UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROTÓTIPO DE SISTEMA DE MONITORAMENTO REMOTO UTILIZANDO TCP/IP SOBRE ETHERNET (802.3)

ARIBERTO MONTIBELLER JUNIOR

BLUMENAU
2005

2005/1-05

ARIBERTO MONTIBELLER JUNIOR

PROTÓTIPO DE SISTEMA DE MONITORAMENTO REMOTO UTILIZANDO TCP/IP SOBRE ETHERNET (802.3)

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Miguel Alexandre Wisintainer – Orientador

BLUMENAU
2005

2005/1-05

PROTÓTIPO DE SISTEMA DE MONITORAMENTO

REMOTO UTILIZANDO TCP/IP SOBRE ETHERNET (802.3)

Por

ARIBERTO MONTIBELLER JUNIOR

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Miguel Alexandre Wisintainer – Orientador, FURB
Membro:	Prof. Antonio Carlos Tavares – FURB
Membro:	Prof. Francisco Adell Péricas – FURB

Blumenau, 04 de maio de 2005

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

Aos meus pais Ariberto e Eliana, pela educação que proporcionaram, pelo apoio, amor dedicação, aos meus Avôs Clarindo, Elvira, José e Aridene, mesmo que indiretamente, me ajudaram com pensamento positivo e principalmente acreditando em mim.

Aos meus irmãos, Aryel, Anelise e especial a Ariana, que compreenderam, mais ou menos, que tinha que me concentrar. À minha namorada Rosiane que compreendeu a falta de atenção e mesmo assim sempre esteve do meu lado.

Aos meus amigos, especialmente aos da monitoria, pelas trocas de idéias, brincadeiras, empurrões e tudo mais. Pois sem amigos você não é nada, não vou citar nomes, pois sou capaz de esquecer algum.

Aos professores, em especial Tavares e Péricas, mesmo que com pequenas dicas, foram dicas fundamentais para esse trabalho.

À empresa LZT, em especial ao “chefe” Ricardo, pela compreensão de minha ausência em alguns momentos.

À Ângela, Rafael e Emerson, que emprestaram partes do hardware para esse trabalho.

Em especial ao meu orientador, Miguel, por ter acreditado na conclusão deste trabalho, e pelas dicas muito valiosas que passou e pelo que aprendi enquanto monitor de suas disciplinas.

Não tem como agradecer todas as pessoas, pois para isso teria que colocar em anexo algumas páginas, a todos um forte abraço.

Jamais considere seus estudos como uma obrigação, mas como uma oportunidade invejável para aprender a conhecer a influência libertadora da beleza do reino do espírito, para seu próprio prazer pessoal e para proveito da comunidade à qual seu futuro trabalho pertence.

Albert Einstein

RESUMO

Este trabalho apresenta uma alternativa aos tradicionais dispositivos de vigilância. Integrando em um só dispositivo, câmera e diversos sensores, onde o mesmo conecta-se a uma rede TCP/IP, tornando assim possível monitorar ambientes em qualquer lugar do mundo. Para o mesmo se fez necessário implementar um hardware, cliente, que é responsável por gerenciar a câmera, os sensores e comunicar-se, através de uma rede TCP/IP, com o servidor, implementado em Java, independente de plataforma, responsável por gerenciar e exibir os dados capturados pelos clientes.

Palavras-chave: Dispositivo remoto; Captura de imagens; Rede de computadores; Ethernet (802.3); Protocolos TCP/IP.

ABSTRACT

This work presents an alternative to the traditional devices of monitoring. Integrating in one only device, camera and diverse sensors, where the same it is connected a net TCP/IP, thus becoming possible to monitor environments in any place of the world. For the same one became necessary to implement the hardware, client, who is responsible for managing the camera, the sensors and communicating themselves, through a net TCP/IP, with the server, implemented in Java, independent of platform, responsible for managing and showing the data captured for the client.

Key-Words: Remote device; Capture of images; Computer network; Ethernet (802.3); Protocols TCP/IP.

LISTA DE ILUSTRAÇÕES

Figura 1 - TCP/IP x OSI/ISO 15
Figura 2 - Arquitetura da rede Internet 15
Figura 3 – Estabelecimento de um *socket* para troca de mensagens TCP 18
Figura 4 – Criação de um *socket* para troca de mensagens UDP 19
Figura 5 - Amostragem e quantização 21
Figura 6 - Imagem em diferentes resoluções 22
Figura 7 – Chips CCD 24
Figura 8 - Captura e acúmulo de luz em um sensor CCD 25
Figura 9 – Arquitetura proposta por Von Neumann 27
Figura 10 - Placa disponibilizada junto ao Kit 28
Quadro 1 – Características Rabbit 2000 TCP/IP 28
Figura 12 - Diagrama de estados do cliente 34
Figura 13 - Diagrama de atividades do cliente 35
Figura 14 – Protocolo id / status 36
Figura 15 – Digrama de atividades para captura da imagem 37
Figura 16 – Diagrama de classe Dispositivo 39
Figura 17 – Digrama de atividades *loop* principal server 40
Figura 18 – Diagrama atividades *thread* de conexão do cliente 41
Figura 19 - Diagrama de atividades troca de mensagem server/cliente 42
Figura 20 – *Layout* do hardware 44
Quadro 2 – Definição de variáveis para conexão de TCP/IP 45
Quadro 3 – Variáveis Básicas 46
Quadro 4 – Alocação de memória 46
Figura 21 – Disposição da memória alocada 47
Quadro 5 – Inicialização e abertura do *socket* 47
Quadro 6 – *Loop* principal do software 48
Quadro 7 – Função de captura da imagem 50
Quadro 8 – Classe imagem 51
Quadro 9 – Criação da imagem a ser exibida 52
Quadro 10 – *Thead* principal do sevidor 53
Quadro 11 – Método *rum* da classe *ServerThead* 54

Quadro 12 – *Loop* da função *processConnection* 55
Figura 22 – Tela principal do software 57
Figura 23 – Menu Arquivo 58
Figura 24 – Propriedades 58
Figura 25 – Menu dispositivo 58
Figura 26 – Propriedades do dispositivo 59
Figura 27 – Janela do dispositivo 60
Figura 28 – Menu Arquivo do dispositivo 61

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS DO TRABALHO 13
1.2 ESTRUTURA DO TRABALHO 13
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 REDE TCP/IP 14
2.1.1 Camada de rede 15
2.1.2 Camada de transporte 16
2.1.2.1 Protocolo TCP 17
2.1.2.2 Protocolo UDP 18
2.1.2.3 Socket 19
2.2 IMAGEM DIGITAL 20
2.3 SENSORES 22
2.3.1 Sensores CCD 24
2.4 MICROCONTROLADORES 26
2.4.1 Kit Rabbit 2000 TCP/IP 27
2.5 TRABALHOS CORRELATOS 29
3 DESENVOLVIMENTO DO TRABALHO 31
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 31
3.2 ESPECIFICAÇÃO 31
3.2.1 Especificação do hardware 32
3.2.2 Especificação do software embarcado 34
3.2.3 Especificação do software para PC 38
3.3 IMPLEMENTAÇÃO 43
3.3.1 Técnicas e ferramentas utilizadas 43
3.3.2 Hardware 44
3.3.3 Software embarcado 45
3.3.4 Software PC 50
3.3.5 Operacionalidade da implementação 56
3.4 RESULTADOS E DISCUSSÃO 61
4 CONCLUSÕES 62
4.1 E7XTENSÕES 63

REFERÊNCIAS BIBLIOGRÁFICAS 64

1 INTRODUÇÃO

Hoje em dia é cada vez maior a preocupação com segurança, tendo em vista o aumento do número de furtos a residências, empresas e instituições. De uma forma geral, todos tentam proteger-se, seja contratando seguranças, instalando sistema de alarme ou de vigilância eletrônica. Os sistemas de vigilância eletrônica, por exemplo, podem custar muito caro, pois para a implantação dos mesmos é necessária toda uma estrutura de cabeamento para atender às necessidades de instalação.

Com a evolução e popularização das redes de computadores, melhor dizendo a necessidade delas nos dias de hoje, a maioria das novas edificações já são projetadas prevendo a implantação de uma rede para computadores. Além disso, construções mais antigas vêm passando por adaptações para implantação dessas estruturas. Sendo assim, se já existe toda uma estrutura de rede, por que gastar com mais a instalação de uma estrutura de cabos especialmente para vigilância? Não seria mais fácil utilizar a estrutura de rede já existente?

Esse é o ponto a ser abordado neste trabalho, o desenvolvimento de um hardware que possa ser conectado a um ponto de rede e disponibilizar alguns dos recursos que existem em sistemas de vigilância atuais, como por exemplo, câmeras, sensores de presença e sensores de temperatura.

Esse dispositivo será gerenciado por um software em um microcomputador na mesma rede. Através desse software o usuário poderá visualizar as imagens captadas pela câmera e os dados dos sensores. O software também receberá alertas enviados pelo dispositivo quando ocorrer algo de não esperado, por exemplo, movimento ou um aumento de temperatura no local monitorado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um hardware e um software para monitoramento remoto de ambientes, através de imagens e sensores de presença.

Os objetivos específicos do trabalho são:

a) construir um hardware para captura de dados dos sensores e da câmera;
b) estabelecer comunicação entre o hardware e o microcomputador através de uma rede ethernet (802.3);
c) disponibilizar um software no PC para recebimento e envio dos dados para o dispositivo;
d) disponibilizar um software no PC para gerenciar o dispositivo e exibir os dados e imagens recebidas do dispositivo.

1.2 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta uma introdução sobre rede TCP/IP, imagem digital, sensores e microcontroladores. No capítulo 3 serão abordados detalhes sobre a especificação e desenvolvimento do protótipo. Por fim, o capítulo 4 traz as considerações finais e sugestões para extensões no trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

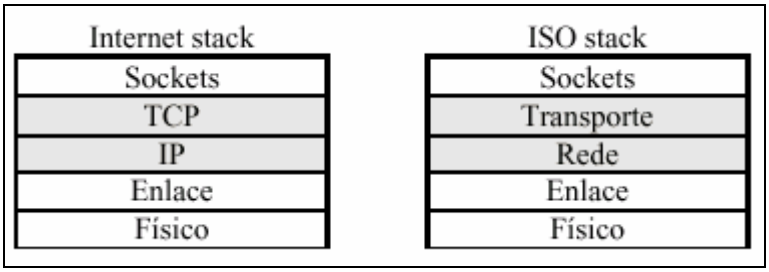
Neste capítulo são apresentados alguns conceitos fundamentais para o desenvolvimento deste trabalho. Tendo ênfase em redes TCP/IP, onde serão abordadas as camadas de maior relevância para o trabalho com seus respectivos protocolos. Também se faz necessário falar de formatos de imagens digitais, sensores, assim como de trabalho correlatos.

2.1 REDE TCP/IP

Segundo Seixas Filho (2002), TCP/IP é na verdade o nome genérico para uma família de protocolos e utilidades também conhecido por *Internet Protocol Suite*, onde *suite* designa uma pilha de protocolos. Estes protocolos originalmente faziam parte da internet, uma *Wide Area Network* (WAN) que evoluiu a partir da *Advanced Research Projects Agency Network* (ARPANET), criada pelo *United States Department of Defense* (DoD) para interligar centros de pesquisa que trabalhavam para o governo. TCP/IP é hoje o padrão de fato na interligação de redes heterogêneas locais (LAN), a grande distância (WAN) e a internet.

O modelo de referência *Open Systems Interconnectio* (OSI), especificado em 1983, é composta por sete camadas de protocolos: física, enlace, rede, transporte, sessão, apresentação e aplicação. Já o modelo de referência TCP/IP tem sua especificação baseada em um conjunto de cinco camadas de protocolos: física, enlace, rede, transporte e aplicação (PERICAS, 2003, p. 32-36).

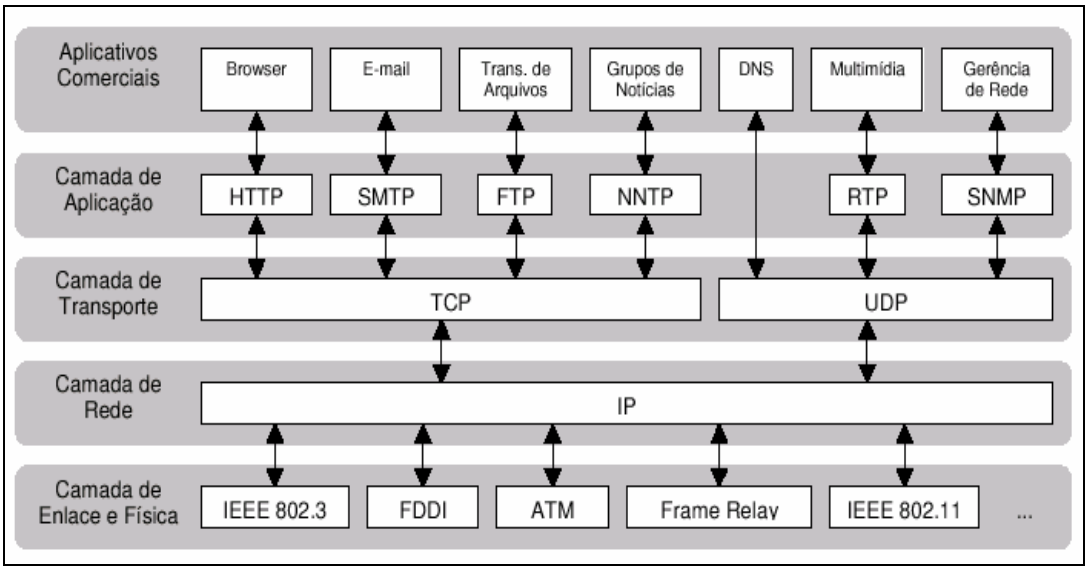
Comparando o padrão de protocolos OSI com o TCP/IP, tem-se que os protocolos TCP e IP correspondem aos níveis de transporte e de rede como e apresentado na Figura 1 (SEIXAS FILHO, 2002).



Fonte: Seixas Filho (2002)

Figura 1 - TCP/IP x OSI/ISO

A Figura 2 mostra uma visão geral das camadas implementadas pelo modelo de referência TCP/IP e seus principais protocolos e aplicações.



Fonte: Péricas (2003, p. 37)

Figura 2 - Arquitetura da rede Internet

2.1.1 Camada de rede

Segundo Seixas Filho (2002), os serviços proporcionados por esta camada se referem basicamente ao endereçamento, roteamento e segmentação de pacotes de modo a compatibilizá-los com os padrões adotados pelas duas entidades comunicantes.

De acordo com Gasparini e Barrella (1993), os endereços dos pacotes IP possuem 32 bits, os quais são basicamente constituídos de dois campos: o *netid* que identifica a qual rede

um *host* pertence e *host id* que identifica o *host* na rede. Sendo assim, dentro de uma rede todos os *hosts* têm o mesmo *netid*.

Para Péricas (2003, p. 69), a principal função da camada de rede é o roteamento de pacotes entre origem e o destino. O equipamento de origem determinará se a equipamento de destino faz parte da rede local, neste caso o pacote é enviado diretamente ao destino. Se o destino não pertence à mesma rede, a tabela de roteamento é consultada para verificar para qual *gateway* o pacote deve ser enviado. O *gateway* ou *router* deverá conduzir o pacote ao seu destino final.

2.1.2 Camada de transporte

Conforme Péricas (2003, p. 84), a camada de transporte é o centro do modelo referência TCP/IP. Sua função é garantir a transferência confiável de dados entre os *hosts* de origem e de destino, transmitindo segmentos de dados independentes da rede física utilizada.

Péricas (2003, p. 84-85) enumera as principais funcionalidades que esta camada deve fornecer:

a) multiplexação e demultiplexação: procedimento de reunião de dados, no computador de origem, de diferentes processos, encapsulá-los com informações de cabeçalho apropriadas e passar os segmentos resultantes para camada de rede, e vice-versa;
b) transferência confiável de dados: garantir que os dados enviados por um processo sejam recebidos de forma confiável e ordenados no processo de destino. Isso é denominado *Quality of Service* (QoS);
c) segmentação: dividir as mensagens em partes menores para transmissão, conforme a capacidade da camada de rede, e reagrupar os fragmentos de

mensagens no destino, tendo assim a mensagem original;

d) controle de congestionamento: prevenir para que conexões na camada de transporte não sejam inundadas com um tráfego excessivo de dados nos enlaces constituintes da rede.

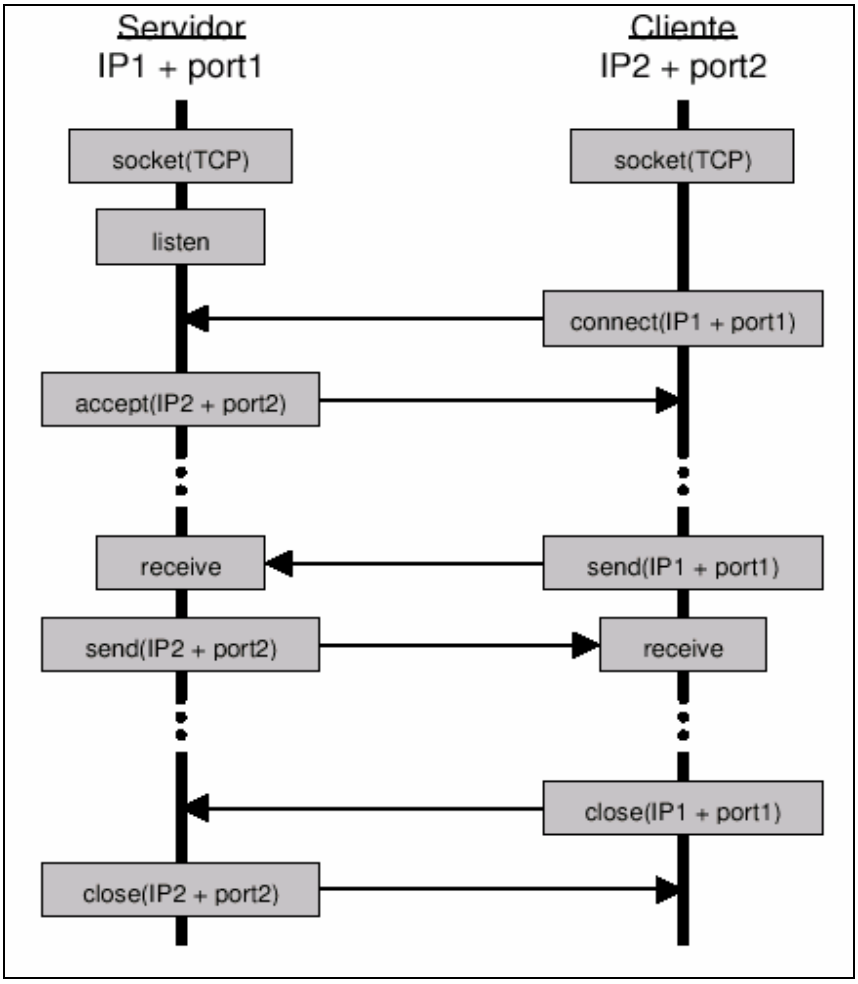
2.1.2.1 Protocolo TCP

O *Transmission Control Protocol* (TCP) é um protocolo orientado a conexão que fornece um serviço confiável de troca de dados fim-a-fim. Esse protocolo implementa todas as funcionalidades citadas anteriormente.

Conforme Soares et al. (1995), TCP não exige um serviço de rede e do protocolo de internet confiável. Ele garante a qualidade e seqüência dos dados entregues no destino, ou seja, garante a QoS da comunicação.

Ainda segundo Soares et al. (1995), para que várias aplicações possam transmitir simultaneamente, o TCP usa o conceito de porta, onde é atribuída uma porta diferente para cada aplicação que estiver utilizando o protocolo. Exceto para aplicações muito usadas, onde são atribuídas portas fixas, por exemplo, FTP porta 20 e Telnet porta 23.

Como TCP é um protocolo orientado a conexão, pode-se tratá-lo, segundo Péricas (2003, p. 92), como um canal virtual entre um *socket* cliente e *socket* servidor. *Socket* pode ser considerado um ponto de referência para o qual as mensagens podem ser enviadas e partir do qual elas são recebidas (CARVALHO, 1994). Na Figura 3 é representada como ocorre a troca de mensagens entre o cliente e o servidor.



Fonte: Péricas (2003, p. 93)

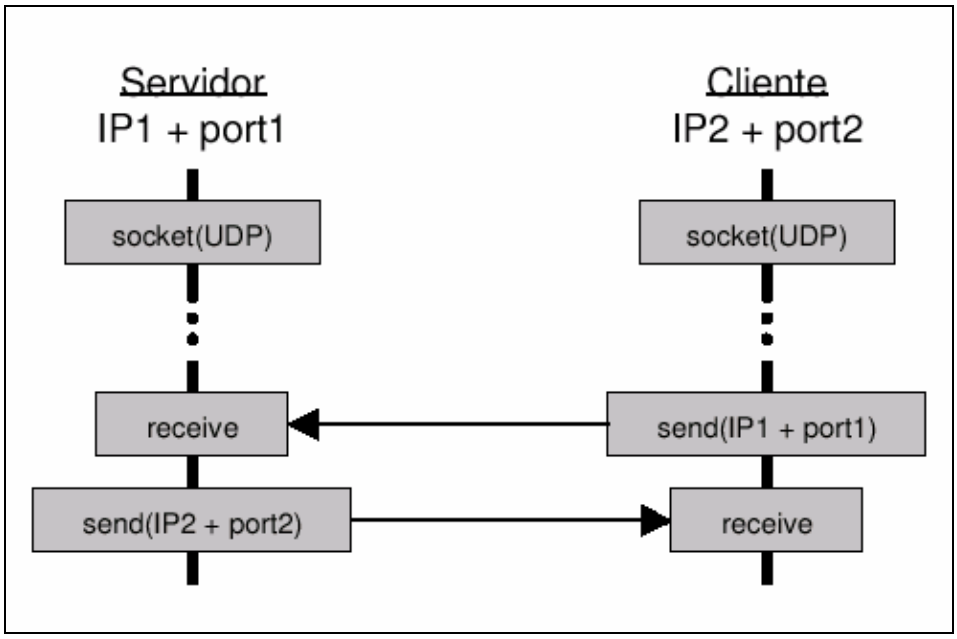
Figura 3 – Estabelecimento de um *socket* para troca de mensagens TCP

2.1.2.2 Protocolo UDP

O protocolo *User Data Protocol* (UDP), segundo Torres (2001), não é orientado a conexão. Isto é, ao contrário do TCP, ele não verifica se o pacote de dados chegou ou não ao seu destino, sendo assim não tem QoS na comunicação.

Conforme Torres (2001), existem vantagens em usar UDP em vez de TCP. Em aplicações que necessitam de velocidade, pois com a evolução das redes quase não há perda de pacotes. Mas para redes maiores e principalmente na internet, a taxa de perda de pacotes pode ser muito alta, inviabilizando sua utilização.

UDP oferece um serviço sem conexão, onde apenas existe o envio de mensagens entre o cliente e o servidor, sem necessidade de confirmação de recebimento (Figura 4).



Fonte: Péricas (2003, p. 94)

Figura 4 – Criação de um *socket* para troca de mensagens UDP

2.1.2.3 Socket

Um *socket* é considerado um ponto de referência para o qual as mensagens podem ser enviadas e a partir do qual as mensagens são recebidas. Qualquer processo pode criar um *socket* para se comunicar com outro processo, mas os dois processos devem criar seus próprios *sockets*, visto que os dois *sockets* são usados como um par, (BOCKENSKI, 1995).

O *socket* é a associação do endereço IP de uma máquina mais o endereço de uma porta, atribuído por determinada aplicação (CARVALHO, 1994).

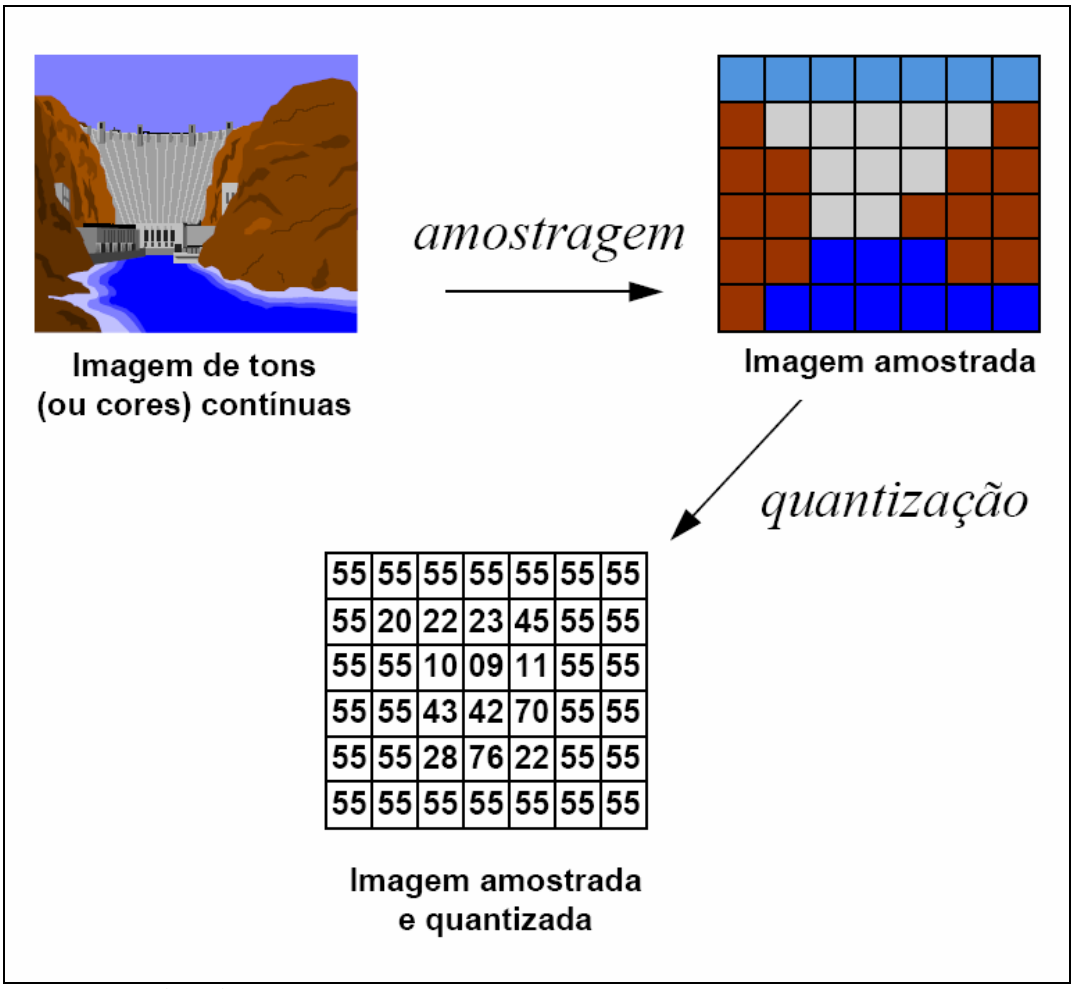
Concluindo a idéia de Carvalho (1994), as interfaces *sockets* acessam diretamente a camada de transporte, TCP ou UDP, e escondem das aplicações toda a complexidade existente nos respectivos protocolos.

2.2 IMAGEM DIGITAL

Segundo Vianna (2002), uma imagem natural pode ser descrita por uma variação contínua de tons e cores. No caso de fotografia, por exemplo, os tons variam de claro a escuro e as cores variam de vermelho a azul, abrangendo desta forma todo o espectro de cores visíveis.

Crosta (1992) caracteriza uma imagem digital como uma matriz, de dimensões *x* linhas por *y* colunas, com cada elemento (cada célula) possuindo um atributo *z* que representa o nível de cinza. Já Gonzalez e Woods (2000) definem uma imagem por uma função bidimensional de intensidade da luz *f(x,y)*, onde *x* e *y* são as coordenadas do ponto e o valor de *f* em qualquer ponto *(x,y)* é proporcional ao brilho (ou nível de cinza) da imagem naquele ponto.

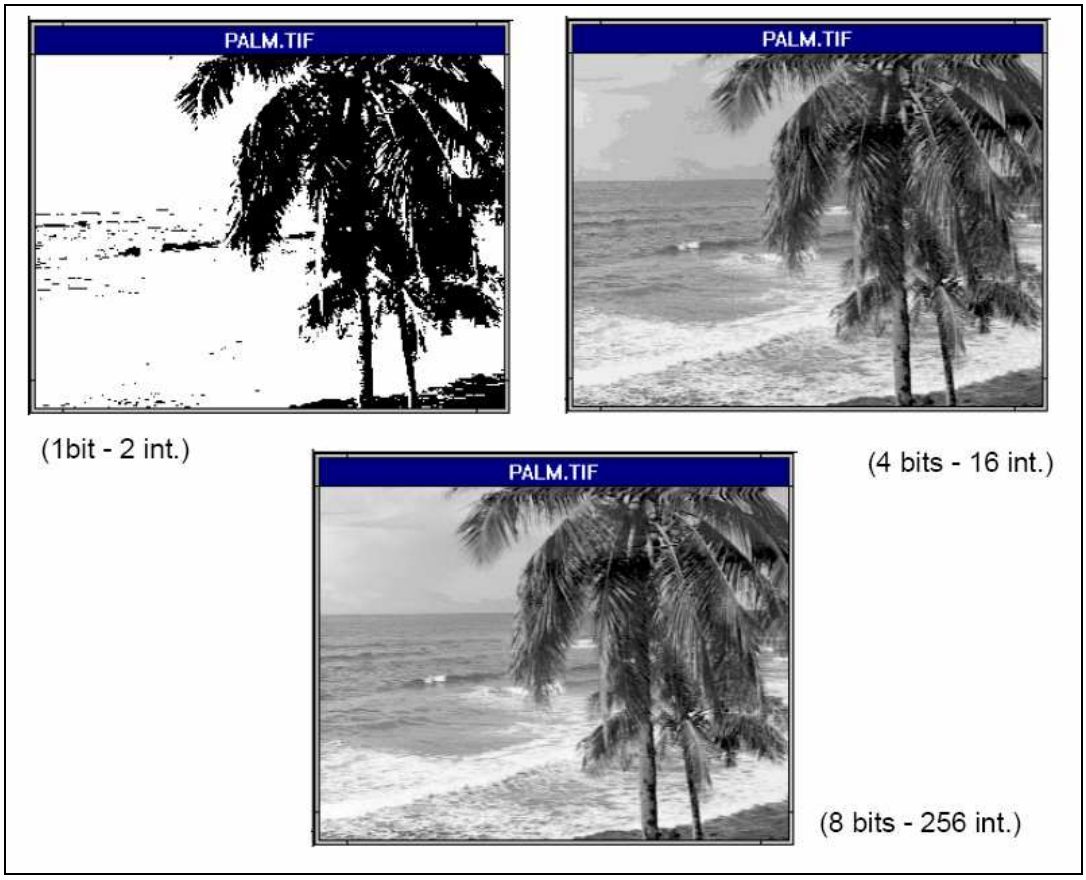
Uma imagem digital, entretanto, é composta por pontos discretos de tons e/ou cores, ou brilho, e não por uma variação contínua. Para a criação de uma imagem digital, deve-se dividir a imagem contínua em uma série de pontos que irão possuir uma determinada tonalidade (níveis de cinza) ou cor (colorido). Adicionalmente a este processo de divisão, deve-se descrever cada ponto por um valor digital. Os processos de divisão da imagem contínua e determinação dos valores digitais de cada ponto são chamados de amostragem e quantização, respectivamente. A combinação destes dois processos é o que se denomina de digitalização de imagens (VIANNA, 2002). Na Figura 5 é exemplificado os processos de amostragem e quantização.



Fonte: Vianna (2002)

Figura 5 - Amostragem e quantização

Ainda de segundo Vianna (2002), cada *pixel* em uma imagem digital representa a intensidade luminosa de um determinado ponto da imagem original. Sendo assim, o conceito de "resolução de brilho" refere-se à quão preciso é o brilho de cada *pixel* para representar a intensidade luminosa da imagem original. Após o processo de amostragem, cada amostra é quantizada. Este processo de quantização converte uma intensidade de tons contínuos, em um valor de brilho. A precisão deste valor digital está diretamente relacionada com o número de *bits* que serão utilizados na quantização. Como exemplo, adotando-se uma imagem digital que possua somente tons de cinza, se forem utilizados 3 *bits*, o brilho pode ser convertido em somente 8 tons de cinza, ao passo que se forem utilizados 8 *bits*, este valor passará para 256 tons. Na Figura 6 são apresentadas imagens com diferentes resoluções.



Fonte: Vianna (2002)

Figura 6 - Imagem em diferentes resoluções

2.3 SENSORES

Segundo Passos (2002) sensores, são dispositivos que detectam movimentos e ações que ocorrem nos processos. Pode-se dizer também que são elementos dotados e encarregados de gerar informações para os sistemas de automação. Como era de se esperar, existem muitos tipos de sensores, cada um para uma atividade e aplicação específica.

A seguir serão enumerados os principais tipos de sensores definidos por Passos (2002).

a) sensor magnético ou *reed-switch*: sensores que são acionados quando entram em contato com um campo magnético. Geralmente, é constituído de um material denominado ferro-magnético. O seu funcionamento ocorre da seguinte maneira,

quando um imã entra em contato com o sensor este atrai um par de chapas que fecha o circuito acionando uma determinada carga;

b) sensor capacitivo**:** esse tipo de sensor funcionam seguindo os princípios de funcionamento do capacitor, como o próprio nome sugere. Ele se opõe às variações de tensões do circuito como o capacitor. O sensor capacitivo é constituído de duas chapas metálicas separadas por um material isolante denominado dielétrico que no caso é o ar, pois suas chapas são colocadas uma ao lado da outra, diferente do capacitor que possui suas placas uma sobre a outra. O acionamento do sensor ocorre quando um corpo constituído de material não magnético se aproxima aumentando a sua capacitância. Quando isso ocorre, o circuito de controle detecta a variação na capacitância. Geralmente, este tipo de sensor é utilizado para medir níveis de água ou para serem empregados em esteiras em uma linha de produção;

c) sensor indutivo: funciona seguindo os conceitos de funcionamento do indutor. O indutor é um componente eletrônico composto por um núcleo no qual existe uma bobina em sua volta. Quando uma corrente percorre esta bobina um campo magnético é formado. Por sua vez, o campo magnético é concentrado no centro do núcleo fazendo com que se armazene energia por algum tempo. O sensor indutivo utiliza este tipo de funcionamento para ser acionado e informar o sistema da presença de algum corpo. O núcleo do sensor indutivo é aberto e assim sendo o campo magnético passa pelo ar em uma intensidade menor. Porem quando um corpo metálico é aproximado seu campo magnético passa pelo corpo aumentando sua intensidade acionado o circuito;

d) sensor óptico: é constituído por dois componentes denominados, emissor de luz e receptor de luz. Geralmente, os emissores de luz são os *leds* (diodos emissores de luz) ou lâmpadas comuns. Já o receptor é um componente eletrônico foto-sensível tais como os fototransistores ou fotodiodos. O funcionamento ocorre da seguinte maneira: uma onda é gerada por um circuito oscilador e esta é convertida em luz pelo emissor. Quando um corpo se aproxima este reflete a luz do emissor para o receptor acionando o circuito de controle;

e) sensor de pressão ou chave fim de curso: este tipo de sensor está presente em muitos dispositivos mecânicos e pneumáticos. Estes sensores são utilizados para detectar o fim de um curso de um determinado dispositivo. Estes dispositivos podem ser atuadores mecânicos tais como cilindros e alavancas. Seu funcionamento

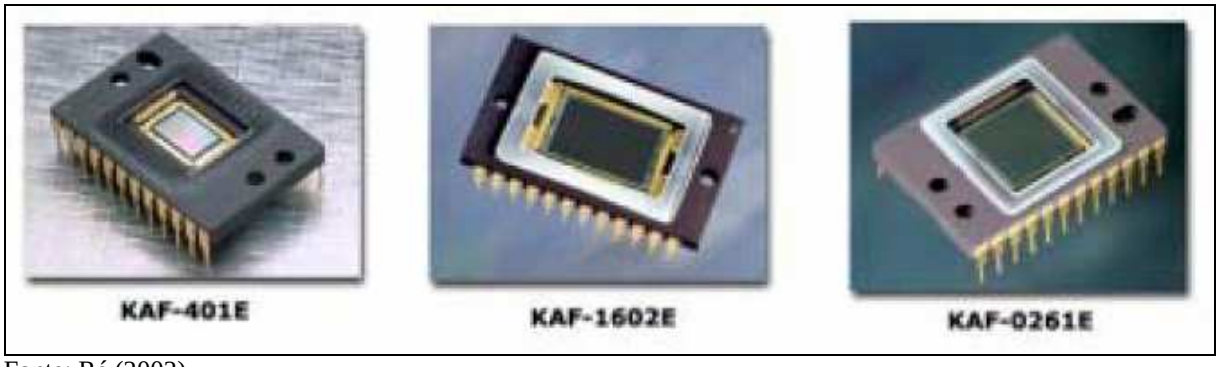
se mostra muito simples pois seu acionamento é totalmente mecânico. Este sensor pode ser normalmente aberto (NA) ou normalmente fechado (NF);

f) *encoder*: são dispositivos eletromecânicos que convertem posicionamentos mecânicos em sinais eletrônicos digitais para o sistema de controle. Este equipamento é capaz de medir deslocamentos angulares e lineares em máquinas e robôs.

Existe um tipo especial de sensor para captura de imagens digitais, *"charge coupled device"*(CCD), sobre o qual será realizado um estudo mais detalhado no tópico 2.3.1.

2.3.1 Sensores CCD

Basicamente *charge coupled device* (CCD) é um circuito eletrônico constituído de milhões de sensores microscópicos sensíveis à luz. A quantidade destes sensores expressa a resolução, ou seja quanto mais sensores, maior é a resolução da imagem. Cada um desses sensores representa um *pixel* e como ele não distingue cores, somente intensidade de luz, sobre eles são colocados filtros de cores básicas, que são azul, verde e vermelho (POVARESKIM, 2005). Na Figura 7 são apresentados alguns chips de CCD.

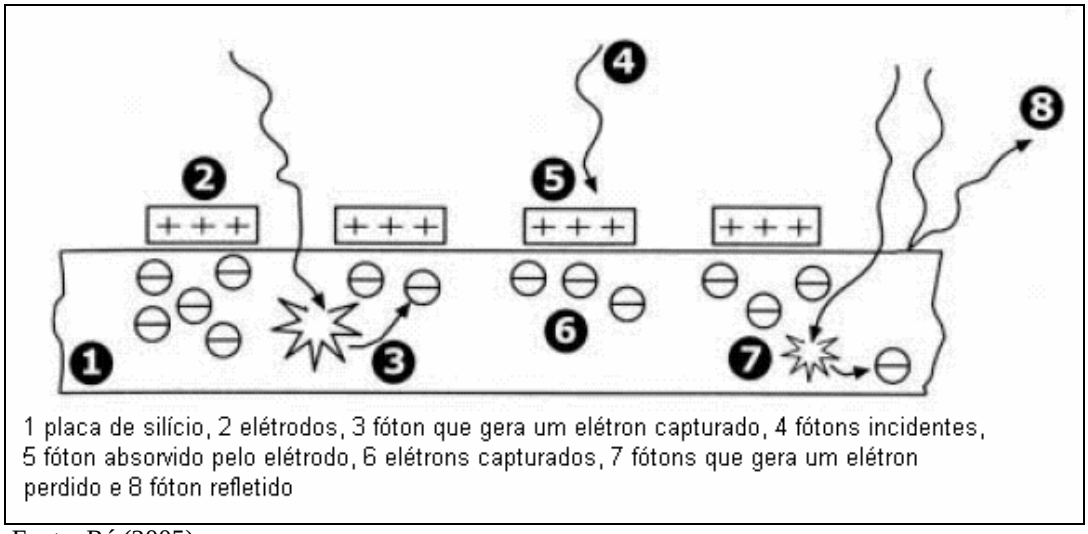


Fonte: Ré (2002)

Figura 7 – Chips CCD

Segundo Ré (2002), o funcionamento dos sensores CCD baseia-se no efeito fotoelétrico. Algumas substâncias têm a propriedade de absorver fótons e libertar no processo

um eletro. Os painéis solares fotovoltaicos foram concebidos com base neste princípio. Nestes últimos, os elétrons gerados ao incidir a luz, são recolhidos e convertidos em energia elétrica. O mesmo material empregado nos painéis solares, o silício, constitui a matéria-prima para a construção dos sensores CCD. Tipicamente consiste de uma placa de silício de 125 a 500 micrometros de espessura, onde é implantada uma rede de eletrodos que capturam e analisam os elétrons gerados pelo efeito fotoelétrico. Ainda segundo Ré (2002) cada trio de eletrodos atua como uma "ratoeira" eletrostática, acumulando os elétrons gerados na placa de silício. O elétrodo central de cada trio mantém uma carga positiva, enquanto que os restantes mantêm um potencial nulo. Deste modo os elétrons, cuja carga é negativa, acumulam-se em volta do eletrodo central à medida que a luz incide no detector. A Figura 8 demonstra a captura e acúmulo de luz em um sensor CCD.



Fonte: Ré (2005)

Figura 8 - Captura e acúmulo de luz em um sensor CCD

Ainda segundo Ré (2005), os trios de elétrodos são dispostos em colunas que cobrem a totalidade do sensor CCD. As diferentes colunas são isoladas entre si por um material que gera um potencial negativo permanente ao entrar em contato com a placa de silício, o que evita a contaminação entre colunas. As linhas de elétrodos consideradas perpendicularmente às colunas são designadas como linhas. Cada trio de elétrodos é uma peça fundamental do

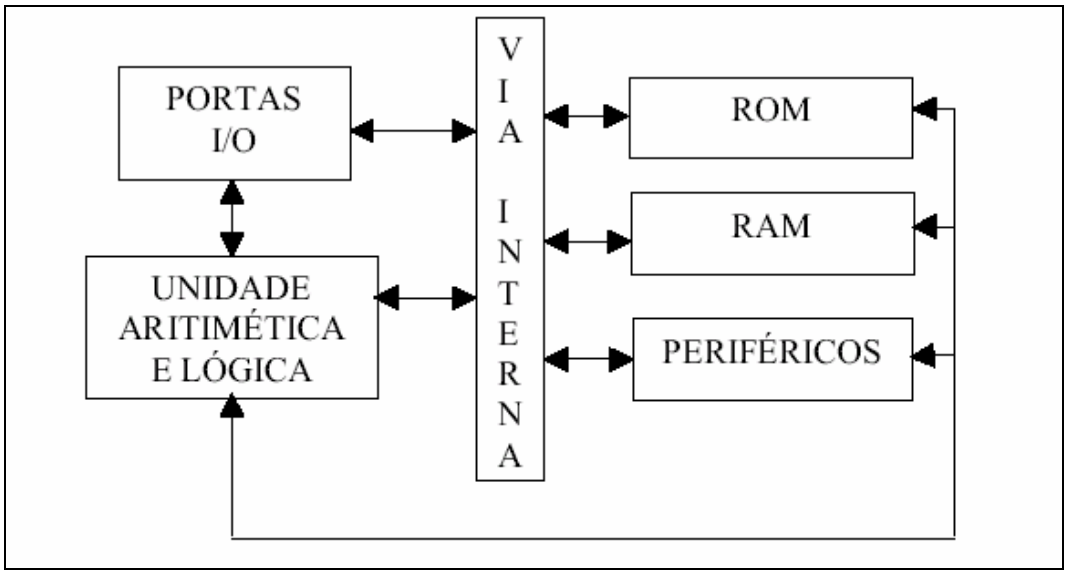
detector CCD e corresponde a um elemento da imagem digital final, que é designado *pixel*. O tamanho físico do *pixel* é variável. Existem *pixeis* retangulares e *pixeis* quadrados. As suas dimensões variam habitualmente entre 6 e 27 micrômetros.

Existem também sensores com tecnologia CMOS, que usam fotodiodos e transistores para os pixels. Nestes, o acesso a cada pixel é individual, não há movimento de cargas. Em geral, os sensores CMOS apresentavam qualidade de imagem inferior, menores resoluções e sensibilidades. Mas o custo é menor e o consumo de energia também. E a velocidade é maior pois não há o processo de transferência de cargas.

2.4 MICROCONTROLADORES

Basicamente microcontrolador é um componente eletrônico digital, que em uma só pastilha, *chip*, contempla os recursos encontrados em microcomputador comum. Abrindo assim um leque de possibilidades para implementação de sistemas embarcados, assim facilitando o desenvolvimento de sistemas de pequeno porte, sistemas para aplicações específicas.

Segundo Silva Junior (1998), a maioria dos microntroladores tem sua estrutura interna baseada na arquitetura proposta por Von Neumann, que pode ser observada na Figura 9. Ela prevê um único barramento de comunicação entre as memórias, CPU e seus periféricos.



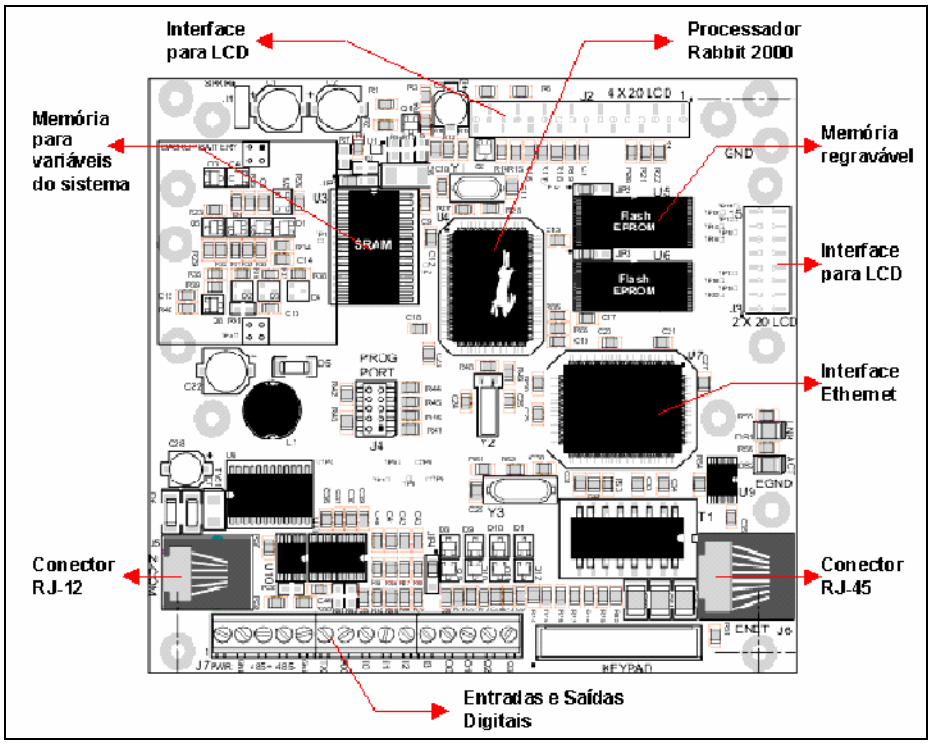
Fonte Silva Junior (1997)

Figura 9 – Arquitetura proposta por Von Neumann

2.4.1 Kit Rabbit 2000 TCP/IP

Denominado pelo fabricante de "*starter kit*", tem como principal vantagem fornecer uma série de facilidades, por exemplo, quatro portas seriais, saídas e entradas digitais, saídas para *display* digital, entrada para teclado, assim como já possuir memória FLASH e SRAM, e sua principal vantagem que é possuir interface Ethernet 802.3, que vem facilitar a início de projetos que utilizem algumas das facilidades citadas acima (Rabbit, 2000).

Na Figura 10 é apresentada à placa disponibilizada junto ao kit rabbit 2000 TCP/IP.



Fonte: Rabbit (2000)

Figura 10 - Placa disponibilizada junto ao Kit

O Quadro 1 descreve com mais detalhes as características da placa microcontroladora.

Componente	Especificação
Tamanho da placa	4.39” x 4.71” x 0.79” (109mm x 120 mm x 20mm)
Conectores	15 terminais, 1 RJ-12 e 1 RJ-45
Temperatura de operação	-20˚C até 70˚C
Umidade	5% até 95%
Tensão de entrada	9V até 40V DC
Corrente	100 mA – 12 VDC
Interface Ethernet	Conexão direta 10BaseT Ethernet via RJ-45
Entradas digitais	4 protegidas, 0V até 5V DC (proteção de –36V até 36 VDC máx.)
Saídas Digitais	4 open colletor, sinking (200 mA, 40VDC Máx.)
Microprocessador	Rabbit 2000TM compatível com Z180
Clock	18.432 MHz
SRAM	128K (suporta 32K-512K)
Flash EPROM	256K para programas de dados, 256K para armazenar arquivos (suporta 128K-512K)
Timers	7
Taxa de transmissão serial	Velocidade máxima assíncrona 115.200 bps para ambas as portas seriais
Watchdog/Supervisor	Sim
Time/Date Clock	Sim

Fonte: Censi (2001).

Quadro 1 – Características Rabbit 2000 TCP/IP

Agregado ao kit existe um ambiente para desenvolvimento para o software do microcontrolador, denominado Dynamic C, desenvolvido pela Z-World, que inclui em um mesmo ambiente as funções de edição, compilação-montagem, carregamento e depuração dos programas para os microcontroladores Rabbit e bibliotecas de função(CENSI, 2001). Algumas desta bibliotecas são:

a) costate.lib, cofunc.lib – inclui funções para gerenciamento multitarefa;
b) tcpip.lib – funções para conexão TCP/IP e troca de menssagem
c) math.lib – funções matemáticas;
d) rs232.lib – funções para transferência de dados;
e) rtclock.lib – funções para controle de tempo;
f) string.lib – funções para operações com cadeias de caracteres;
g) xmem.lib – funções para acesso a memória estendida;
h) sysio.lib – funções de suporte ao sistema de entrada e saída.

2.5 TRABALHOS CORRELATOS

Na seqüência são enumerados alguns trabalhos que possuem relação com trabalho a ser realizado:

a) protótipo de um sistema coletor de dados microcontrolado conectado a uma rede TCP/IP: trabalho desenvolvido por Vasques (2003), que consiste em um hardware, com um teclado, para coleta e armazenamento de dados, para posterior envio através de uma rede TCP/IP a um software específico, instalado em um computador conectado à mesma rede;
b) protótipo de um hardware para controle de freqüência acadêmica: trabalho de

desenvolvido por Silva (2002). Foi implementado um hardware para registrar a freqüência dos alunos e no fim da aula era enviado um e-mail ao professor com a lista de presenças. Esse dispositivo era conectado a uma rede TCP/IP;

c) protótipo de hardware e software para captura e visualização de imagens compartilhadas via interface digital serial diferencial balanceada: trabalho realizado por Santos (2002), onde foram implementados dois módulos, um para capturar as imagens e outro módulo com um *display* de cristal líquido, no qual é exibida essas imagens. Esses dois módulos comunicam-se através de uma interface serial;

d) construção de um protótipo (hardware e software) para segurança predial através de monitoração via câmera digital e *display* gráfico: trabalho implementado por Merege Neto (2004). Foi uma continuação do trabalho realizado por Santos (2002), mas o foco neste era aumentar a velocidade de transmissão da imagem.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo, são detalhadas a especificação e implementação do protótipo, através de diagramas e trechos do código fonte implementado.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O trabalho será composto de hardware e software. Abaixo são detalhados os requisitos funcionais (RF) e os requisitos não funcionais (RNF) de cada uma das partes.

O hardware deve:

a) coletar e armazenar os dados captados pela câmera (RF);

b) conectar-se à rede ethernet para comunicação com o microcomputador (RF);

c) enviar alerta quando ocorrer um imprevisto no ambiente que está sendo monitorado (RF).

O software deve:

a) gerenciar o dispositivo (RF);

b) receber os dados enviados pelo dispositivo através de uma rede ethernet (RF);

c) possibilitar a visualização das imagens captadas pelo dispositivo, assim como os dados dos demais sensores (RF);

d) ser independente de plataforma e sistema operacional (RNF).

3.2 ESPECIFICAÇÃO

O protótipo consiste de duas partes distintas, um hardware, que é um cliente TCP, e um software, servidor que rodará em um PC. Para realizar este trabalho se faz necessário

dividir sua especificação em três partes. São elas: especificação do hardware, do software embarcado para o micro controlador e o software a ser implementado para o micro computador. A especificações do hardware e do software embarcado são complementares.

A ferramenta Enterprise Architect 4.51 (SPARX, 2005), foi utilizada para desenvolvimento dos digramas da *Unified Modeling Language* (UML) sendo que os diagramas utilizados foram, de atividades, estados e classes (FURLAN, 1998).

3.2.1 Especificação do hardware

O hardware tem com função coletar as informações dos sensores, armazenar em memória para ser enviado através de uma rede ethernet ao software servidor desenvolvido para o PC.

O dispositivo é constituído de seis partes fundamentais: microcontrolador, memória, interface com rede *ethernet*, câmera para a captura das imagens, interface de controle para câmera e sensores. Seu esquemático é apresentado na Figura 11.

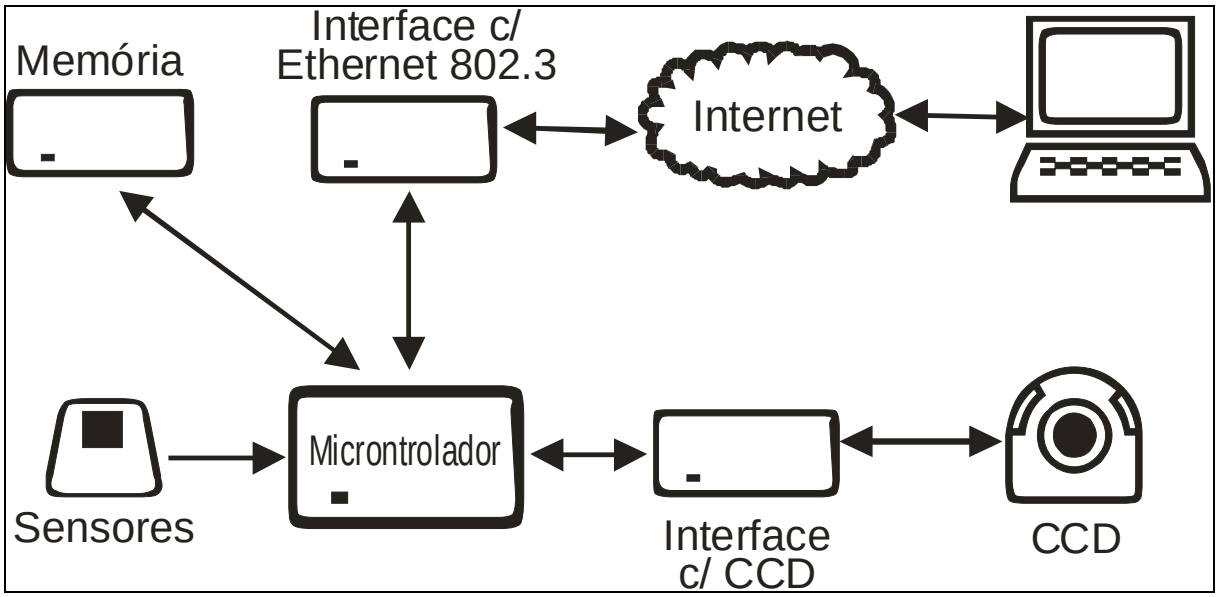


Figura 11 – Esquemático do hardware

Detalhando melhor a funcionalidade de cada uma das partes do dispositivo:

a) microcontrolador, é responsável por gerenciar a interface com a rede TCP/IP e a aquisição dos dados coletados, pelos sensores;

b) CCD é o principal dos sensores, realiza a captura das imagens, que pode ter uma resolução de até 360 x 270 *pixel*, onde cada *pixel* é de 8 bits representando assim uma imagem em 256 tons de cinza;

c) interface com o CCD, tem com finalidade facilitar o envio de comandos de configuração ao CCD e recebimento de dados do sensor CCD (imagem);

d) interface com rede, tem como função ser a ponte entre o microcontrolador e a rede ethernet 802.3, que por sua vez pode prover uma saída para a Internet;

e) sensores, de uso comercial, o dispositivo terá disponibilidade para conectar vários sensores que tenham respostas em nível lógico 0 ou 1, por exemplo sensores de presença e sensores de incêndio;

f) memória, será responsável para armazenar os dados capturados pelo CDD, pois estes podem ter o tamanho de até 95 kbytes.

Para aquisição das imagens é utilizado um módulo M4088, para controlar esse módulo existe um hardware que faz interface com o microcontrolador, tornando mais fácil à utilização deste sensor, pois ele se encarrega de configurar e controlar o módulo da câmera.

Para atender vários requisitos de uma só vez, optou-se pela utilização do Kit Rabbit 2000 TCP/IP, que possui um módulo de memória Flash EPROM de 256K, uma interface *ethernet* 802.3 de 10 mega bits, interface serial RS 232 que é utilizada para comunicar com a interface da câmera, 8 entradas a saídas digitais (I/O) para conectar os sensores e principalmente um microcontrolador Rabbit 2000, que dá todo um suporte e velocidade para controlar todos os recursos disponibilizados pelo kit, entre outros recursos não utilizados.

3.2.2 Especificação do software embarcado

Para a especificação do software embarcado foram utilizados os diagramas de estados e atividades propostos pela UML (FURLAN, 1998).

O dispositivo terá seu software implementado no conceito de um cliente TCP/IP. Optou-se por uma abordagem de programação *monothread* (somente uma linha de execução), que terá seu *loop* principal representado na Figura 12 através de seu diagrama de estados.

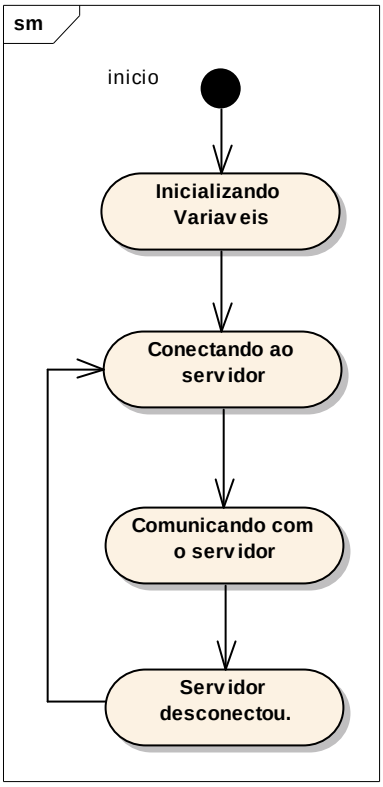


Figura 12 - Diagrama de estados do cliente

No diagrama acima, o primeiro estado é quando o software vai inicializar e alocar todos os recursos necessários, em um segundo momento o software solicita uma conexão com o servidor definido na inicialização e somente passa para o próximo estado quando a conexão estiver concluída. O terceiro estado é efetivamente quando o software está no seu maior nível de processamento, esse estado será mais bem detalhado posteriormente através de seu diagrama de atividades, já o quarto estado somente vai ocorrer quando houver algum

problema com a conexão entre o cliente, hardware, e o servidor que estará localizada no PC.

Para detalhar melhor o terceiro estado onde o software está realizando troca de mensagens com o servidor, o diagrama de atividades do estado é demonstrado na Figura 13.

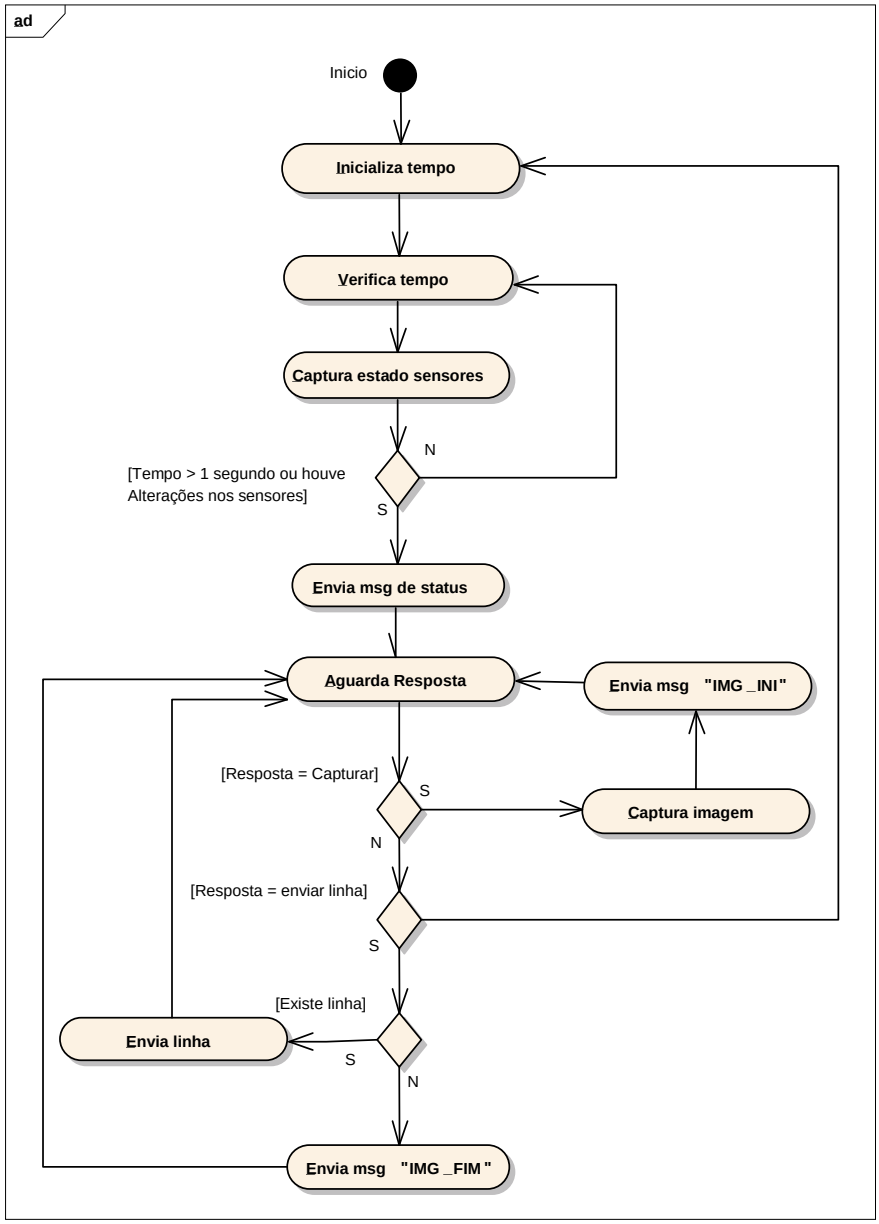


Figura 13 - Diagrama de atividades do cliente

Primeiro é inicializar o contador de tempo, pois a cada intervalo de tempo pré-definido ou quando ocorrer alguma alteração de estados dos sensores, o cliente vai se comunicar com o servidor, informando o seu id, número de identificação única, e mais os estado dos sensores, como demonstrado na Figura 14, onde os 10 primeiros bytes são o *id* dispositivo e os 4 bytes

restantes é o estado dos sensores, após isso o software fica aguardando uma resposta do servidor.

id	status
10	4

Figura 14 – Protocolo id / status

Se a resposta for “C”, captura, o cliente vai realizar a captura da imagem armazenando em uma estrutura de vários *arrays* onde cada *array* guardará o conteúdo de uma linha da imagem, (o processo de captura da imagem será detalhado melhor posteriormente), zera o contador de linhas enviadas e envia ao servidor uma mensagem “IMG_INI”, representando que a imagem foi capturada e está pronta a ser enviada.

Já se a resposta for “N”, *next*, e o contador de linhas enviadas for menor que o número de linhas a enviar o cliente envia a linha da imagem que está na vez e incrementa o contador de linhas enviadas, senão houver mais linhas a enviar ele transmite “IMG_FIM”, que representa o fim da imagem.

Se não for nenhuma das mensagens definida o cliente desconsidera a mensagem e volta ao início do *loop* reinicializando o contador de tempo. O software só sairá deste *loop* se houver desconexão com o servidor.

No processo de captura da imagem o microcontrolador não realiza comunicação diretamente com a câmera, mais se comunica via porta serial RS232, com a interface serial da câmera que é responsável por todo gerenciamento e captura das imagens. A velocidade, dessa transmissão é de 57.600 bps, essa velocidade torna a captura da imagem lenta, demorando em torno de 5 segundos.

Para operar a interface com câmera, o microcontrolador envia um comando via serial, e aguarda uma resposta da interface, por exemplo, para obter uma imagem é enviado o caractere “E” (45H), ao receber esse caractere, a interface captura a imagem da câmera, com

dimensões de 270 linhas com 361 colunas, ao término, inicia a transmissão da imagem via serial. A interface transmite os 97470 bytes capturados em série onde cada byte representa um *pixel*. Cara *pixel* tem seu valor entre 0 e 255, representando assim 256 tons de cinza cada.

Na Figura 15 apresenta o processo de captura de imagem através de seu diagrama de atividades.

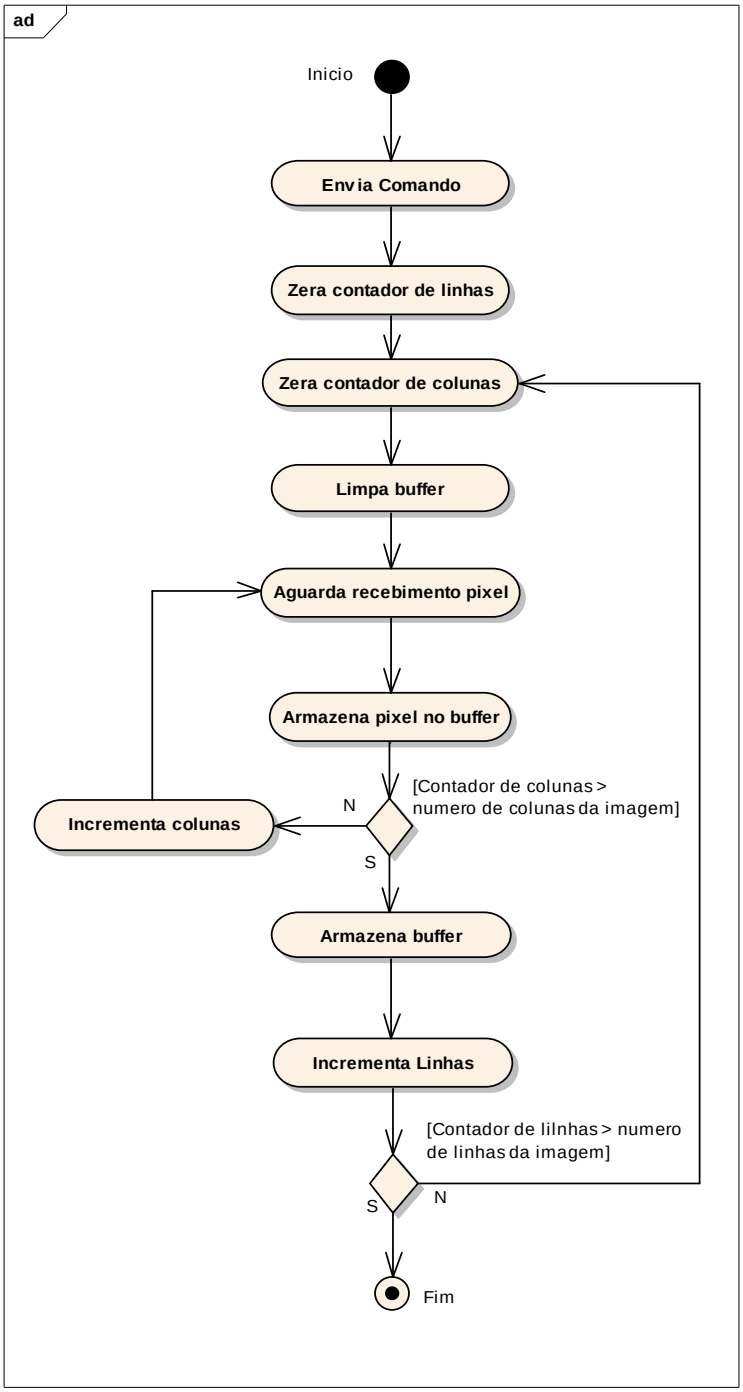


Figura 15 – Digrama de atividades para captura da imagem

O microcontrolador envia, via serial, para a interface de controle da câmera um caractere ‘E’ (45H), que faz com que a interface realize a captura.

Imediatamente após a transmissão do comando de captura, inicia-se a recepção dos bytes que representam a imagem capturada. Para cada ciclo do processo serão recebidos 270 linhas com 361 bytes cada.

3.2.3 Especificação do software para PC

O software é especificado com uma abordagem de orientação a objetos. Para a especificação desta tarefa foram utilizados os diagramas de atividades e de classes (FURLAN, 1998).

É dividido em duas camadas, uma é a camada da aplicação, responsável pela visualização dos dados e gerenciamento dos dispositivos, e uma segunda camada que é o servidor que realiza toda a comunicação com os dispositivos. O uso de duas camadas torna mais fácil posteriores manutenções e alterações.

A aplicação não tem nenhum acesso à rede, ela apenas vai se comunicar com a segunda camada. O objeto em comum entre as duas camadas é o Dispositivo que tem seu diagrama de classes demonstrado na Figura 16. Outra grande vantagem da utilização de duas camadas é abrir espaço para possível visualização dos dados em mais de um software.

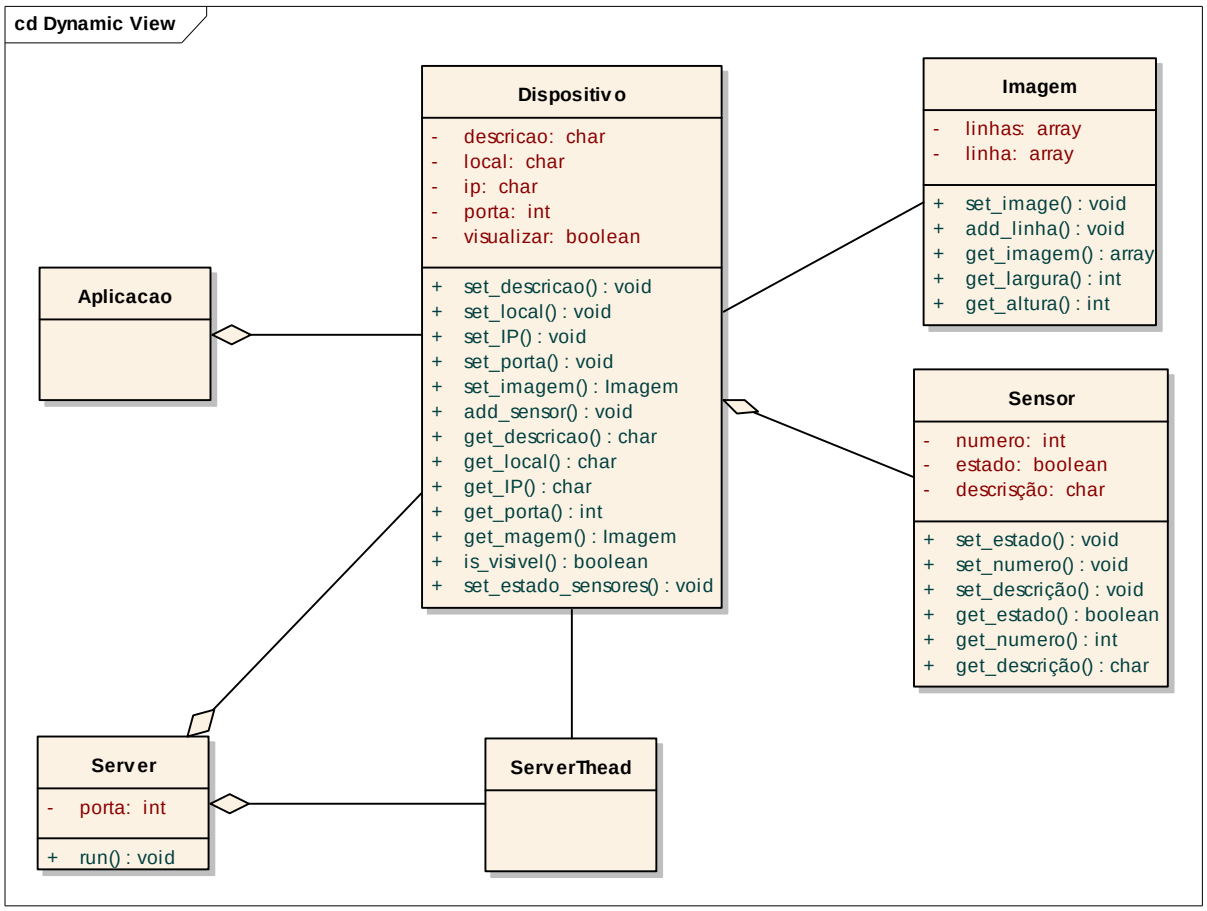


Figura 16 – Diagrama de classe Dispositivo

Cada objeto do tipo dispositivo é uma representação lógica dos dispositivos remotos, contendo como atributos todos os atributos que cada dispositivo remoto contém, e mais alguns, como por exemplo descrição, local onde está instalado e se este dispositivo está sendo visualizado por alguma aplicação.

O servidor está dividido em duas partes distintas, o *loop* principal e o tratamento da conexão com os dispositivos. O *loop* principal é uma *thread*. Para cada nova conexão de um cliente é iniciada uma nova *thread*, para realizar toda troca de mensagens. Essa nova *thread* só será destruída quando ocorrer uma desconexão do cliente.

O *loop* principal do servidor tem seu diagrama de atividades representado na Figura 17.

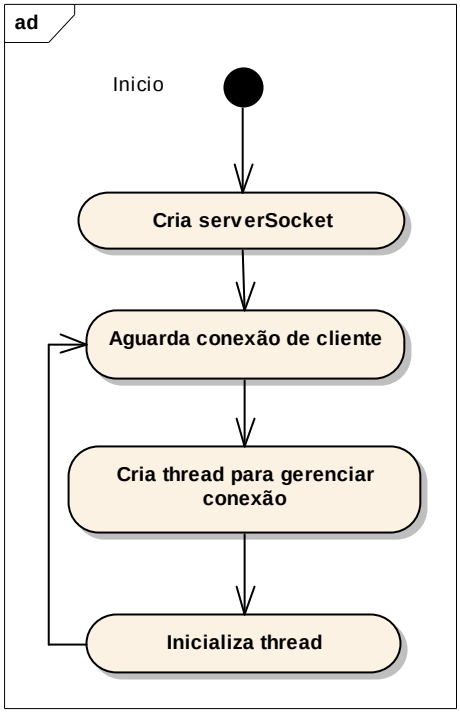


Figura 17 – Digrama de atividades *loop* principal server

O primeiro passo para o servidor e instanciar o *ServerSocket* para ficar monitorando a porta 5000, após isso fica esperando a conexão de algum cliente. Assim que reconhece a conexão do cliente cria e inicializa uma *thead* para gerenciar a mesma, após inicializar a nova *thead*, volta imediatamente a esperar um novo cliente conectar.

A Figura 18 mostra a *thread* principal de controle da conexão do cliente.

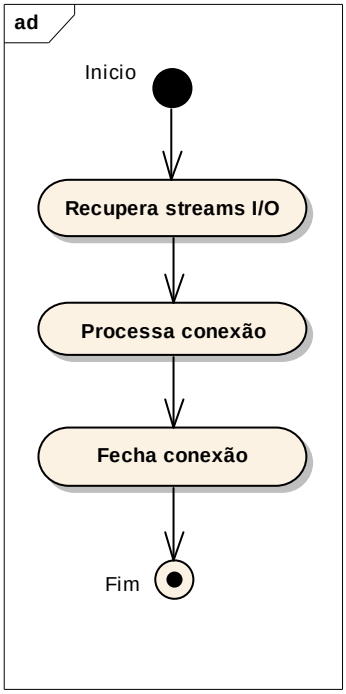


Figura 18 – Diagrama atividades *thread* de conexão do cliente

Ao iniciar a *thread*, a primeira coisa a fazer é recuperar os canais de comunicação através dos *sockets*, *InputStream* e *OutputStream*. Após isso, o processo mais complexo que seria a troca de mensagens com o cliente, se por ventura ocorrer algum problema durante a troca de mensagem, a *thread* irá fechar a conexão.

Através da Figura 19 é demonstrado o processo principal da *thread*, a troca de mensagens entre o servidor e o cliente.

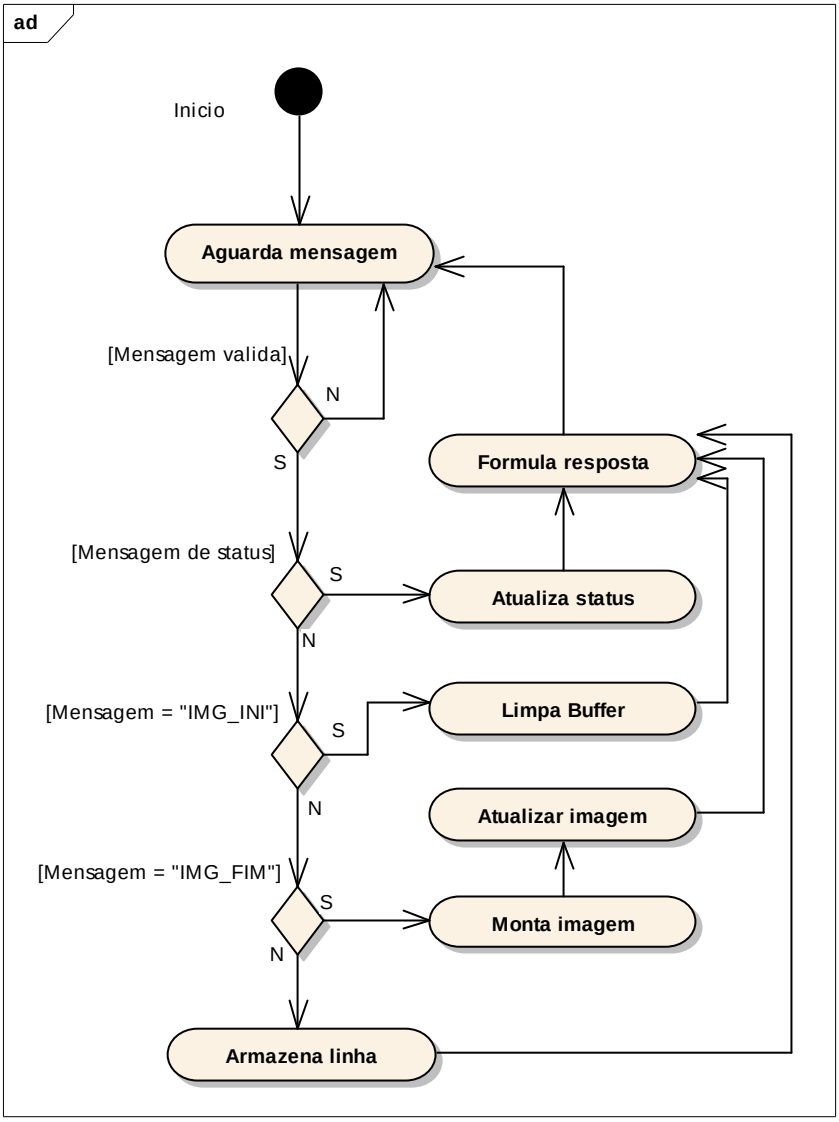


Figura 19 - Diagrama de atividades troca de mensagem server/cliente

A *thread* que trata a conexão sempre fica esperando alguma mensagem, e conforme a mensagem recebida, realiza uma ação e envia ao cliente uma resposta, o servidor fica esperando quatro tipos de mensagens diferentes, e desconsidera mensagens que são inválidas. As mensagens que o servidor espera são:

a) mensagem de *status*, já detalhada na especificação do software embarcado, é uma mensagem simples onde consta o *id* do dispositivo concatenado junto com os estados dos sensores. Recebendo essa mensagem o servidor verifica se o dispositivo em questão está sendo utilizado para exibir as imagem envia a mensagem “C” ao dispositivo, para ele realizar a captura da imagem senão envia

apenas uma mensagem de “K” que significa uma confirmação de recebimento;

b) “IMG_INI”, o servidor limpa o *buffer* que usa para armazenar a imagem, e envia ao dispositivo a mensagem “N”, que representa para o dispositivo enviar próxima linha da imagem;

c) “IMG_FIM”, o servidor monta a imagem de forma que possa ser exibida pelo dispositivo e em seguida dispara o método responsável por exibir a imagem, e envia ao dispositivo uma resposta “K”, com significado de confirmação do recebimento;

d) se não é nenhuma das mensagens acima e é uma mensagem válida, o servidor considera como uma nova linha da imagem, assim concatenando essa nova linha ao buffer, e envia uma resposta “N”.

3.3 IMPLEMENTAÇÃO

Nesta seção é apresentada a forma como foi implementado o software e os resultados obtidos.

3.3.1 Técnicas e ferramentas utilizadas

Para implementação do protótipo, na parte de hardware foi utilizado o *starter kit Rabbit 2000 TCP/IP*, que agrega em uma só placa muita dos requisitos exigidos na especificação e ainda vem acompanhado de um ambiente de programação, *Dynamic C,* para desenvolvimento do software embarcado na linguagem C. O ambiente *Dynamic C* possui várias bibliotecas, que facilitam o uso dos recursos do kit, e tem como ponto positivo a possibilidade de depuração do software (RABBIT, 2000).

Para a implementação do software para o PC, optou-se pela linguagem Java, por se tratar de uma linguagem multiplataforma, ter ampla documentação e possuir vários ambientes de programação gratuitos. Para realizar a implementação foi utilizado o ambiente *NetBeans 3.6*, pois provê a possibilidade de desenvolvimento de interfaces com o usuário com maior facilidade.

3.3.2 Hardware

Com a utilização do kit Rabbit 2000, o desenvolvimento do hardware tornou-se mais simples, não existindo a necessidade de implementar vários dos requisitos levantados na especificação, pois na placa microcontroladora do kit já contemplava vários desses requisitos. Tendo seu *layout* apresentado na Figura 20.

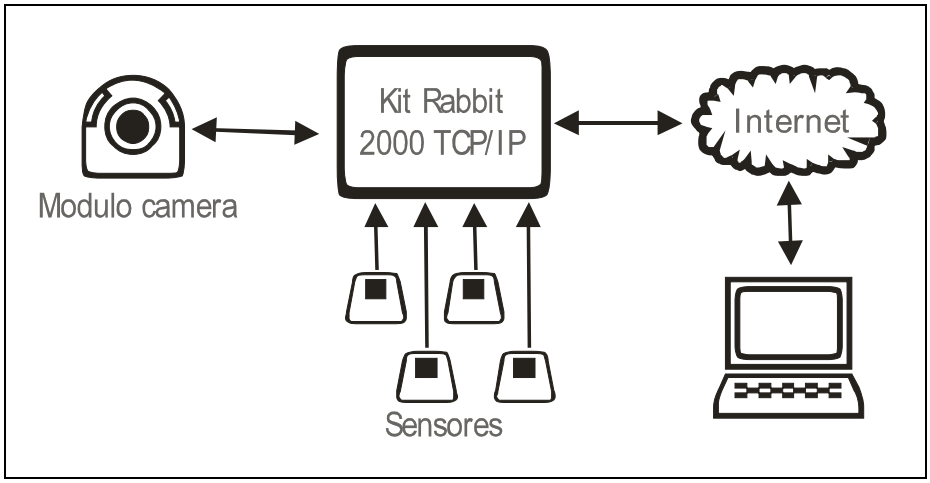


Figura 20 – *Layout* do hardware

O centro do protótipo é a placa do kit Rabbit, ela é conectada ao módulo da câmera através de uma porta serial, e os sensores estão conectados nas entradas digitais e por fim o hardware conecta-se a rede 802.3 através de um conector RJ 45.

3.3.3 Software embarcado

O Quadro 2 demonstra a definição de algumas variáveis necessárias para realizar uma conexão TCP/IP.

```
#define MY_IP_ADDRESS       "10.0.0.200"      // Endereço IP do dispositivo
#define MY_NETMASK          "255.255.255.0"   // Mascara de rede
#define MY_GATEWAY          "10.0.0.2"        // Endereço IP do Gateway

//#define TCPCONFIG 5        // Obter IP via servidor DHCP

#use "dcrtcp.lib"
```

Quadro 2 – Definição de variáveis para conexão de TCP/IP

O primeiro bloco demonstra a definição das variáveis necessárias pra conexão TCP/IP, mas existe a possibilidade de utilizar um servidor DHCP para obter um IP dinamicamente, para usar essa opção e só utilizar como mostrado na linha 5, quando a variável TCPCONFIG é inicializada com valor 5. As bibliotecas utilizadas pelo *Dynamic C* desconsideram as outra variáveis e realizam toda a negociação para obter um IP. Também é definido o uso da biblioteca dcrtcp.lib que contém todas as funções para abertura, inicialização, envio e recebimento das mensagens através de *sockets*.

O Quadro 3 apresenta a inialização de algumas variáveis básicas para o funcionamento do protótipo. São definidos o endereço e a porta do servidor, tamanho do *buffer* da porta serial, velocidade da comunicação serial, uma variável do tipo *socket*, um *buffer* pra guardar as linha temporárias e por fim um *array* para armazenar os ponteiros para as linhas armazenadas através de alocação dinâmica de memória.

```
#define DEST                        "10.0.0.2"      // Definição do IP do Servidor
#define PORT                        5000            // Porta do servidor

#define CINBUFSIZE 15               // Define tamanho do buffer para leitura da porta serial
#define COUTBUFSIZE 15

#ifndef _232BAUD                    // Define a velocidade da comunicação serial
#define _232BAUD 57600
#endif

tcp_Socket socket;                  // Variável do socket

static char Buf[244];               // Array do buffer de leitura temporário
static unsigned long linhas[160];   // Array que guarda ponteiro da linhas alocadas
```

Quadro 3 – Variáveis Básicas

Como mencionado anteriormente, as linhas da imagem são alocadas dinamicamente, para evitar perda de sincronia durante a captura das imagens é realizado a alocação de toda a memória necessária no início do programa. No Quadro 4 é exemplificado esse processo, onde a função *xalloc* recebe como parâmetro a quantidade de bytes que deseja alocar e após alocar a memória retorna um ponteiro, que é armazenado no *array* linhas.

```
for (i = 0; i < 160; i++){
        linhas[i] = xalloc(244);
}
```

Quadro 4 – Alocação de memória

Este processo resultará em um array de ponteiros para os blocos de memória alocados dinamicamente, como representado na Figura 21, onde é ilustrado como fica a memória após realizar o procedimento acima.

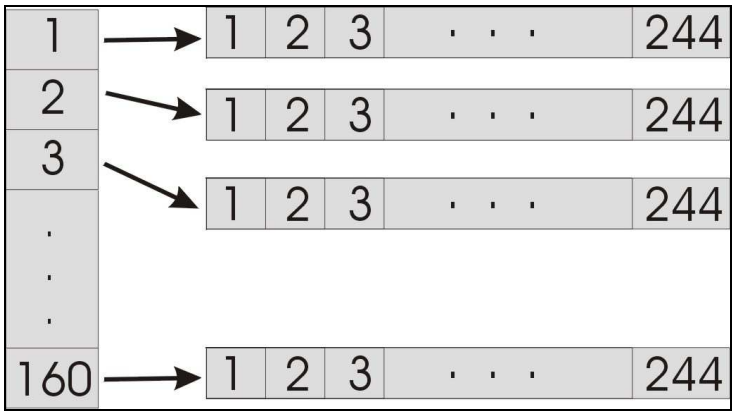


Figura 21 – Disposição da memória alocada

Para realizar uma comunicação TCP/IP é necessário inicializar e abrir um *socket*, o processo é exibido no Quadro 5. São três passos a serem realizados, primeiro é chamar a função *Sock_init* que é responsável por iniciar o módulo *ethernet* com os parâmetros definidos no início do código. Em seguida é resolvido o endereço do servidor pré-definido, se não for possível resolver esse endereço o sistema é abortado. Por fim é realizada a abertura do *socket* através da função *tcp_open*, que tem como parâmetros um ponteiro para variável do tipo *socket* e a porta local de comunicação, se informado zero e função recupera a próxima porta disponível, o endereço, resolvido anteriormente, e a porta do servidor e o último parâmetro é ignorado.

```
Sock_init();

if( 0L == (destIP = resolve(DEST)) ) {
        exit(2);
}

tcp_open(&socket,0,destIP,PORT,NULL);
```

Quadro 5 – Inicialização e abertura do *socket*

No Quadro 6 é exibido o código do *loop* principal do software embarcado, onde é realizado o recebimento, envio de mensagens e chamada de funções.

```
Do {
      // define time out de recebimento de mensagem
     sock_wait_input(&socket, 20, NULL, &status);

     // aguarda recebimento de alguma mensagem
     bytes_read = sock_fastread(&socket, buffer, sizeof(buffer)-1);

     if(bytes_read>0) {
          buffer[bytes_read] = '\0';                    // Insere character de fim de palavra

          if (buffer[0] == 'K'){
              waitfor( DelayMs(500) );                 // Aguarda meio segunda
              str_tmp = monta_str_status()             /* Chama função que monta string contendo
                                                       ID concatenado com o estado dos sensores*/
              sock_write(&socket, str_tmp, 14);        // Envia retorno da função
           }

           if (buffer[0] == 'C'){
              captura_imagem();                          //Chama função de captura da imagem
              sock_write(&socket, "IMG_INI", 7);  // Envia msg de inicio da imagem
              printf("iniciando transmição\n");
              i = 0;                                     // Inicia contador de linhas enviadas
           }

           if ((buffer[0] == 'N') && (i<160 )){
               xmem2root(Buf, linhas[i], 244);       // Recupera linha alocada dinamicamente
               sock_write(&socket,Buf,244);          // Envia linha
               i++;
           }

           if ((buffer[0] == 'N') && (i == 160)){
               sock_write(&socket,"IMG_FIM",7);  // Envia msg de fim da imagem
               printf("Fim da Transmição\n");
               i++;
           }
     }
} while(tcp_tick(&socket));     // Sai do loop se perder a conexão
```

Quadro 6 – *Loop* principal do software

Esclarecendo algumas das funções utilizadas que são disponibilizadas pela biblioteca dcrtcp.lib:

a) *sock_wait_input*, é utilizada para definir um *time out* para o recebimento das mensagens, tem como parâmetros, o ponteiro para *socket*, o tempo para ocorrer o *time out*, um terceiro parâmetro que na versão 8.10 deve ser informado *NULL* e o

último parâmetro é ponteiro para função que será chamada quando ocorrer o *time out*;

b) *sock_fastread*, fica aguardando até o recebimento de alguma mensagem enviada pelo servidor, mais tem um tempo máximo definido pela função *sock_wait_input*, essa função tem como parâmetros o ponteiro para o *socket*, *buffer* onde vai ser armazenado o conteúdo recebido e a quantidade máxima de bytes à serem lidos. O retorno é a quantidade de bytes lidos;

c) *sock_write*, sua função é realizar o envio das mensagens, tem como parâmetros o ponteiro para o *socket*, o conteúdo a ser enviado e o número de bytes a ser enviado.

Além destas funções foram utilizadas algumas outras funções disponibilizadas pelo ambiente de desenvolvimento:

a) *waitfor*, nada mais é do que um *delay*;

b) *xmem2root*, tem função de recuperar conteúdos alocados na memória, tem como parâmetros o *buffer* onde vai ser armazenado o conteúdo recuperado, o ponteiro para a memória e quantidade de bytes a ser recuperado.

A função para captura da imagem consiste em enviar para a câmera o comando para captura representado pelo caractere ‘E’ (45H).

Imediatamente após a transmissão do comando de captura, inicia-se a recepção dos *bytes* que representam a imagem capturada. Para cada ciclo do processo serão recebidas 270 linhas com 361 bytes, sendo que somente serão considerados 160 linhas com 244 colunas, para reduzir o tamanho da imagem a ser transmitida.

Para a comunicação serial, as seguintes funções, pertencentes à biblioteca de funções rs232.lib, são utilizadas:

a) *serCopen*, efetua a abertura do canal serial RS-232 com a velocidade de

transmissão especificada no parâmetro;

b) b) *serCputc*, transmite um caractere, informado no parâmetro, pelo canal serial;

c) c) *serCgetc*, efetua a recepção de um caractere, disponível no *buffer* de leitura do canal serial. Caso nenhum caractere esteja disponível para leitura a função retornará o valor "–1".

No Quadro 7 é exibida a função responsável pela captura das imagens.

```
Void captura_imagem(){

   serCputc('E');       // Envia character 'E' via serial para interface da camera

    for (j=0;j<161;j++){                    // percorre 161 linhas esperadas
        for (i=0;i<361;i++) {               // percorre 361 pixels de cada linha
            while ((Pxl=serCgetc()) == -1);        // Agurada dado valido
            if  ( i < 244)
                Buf[i] = Pxl;                      // concatena pixel do buff
             if ((i == 244) && (j < 160))
                root2xmem(linhas[j], Buf, 244);  // aloca buffer na memoria
        }
    }
}
```

Quadro 7 – Função de captura da imagem

3.3.4 Software PC

Como já detalhado na especificação, o software do PC utilizará uma arquitetura de duas camadas e é orientado a objetos. As duas camadas são: interface com o usuário e o servidor responsável por toda a troca de mensagem.

Da interface é apresentado o trecho do código responsável por exibir as imagens. Para exibir as imagens implementou-se a classe Imagem *extends* de *JLabel*, sobrescrever os método *paint*, pois o mesmo é responsável pelo visualização do objeto. No Quadro 8 é exibido o código dessa implementação.

```
class Imagem extends JLabel {

    private BufferedImage img;

    public void setImagem(BufferedImage img){
        this.img = img;
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.drawImage(img, 0, 0, 244, 160, null);
    }
}
```

Quadro 8 – Classe imagem

Nessa classe foram adicionados um atributo do tipo *BufferedImage*, que é como próprio nome sugere um *buffer* para armazenar uma imagem, um método *setImagem*, que é usado para atualizar a imagem em exibição, e por fim foi sobrescrito o método *paint* onde e forçada a exibição da imagem adiciona anteriormente, através do método *drawImage* que recebe como parâmetros a imagem a ser exibida, ponto inicial em *x* e *y*, onde será iniciado a impressão da imagem, em seguida a largura e a altura da imagem, e o quinto parâmetro é ignorado.

O Quadro 9 exibe o trecho do código. O Quadro 9 exibe o código para a criação do objeto *BufferedImag*.

```
BufferedImage img = new BufferedImage( ((ArrayList)imagem.get(1)).size(),
                                       imagem.size(),
                                       BufferedImage.TYPE_INT_RGB );
Color c;
int pixelInt;
for (int i = 0; i < imagem.size(); i++ ){
      ArrayList linha = (ArrayList)imagem.get(i);
      for(int j = 0; j < linha.size(); j++){
            try{
                pixelInt = Integer.parseInt( linha.get(j).toString() ) ;
                c = new Color( pixelInt, pixelInt, pixelInt);
            }catch( Exception e ){
                c = new Color( 125, 125, 125);
            }
            img.setRGB(j, i, c.getRGB() );
       }
}
```

Quadro 9 – Criação da imagem a ser exibida

Nesse processo é criado um objeto img do tipo *BufferedImage*, na criação são passados como parâmetros a largura, altura e tipo de dados esperados no momento de inserir os *pixels*. Como definido na especificação a imagem é um *array* de linhas, onde cada linha é um *array* de *pixel*, essa estrutura é percorrida, criando um pixel para cada posição, para cada pixel é necessário criar um objeto do tipo *Color*, na criação desse tipo de objeto são passados como parâmetro os valores que compõem uma cor, *red* (R), *green* (G) e *blue* (B), esses valores variam de 0 a 255, no caso de uma imagem em tons de cinza os calores devem ser iguais, é importante prever uma exceção na criação da cor de cada *pixel*, pois alguns valores recebidos podem estar fora da faixa esperada, quando isso ocorre é colocado um valor de cinza intermediário. E por fim é inserida essa cor em cada ponto do objeto imagem.

Agora vamos tratar da implementação da segunda camada, o servidor responsável pela comunicação com os dispositivos. Esta camada é uma nova *thread* do software, que por fim ira criar mais uma *thread* para cada dispositivo que conectar-se ao servidor. A *thread* principal é exibida no Quadro 10.

```
public class Server implements Runnable {

   private static int porta = 5000;
   private HashMap dispositivos;

   Server(HashMap d){
         this.dispositivos = d;
   }

   Server(int porta, HashMap d){
         this.dispositivos = d;
         this.porta = porta;
   }

   public void run() {
         try{
               ServerSocket listener = new ServerSocket(porta);
               Socket socket;

               while(true){
                    socket = listener.accept();

                    ServerThead conn = new ServerThead(socket, this.dispositivos);
                    (new Thread(conn) ).start();
               }
         } catch (IOException ioe) {
                System.out.println("IOException on socket listen: " + ioe);
         }
    }
}
```

Quadro 10 – *Thead* principal do sevidor

Esta classe tem como atributos a porta, que o *ServerSocket* vai escutar, e um ponteiro para um *HashMap* com os dispositivos já reconhecidos anteriormente, e também um método *run* que terá um *loop* eterno, que somente será quebrado se ocorrer alguma exceção. Este *loop* fica aguardando a conexão de algum cliente, através do método *accept*, quando recebe a solicitação de uma conexão, cria um *socket* e retorna o mesmo. Assim que o ocorre a conexão, o servidor cria uma *thread* para o tratamento dessa conexão, passando como parâmetro o *socket* recebido e um ponteiro para a lista de dispositivos, em seguida apenas inicia a execução da *thread* e volta a aguardar uma nova conexão.

No Quadro 11 é detalhado o método *run* da *thread* criada para cada conexão.

```
Public void run () {

   try {
      getStreams();
      processConnection();
   }

   catch ( EOFException eofException ) {
       System.out.println( "Cliente terminou a conexão" );
   }

   finally {
        try {
            closeConnection();
        }
        catch ( Exception e) {
            System.out.print(" Erro ao fechar connection ");
         }
   }
```

Quadro 11 – Método *run* da classe *ServerThead*

Este método faz a chamada de outros dois métodos, *getStreams*, que recupera os canais de leitura e escrita como o cliente, e *processConnection*, responsável pelo processamento da conexão. O que mais vale ressaltar neste método é o tratamento de exceções que contém, o método *processConnection*, possui um também um *loop* eterno, mais ocorrendo qualquer problema com a comunicação entre o servidor e o cliente, o sistema abortará o método *processConnection* e entra no tratador de exceções que posteriormente realizar o fechamento do *socket*.

No Quadro 12 é detalhado melhor o *loop* eterno do método *processConnection*.

```
while (true){
      int i = input.read(aux);
      msg = new String(aux, 0, i);
      if ( msg.length() == 14 ){
         id = getId(msg);
         estadoSensores = getEstado(msg);
         if ( !dispositivos.containsKey( id ) ){
                  DispositivoRemoto disp = new DispositivoRemoto(
                                                      connection.getInetAddress().toString(),
                                                      connection.getPort() );
                  dispositivos.put( id, disp );
         }
         if ( ( (DispositivoRemoto)dispositivos.get(id) ).isVisivel() ){
                  output.write( "C".getBytes() );
         }else{
                  output.write( "K".getBytes() );
         }
         ( (DispositivoRemoto)dispositivos.get(id) ).setEstadoSensores(estadoSensores);
         output.flush();
         continue;
      }
      if ( msg.equals("IMG_INI") ) {
         imagem.clear();
         output.write( "N".getBytes() );
         output.flush();
         continue;
      }
      if ( msg.equals("IMG_FIM") ) {
         ((DispositivoRemoto)dispositivos.get(id)).setImagem(imagem);
         output.write( "K".getBytes() );
         output.flush();
         continue;
      }
      // se nenhuma das msg esperadas considera linhas da imagem
      ArrayList linha = new ArrayList();
      for (int ind = 0; ind < msg.length(); ind++ ){
            linha.add( new Integer( aux[ind] ) );
      }
      imagem.add( linha );
      output.write( "N".getBytes() );
      output.flush();
}
```

Quadro 12 – *Loop* da função *processConnection*

Este *loop* segue as especificações realizadas anteriormente, fica aguardando uma mensagem do cliente, se a mensagem recebida tem tamanho igual a 14 considera como uma mensagens de status contendo o ID e o status dos sensores, se é a primeira vez que esse

dispositivo está se conectando, cria um novo objeto do tipo *dispositivoRemoto*, após se o dispositivo está sendo visualizado por alguém, envia o caractere “C” solicitando o envio de uma nova imagem, se não está sendo visualizado apenas envia o caractere “K” somente confirmando o recebimento da mensagem. Se a mensagem recebida for igual o “IMG_INI”, limpa o *buffer* de imagem e envia o caractere “N” solicitando o envio da próxima linha da imagem. Se a mensagem for “IMG_FIM”, seta essa a nova imagem recebida no dispositivo e envia uma resposta “K” confirmando o recebimento. E por fim se não é nenhuma das possibilidades acima o software considera uma linha da nova imagem, converte os dados recebidos em números inteiros e posteriormente armazena a linha no buffer e envia uma resposta “N”, solicitando o envio de mais uma linha.

3.3.5 Operacionalidade da implementação

A operacionalidade do hardware é bastante simples, basta conectar o hardware em uma rede, conectar os sensores e a fonte de alimentação.

Já o software implementado para o PC, independente de plataforma, tem sua tela principal apresentada na Figura 22.

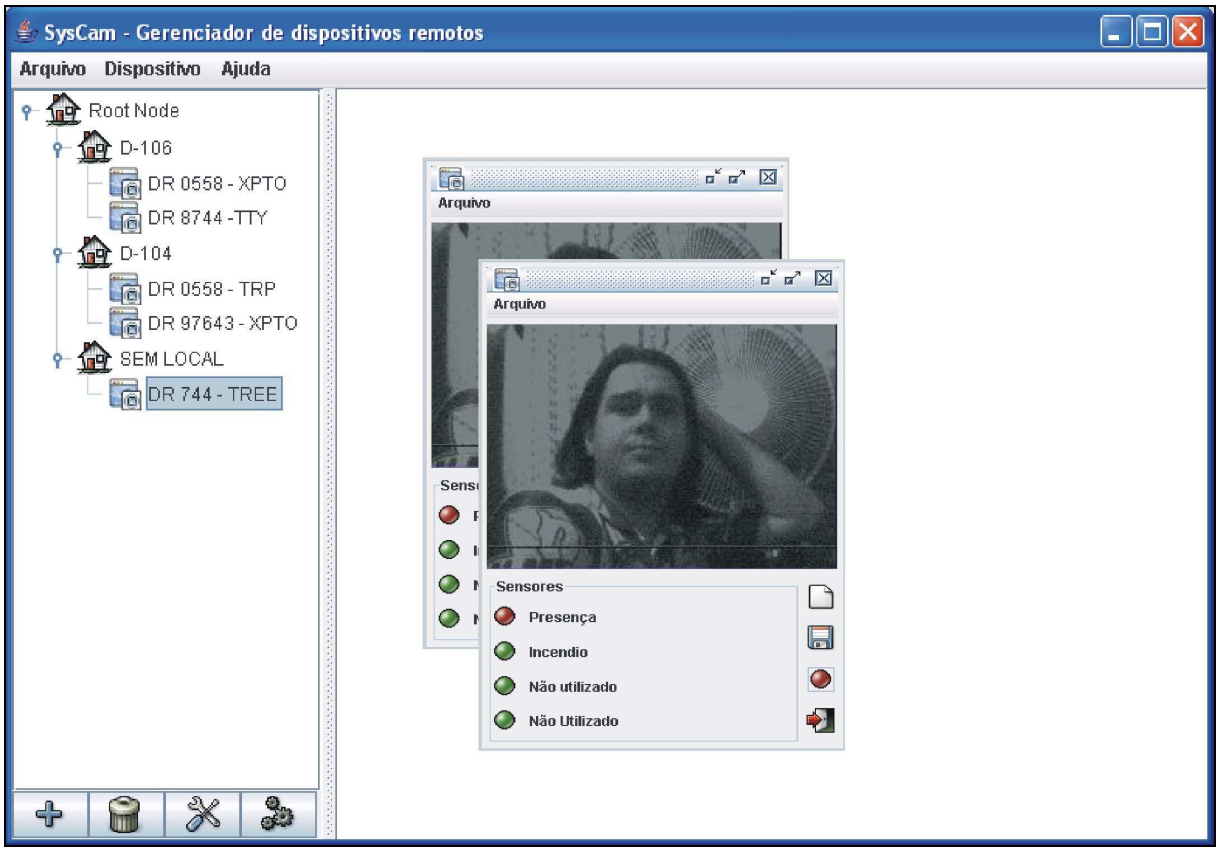


Figura 22 – Tela principal do software

Esta tela está dividida em três partes, no lado esquerdo existe uma árvore onde é exibida como os dispositivos estão distribuídos, onde os filhos de “Root Node” são os locais onde os dispositivos estão instalados, enquanto os filhos dos locais, folhas da árvore, são os dispositivos. Na parte inferior da árvore existem alguns atalhos para manipulação da mesma, que serão melhor detalhados quando for demonstrado o menu Dispositivo. Para visualizar, o conteúdo de um dispositivo, basta dar um duplo *click* na folha correspondente. Ao realizar está operação entra em questão a segunda parte da tela, o *desktop* na parte central da tela, onde são exibidos os dispositivos com seu conteúdo. E na parte superior do software está localizado o menu, Arquivo, Dispositivo e Ajuda.

O menu arquivo tem seu conteúdo demonstrado na Figura 23.

Figura 23 – Menu Arquivo

Neste menu existe apenas opções de sair e propriedades do software, que tem sua tela demonstrada na Figura 24. Até o momento a única opção de configuração é poder configurar qual porta o servidor vai monitorar.

Figura 24 – Propriedades

O menu Dispositivo tem com opções de Adicionar novos nodos na árvore, Remover e editar as Propriedades, essas opções são válidas para os dispositivos e para os locais. Já opção de Executar é válida somente para os dispositivos. Este menu está demonstrado na Figura 25.

Figura 25 – Menu dispositivo

Ao adicionar ou editar propriedades de um dispositivo, é exibida a janela demonstrada na Figura 26.

Figura 26 – Propriedades do dispositivo

Esta janela está dividida em duas partes, as propriedades do dispositivo, onde a descrição e local podem ser alterados e ID, IP e porta são enviados pelo dispositivo e não podem ser alterados, e o detalhamento dos sensores, onde pode ser incluída uma descrição de cada um dos sensores e também pode ser marcado se o mesmo irá produzir alerta quando seu estado for alterado.

Na Figura 27 é demonstrada a janela onde é exibida a imagem e os estados dos sensores do dispositivo.

Figura 27 – Janela do dispositivo

Esta janela está dividida em duas partes, a parte superior onde é exibida a imagem, e a parte inferior onde é exibido o estado dos sensores, se o indicador estiver verde indica que seu estado é inativo e se vermelho indica que o mesmo está em nível lógico alto. Ao lado do estado dos sensores existe vários atalhos para as funções disponibilizadas no menu arquivo, a única opção que não consta no menu é a realizada pelo terceiro ícone com um círculo vermelho, se esta opção está ativa, o círculo vermelho e alterado para um círculo verde, o software grava em arquivo todas as imagens recebidas do dispositivo. Na Figura 28 é exibido o menu arquivo desta janela.

Figura 28 – Menu Arquivo do dispositivo

As opções do menu são, Salvar a imagem que está sendo exibida, Enviar a imagem que em questão por *e-mail*, Limpar a imagem, Editar as propriedades do dispositivo e Fechar.

3.4 RESULTADOS E DISCUSSÃO

A utilização de uma interface de controle para a câmera provê a facilidade na sua utilização, mas por outro lado existe um custo, perda de velocidade, pois este módulo se comunica com o microcontrolador através de uma porta serial RS 232 em uma velocidade de 57600 *bps*, demorando assim em torno de 4 segundos para transferir cada imagem capturada para o microcontrolador. Essa é a grande limitação encontrada no trabalho realizado, foram realizados esforços, tentado por mais de 3 semanas realizar o gerenciamento do modulo CCD, não tendo progressos e por questão de tempo optou-se por utilizar a interface com a câmera, tornando assim a processo de captura da imagem lento.

Exceto pelo problema citado acima, todas as outras funcionalidades esperadas do protótipo foram alcançadas.

4 CONCLUSÕES

Todos os objetivos propostos foram alcançados, desde a captura da imagem até exibição da mesma pelo software disponibilizado. Passado por todo o processo de elaboração e comunicação do hardware através da rede 802.3, elaboração de um servidor no PC para conexão dos clientes (hardware) e software para exibição dos dados.

Para realização deste trabalho se fez necessário realizar estudos sobre microcontroladores, sensores, padrões de redes de computadores, protocolos de comunicação, utilização de *sockets*, linguagem C para microcontroladores e suas bibliotecas, assim como um maior aprimoramento na linguagem Java.

Verificou-se que o número de aplicações que utilizam rede TCP/IP ou que possivelmente podem vir a utilizar essa tecnologia, vem crescendo exponencialmente, com o desenvolvimento de hardwares mais aprimorados que integram essa tecnologia, cada vez mais será indispensável nos tornarmos adeptos da mesma.

O kit Rabbit 2000 TCP/IP, foi de fundamental importância para a realização do trabalho, pois agrega em um hardware diversos recursos necessários para realizar o mesmo, assim como o ambiente de programação Dynamic C, que proporcionou todo o suporte, através de suas bibliotecas, para melhor utilização dos recursos disponibilizados kit rabbit. E linguagem Java também se mostrou mais do que satisfatória tanto para desenvolvimento do servidor e da aplicação para visualização dos dados.

Finalizando, todos os objetivos foram alcançados e muito conhecimento foi adquirido, mas como dizia Albert Einstein, “a imaginação é mais importante que o conhecimento”.

4.1 E7XTENSÕES

Existem muitos pontos que podem ser agregados ou melhorados no protótipo, como sugestão pode-se citar:

a) aumentar a velocidade de captura das imagens, retirando a interface serial e acessando diretamente o sensor CCD pelo microcontrolador;
b) utilizar um CCD colorido e de maior resolução;
c) aumentando a velocidade da captura, ao invés de exibir e gravar imagens estáticas, fotos, exibir e gravar vídeos;
d) utilizar servo motores para direcionar a câmera;
e) mostrar mais informações dos sensores, não somente dois estados, por exemplo, ser captada a temperatura do local e demonstrar em escala de graus Celsius;
f) criptografar os dados que trafegam pela rede;
g) tornar a *thread* do servidor um software independente para que vários programas possam exibir os dados simultâneos, uma sugestão seria utilizar Java RMI.

REFERÊNCIAS BIBLIOGRÁFICAS

BOCKENSKI, Bárbara. **Implementando sistemas cliente / servidor de qualidade**. São Paulo: Makron Books, 1995.

CARVALHO, Tereza Cristina Melo de Brito. **Arquitetura de redes locais de computadores OSI e TCP/IP.** Rio de Janeiro: Makron Books, 1994.

CENSI, Angela. **Sistema para automação e controle residencial via e-mail.** 2001. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

CROSTA, Alvaro Penteado. **Processamento digital de imagens de sensoriamento remoto**. Campinas, SP: Unicamp, 1992.

FURLAN, José Davi. **Modelagem de objetos através da UML**: unified modeling language. São Paulo: Makron Books, 1998.

GASPARINI, Anteu Fabiano Lucio; BARRELLA, Francisco Eugênio. **TCP/IP solução para conectividade**. Tatuapé: Erica, 1993.

GONZALEZ, Rafael C; WOODS, Richard E. **Processamento de imagens digitais**. São Paulo: Edgard Blucher, 2000.

MEREGE NETO, Jorge de Assis. **Construção de um protótipo (hardware e software) para segurança predial através de monitoração via câmera digital e display gráfico**. 2004. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

PASSOS, Julia César Ferreira. **Sensores**: os olhos mecânicos da mecatronica industrial. São Paulo: Editora Saber, 2002. Disponível em <http://www.mecatronicafacil.com.br/oficina/old/diversos/sensores.htm>. Acesso em: 10 maio 2005.

PÉRICAS, Francisco Adell. **Redes de computadores**: conceitos e a arquitetura internet. Blumenau: Edifurb, 2003.

POVARESKIM: Povareskim Soft & System Ltda. São Paulo, 2005. Disponível em: <http://www.povareskim.com.br/chromephoto/dicas/dicas.html>. Acesso em: 27 abr. 2005.

RABBIT. **Rabbit 2000**: TCP/IP development kit. California: Rabbit Semiconductor, 2000.

RÉ, Pedro. **Pedro Ré's astronomical CCD imaging page**. [S.l.], 2005. Disponível em: <http://astrosurf.com/re/>. Acesso em: 25 abr. 2005

SANTOS, Ângelo Dias. **Protótipo de hardware e software para captura e visualização de imagens compartilhadas via interface digital serial diferencial balanceada**. 2002. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SEIXAS FILHO, Constantino. **Comunicação através de sockets sobre TCP/IP**. [Belo Horizonte], [2002?]. Disponível em: <http://www.cpdee.ufmg.br/~seixas/PaginaSDA/Download/SDADownload.htm>. Acesso em: 15 set. 2004.

SILVA, Fernando Luiz Melati. **Protótipo de hardware para controle de freqüência acadêmica**. 2002. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SILVA JUNIOR, Vidal Pereira da. **Microcontrolador PIC**: teoria e prática. São Paulo: Vidal Pereira da Silva Junior, 1998.

SOARES, Luiz Fernando et al. **Redes de computadores**: das LANs, MANs e WANs às redes ATM. Rio de Janeiro: Campus, 1995.

SPARX: **UML tools for software development and modelling**. [S.l.], 2005. Disponível em: <http://www.sparxsystems.com.au>. Acesso em: 25 maio. 2005.

TORRES, Gabriel. **Redes de computadores**: curso completo. Rio de Janeiro: Axcel Books, 2001.

VASQUES, Anderson Luiz. **Protótipo de um sistema coletor de dados microcontrolado conectado a uma rede TCP/IP**. 2003. Trabalho de Conclusão de Curso (Engenharia de Telecomunicações) - Centro de Ciências Tecnológicas, Universidade Regional de Blumenau, Blumenau.

VIANNA, Arlindo Cardarett. **Computação gráfica**. [Rio de Janeiro], 2002. Disponível em: <http://www.inf.ufes.br/~thomas/graphics/www/disc_cg.html>. Acesso em: 16 set. 2004.