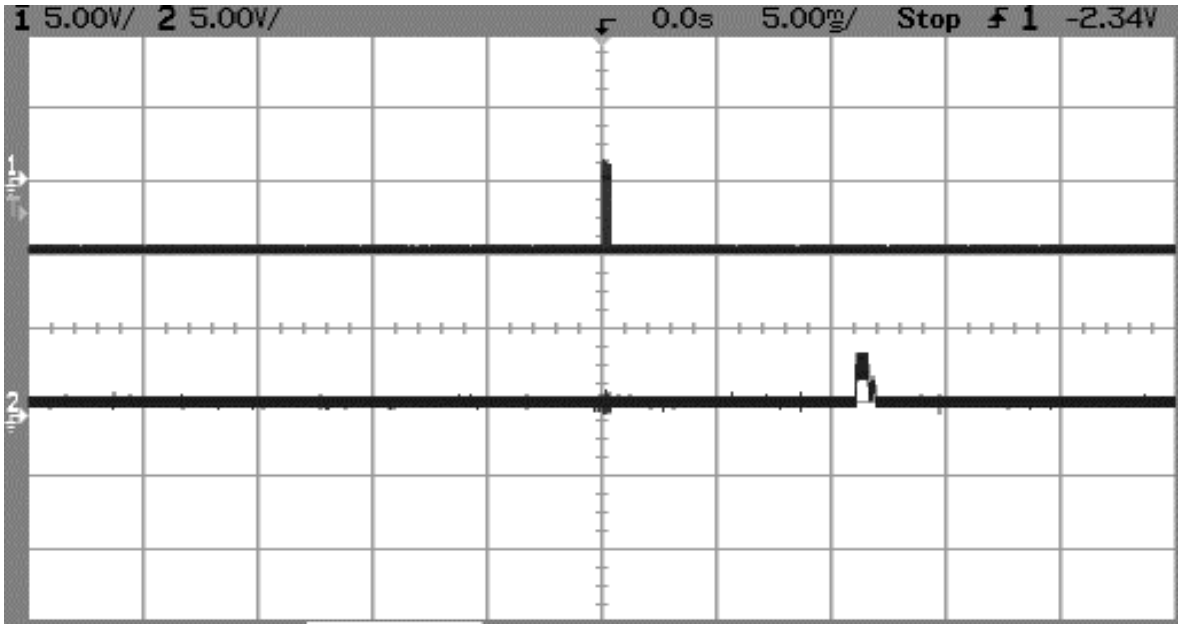


Figura 34 – Obstáculo a cerca de 2 metros

4.2. VALIDAÇÃO DAS MEDIDAS

De nada valeria um sistema de medição no qual não se poderia ter qualquer confiabilidade no valor medido. Assim, para validação das medidas, foi feito um procedimento no qual se variou a posição do sensor de cerca de 20 centímetros a 3 metros de uma parede em intervalos de 10 centímetros. A altura do sensor em relação ao chão foi de 1 metro e procurou-se também manter o ângulo de incidência normal à parede. Para cada posição foram coletadas 50 medidas de distância a fim de estimar-mos também a variância dos valores à medida que aumentássemos a distância.

É importante ressaltar aqui que, como a encomenda dos transdutores da murata chegou a menos de um mês da defesa deste projeto, os dados aqui obtidos foram coletados quando da conclusão do módulo sensor, isto é, da placa de circuito impresso do transceptor e da montagem mecânica. Os transdutores utilizados foram de mesma especificação em relação à freqüência nominal e dimensões.

Na época, foi utilizado apenas um microcontrolador ATmega8 com oscilador interno de 8MHz conectado diretamente a um computador. A um comando feito pelo usuário no computador, o microcontrolador executava exatamente 50 vezes a rotina de medição descrita na seção 3.6.2 em intervalos regulares de 200 milissegundos. As informações eram passadas via serial para o programa MatLab do computador onde eram

gravadas num arquivo texto. Além dos valores de distância medidos, foram também armazenados no arquivo o tempo de vôo e a temperatura em cada medição, bem como a distância real obtida com trena, fornecida pelo usuário.

A curva entre as distâncias estimadas e reais é mostrada na Figura 35 abaixo, juntamente com a reta ideal esperada. A curva mostrada foi obtida por ajuste de reta por mínimos quadrados, onde logo se nota uma característica linear.

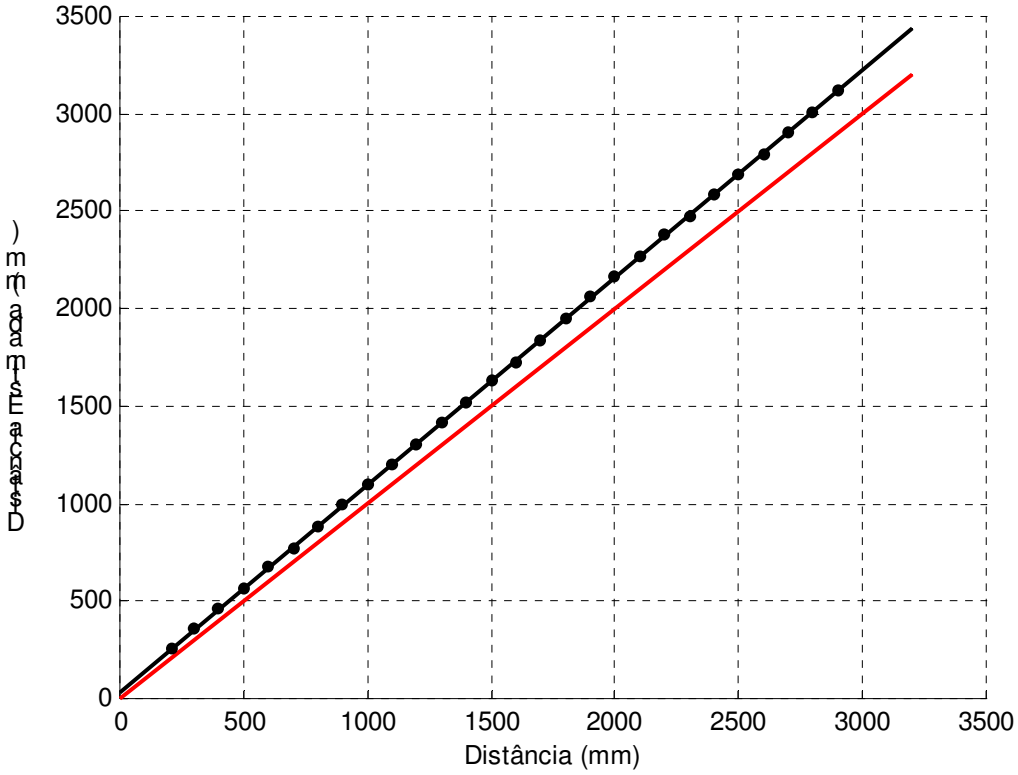


Figura 35 – Curva de validação das medidas

Já a Figura 36 mostra o desvio padrão das medidas em função da distância da parede. Um ajuste de reta por mínimos quadrados também foi feito, mostrando certa tendência do aumento da variância com a distância medida.

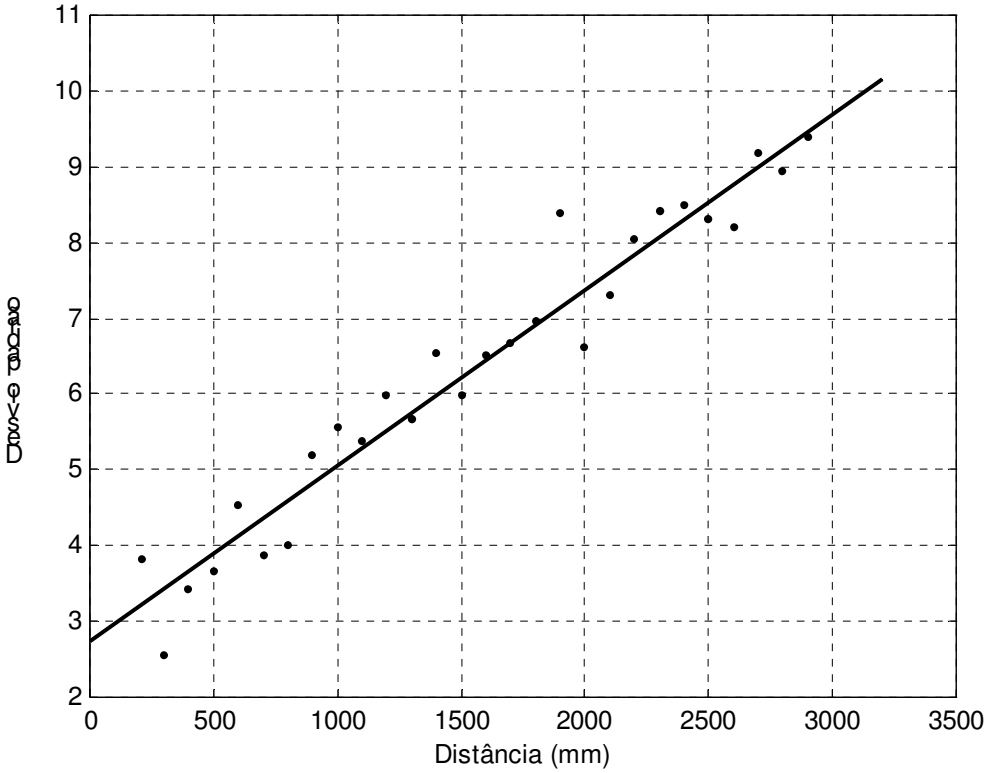


Figura 36 – Desvio padrão versus distância

Nota-se, a partir da Figura 35,que, apesar da linearidade do sistema, a curva diverge da reta ideal de resposta, apresentando um erro absoluto crescente chegando a 200mm de erro na distância de 3 metros. Como a maior parte do erro não é de *offset*, isto é, de deslocamento da curva obtida em relação à reta ideal, infere-se que a principal fonte de erro estaria na expressão [7] da velocidade do som, deduzida na seção 2.1.2. e utilizada na equação [12] para cálculo da distância, pois uma velocidade sobre-estimada quando multiplicada pelo tempo para obtenção da distância resultaria na característica da curva vista figura.

Já o circuito transceptor de ultra-som provocaria principalmente um erro de *offset,* no sentido que poderia atrasar o sinal de resposta ou tempo de detecção dependendo no nível do comparador.

Para resolver este problema, é necessário um procedimento de calibração para cada sensor. Porém, visto à falta de tempo, uma proposta de calibração será feita no capítulo 5, estimando-se novos parâmetros para a equação [7] e utilizando estes dados já coletados para ilustração e comparação com os resultados obtidos após o procedimento de calibração.

4.3. DESEMPENHO DO SOFTWARE

A validadação da funcionalidade do barramento RS485 e do protocolo de comunicações pode ser percebida pela simples verificação de que os resultados obtidos estão corretos. Para tal verificação, um programa que executa no ambiente do Windows NT apenas imprime na tela de forma organizada os valores gravados pelo programa que efetua os comandos de varredura dos sensores, que por sua vez, executa em ambiente RTX, em uma memória compartilhada.

Colocando apenas dois sensores dispostos lado a lado, relativamente alinhados, posicionou-se um objeto a aproximadamente 650 milímetros dos sensores, e os resultados obtidos, assim como o tempo levado para executar a varredura impressos na tela estão na Figura 37.

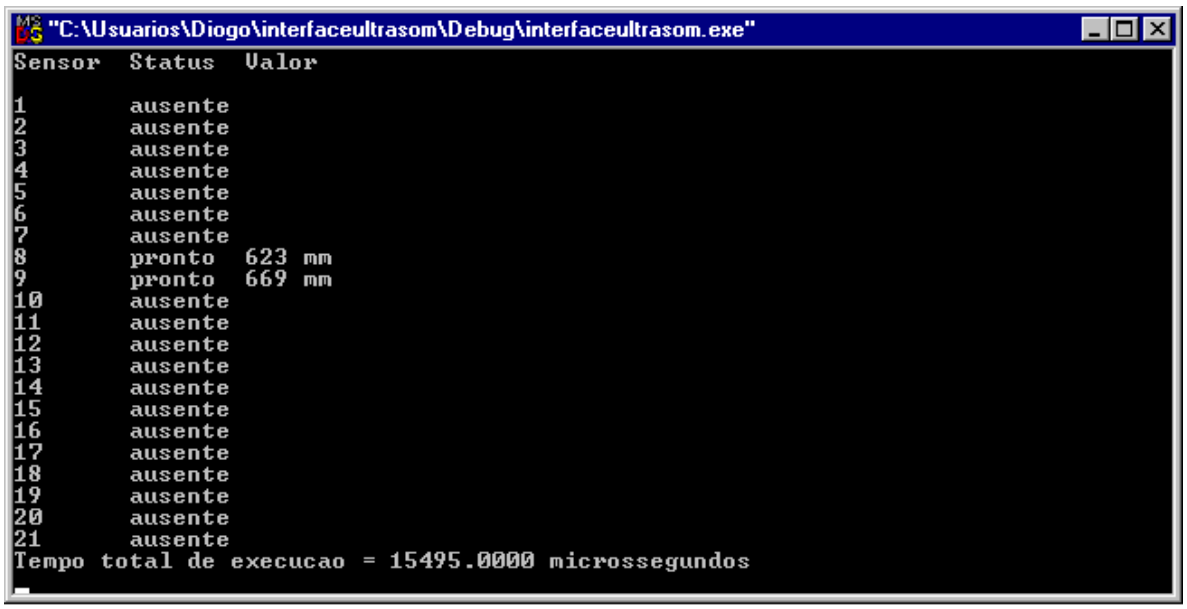


Figura 37 -Impressão de resultados na tela

Também fizemos algumas capturas nos pinos de habilitação do barramento no transceptor RS485 do PC e de um ATMega8 quando o PC faz uma solicitação de status, fazendo com que o ATMega8 retorne apenas um byte. O resultado está na Figura 38:

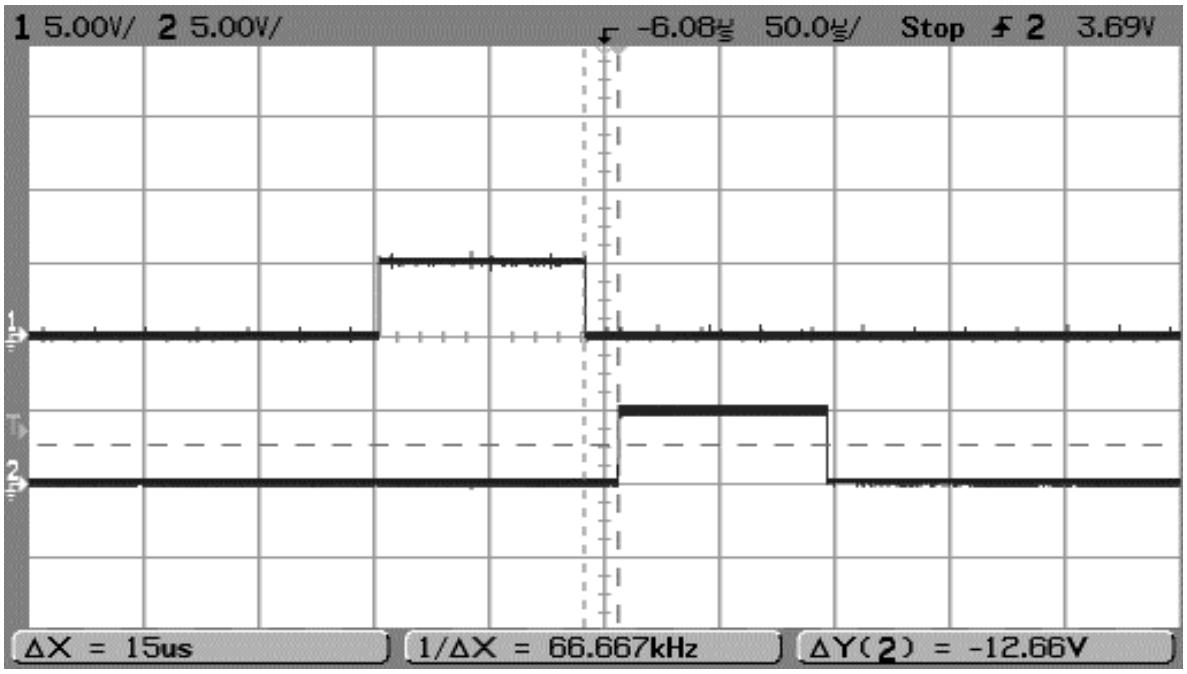


Figura 38 - Instantes de transmissão de um comando e de uma resposta

O canal 1 mostra o byte transmitido pelo PC e o canal 2 mostra o byte respondido pelo ATMega8. Como capturamos apenas o sinal de habilitação do barramento, o conteúdo do byte não pode ser visto, apenas o intervalo de transmissão.

O parâmetro ΔX medido é o tempo que o ATMega8 demora a processar o byte recebido e começar a transmitir uma resposta. Este intervalo de 15 microssegundos chega a ser 6 vezes menor que o tempo de transmissão de um byte. Assim, o tempo que o RTX deve esperar para receber uma resposta a partir do momento em que ele terminou a transmissão de uma interrogação até o recebimento completo de uma resposta é de cerca de 105 microssegundos.

A característica determinística da plataforma RTX pode ser avaliada na Figura 39 abaixo:

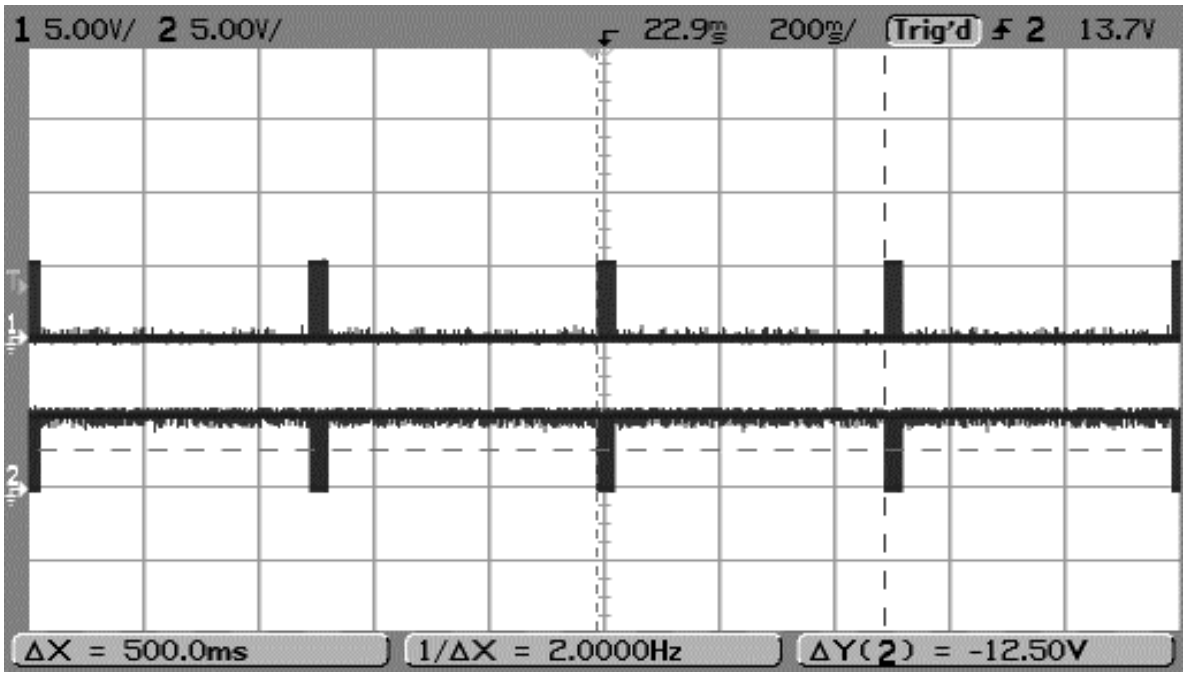


Figura 39 - característica determinística da plataforma RTX

Esta captura foi realizada com o programa exemplo (apêndice B) onde se determinou que uma varredura devesse ser feita a cada 500 milissegundos. Olhando o ΔX medido agora, podemos comprovar a precisão de temporização das tarefas lançadas através da plataforma RTX.

5. PROPOSTA DE CALIBRAÇÃO

A partir das equações [7] e [12], a expressão completa da distância em função do tempo de vôo t_T e temperatura T em ºC é dada por:

$$l = \frac{t_T}{2} V_0 \sqrt{273.15 + T} \tag{13}$$

em que V_0 é a constante de 20,04 m.s^{-1}.ºC$^{-1/2}$ obtida anteriormente.

Se supusermos V_0 desconhecido e $t_T = \bar{t}_T - \Delta t_T$, isto é, o tempo de vôo igual ao tempo medido menos um delta relativo ao atraso de detecção no circuito receptor, a expressão acima fica

$$l = \frac{\bar{t}_T}{2} V_0 \sqrt{273.15 + T} - \frac{\Delta t_T}{2} V_0 \sqrt{273.15 + T} \tag{14}$$

Definindo $\alpha = \Delta t_T V_0$ e escrevendo a equação acima de forma matricial, temos:

$$l = \begin{bmatrix} \frac{\bar{t}_T}{2}\sqrt{273.15+T} & \frac{1}{2}\sqrt{273.15+T} \end{bmatrix} \begin{bmatrix} V_0 \\ \alpha \end{bmatrix} \tag{15}$$

ou de forma simplificada

$$l = \varphi(\bar{t}_T, T).\theta \tag{16}$$

De posse de N medidas de distância l_n, obtidas com trena, tempo t_{Tn} e temperatura T_n, pode-se estimar o parâmetro $\theta = [V_0 \quad \alpha]^T$ minimizando-se a seguinte função de custo:

$$J = \frac{1}{N} \sum_{n=1}^{N} \left(l_n - \varphi(\bar{t}_{Tn}, T_n)\theta \right)^2 \tag{17}$$

O vetor $\hat{\theta}$ que minimiza então a função acima é dado por:

$$\hat{\theta} = \left(\sum_{n=1}^{N} \varphi^T(\bar{t}_{Tn}, T_n) \cdot \varphi(\bar{t}_{Tn}, T_n) \right)^{-1} \cdot \left(\sum_{n=1}^{N} \varphi^T(\bar{t}_{Tn}, T_n) \cdot l_n \right) \tag{18}$$

Utilizando então os dados coletados anteriormente como exemplo e com auxílio do Matlab, encontramos $\hat{\theta}$=[18.7830 3.6397], ou melhor, $\hat{V}_0$ = 18,783 m.s^{-1}.ºC$^{-1/2}$ e $\hat{\Delta}t_T$ = 193,8 μs.

Refazendo agora os cálculos de distância utilizando os parâmetros estimados, traçou-se novamente o gráfico da Figura 35, obtendo-se o resultado mostrado na Figura 40, onde se nota que a curva obtida é praticamente a reta ideal com inclinação unitária.

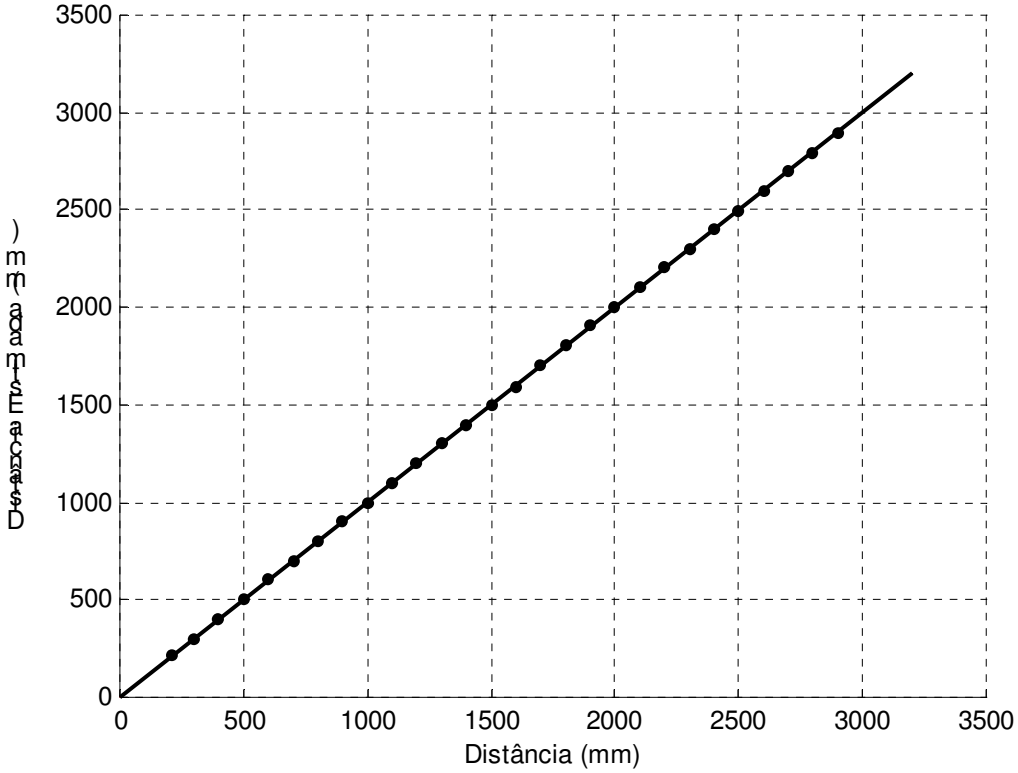


Figura 40 – Validação das medidas depois da calibração

Para determinar o erro associado às medidas, é preciso calcular as covariâncias dos parâmetros estimados $\hat{V}_0$ e $\hat{\Delta}t_T$, dadas pela matriz $P_{\hat{\theta}}$ pela seguinte expressão:

$$P_{\hat{\theta}} = E\left\{\left(\theta - \bar{\theta}\right)\left(\theta - \bar{\theta}\right)^T\right\}$$

$$= \sum_{i=1}^{N} \frac{\partial \hat{\theta}}{\partial \bar{t}_i} \sigma_{\bar{t}_i} \frac{\partial \hat{\theta}}{\partial \bar{t}_i}^T + \sum_{i=1}^{N} \frac{\partial \hat{\theta}}{\partial T_i} \sigma_{T_i} \frac{\partial \hat{\theta}}{\partial T_i}^T + \sum_{i=1}^{N} \frac{\partial \hat{\theta}}{\partial l_i} \sigma_{l_i} \frac{\partial \hat{\theta}}{\partial l_i}^T \tag{19}$$

em que $\sigma_{\bar{t}_i}$, σ_{T_i} e σ_{l_i} são as incertezas associadas às medidas de tempo, temperatura e distância. Para os dados utilizados, $\sigma_{\bar{t}_i}$ =1μs (precisão do cronômetro), σ_{T_i} =0,5ºC (precisão do LM35) e σ_{l_i} =0,5cm (precisão da trena usada). Calculando, obteve-se:

$$P_{\hat{\theta}} = \begin{bmatrix} 0.0005 & 0.0001 \\ 0.0001 & 0.4199 \end{bmatrix}$$

A variância do valor medido pode então ser obtida por

$$\sigma_l{}^2 = \frac{\partial l\left(\hat{\theta}, \bar{t}_T, T\right)}{\partial \hat{\theta}} P_{\hat{\theta}} \frac{\partial l\left(\hat{\theta}, \bar{t}_T, T\right)^T}{\partial \hat{\theta}} \tag{20}$$

Calculando as variâncias, traçou-se o gráfico dos desvios padrões das distâncias obtidas com os novos parâmetros para cada posição do sensor na Figura 41, em que se nota a característica crescente da incerteza em função da distância.

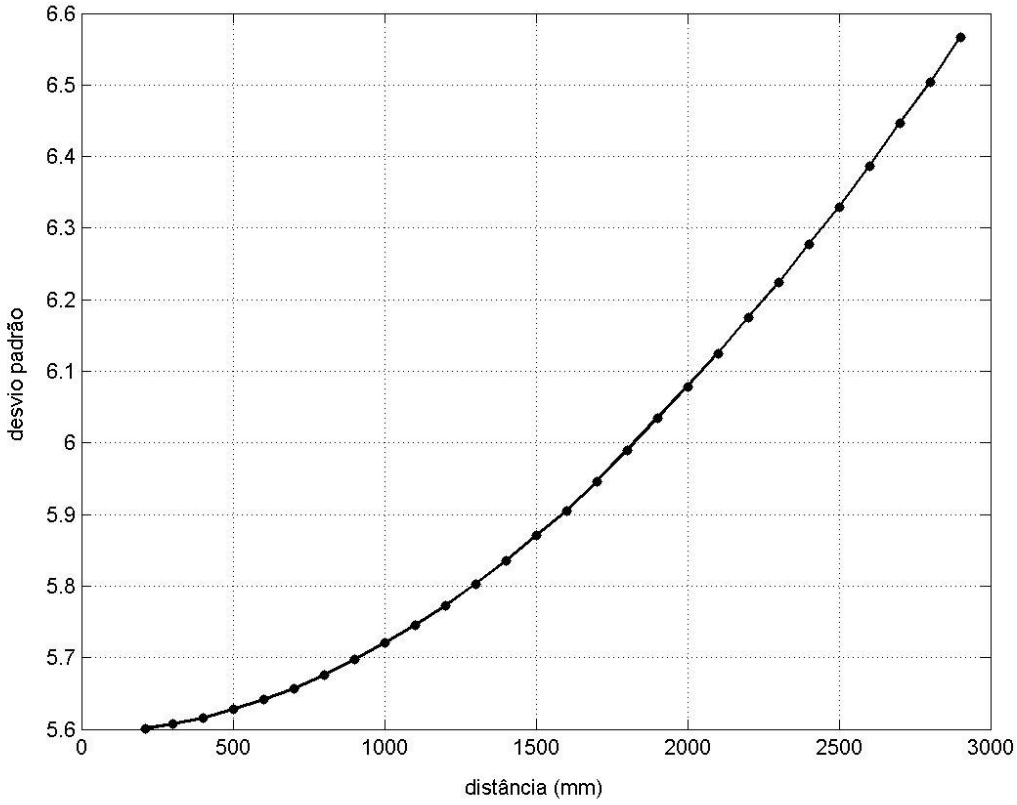


Figura 41 – Desvio padrão das distâncias medidas após calibração

Para uma melhor análise traçaram-se no mesmo gráfico os desvios padrões das distâncias obtidas antes e depois do procedimento de calibração na Figura 42. Nota-se que até 1500 mm, a incerteza estimada é maior do que o desvio padrão das medidas, pois para curtas distâncias o efeito do espalhamento do sinal de ultra-som não é importante. Já acima de 1500 mm, o espalhamento da frente de onda deve causar uma incerteza que não foi modelada. O mais certo seria usar uma distribuição de Rayleigh para a amplitude do sinal de retorno, porém isto foge do escopo deste trabalho de graduação e requer estudos mais aprofundados.

O importante é que para curtas distâncias, onde evitamento de obstáculos é feito, tem-se uma estimativa, ou melhor, uma sobre-etimativa das incertezas das medidas.

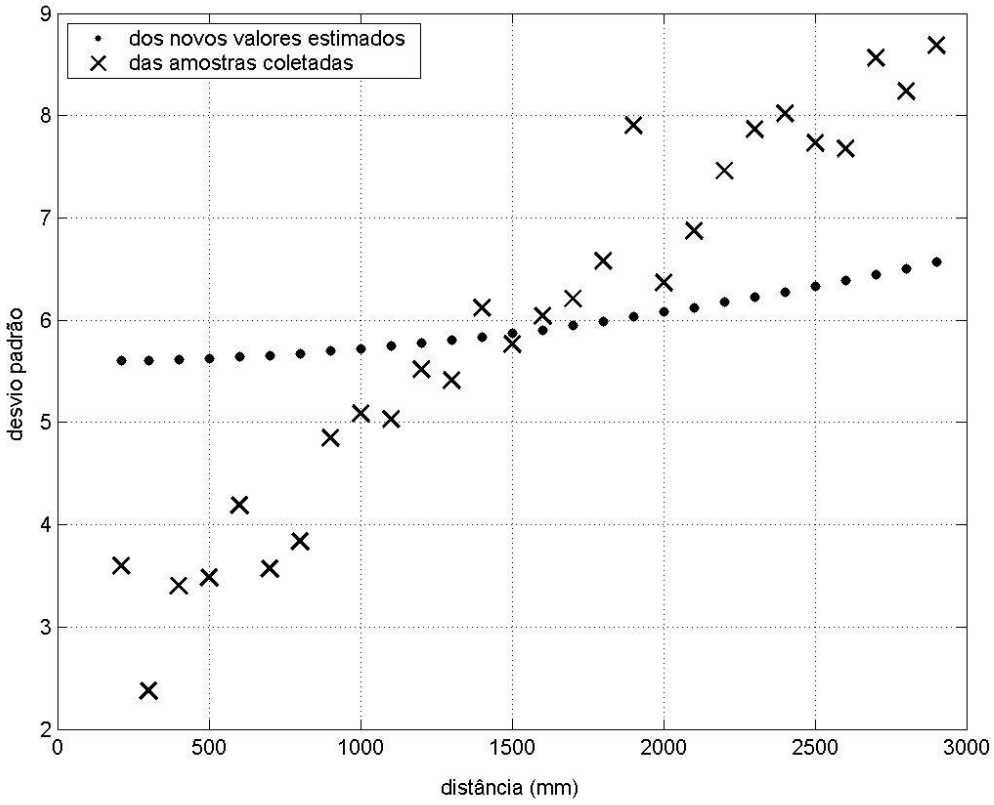


Figura 42 – Comparativo entre os devios padrões antes e depois da calibração

6. CONCLUSÕES E RECOMENDAÇÕES

A proposta inicial de projetar uma rede de sensores ultra-sônicos foi praticamente cumprida. Conseguiu-se desenvolver o sistema de medição, composto pelo circuito transceptor e transdutores, e o seu controle pelo microcontrolador ATmega8 para medir distância e detectar obstáculos. Além disso, foi também desenvolvido a rede de microcontroladores e sua interligação com o PC, bem como integrar os programas de medição e comunicação.

Alguns pontos do sistema não foram observados, como potência consumida, a distância máxima que pode ser medida e a verificação da compensação da temperatura na medição. A potência consumida é um fator importante principalmente em veículos robóticos que têm limitações de espaço e peso para baterias embarcadas. A faixa de distância de operação dos sensores, bem como a questão da influência da temperatura, não foram verificadas devido à falta de um ambiente controlado para aferição e disponibilidade de tempo. Outra pendência foi a questão da fixação do cinturão no robô.

Sugere-se que para trabalhos futuros se explore essas questões pendentes, bem como a proposta de calibração do capítulo 5 para então depois partir para aplicação em evitamento de obstáculos e navegação autônoma.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ARAGONES, J. *Commande et evaluation des performances d'un robot omnidirectionnel a roués*. Tese de Doutorado, Universidade de Montpellier 2. 2002

[2] BENTLEY, J.P., *Principles of measurements systems*. John Wiley & Sons, Nova Iorque, E.U.A.. 1988.

[3] BIQUARD, P., *Les ultrasons,* Que sais-je?. Presses Universitaires de France. 1972.

[4] BLITZ, J. *Fundamentals of Ultrasonics.* Butterworths, Londres. 1967.

[5] BORENSTEIN, J., EVERETT, H.R., FENG, L., *Where am I? Sensors and Methods for Móbile Robot Positioning*. Universidade de Michigan. 1996.

[6] CARLIN, B., *Ultrasonics*. McGraw-Hill, Nova Iorque. 1960.

[7] GIOZZA, W.F., ARAÚJO, J.F.M.de, MOURA, J.A.B., SAUVÉ, J.P., *Redes locais de computadores – Tecnologia e Aplicações*. McGraw Hill, São Paulo. 1986.

[8] JOHNSON, D.E., HILBURN, J.L., *Rapid, practical desings of active filters*. John Wiley & Sons, Nova Iorque, E.U.A.. 1975.

[9] SCHUNK, L.M., LUPPI, A., *Microcontroladores AVR – Teoria e Aplicações*. Ed. Érica, São Paulo. 2001.

[10] Data sheet Ultrasonic Sensors. Murata Manufacturing Co., Ltd. Disponível em: <http://www.murata.com>. Acesso em: 10 ago. 2004.

[11] Data sheet DS485, National Semiconductor Corporation. Disponível em <http://www.national.com>. Acesso em 22 jan. 2005

[12] Application Notes ST485, STMicroelectronics. Disponível em <http://www.st.com>. Acesso em 23 jan. 2005

[13] Data sheet AVR ATmega8. Atmel Corporation. Disponível em: <http://www.atmel.com>. Acesso em: 31 mar. 2004

[14] Data sheet TL074. Texas Instrument Incorporated. Disponível em: <http://www.ti.com>. Acesso em: 11 jan. 2005

[15] Application Report, *Ultrasonic Distance Measurement With the MSP430.* Texas Instrument Incorporated. Disponível em: <http://www.ti.com>. Acesso em: 4 jan. 2005

[16] Data sheet MAX232. Maxim Integrated Products. Disponível em: <http://www.maxim-ic.com>. Acesso em: 8 jan. 2005

[17] User Guide RTX 4.3, VenturCom, Inc.

[18] Reference Guide RTX 4.3, VenturCom, Inc.

[19] Site: <http://www.arcelect.com/rs232.htm> Acesso em: 21 jan. 2005

[20] Site: <http://www.beyondlogic.org/serial/serial.htm> Acesso em: 1 dez. 2004

APÊNDICE

APÊNDICE A. PROGRAMA DO ATMEGA8

```
/*********************************************************************/
/* UNIVERSIDADE DE BRASÍLIA - UnB                                    */
/* PROJETO DE GRADUAÇÃO 2                                            */
/* ENGENHARIA MECATRÔNICA                                            */
/* MÓDULO ESCRAVO DE CONTROLE DOS SENSORES                           */
/* DO CINTURÃO DE ULTRA-SOM DO ROBÔ OMNI                             */
/* Alexandre Simões Martins                  00/53104                */
/*                                                                   */
/*************************APRESENTAÇÃO*******************************/
/*                                                                   */
/* ultra-som.c                                                       */
/*                                                                   */
/* Criado em 22/03/2004                                              */
/* Versão final em 18/01/2005                                        */
/*                                                                   */
/* Descrição:                                                        */
/*                                                                   */
/*     Este programa interpreta e responde comandos de uma unidade   */
/*     mestre de um barramento de comunicações RS485.                */
/*     Realiza medições de distância                                 */
/*     em até três módulos sensores de ultra-som sob demanda, enviando */
/*     o valor medido ä unidade mestre do barramento quando solicictado */
/*                                                                   */
/*********************************************************************/



/****************************** INCLUDES *****************************/
#include <math.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <stdio.h>
/*********************************************************************/

/****************************** DEFINES ******************************/
#define pulsos            15

#define COM_STATUS        0x00
#define COM_DADO          0x20
#define COM_MEDIR         0x40

#define MEDINDO           0x00
#define PRONTO            0x20
#define INATIVO           0x40

#define SIM               1
#define NAO               0

#define AVR2PC            0x00
#define PC2AVR            0x80

#define ENDERECO          0x1F
#define COMANDO           0X60
#define END_SENSOR        0X03
#define END_AVR           0x1C

/*********************************************************************/

/************************** PROTÓTIPOS*******************************/
```

```
float  conversor_ad        (void);
void   USART_Init          (void);
void   USART_Transmit      (unsigned char);

/*********************************************************************/

/************************* VARIÁVEIS GLOBAIS ***************************/

unsigned char buffer;            //buffer de recpeção serial
unsigned char endereco;          //endereço do dispositivo
unsigned char status[3];         //status de cada sensor
unsigned char sensor;            //endereço do sensor atual
unsigned char medir;             //flag para iniciar medição
unsigned int  distancia[3];      //variavel que armazena a ultima
                                 //medida realizada de cada sensor
unsigned char ciclo = 0;         //variavel para emissão dos pulsos
                                 //de ultra-som no estouro do timer0
unsigned char ciclo_r = 0;       //variável para esperar o inicio da recepcao
                                 //do eco de ultra-som (tira ruído de
                                 //acoplamento)
unsigned char tx_ultrasom;

/*********************************************************************/

/******** Interrupção de estouro do timer0 que gera pulsos 40kHz **********/

SIGNAL (SIG_OVERFLOW0)
{
       TCNT0+=122;
       if ((ciclo%2)!=0) PORTB |= tx_ultrasom;
       else PORTB &= ~tx_ultrasom;
       if (ciclo)   --ciclo;
       if (ciclo_r) --ciclo_r;
}

/*********************************************************************/

/* interrupção de estouro do timer1 (que mede o tempo de vôo)
   sinalizando timeout para a recepção do eco de ultra-som *****************/

SIGNAL (SIG_OVERFLOW1)
{
       unsigned char sreg;
       sreg=SREG;
       cli();
       TCCR1B = 0x00;                   //Pára o cronômetro
       ACSR &= ~(_BV(ACIE)); //Desativa Interrupção do comparador
       distancia[sensor]=0xFFFF;
       status[sensor]=PRONTO;
       SREG=sreg;
       sei();
}
/*********************************************************************/

/* interrupção do comparador analógico que sinaliza
   recepção do eco de ultra-som e para contagem do timer1 ******************/

SIGNAL (SIG_COMPARATOR)
{
       unsigned char sreg;
       float temperatura,velo,tempo;
       ACSR &= ~(_BV(ACIE));     //Desativa Interrupção do comparador
       sreg=SREG;
       cli();
       TCCR1B = 0x00;                   //Pára o cronômetro

       tempo=TCNT1*0.72337962962962962962962962963;  //tempo em microsegundo
```

```
                                        //com fator de escala para 11.0592MHz
        SREG=sreg;
        sei();
        temperatura=conversor_ad();
        velo=20.03*sqrt((temperatura/10)+273);        //(m/s)
        distancia[sensor]=(tempo/2000.0)*(unsigned)velo;          //(mm)
        status[sensor]=PRONTO;
}
/*****************************************************************************/

/******* interrupção que indica tranmissão completa (buffer vazio) **********/

SIGNAL (SIG_UART_TRANS)
{
        //reseta o enable do barramento rs-485
        PORTD &= ~_BV(PD2);
}
/*****************************************************************************/

/******** interrupção que indica recebimento de um dado pela serial *********/

SIGNAL (SIG_UART_RECV)
{
        unsigned char i,j,k,funcao;
        buffer=UDR;
        //checa se direção é PC->AVR
        if (buffer & PC2AVR)
        {
                //checa se o endereço é deste dispositivo
                i=buffer&END_AVR;
                if (i==endereco || i==0)
                {
                        //pega o comando
                        funcao = buffer & COMANDO;
                        switch (funcao)
                        {
                                case COM_MEDIR:
                                //pega o endereço  do sensor e define como o sensor
                                //atual
                                        sensor = buffer & END_SENSOR;
                                        switch (sensor)
                                        {
                                                case 0:
                                                        tx_ultrasom=_BV(PB0);
                                                        break;
                                                case 1:
                                                        tx_ultrasom=_BV(PB1);
                                                        break;
                                                case 2:
                                                        tx_ultrasom=_BV(PB2);
                                                        break;
                                        }
                                        if (status[sensor]==PRONTO)
                                        {
                                                //seta flag para iniciar aquisicao
                                                medir = SIM;
                                                //escolhe pino d entrada para a resposta
do sensor
                                                ADMUX &= ~END_SENSOR;
                                                ADMUX |= sensor;
                                        }
                                        break;

                                case COM_STATUS:
                                    //coloca o endereço completo (dispositivo + sensor)
                                    i = buffer & ENDERECO;
                                        //coloca O status
                                        i |= status[buffer&END_SENSOR];
```

```
                                    //transmite
                                    USART_Transmit(i);
                                    break;

                             case COM_DADO:
                                    //coloca o endereço completo
                                 i = buffer & ENDERECO;
                                    //coloca o status
                                    i |= status[buffer&END_SENSOR];
                                    j = 0x7F &
                                        ((unsignedchar)
                                        (distancia[buffer&END_SENSOR]>>7));
                                    k  = 0x7F &
                                    (unsigned char) distancia[buffer&END_SENSOR];

                                    USART_Transmit(i);
                                    USART_Transmit(j);
                                    USART_Transmit(k);
                                    break;
                        }
                  }
            }
}
/*****************************************************************************/

/***** função que  faz conversão A/D e retorna o valor da tensão em mV ******/

float conversor_ad (void)
{
      // Desativa Multiplexação do Comparador Analógico
      //(permite ligar o conversor A/D)
      SFIOR &= ~(_BV(ACME));
      // Ativação do ADC
      ADCSRA |= _BV(ADEN);
      //selecao do prescaler clock_io/64
      ADCSRA |= _BV(ADPS2) | _BV(ADPS1);
      // seleção da porta de entrada para ADC3 (PC3 ou pin 26)
      ADMUX &= ~END_SENSOR;
      ADMUX |= 0X03;
      //inicio da conversão
      ADCSRA |= _BV(ADSC);
      // espera a conversão terminar
      while (ADCSRA & _BV(ADSC) );
      // Desliga o ADC
      ADCSRA &= ~_BV(ADEN);
      //Volta o multiplexador para o comparador
      SFIOR |= _BV(ACME);
      // captura o valor no registrador adc
      return(ADC*4.8828125);
}
/*****************************************************************************/

/**************** função que configura trasmissão serial ********************/

void USART_Init(void)
{
      UDR=0;
      UBRRL = 11;               //Seta baud rate de 115200 para fosc= 11059200
      UCSRA = _BV(U2X);         //seta bit indicando double speed,
      UCSRB = _BV(TXEN) |       //Habilitação da transmissão serial
              _BV(RXEN) |       //Habilitação da recepção serial
              _BV(RXCIE) |      //Habilitação da interrupção de recepção serial
              _BV(TXCIE);       //Habilitação da interrupção de transmissão
                                //completa
```

```
	//Setar tamanho palavra
	//URSEL = 1 ->seleciona acesso para UCSRC (0 is UBRRH) reading and writing

	//UMSEL = 0 -> Operação assíncrona
	//UPM1 e UPM0 = 00 -> sem paridade
	//USBS = 1 -> seleciona 2 bits de parada a ser inserido pelo transmissor
	//UCSZ 2 1 0 = 011 -> escolhe tamanho de caractere de 8 bits p/ dado
	UCSRC = _BV(URSEL) | _BV(UCSZ1) | _BV(UCSZ0);
}
/*****************************************************************************/

/************** função para transmitir um byte por rs-485 ********************/

void USART_Transmit( unsigned char data )
{
	//Seta o enable do barramento rs-485
	PORTD |= _BV(PD2);
	// Aguarda o esvaziamento do buffer de transmissão
	while ( !( UCSRA & (1<<UDRE)) );
	// põe o dado no buffer e o transmite
	UDR = data;
}
/*****************************************************************************/

/***************** função que dispara a medição dos sensores ****************/

void medicao (void)
{
	unsigned char sreg;
	TCCR0=0;                    //timer0 parado
	sreg=SREG;
	cli();
	TCNT1=0;                    //reseta o cronômetro
	SREG=sreg;
	sei();
	ciclo = 2*pulsos;           //Define o número de pulsos de ultra-som a
	                            //serem emitidos
	ciclo_r=ciclo+5;
	TCNT0=255;                  //coloca timer0 proximo do estouro
	TCCR0=1;                    //Inicia a emissão, i.é., liga o timer0
	TCCR1B=0X02;                //Liga o cronômetro - inicia contagem do
	                            //timer1 com prescaler = clk_i_o / 8
	while (ciclo_r);            //Espera o tempo mínimo para iniciar a medição
	TCCR0=0;                    //timer0 parado
	if (ACSR<32)
	{
	    distancia[sensor] = 0;
	      status[sensor]=PRONTO;
	}
	else
	{
	      ACSR |= _BV(ACIE);  //Ativa interrupção do comparador
	}

}
/*****************************************************************************/

/******************************** PROGRAMA PRINCIPAL *************************/

void main (void)
{
	//Configura Pino PD2 (enable do barramento rs-485) como saída
	DDRD |= _BV(PD2);

	//inicializa Pino PD2 (enable do barramento rs-485) em  0
	PORTD &= ~_BV(PD2);
```

```
	//Configura Pinos PB0 PB1 e PB2 como saída
	//(tx dos sensores 1, 2 e 3 respectivamente)
	DDRB |= _BV(PB0) | _BV(PB1) | _BV(PB2);

	//Configura Pinos PD3, PD4 e PD5 (status de conexao dos sensores)
	//como entrada e pull-up ativado
	DDRD &= ~( _BV(PD3) | _BV(PD4) | _BV(PD5) );
	PORTD |=  _BV(PD3) | _BV(PD4) | _BV(PD5);

	//inicializa status
	status[0]=PRONTO;
	status[1]=PRONTO;
	status[2]=PRONTO;

	// Desliga o ADC
	ADCSRA &= ~_BV(ADEN);

	//Define que a entrada Negativa do comparador será do multiplexador
	//(Analog Comparator Multiplexer Enable)
	SFIOR |= _BV(ACME);

	//INterrupcao é gerada na borda de DESCIDA da saída do comparador
	ACSR |= (_BV(ACIS1));
	ACSR &= ~(_BV(ACIS0));

	//Interrupção do Comparador inicialmente desativada
	ACSR &= ~(_BV(ACIE));

	//timer0 e timer1 parados
	TCCR0 = 0;
	TCCR1B = 0x00;

	//interrupções do timer0 e timer1 ativadas
	TIMSK |= _BV(TOIE0) | _BV(TOIE1);

	//Configura USART
	USART_Init();

	//Configura Pinos PC4, PC5 e PD7 (endereco do avr)
	//como entrada e pull-up ativado
	DDRD &= ~(_BV(PD7));
	PORTD |= _BV(PD7);
	DDRC &= ~( _BV(PC4) | _BV(PC5));
	PORTC |= _BV(PC4) | _BV(PC5);

	endereco=END_AVR&~(((PINC & _BV(PC4))>>2) |
	                   ((PINC & _BV(PC5))>>2) |
	                   ((PIND & _BV(PD7))>>3));
	//endereco = 0x04;

	//habilita vetor de interrupções globais
	sei ();
/*****************************  LOOP PRINCIPAL *****************************/

	while (1)
	{
		if ((medir==SIM) && (status[sensor]==PRONTO))
		{
			medir=NAO;
			status[sensor] = MEDINDO;
			medicao();
		}
		if (PIND & _BV(PD3))
			status[0] = INATIVO;
		else if (status[0] == INATIVO)
			status[0] = PRONTO;

		if (PIND & _BV(PD4))
```

```
                status[1] = INATIVO;
            else if (status[1] == INATIVO)
                status[1] = PRONTO;

            if (PIND & _BV(PD5))
                status[2] = INATIVO;
            else if (status[2] == INATIVO)
                status[2] = PRONTO;
        }
}

/*****************************************************************************/
```

APÊNDICE B. PROGRAMA DA PLATAFORMA RTX

```
//*************************************************************************/
/* UNIVERSIDADE DE BRASÍLIA - UnB
/* PROJETO DE GRADUAÇÃO 2
/* ENGENHARIA MECATRÔNICA
/* MÓDULO ESCRAVO DE CONTROLE DOS SENSORES
/* DO CINTURÃO DE ULTRA-SOM DO ROBÔ OMNI
/* Diogo Andrade                              00/53121
/*
/*
/**************************APRESENTAÇÃO*********************************/
/*
/* ultra-somLibRTX.h
/*
/* Criado em 20/11/2004
/* Versão final em 18/01/2005
/*
/* Descrição:
/*
/*     Biblioteca de definições e prototipagem das funções descritas
/* na biblioteca ultra-somLibRTX.c.
/*************************************************************************/

#define SERIALLIBRTX_COMPORT_1 0x3F8
#define SERIALLIBRTX_COMPORT_2 0x2F8
#define SERIALLIBRTX_COMPORT_3 0x3E8
#define SERIALLIBRTX_COMPORT_4 0x2E8

#define SERIALLIBRTX_BAUDRATE_38400          0x03
#define SERIALLIBRTX_BAUDRATE_115200         0x01
#define SERIALLIBRTX_BAUDRATE_57600          0x02
#define SERIALLIBRTX_BAUDRATE_19200          0x06
#define SERIALLIBRTX_BAUDRATE_9600           0x0C
#define SERIALLIBRTX_BAUDRATE_4800           0x18
#define SERIALLIBRTX_BAUDRATE_2400           0x30


#define PC2AVR                  0x80
#define AVR2PC                  0x00

#define COM_STATUS              0x00
#define COM_DADO                0x20
#define COM_MEDIR               0x40

#define MEDINDO                 0x00
#define PRONTO                  0x20
#define INATIVO                 0x40

#define CLASSE_0                0x00
#define CLASSE_1                0x01
#define CLASSE_2                0x02

#define COMANDO                 0x60
#define ENDERECO                0x1F
#define STATUS                  0X60

#define NUM_DISP                7
#define NUM_CLASSES             3
#define NUM_SENSORES            NUM_DISP*NUM_CLASSES

#define BCAST                   0x00
#define DISP_1                  0x04

#define END_SENSOR              0X03
```

```
#define END_AVR                 0x1C
#define DIRECAO                 0x80
#define false                   0
#define true                    1

void ChecaStatus(unsigned char *);
BOOL sChecaStatus( int );
void MedirClasse(int);
void MedirSensor(int);
int RetornoDeDados ( int , int *);


BOOL SerialLibRTX_Iniciar(int Comport, unsigned char bps);
BOOL SerialLibRTX_EnviarByte(unsigned char *pData);
BOOL SerialLibRTX_ReceberByte(unsigned char *pData, double MaximaEsperaUS);
```

```
//*********************************************************************/
/* UNIVERSIDADE DE BRASÍLIA - UnB
/* PROJETO DE GRADUAÇÃO 2
/* ENGENHARIA MECATRÔNICA
/* MÓDULO ESCRAVO DE CONTROLE DOS SENSORES
/* DO CINTURÃO DE ULTRA-SOM DO ROBÔ OMNI
/* Diogo Andrade                          00/53121
/*
/*
/***************************APRESENTAÇÃO*********************************/
/*
/* ultra-somLibRTX.c
/*/* Criado em 20/11/2004
/* Versão final em 18/01/2005
/*
/* Descrição:
/*
/*           Biblioteca de funções genéricas de uso da porta serial e
/* funções básicas de manipulação do cinturão de sensores de ultra-som
/* desenvolvido
/*********************************************************************/

#include <windows.h>
#include <rtapi.h>
#include <stdio.h>
#include "SerialLibRTX.h"

// porta de comunicação default
static int SerialPortAddress = SERIALLIBRTX_COMPORT_1;

// taxa de comunicação default.
static unsigned char SerialPortBaudrate = SERIALLIBRTX_BAUDRATE_9600;


/******* Rotinas Genéricas de Manipulação da porta serial ****************/

// abertura da porta serial
BOOL SerialLibRTX_Iniciar(int Comport, unsigned char bps)
{
       SerialPortAddress = Comport;
       SerialPortBaudrate = bps;

       //Desativar interrupções
       RtWritePortUchar(((PUCHAR)(SerialPortAddress + 1)), 0);
```

```
	// DLAB ON
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 3)), 0x80);
	// Baud Rate - DL Byte
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 0)), bps);
	// Baud Rate - DH Byte
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 1)), 0x00);
	// 8 Bits, No Parity, 1 Stop Bit
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 3)), 0x03);
	// FIFO: 14 bytes, limpa TX e RX fifos, habilita
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 2)), 0xC7);
	// Ativa DTR, RTS = 0, e OUT2
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 4)), 0x0B);

	return TRUE;
}


// rotina de transmissão de um byte com controle do pino RTS
BOOL SerialLibRTX_EnviarByte(unsigned char *pData)
{
	while(!(RtReadPortUchar(((PUCHAR)(SerialPortAddress + 5))) & 0x40)){
		Sleep(1);
	} // Espera fim da última transmissão

	//setar RTS = 1 mantendo OUT2
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 4)), 0x09);
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 0)), *pData); // Envia.
	// Espera fim da última transmissão
	while(!(RtReadPortUchar(((PUCHAR)(SerialPortAddress + 5))) & 0x40));
	//desetar rts mantendo OUT2
	RtWritePortUchar(((PUCHAR)(SerialPortAddress + 4)), 0x0B);
	return TRUE;
}

// Rotina que recebe um byte pela porta serial, com um timeout customizável
BOOL SerialLibRTX_ReceberByte(unsigned char *pData, double MaximaEsperaUS)
{
	LARGE_INTEGER ClockInit,Clock;
	double ElapsedTime;

	if(MaximaEsperaUS<0) MaximaEsperaUS = 0; // Espera minima de 0 us.

	RtGetClockTime(CLOCK_2,&ClockInit);

	while(1){
		if(RtReadPortUchar(((PUCHAR)(SerialPortAddress + 5))) & 1)
		{
			*pData = RtReadPortUchar(((PUCHAR)(SerialPortAddress + 0)));
			return(TRUE);
		}
		RtGetClockTime(CLOCK_2,&Clock);
		// Tempo em uS;
		ElapsedTime = (Clock.QuadPart-ClockInit.QuadPart)*0.1;
		if(ElapsedTime >= MaximaEsperaUS){
			return(FALSE);  // Dado não chegou no tempo estipulado.
		}
	}
}
/*************************************************************************/


/******* Rotinas de Controle do Cinturão de Sensores de Ultra-Som **********/
/******************** Desnevolvidas por Diogo Andrade **********************/

// Função que percorre faz uma varredura no cinturão,
// guardando o estado dos sensores em um vetro de bytes
void ChecaStatus( unsigned char *Vivos)
{
```

```
	int i;

	for(i=0;i<(NUM_SENSORES);i++)
			*(Vivos + i)= sChecaStatus( i + 1);
}

// Função que pede o status de um sensor particular, retornando
// verdadeiro se o sensor está presente, e falso cas contrário
BOOL sChecaStatus( int Sensor)
{
	int i,j;

	unsigned char Transmite, Responde;

	i = (Sensor - 1)%3;
	j = (Sensor - 1)/3;

	Transmite = PC2AVR | COM_STATUS | (DISP_1+(j<<2)) | (CLASSE_0 + i);

	SerialLibRTX_EnviarByte(&Transmite);

	if(!SerialLibRTX_ReceberByte(&Responde, 120.0))
		return false;
	else
		if (((Transmite = (Transmite & ENDERECO)) != (Responde & ENDERECO))
	|| ((Responde & COMANDO) == INATIVO))
		return false;
	else
		return true;
}

// Função que envia um comando de medição a um sensor específico.
void MedirSensor( int Sensor)
{
	int i,j;
	unsigned char Transmite;

	i = (Sensor - 1)%3;
	j = (Sensor - 1)/3;

	Transmite = PC2AVR | COM_MEDIR | (DISP_1+(j<<2)) | (CLASSE_0 + i);

	SerialLibRTX_EnviarByte(&Transmite);
}

// Função que dá um ocmando de medição por classede sensores
void MedirClasse( int Classe)
{

	unsigned char Transmite;


	Transmite = PC2AVR | COM_MEDIR | BCAST | (CLASSE_0 + Classe);

	SerialLibRTX_EnviarByte(&Transmite);
}

// Função que solicita ao sensor o valor da medição.
// os valores de retorno são:
//		0 - o sensor não está presente
//		1 - a medida está pronta, e o resultado é gravado em "Valor"
//		2 - o sensor ainda não temrminou a medição
int RetornoDeDados ( int Sensor, int *Valor)
{
	int i,j,k;
	unsigned char Transmite, Recebe[3];

	i = (Sensor - 1)%3;
```

```
	j = (Sensor - 1)/3;

	Transmite = PC2AVR | COM_DADO | (DISP_1+(j<<2)) | (CLASSE_0 + i);

	SerialLibRTX_EnviarByte(&Transmite);

	for (k=0;k<3;k++)
		if(!SerialLibRTX_ReceberByte(&Recebe[k], 120.0))
			return 0;

		if( (Recebe[0] & COMANDO) == PRONTO &&
		    (Recebe[0] & DIRECAO) == AVR2PC )
		{
			*Valor = ((int)Recebe[1] << 7) + (int)Recebe[2];
			return 1;
		}
		else if( (Recebe[0] & COMANDO) == INATIVO &&
			(Recebe[0] & DIRECAO) == AVR2PC )
			return 0;
		else if( (Recebe[0] & COMANDO) == MEDINDO &&
			   (Recebe[0] & DIRECAO) == AVR2PC )
			return 2;
		else return 0;

}

/*****************************************************************************/




/*****************************************************************************/
/* UNIVERSIDADE DE BRASÍLIA - UnB
/* PROJETO DE GRADUAÇÃO 2
/* ENGENHARIA MECATRÔNICA
/* MÓDULO ESCRAVO DE CONTROLE DOS SENSORES
/* DO CINTURÃO DE ULTRA-SOM DO ROBÔ OMNI
/* Diogo Andrade                              00/53121
/***************************APRESENTAÇÃO*********************************/
/*
/* ultra-somRTX.c
/*
/* Criado em 20/11/2004
/* Versão final em 18/01/2005
/*
/* Descrição:
/*
/*           Este é um programa exemplo para a utilização da biblioteca
/* ultra-somRTX.h para a utilização do cinturão de sensores de
/* ultra-som desenvolvido.
/*      O programa faz a varredura dos sensores a cada 150 ms
/* e grava os resultados medidos e o tempo da varredura em uma
/* memoria compartilhada.
/*
/*****************************************************************************/

#include <windows.h>
#include <rtapi.h>
#include <stdio.h>
#include "SerialLibRTX.h"

//prototipo da função de tarefa periódica
void RTFCNDCL SerialTimerTask(void *   nContext);

//variáveis globais
unsigned char       Status[3][7][3];       // matriz que guarda os bytes
                                           // recebidos do cinturão
int          SerialTimerCounter=0;     // contador de execuções
```

```
double       SerialTimerTaskExecutionTimeUS = 0.0, //tempo gasto
                                                   //para executar uma tarefa
LARGE_INTEGER       SerialTimerTaskPeriod;  // periodo do timer
HANDLE              hSerialTimerTask,       // handle do timer
                    mutex,                  // handle do mutex
                    memoria,                // handle das memorias
                    memoria2;               // compartilhadas

void _cdecl main(int    argc, char **argv, char **envp)
{


      int i,j,k,z;                              //contadores

      unsigned char **copia;           //aponta para a memoria compartilhada
                                       //que guarda os valores meiddos
      double       **copiatempo;       //aponta para a memoria compartilhada
                                       //que guarda o tempo decorrido

    //período deo timer em 100 ns.
    SerialTimerTaskPeriod.QuadPart = 1500000; // 150ms

    // Criação do timer periódico
    if (!(hSerialTimerTask = RtCreateTimer(
                                  NULL,
                                  0,                /
                                  SerialTimerTask, // handler do timer
                                  NULL,
                                  RT_PRIORITY_MAX, // prioridade
                                  CLOCK_2) ))      // RTX HAL timer
    {
        RtPrintf("RtCreateTimer error = %d\n",GetLastError());
        ExitProcess(1);
    }


      // Iniciar a porta serial:
      if(!SerialLibRTX_Iniciar(SERIALLIBRTX_COMPORT_2,
                               SERIALLIBRTX_BAUDRATE_115200))
      {
       RtPrintf("Erro em SerialLibRTX_Iniciar\n");
       ExitProcess(1);
      }

      RtPrintf("Porta aberta com suscesso\n");


      //alocação de memoria para os ponteiros
      //para memorias compartilahdas mapeadas
      //no espaço de endereçamento virtualdo usuario
      copia = malloc(63);
      copiatempo = malloc(sizeof(double));

      //criação do mutex
      if(!(mutex = RtCreateMutex(NULL,FALSE,"mutex")))
      {
            RtPrintf("RtCreateMutex memory error = %d\n",GetLastError());
            ExitProcess(1);
      }

      //criação das memória compartilhada que irá guaradar os dados medidos
      if(!(memoria = RtCreateSharedMemory(    PAGE_READWRITE,
                                              0,
                                              63*sizeof(unsigned char),
                                             "memoria",
                                              copia)))
      {
            RtPrintf("RtCreateShared memory error = %d\n",GetLastError());
```

```
            ExitProcess(1);
        }
        //criação das memória compartilhada que irá guaradar os dados medidos
        if(!(memoria2 = RtCreateSharedMemory(   PAGE_READWRITE,
                                                0,
                                                sizeof(double),
                                               "memoria2",
                                                copiatempo)))
        {
            RtPrintf("RtCreateShared memory error = %d\n",GetLastError());
            ExitProcess(1);
        }

       //inicialização do contador de tarefas
        SerialTimerCounter = 0;

        //Inicialização do timer criado
       if (! RtSetTimerRelative( hSerialTimerTask,
                              &SerialTimerTaskPeriod,
                              &SerialTimerTaskPeriod) )
        {
             RtPrintf("RtSetTimerRelative error = %d\n",GetLastError());
             ExitProcess(1);
        }


        // Laço principal

        //irá contar 500 execuções da tarefa periódica e terminar o programa
        while(SerialTimerCounter < 500 && SerialTimerCounter >= 0)
        {
            Sleep(100);
            //seção crítica
            RtWaitForSingleObject(mutex,INFINITE);
            z=0;
            //copia dos valores medidos para a memoria compartilhada
            for(j=0;j<NUM_DISP;j++)
                for (i=0;i<NUM_CLASSES;i++)
                    for (k=0;k<3;k++)
                    {
                        *(*copia + z) = Status[i][j][k];
                        z++;
                    }
            **copiatempo = SerialTimerTaskExecutionTimeUS;
            RtReleaseMutex(mutex);
            //fim da seção crítica
        }
        RtPrintf("\nfim\n");
    ExitProcess(0);
}

//corpo da tarefa periódica, que consiste em varrer todos os sensores do cinturão
void RTFCNDCL SerialTimerTask(PVOID    context)
{
        unsigned char Vivos[NUM_SENSORES],Prontos[7];
        int i,terminou = 0, j,l,Valor;
        LARGE_INTEGER ClockInit,Clock;
        LARGE_INTEGER delay;


        delay.QuadPart = 10000; //intervalo de pedido de dados

        RtGetClockTime(CLOCK_2,&ClockInit); //tempo de inicio de
                                            //execução da tarefa

        ChecaStatus(Vivos); //guardar em "Vivos" os sensores que estão presentes

        RtWaitForSingleObject(mutex,INFINITE); //inicio de seção crítica
```

```
for(i=0;i<NUM_CLASSES;i++)
{

	MedirClasse( i ); //medição em broadcast por classe

	//inicialização da lista de sensores prontos
	for(j=0;j<NUM_DISP;j++)
		if(!Vivos[j*3+i])
			{
				Prontos[j] = true;
				Status[i][j][0] = 0;
			}
			else Prontos[j] = false;

	terminou = false;
	while(!terminou) //laço que só termina até todos os sensores da
			//classe saiam do estado de "medindo"

		for(j=0;j<NUM_DISP;j++)
		{
			if(Vivos[j*3+i] && !Prontos[j])
			{
				//pedido de retorno de dados
				Status[i][j][0] = (unsigned char)
							RetornoDeDados( j*3+i+1,
									&Valor);
				switch(Status[i][j][0])
				{
					case 2: //medindo
					break;

					case 1: //pronto
					Prontos[j] = true;
					Status[i][j][1] = (Valor >> 7) & 0x7F;
					Status[i][j][2] = Valor & 0x7F;
					break;

					case 0: //sensor ausente
					Prontos[j] = true;
					Vivos[j*3+i] = false;
					break;

					default:
					break;
				}
			}
		}
		RtSleepFt(&delay);//intervalo de pergunta para não
				//sobrecarregar o barramenbto de comunicações
		//rotina de parada
		for(l=0;l<NUM_DISP;l++)
		{
			if(Prontos[l])
				terminou = true;
			else
			{
				terminou = false;
				break;
			}
		}
	}
}

++SerialTimerCounter; //contador de tarefas lançadas

//medir o instante de conclusão da tarefa
RtGetClockTime(CLOCK_2,&Clock);
```

```
       //calcular o tempo transcorrido em microssegundos
       SerialTimerTaskExecutionTimeUS = (Clock.QuadPart -
                                         ClockInit.QuadPart)*0.1;
       RtReleaseMutex(mutex);//fim da seção crítica
}



/*********************************************************************/
/* UNIVERSIDADE DE BRASÍLIA - UnB
/* PROJETO DE GRADUAÇÃO 2
/* ENGENHARIA MECATRÔNICA
/* MÓDULO ESCRAVO DE CONTROLE DOS SENSORES
/* DO CINTURÃO DE ULTRA-SOM DO ROBÔ OMNI
/* Diogo Andrade                              00/53121
/*
/**************************APRESENTAÇÃO*********************************/
/*
/* InterfaceUltra-somRTX.c
/*
/* Criado em 20/11/2004
/* Versão final em 18/01/2005
/*
/* Descrição:
/*
/*           Este é um aplicativo simples que apenas lê os valores de
/* medidas gravadas na memória compartilhada gravada pela aplicação
/* RTX. Os valores são organizados e impressos em uam janela DOS
/*´para monitoramento
/*
/*********************************************************************/


#include <windows.h>
#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <errno.h>
#include <rtapi.h>




#define NUM_DISP          7
#define NUM_CLASSES       3
#define NUM_SENSORES      NUM_DISP*NUM_CLASSES



//função para posicionamento do cursor no terminal
void gotoxy(int xpos, int ypos)
{
  COORD scrn;

  HANDLE hOuput = GetStdHandle(STD_OUTPUT_HANDLE);

  scrn.X = xpos; scrn.Y = ypos;

  SetConsoleCursorPosition(hOuput,scrn);
}
```

```
void main(void)
{

	int i,j,k,z,l,auxvalor;
	unsigned char Status[3][7][3], **copia;
	HANDLE mutex,memoria,memoria2;
	double TempoDeExecucao, **copia2;


	//alocação de memoria para os ponteiros para
	//memorias compartilahdas mapeadas
	//no espaço de endereçamento virtual do usuario
	copia = malloc(63);
	copia2 = malloc(sizeof(double));

	//criação do mutex
	if(!(mutex = RtCreateMutex(NULL,FALSE,"mutex")))
	{
		RtPrintf("RtCreateMutex memory error = %d\n",GetLastError());
		ExitProcess(1);
	}

	//criação das memória compartilhada que irá guaradar os dados medidos
	if(!(memoria = RtCreateSharedMemory(	PAGE_READWRITE,
						0,
						63*sizeof(unsigned char),
						"memoria",
						copia)))
	{
		RtPrintf("RtCreateShared memory error = %d\n",GetLastError());
		ExitProcess(1);
	}
	//criação das memória compartilhada que irá guaradar os dados medidos
	if(!(memoria2 = RtCreateSharedMemory(	PAGE_READWRITE,
						0,
						sizeof(double),
						"memoria2",
						copia2)))
	{
		RtPrintf("RtCreateShared memory error = %d\n",GetLastError());
		ExitProcess(1);
	}



	//laço principal
	while(1)
	{
		RtWaitForSingleObject(mutex,INFINITE);
		//seção critica
		z=0;
		for(j=0;j<NUM_DISP;j++)
			for (i=0;i<NUM_CLASSES;i++)
				for (k=0;k<3;k++)
				{
					Status[i][j][k] = *(*copia + z);
					z++;
				}
		TempoDeExecucao = **copia2;
		z = 0;
		//fim da seção crítica
		RtReleaseMutex(mutex);

		//Impressão dos dados na tela

		gotoxy(4,1);
		printf("\n\nSensor\tStatus\tValor");
		//limpar a tela
```

```
            for(l=0;l<22;l++)
                  printf("\n
");
            gotoxy(4,3);
            for(j=0;j<NUM_DISP;j++)
            {
                  for (i=0;i<NUM_CLASSES;i++)
                        if (Status[i][j][0] != 0)
                        {
                              auxvalor = ((int)Status[i][j][1] << 7) +
                                          (int)Status[i][j][2];

                              if (Status [i][j][0] == 2)
            printf("\n%d\tmedindo\t%ld mm\t\t\t\t\t",3*j+i+1,auxvalor);
                              else if (Status [i][j][0] == 1)

            printf("\n%d\tpronto\t%ld mm\t\t\t\t\t",3*j+i+1,auxvalor);
                        }
                        else
            printf("\n%d\tausente\t\t\t\t\t\t",3*j+i+1);
            }
printf("\nTempo total de execucao = %6.4f microssegundos\n",TempoDeExecucao);
            Sleep(200); //taxa de atualização da tela

      }

    ExitProcess(0);
}
```

ANEXOS

A. TABELAS

Specific Heat Ratio of Air at Standard Atmospheric Pressure in SI Units:

Temperature - *t* (°C)	Specific Heat Ratio - *k*
-40	1.401
-20	1.401
0	1.401
5	1.401
10	1.401
15	1.401
20	1.401
25	1.401
30	1.400
40	1.400
50	1.400
60	1.399
70	1.399
80	1.399
90	1.398
100	1.397
200	1.390
300	1.379
400	1.368
500	1.357
1000	1.321

FONTE: www.engineeringtoolbox.com, acessado em 10/01/2005.