CENTRO UNIVERSITÁRIO DE BRASÍLIA - UNICEUB
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ALLAN MOTA E SILVA

MONITORAMENTO DE AMBIENTE ATRAVÉS DE UMA UNIDADE MÓVEL CONTROLADA PELO COMPUTADOR.

Orientador: José Julimá Bezerra Júnior

BRASÍLIA
2010

ALLAN MOTA E SILVA

MONITORAMENTO DE AMBIENTE ATRAVÉS DE UMA UNIDADE MÓVEL CONTROLADA PELO COMPUTADOR.

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. José Julimá Bezerra Júnior

BRASÍLIA
2010

ALLAN MOTA E SILVA

MONITORAMENTO DE AMBIENTE ATRAVÉS DE UMA UNIDADE MÓVEL CONTROLADA PELO COMPUTADOR.

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. José Julimá Bezerra Júnior

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas -FATECS.

_____

Prof. Abiezer Amarilia Fernandez
Coordenador do Curso

Banca Examinadora:

_____

Prof. José Julimá Bezerra Júnior, Mestre em Engenharia Elétrica.
Orientador

_____

Prof. Flávio Antônio Klein, Mestre em Estatística e Métodos Quantitativos.
Centro Universitário de Brasília - UniCEUB

_____

Prof. Gil Renato Ribeiro Gonçalves, Doutor em Física.
Centro Universitário de Brasília - UniCEUB

DEDICATÓRIA

Dedico este trabalho aos meus pais, por todo investimento que fizeram por mim e por sempre acreditarem nas minhas decisões, a minha esposa e meus filhos pela paciência, dedicação e por sempre estarem ao meu lado.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela oportunidade de chegar até aqui.
Ao Professor Julimá pela paciência e orientação.
Aos amigos Cristiano Matos, Daniel Stolf e Rubens Bardelli por todo o apoio.
Aos colegas do curso de Engenharia da Computação pelo longo caminho que percorremos juntos, ajudando uns aos outros.

RESUMO

O presente trabalho apresenta uma proposta de sistema capaz de monitorar a distância, ambientes insalubres ou de pequenas dimensões, que tornam o acesso humano perigoso ou praticamente impossível. Este sistema foi criado, exatamente, para evitar a presença humana nestes locais. O projeto consiste na utilização de um veículo e uma microcâmera para registrar imagens do ambiente a ser monitorado. Além da câmera, o carro é dotado de um sensor que registra, a todo instante, a temperatura do ambiente, um sensor de luminosidade responsável pela ativação dos faróis e um localizador sonoro, avisando ao usuário, o local onde o veículo se encontra, evitando a perda do equipamento. Também foi desenvolvido uma aplicação *Pan & Tilt* para controlar a câmera, ampliando, assim, o campo visual a ser monitorado. Tanto o veículo como a microcâmera são controlados remotamente através de um computador. Os sinais de controle do veículo e da câmera, bem como as imagens captadas pela mesma e os valores das medições da temperatura do ambiente, são transmitidas pelo ar, via radiofrequência, dispensando assim, a utilização de fios e cabos.

ABSTRACT

This paper proposes a system capable of monitoring the distance, unhealthy or small environments, that make human access dangerous or impossible. This system was created precisely to avoid the human presence on these sites. The project is the use of a vehicle and a miniature camera to record images of the environment to be monitored. Besides the camera, the car is equipped with a sensor that registers at any instant the room temperature, a light sensor responsible for activation of lights and a sound locator, warning the user where the vehicle is, avoiding loss of equipment. Was also developed an Pan & Tilt application to control the camera, extending the visual field to be monitored. Both vehicle and a miniature camera is controlled remotely by a computer. The control signals of the vehicle and the camera, the images captured by the same and the values of measurements of temperature, are transmitted by air, via radio frequency, eliminating the use of wires and cables.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SÍMBOLOS / DEFINIÇÕES

USB – Universal Serial Bus;
LED - Light-emitting Diode;
PC- Personal Computer;
RS-232 - Recommended Standard 232;
RF - Radiofrequência;
MHz – Megahertz;
KHz - Quilohertz;
DC – Tensão Contínua;
AC – Tensão Alternada;
PWM - Pulse Width Modulation;
ms – milissegundos;
LDR - Light Dependent Resistor;
CDC - Communications Device Class;
ASK - Amplitude Shift Keying;
mV – MiliVolts ;
mA – Miliamperes;
A/D – Analógico / Digital;
AmpOP – Amplificador Operacional.
RCA – Radio Corporation of America

CAPÍTULO 1 – INTRODUÇÃO

Nos últimos anos, foi possível testemunhar o rápido avanço da tecnologia na área da comunicação e transmissão de dados. Atualmente, dois ou mais dispositivos podem transmitir e receber informações entre eles sem a necessidade de fios, utilizando o ar como meio de transmissão. Os maiores exemplos desta evolução são os telefones celulares e as redes de computadores *wireless*. Ambos, utilizam a transmissão via radiofrequência para que se comuniquem entre os seus pares.

Tomando como base essa tecnologia, este projeto visa o desenvolvimento de um equipamento portátil, capaz de transmitir e receber dados através do ar, sem a utilização de fios ou cabos, tornando assim, o sistema com maior mobilidade.

1.1 – MOTIVAÇÃO

A motivação para a realização deste projeto surgiu a partir de:

a) O desenvolvimento de um sistema valendo-se da eletrônica, desde a montagem e desenho dos circuitos até a fabricação das placas de circuito impresso.

b) Dotar este sistema de uma porta USB por onde é realizada a conexão com o computador. A porta USB, comparada com as portas paralela e serial, possui tecnologia mais recente e, atualmente, é o principal meio de conexão existente entre os mais diversos dispositivos e o computador.

c) A utilização da transmissão e recepção de dados através da comunicação sem fio, tecnologia esta que está em bastante evidência e ascensão nos dias de hoje.

d) O emprego dos conhecimentos adquiridos ao longo do curso de Engenharia de Computação, em diversas matérias como: Circuitos Eletrônicos; Linguagens e Técnicas de Programação; Lógica Digital; Microcontroladores e Microprocessadores; entre outras.

1.2 – OBJETIVO

O objetivo deste projeto é elaborar um sistema de medidas reduzidas controlado a

distância capaz de se movimentar e monitorar ambientes insalubres, ou seja, locais que possam acarretar algum perigo à saúde humana. Como exemplo, pode-se citar ambientes em que há presença de ruídos ou temperaturas elevadas, gases tóxicos, substâncias químicas ou radiológicas nocivas ao homem ou outros fatores que possam oferecer riscos a integridade e a saúde do ser humano. A sua utilização poderia se estender a inspeções de tubulações ou dutos visando a detecção de vazamentos e medições de concentração de gases e, até mesmo, a detecção de bombas em zonas de guerra. O projeto também pode assistir pessoas com dificuldades de locomoção, sendo utilizado para monitorar uma residência, por exemplo.

1.3 – METODOLOGIA

O projeto teve início a partir de pesquisas teóricas baseadas em livros, artigos e sites da internet que pudessem auxiliar na elaboração e desenvolvimento do mesmo. Estes estudos, inclusive, foram utilizados no desenvolvimento do capítulo *'Referencial Teórico e Tecnológico'* da Monografia.

Uma nova pesquisa foi feita, desta vez, em busca de componentes e dispositivos que seriam necessários para a evolução do trabalho. A pesquisa incluiu, também, os preços e a disponibilidade dos mesmos no mercado brasileiro para identificar a viabilidade do projeto.

Ao término das pesquisas, iniciou-se a montagem dos circuitos em protoboard para testes e, após obter a certificação do funcionamento esperado, os esquemas elétricos foram desenhados no computador, através do *software* Cadsoft Eagle, responsável por gerar, a partir destes esquemas, as trilhas a serem desenhadas nas placas de circuito impresso. Com as trilhas impressas, as placas de circuito impresso foram construídas de forma artesanal.

Em sequência, foi desenvolvido um programa para que o computador fosse capaz de enviar os sinais de comando aos circuitos de controle e imprimir os registros de temperatura. Com as placas de circuito impresso e o programa finalizados, o projeto foi montado e foram feitos alguns testes utilizando como parâmetro, diferentes distâncias para verificar o recebimento de dados entre o computador e o veículo.

Por fim, todo o desenvolvimento deste projeto, os testes realizados e a estimativa de custo foram relatados no capítulo 3.

1.4 – VISÃO GERAL DO PROJETO

O diagrama geral do projeto pode ser visto na figura 1.1.

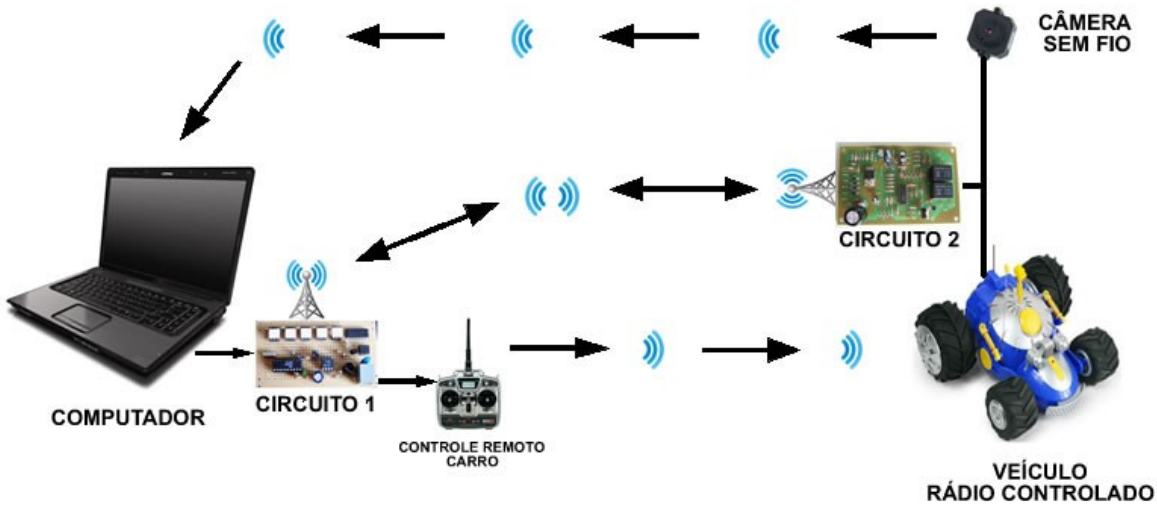


Figura 1.1 – Diagrama Geral do Projeto

O computador é a interface utilizada pelo usuário para se comunicar com todo o sistema. Com ele, o usuário controla o veículo e a microcâmera a partir do teclado e, por intermédio do monitor, consegue visualizar as imagens capturadas pela câmera e as medidas de temperaturas registradas.

O Circuito 1 é uma representação de três circuitos:

a) O circuito responsável por fazer a conexão diretamente ao computador pela porta USB. Nele, encontra-se o receptor que intercepta os registros de temperatura enviados pelo respectivo transmissor.

b) O circuito transmissor de sinais de comandos para controle das câmeras.

c) O circuito transmissor de sinais de comandos para o 'Controle Remoto do Carro', fazendo com que o usuário controle o carro diretamente pelo teclado do computador.

Apesar do desenvolvimento do circuito citado no item 'c', é necessário a aplicação do 'Controle Remoto do Carro' pois, nele está instalado o transmissor responsável por enviar os sinais para o receptor do veículo. O 'Veículo Rádio Controlado' tem a função de transportar a parte do sistema responsável pelo monitoramento do ambiente. Foram instalados ao veículo, a microcâmera e o Circuito 2. Este último, descreve os três circuitos presentes no veículo:

a) O circuito leitor e transmissor das medidas de temperatura registradas;

b) O circuito receptor de sinais de comandos responsáveis pela movimentação da câmera e acionador do *buzzer,* após o circuito perder contato com o transmissor.

c) O circuito sensor de luz, responsável pelo acendimento automático dos faróis quando houver pouca ou nenhuma luz no ambiente.

O trabalho está organizado em quatro capítulos, incluindo este. O segundo capítulo engloba o referencial teórico e tecnológico que trata de *hardware*, linguagens de programação, das tecnologias e alguns componentes utilizados no projeto. O terceiro capítulo apresenta o desenvolvimento de todos os circuitos e do aplicativo para controle e impressão dos dados recebidos, bem como as explicações de como eles funcionam. Neste capítulo também são apresentados os resultados dos testes executados para comprovar o funcionamento do sistema a diferentes distâncias. O último capítulo apresenta a conclusão do projeto e sugestões para futuras evoluções do mesmo. Ao final do trabalho situam-se os apêndices, onde são exibidos os *firmwares* e código fonte do aplicativo desenvolvidos e os desenhos esquemáticos dos circuitos representando as placas construídas.

CAPÍTULO 2 – REFERENCIAL TEÓRICO E TECNOLÓGICO

2.1 – *HARDWARE* E INTERFACES PARA COMUNICAÇÃO COM O CIRCUITO

2.1.1 – *Personal Computer*

O computador pessoal ou PC (*Personal Computer*) é um computador de pequeno porte, que se destina ao uso de uma pessoa ou um grupo restrito de pessoas. [SAIBAMAIS.ORG, 2009].

É comum a referência *desktop* ou computador doméstico a essa classe de computador. Atualmente, o computador pessoal está presente nos mais diversos ambientes, sejam empresas, residências, faculdades, etc. Por ser relativamente barato, é vantajoso para as companhias disporem de um equipamento onde podem tornar sua produção mais eficiente, bem como manter e organizar arquivos. No ambiente doméstico, o PC é cada vez mais utilizado para acesso à *Internet* e na substituição da antiga máquina de datilografia. No ambiente escolar, é comprovado que a presença do computador potencializa o ensino dando aos alunos maior capacidade de pesquisa, acesso ao conhecimento e desenvolvimento do raciocínio lógico.

No projeto, foi utilizado um computador para que o usuário pudesse visualizar as imagens captadas pela câmera e os valores das medidas de temperaturas enviadas pelo circuito apresentado no item 3.3. O computador também é responsável por controlar o veículo e a câmera por intermédio do seu teclado. Como o projeto se conecta ao computador através da porta USB, foi possível substituir o *desktop* por um *notebook*, já que atualmente, as portas seriais e paralelas estão em desuso e os computadores portáteis já não dispõem dessas portas.

2.1.2 – Porta USB

O *Universal Serial Bus* (USB) é um tipo de conexão *Plug and Play* que tem como principal característica, permitir a conexão de dispositivos ou periféricos sem a necessidade

de desligar ou reiniciar o computador. Ele surgiu em 1995, através de um consórcio de empresas como Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC e Philips, com o intuito de substituir as portas seriais e paralelas utilizadas até então. [MORAZ, 2006]

Aproveitando o conceito de *Plug and Play*, o USB conseguiu minimizar o esforço de concepção de periféricos, no que diz respeito ao suporte por parte dos sistemas operacionais e hardware. Com isso, surgiu um padrão que permitiu ao Sistema Operacional e à placa-mãe do computador diferenciar, transparentemente:

- A classe do equipamento (dispositivo de armazenamento, impressora, *scanner*, placa de som, placa de rede, etc);
- As necessidades de alimentação elétrica do dispositivo, caso este não possua alimentação própria;
- As necessidades de largura de banda (para um dispositivo de vídeo, serão muito superiores as de um teclado, por exemplo);
- As necessidades de latência máxima;
- Eventuais modos de operação internos ao dispositivo (por exemplo, máquina digital pode operar, geralmente, como uma *webcam* ou como um dispositivo de armazenamento - para transferir as imagens). [MORAZ, 2006]

Uma das primeiras versões foi a USB 1.0 com velocidade de 1,5 Mbs (*Low-Speed*). Logo em seguida, foi concebida a 1.1 com velocidades entre 1,5 Mbps a 12 Mbps (*Full-Speed*). No final de 2000, foi lançada a versão mais atual, a USB 2.0 (*High-Speed*), que além de ser compatível com as versões anteriores, possui aperfeiçoamentos que vão desde a topologia à velocidade de trafego de dados, chegando ao extremo de 480 Mbps, equivalente a cerca de 60 MBps (60 milhões de *Bytes* por segundo). [MORAZ, 2006]

O *hardware* do computador que contém o controlador *host* e o *hub* principal, tem uma interface voltada para o programador que é chamado de dispositivo de Controlador *Host* (*Host Controller Device* - HCD) e é definido pelo implementador do hardware.

O USB foi projetado de modo que possam ser ligados vários periféricos pelo mesmo canal (i.e., porta USB). Assim, mediante uma topologia em árvore, é possível ligar até 127 dispositivos a uma única porta do computador, utilizando *hubs* especialmente criados para essa função. [MORAZ, 2006]

O *host* USB se comunica com os dispositivos através do seu controlador (*chipset* e outros componentes). O controlador *host* pode ser encontrado na própria estrutura base da placa-mãe do computador, ou pode ser adicionada em um dos *slots* de barramento PCI. Na maioria das placas controladoras USB - PCI, além das portas externas, há uma interna, que

permite instalar periféricos USB dentro do gabinete do computador, se isso for preciso.

É responsabilidade do *Host*:

- Detectar a inclusão e remoção de dispositivos;
- Gerenciar o fluxo de controle de dados entre os dispositivos conectados;
- Fornecer alimentação (tensão e corrente) aos dispositivos conectados;
- Monitorar os sinais do *bus* USB.

O cabo USB é composto por quatro fios e uma malha para eliminação de ruídos simples. Destes fios, dois são responsáveis por transportar energia para alimentar dispositivos. Essa energia é fornecida pela controladora e gerenciada pelo *Driver* do Controlador *Host*.

O Cabo USB utiliza cores padrão para os fios, sendo o fio de cor vermelha chamado de *Vbus* (5v), ou seja, é o fio positivo de fornecimento de energia. O fio de cor preta é o terra (0v).

O *bus* USB pode fornecer no máximo cinco Volts de tensão e quinhentos miliamperes (500 mA) de corrente elétrica, isso para cada porta do *Root Hub* do *host*. A quantidade de corrente que o dispositivo necessita para o seu funcionamento, pode ser configurada via software.

Os dois fios restantes, 'D+' (dado+) e 'D-' (dado-) são usados pelo sistema USB para transferência de dados entre o *Host*, *hub* e dispositivos. Todos os sinais de dados trafegam através destes dois fios usando a codificação NRZI (*No Return to Zero Inverted*). Ou seja, o bit 1 é codificado através de uma transição ocorrendo da maior voltagem para a menor, ou também o inverso, da menor para a maior. Já o bit 0 é codificado sem haver transição. Durante o intervalo de um bit, a voltagem é constante. [AXELSON, 2009]

A figura 2.1 apresenta a disposição da pinagem em dois tipos de portas USBs e a tabela 2.1, as funções dos pinos.

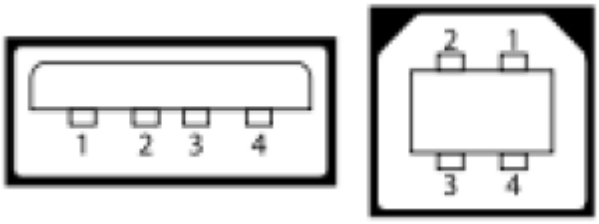


Figura 2.1 – Pinagem das Portas USB
Fonte: http://commons.wikimedia.org

Tabela 2.1 – Pinos da USB e suas funções

Pino	Função
1	VBUS
2	DATA -
3	DATA +
4	TERRA

Fonte: AXELSON, 2009

O protocolo é uma das partes mais importantes do funcionamento do sistema USB, pois, é através dele que um ou mais dispositivos conversam e trocam informações com o *Host*. Nele, a pilha do protocolo está disponível em forma de *drivers* de arquivos *.sys*, *dll*, *drv*, *exe*, etc. Já no dispositivo, o protocolo pode ser encontrado dentro de um microcontrolador específico, como por exemplo um PIC que disponha de uma certa quantidade de memória. Neste caso, o protocolo é escrito com base na norma USB, em uma linguagem específica como C ou *Assembly*; depois é compilado e gravado na memória do microcontrolador através de um gravador de PIC. Este protocolo faz com que a porta USB seja diferente da Porta Serial ou Paralela, onde em ambas, é possível se comunicar com um dispositivo através de sinais elétricos dos pinos e um programa básico. No USB, isso só é possível se o dispositivo carregar o protocolo USB num *chipset* ou mesmo dentro de um microcontrolador. Assim, conclui-se que o combustível do sistema USB é o seu protocolo e, sem ele, não há troca de informação entre os dispositivos.

O protocolo USB tem vários recursos como CRC (*Cyclic Redundancy Check*), detecção e correção de erros, detecção de conexão e remoção de dispositivos, controle de fluxo de dados assegurando transmissões isossíncronas (trafego contínuo de dados), disponibilidade assegurada de largura de banda, entre outros. [AXELSON, 2009]

Todos os dispositivos USB têm uma hierarquia de descritores que informam ao *Host* o que o dispositivo é, ou seja, sua "personalidade", suas características de funcionamento, como o número de série do produto, identificação do fabricante, tipo do dispositivo (impressora, *scanner*, *modem*, *mouse*, etc.), número de configurações, número de *Endpoint*, tipo de transferência, tipo de interface, etc.

Os dispositivos que podem se conectar à porta USB, precisam de um *driver* específico para serem usados ou podem pertencer a uma classe de dispositivo. Estas classes definem um comportamento já esperado de determinados dispositivos, sendo assim, pode ser utilizado um mesmo *driver* para qualquer equipamento que pertença a mesma classe. Um sistema operacional geralmente possui *drivers* genéricos para todas as classes. As classes

estão disponíveis na tabela 2.2.

Tabela 2.2 – Classes de Dispositivos USB

Classes	Uso	Descrição	Exemplos
00h	Dispositivo	Não especificado[class 0]	
01h	Interface	Audio	Microfone, caixa de som
02h	Ambos	Comunicações e Controle CDC (Communications Device Class)	Adaptador Ethernet, modem.
03h	Interface	Dispositivo de Interface Humana (HID)	Mouse, teclado
05h	Interface	Dispositivo de Interface Física (PID)	Joystick
06h	Interface	Imagem	Câmeras Digitais (Muitas câmeras utilizam a classe 08h para acesso direto aos arquivos).
07h	Interface	Impressoras	Impressora jato de tinta
08h	Interface	Armazenamento (Mass Storage)	USB Flash Drive, Mp3 Player, Leitor de cartão de memória
09h	Dispositivo	Hub USB	Hubs hi-speed
0Ah	Interface	CDC-Data	(Esta classe é utilizada juntamente com a classe 02h – Comunicações e Controle CDC.)
0Bh	Interface	Smart Card	Leitores de USB smart cards
0Dh	Interface	Segurança de Conteúdo	-
0Eh	Interface	Vídeo	Webcam
0Fh	Interface	Cuidados Pessoais	-
DCh	Ambos	Dispositivo de Diagnóstico	Dispositivo de Teste USB
E0h	Interface	Controlador Wireless	Adaptador Bluetooth
EFh	Ambos	Misturado	Dispositivos ActiveSync
FEh	Interface	Aplicações Específicas	Dispositivos Infravermelhos
FFh	Ambos	Fornecedor Específico	(Esta classe indica que o dispositivo requer drivers específicos do fornecedor.)

Fonte: AXELSON, 2009.

No sistema USB, existe o processo de enumeração responsável pela conexão, detecção, leitura dos descritivos dos dispositivos e desconexão. É uma atividade ininterrupta. Isso tudo é gerenciado em tempo real pelo controlador *Host* e o software do sistema. Do ponto de vista do usuário do computador, o processo de enumeração é transparente, desde que se tenha instalado anteriormente os *drivers* do fabricante do dispositivo no sistema operacional. Após a instalação, o carregamento dos *drivers* quando um dispositivo é conectado, é automático.

Para a conexão do projeto ao computador, foram necessárias duas portas USB do

computador. Na primeira porta, é conectado o circuito apresentado no tópico 3.2, responsável por receber as informações de controle do carro e da câmera enviados pelo próprio computador e, também, transmitir ao computador os valores das medidas de temperatura registradas pelo circuito transmissor de temperatura presente no carro e que é explicado no item 3.3. Na segunda porta, é conectada a placa de captura *EASYCAP*, que tem a função de enviar ao computador as imagens transmitidas pela microcâmera.

2.1.3 – Porta Serial

A porta serial é utilizada para converter informações em paralelo para informações seriais. Isso significa que a interface serial pega os bytes recebidos em paralelo e converte-os para bits individuais e são enviados separadamente (um a um). A porta serial também executa a operação inversa, ou seja, pega os bits separados, converte-os para bytes e envia-os para o computador em paralelo. [AXELSON, 2007]

A interface também é conhecida como RS-232. O padrão RS-232 foi originalmente definido para uma comunicação por meio de 25 fios diferentes. Porém, a IBM, ao utilizar o padrão para o seu projeto do IBM-PC, definiu que apenas 9 pinos seriam necessários. Entretanto, o conector DB25 foi mantido nos computadores, devido a este ser um padrão da época.

Por muitos anos, o padrão para comunicação serial em quase todos os computadores era algum tipo de porta RS-232. Ele continuou sendo utilizado em grande escala até o fim dos anos 90. Durante este tempo, esta foi a maneira padrão para a conexão de *modems*.

A importância de portas seriais começou a decrescer gradualmente quando redes de alta velocidade tornaram-se disponíveis para a comunicação PC / PC. Hoje, é comum utilizar conexões *Ethernet Base* 10, 100 ou 1000. Num futuro próximo, velocidades maiores serão comuns.

Atualmente, o protocolo de comunicação RS-232 já foi praticamente suprimido pela USB para comunicação local, por ser mais rápido, por possuir conectores mais simples e ter um melhor suporte por *software*.

Por este motivo, alguns computadores produzidos atualmente já não contam mais com a porta serial. Entretanto, essa interface ainda é utilizada em periféricos para pontos de venda (caixas registradoras, leitores de códigos de barra ou fita magnética) e para a área industrial (dispositivos de controle remoto).

A figura 2.2 mostra o conector padrão DB9 de 9 pinos e a tabela 2.3, os pinos com suas respectivas nomenclaturas e funções:

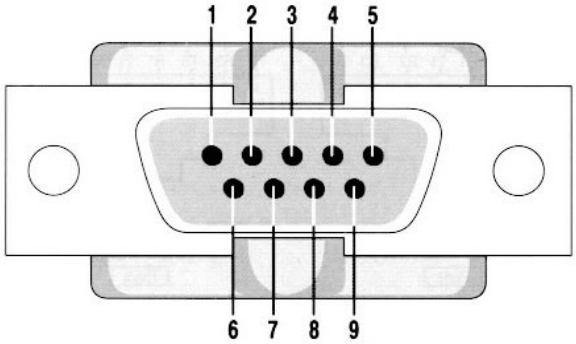


Figura 2.2 – Pinagem do Conector DB9
Fonte: http://www.western-data.com

Tabela 2.3 – Pinos e Funções da Porta Serial

Pino	Nomenclatura	Função
1	DC	Detecção de portadora
2	RXD	Recepção de dados
3	TXD	Transmissão de dados
4	DTR	Terminal de dados pronto
5	GND	Terra do Sinal
6	DSR	Data set pront
7	RTS	Solicitação de Envio
8	CTS	Pronto Para Enviar
9	RI	Indicador de Chamada

Fonte: AXELSON, 2007

Existem dois modos de comunicação serial: Síncrona e Assíncrona.

No modo de comunicação serial síncrona, o transmissor e o receptor devem ser sincronizados para a troca de comunicação de dados. Geralmente, uma palavra de sincronismo é utilizada para que ambos ajustem o relógio interno. Após a sincronização, os bits são enviados sequencialmente, até uma quantidade pré-combinada entre os dispositivos. [AXELSON, 2007]

A comunicação serial assíncrona é a forma mais usual de transmissão de dados. Não existe a necessidade de sincronização entre os dispositivos, uma vez que os caracteres são transmitidos individualmente e não em blocos, como na comunicação síncrona. A transmissão de cada caractere é precedida de um bit de *start* e terminada por 1, 1,5 ou 2 bit(s) de parada. [AXELSON, 2007]

A interface serial já foi utilizada por vários dispositivos. A lista abaixo inclui alguns exemplos:

- Impressoras;
- *Modems* de conexão discada;
- *Mouses*;
- Receptores GPS
- *Displays* de LCD
- Leitor de código de barras
- Atualizar *firmware* de vários equipamentos domésticos como o *DVD-Player*.

2.2 – LINGUAGENS DE PROGRAMAÇÃO UTILIZADAS NO PROJETO

2.2.1 – Linguagem C

A linguagem de programação C foi desenvolvida por Dennis Ritchie durante o começo dos anos 70 para ser utilizada na implementação de sistemas operacionais e outras tarefas de programação de baixo nível. Ela foi derivada de outras duas linguagens anteriores: a Linguagem BCPL e a Linguagem B. A Linguagem BCPL foi desenvolvida por Martin Richards. Esta linguagem influenciou a linguagem inventada por Ken Thompson, chamada B. Logo em seguida, Dennis Ritchie desenvolveu a Linguagem C utilizando o sistema operacional *UNIX*. [KERNIGHAN e RITCHIE, 1998]

O desenvolvimento inicial de C ocorreu entre 1969 e 1973. Em 1973, ela tornou-se poderosa o suficiente para reimplementar o *kernel* do sistema operacional *Unix*.

A linguagem C buscou manter o "contato com o computador real" e, ainda sim, dar ao programador novas condições para o desenvolvimento de programas em diversas áreas como por exemplo, comercial, científica e de engenharia. Ela tornou-se imensamente popular depois de 1980 e, por um tempo, foi a linguagem predominante em programação de sistemas e de aplicações de microcomputadores, devido a sua simplicidade e flexibilidade. O próprio sistema operacional *Windows* foi desenvolvido a partir dela.

A linguagem C possui as seguintes características:

- Portabilidade entre máquinas e sistemas operacionais, ou seja, um código escrito poderá ser executado em diferentes máquinas, independentemente da

sua configuração física (*hardware*) e do sistema operacional residente.

• É estruturada, sendo utilizados diversos tipos de laços e desvios, tais como: *while, do-while, for; if-else, switch*, que permitem ao programador exercer um controle lógico mais eficaz sobre os códigos fontes dos seus programas.

• Possui subrotinas com variáveis locais, isto é, funções cujas variáveis são visíveis apenas dentro desta função e somente no momento em que estas funções estejam sendo usadas. Assim, as variáveis com mesmo nome, que pertencem a funções distintas, são protegidas dos efeitos colaterais, isto é, uma modificação em nível funcional não acarreta mudança na variável em nível global.

• Código compacto e rápido, quando comparado ao código de outras linguagens de complexidade análoga.

• É *Case-Sensitive*, ou seja, letras maiúsculas e minúsculas são tratadas como caracteres distintos.

A principal documentação deste padrão encontra-se na publicação "*The C Programming Language*", de Brian Kernighan e Dennis Ritchie, conhecida por "bíblia da linguagem C".

A Linguagem C é utilizada neste projeto na programação dos três microcontroladores existentes no projeto. Para o desenvolvimento dessa tarefa, foi utilizado a ferramenta C Compiler, versão 4.105 da empresa CCS. Ela provê completa estrutura lógica para a programação e é composta por operadores que foram organizados em bibliotecas que são específicas para os microcontroladores PICs da Microchip. A ferramenta também fornece acesso aos recursos de *hardware* através da linguagem C, evitando assim, a necessidade de se utilizar *assembly*. Entretanto, ela permite a inclusão de códigos nesta linguagem em sua implementação. Deste modo, pode-se desfrutar de todas as vantagens da implementação em alto nível e, também, fazer uso da implementação em baixo nível, explorando rotinas que não podem ser otimizadas através da programação em alto nível.

Esta ferramenta possui, ainda, diversos exemplos, funções e *drivers* em forma de bibliotecas sendo, inclusive, necessário a instalação de um destes *drivers* no computador, para que o mesmo reconheça o dispositivo conectado que, neste caso, é o circuito detalhado no item 3.2.

2.2.2 – Linguagem C++

A Linguagem C havia alcançando seu limite, pois, devido a problemas de gerenciamento do código, os programas escritos na linguagem atingiam somente um certo

tamanho, entre 25.000 e 100.000 linhas. Para resolver este problema, na década de 80, Bjarne Stroustrup, nos laboratórios AT&T Bells, acrescentou a linguagem C, o conceito de classes e de verificação de parâmetros de funções além de algumas outras facilidades e a chamou inicialmente de “C com classes”. Em 1983/84 o "C com classes" foi estendido e reimplementado, sendo adicionada a orientação a objetos, resultando na linguagem conhecida como C++. Este termo, aliás, faz referência ao operador de incremento ++, significando um acréscimo a evolução da linguagem C. Ela é considerada uma linguagem de nível médio, pois combina características de linguagens tanto de alto, quanto de baixo nível. As maiores extensões da linguagem C++ foram as funções virtuais e a sobrecarga de operadores. Após mais alguns refinamentos, a primeira versão da linguagem C++ tornou-se disponível ao público em 1985 e foi documentada no livro "*The C++ Programming Language*". *[STROUSTRUP, 1993]*

Os primeiros compiladores C++ utilizavam um pré-processador, uma vez que, não geravam diretamente código executável, mas convertiam o código C++ em código C para uma posterior compilação através de um compilador C comum. Posteriormente, a linguagem passou a exigir um compilador próprio, escrito pelo próprio Stroustrup.

Em 1989, a segunda versão foi lançada, contendo novas características como herança múltipla, classes abstratas, métodos estáticos, métodos constantes e membros protegidos, incrementando o suporte a orientação a objeto. [STROUSTRUP, 1993]

Assim como a linguagem, sua biblioteca padrão também sofreu melhorias ao longo do tempo. Sua primeira adição foi a biblioteca de E/S, e posteriormente a *Standard Template Library* (STL); ambas tornaram-se algumas das principais funcionalidades que distanciaram a linguagem em relação a C.

Pode-se dizer que C++ foi a única linguagem entre tantas outras que obteve sucesso como uma sucessora à linguagem C, inclusive servindo de inspiração para outras linguagens como *Java* e C# (*C Sharp*).

As principais vantagens e desvantagens da linguagem C++ são listadas a seguir:

- Possibilidade em programação de alto e baixo nível.
- Alta flexibilidade, portabilidade e consistência.
- Compatilidade com C, resultando em vasta base de códigos.
- A compatilidade com o C herdou os problemas de entendimento de sintaxe do mesmo.
- Os compiladores atuais nem sempre produzem o código mais otimizado, tanto em velocidade quando tamanho do código.

- Grande período para o aprendizado.
- A biblioteca padrão ainda não cobre áreas importantes da programação, como *threads*, conexões TCP/IP e manipulação de sistemas de arquivos, o que implica na necessidade de criação de bibliotecas próprias para tal, que pecam em portabilidade.
- Devido à grande flexibilidade no desenvolvimento, é recomendado o uso de padrões de programação mais amplamente que em outras linguagens.

A linguagem C++ foi utilizada no projeto para o desenvolvimento de uma ferramenta dotada de interface gráfica com capacidade para enviar sinais de controle a partir de comandos executados pelo usuário. A ferramenta também tem a função de imprimir os valores das temperaturas registradas bem como, a data e hora do recebimento da informação. Os detalhes sobre o aplicativo estão disponíveis no item 3.6.

2.3 – MEIO DE TRANSMISSÃO DAS INFORMAÇÕES

2.3.1 – Frequências de rádio

As ondas de rádio são ondas eletromagnéticas propagadas através de uma antena. Elas possuem diferentes frequências e, ao sintonizar um receptor de rádio em uma frequência específica, é possível captar um sinal. [BRAIN, 2000]

A comunicação via rádio é simples de operar, oferecendo total mobilidade ao usuário, permitindo a conexão de equipamentos distantes entre si, possibilitando a implantação de uma rede de comunicação de dados completa. Outra vantagem é que as ondas de rádio são mais simples de serem geradas, podendo ir a longas distâncias e atravessar prédios e casas, sendo, por essa razão, largamente utilizada em telecomunicações.

As ondas de rádio são dependentes de sua frequência, sendo que, as ondas de baixas frequências atravessam facilmente obstáculos, mas sua força cai rapidamente conforme aumenta a distância do emissor. As ondas de alta frequência viajam em linha reta e ricocheteiam nos obstáculos, porém, são absorvidas pela chuva, atingem a ionosfera e são refletidas de volta à terra, podendo sobre certas condições meteorológicas serem refletidas diversas vezes percorrendo longas distâncias. [TITTEL, 2002]

A figura 2.3 ilustra os diversos tipos de radiofrequência utilizados.

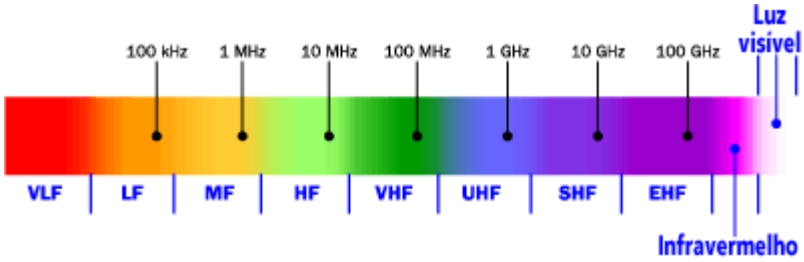


Figura 2.3 – Tipos de frequência
Fonte: BRAIN, 2000

A habilidade de viajar, a longas distâncias, causa um problema: a interferência entre usuários utilizando a mesma frequência. Devido a este fator, todos os governos regulam rigorosamente a licença para uso de rádio transmissores. Nos Estados Unidos, a FCC (*Federal Communications Commission*) é o órgão regulador da radiodifusão. O órgão equivalente no Brasil é a ANATEL. [BRAIN, 2000]

Um aspecto interessante das ondas de rádio é que cada tecnologia *wireless* tem a sua pequena faixa de banda disponível. Existem centenas delas. A tabela 2.4 exibe alguns exemplos:

Tabela 2.4 – Frequências mais comuns

Utilização	Frequência
Rádio AM (Amplitude Modulada)	535 KHz a 1,7 MHz
Rádio de ondas curtas	5,9 MHz a 26,1 MHz
Canais de TV - Canais 2 a 6 - Canais 7 ao 13	 54 a 88 MHz 174 a 220 MHz
Rádio FM (Frequência Modulada)	88 MHz a 108 MHz
Controle remoto de garagens, sistemas de alarmes, etc	em torno de 40 MHz
Telefones sem fios	40 a 50 MHz
Babá eletrônica	49 MHz
Estação espacial MIR	145 MHz e 437 MHz
Telefones celulares	824 a 849 MHz
Radar de controle de tráfego aéreo	960 a 1215 MHz
Sistema de posicionamento global (GPS - Global Positioning System)	1227 e 1575 MHz
Comunicações de rádio no espaço	2290 MHz a 2300 MHz

Fonte: BRAIN, 2000.

2.3.2 – Controles remotos de rádio frequência

O surgimento dos primeiros controles remotos aconteceu durante a Primeira Guerra Mundial. Eles eram equipamentos de rádio frequência e tinham a função de pilotarem navios alemães para colidirem contra barcos aliados. Durante a Segunda Guerra Mundial, os controles remotos foram utilizados para detonar bombas pela primeira vez. Ao fim da grande guerra, os cientistas da época obtiveram uma tecnologia fantástica, porém, não havia campo algum para aplicá-la. Atualmente, mais de sessenta anos depois, os controles remotos que utilizam radiofrequência (RF) são muito comuns, por isso, uma grande parcela da população possui pelo menos um ou mais controles que utilizam a radiofrequência para transmitir um sinal. [LAYTON, 2005]

Eles são empregados em portas de garagens, em alarmes de carros, brinquedos, em televisores, aparelhos de som, entre outros. Uma aplicação que está sendo bastante difundida através dos aparelhos celulares e também utiliza a radiofrequência é o *Bluetooth*.

O controle remoto RF transmite ondas de rádio que correspondem a um comando binário referente ao botão ao qual se está pressionando. A transmissão tem como destino, os receptores de rádio de aparelhos controlados. O problema com este tipo de controle é a quantidade de sinais de rádio puros, invisíveis no ar, a qualquer hora. Os telefones celulares, *walkie-talkies*, aparelhos *WiFi* e telefones sem fio estão todos transmitindo sinais de rádio em frequências variáveis. Os controles remotos RF lidam com o problema da interferência, transmitindo uma frequência de rádio específica e embutindo códigos de endereços digitais nos sinais de rádio (sinal codificado). Isto permite que o receptor de rádio no aparelho de destino saiba quando responder ao sinal e quando ignorá-lo. [LAYTON, 2005]

A maior vantagem dos controles remotos com frequência de rádio é o seu alcance e, o fato de que os sinais de rádio podem atravessar paredes.

2.3.3 – O Carro de Controle Remoto

O carro utilizado no projeto é controlado por rádio e possui quatro componentes principais:

- transmissor – É o controle remoto. Ele é responsável por enviar ondas de rádio ao receptor;

- receptor – É formado por uma antena e uma placa de circuito no interior do carro. Ele recebe os sinais provenientes do transmissor e ativam os motores elétricos no interior do carro de acordo com os comandos recebidos;
- motores elétricos – Os motores têm a função de girar e/ou virar as rodas;
- fonte de energia – Tanto o transmissor quanto o carro possuem fonte de energia. O carro possui uma bateria recarregável de 9,6 Volts, enquanto o controle remoto necessita de duas pilhas pequenas de 1,5 Volts cada uma.

Geralmente, os brinquedos controlados por rádio possuem um pequeno dispositivo portátil que inclui alguns tipos de controles e o rádio transmissor. O transmissor envia um sinal em determinada frequência para o receptor no brinquedo. O transmissor possui uma fonte de alimentação, que fornece a energia para os controles e a transmissão do sinal. A maioria dos carros rádio controlados opera em 27 ou 49 Megahertz. Este par de frequências foi reservado pela FCC. O termo Megahertz significa "milhões de ciclos por segundo". O carro utilizado no projeto opera em 27 MHz. A figura 2.4 apresenta o carro que foi utilizado neste trabalho.

Figura 2.4 – Carro Utilizado no Projeto

Dentro do carro existem dois motores elétricos, uma antena, e uma placa de circuito. Um motor vira as rodas dianteiras para a direita ou esquerda, enquanto o outro motor gira as rodas traseiras para frente ou para trás. Os motores elétricos recebem energia da bateria.

O movimento do carro ocorre da seguinte maneira: Quando se pressiona o *joystick do* controle para uma direção, o controle faz com que um par de contatos elétricos se encostem, como se fosse uma chave, fechando um circuito conectado a um pino específico

de um circuito integrado (C.I.). O circuito completo faz com que o transmissor envie uma sequência de pulsos elétricos.

O transmissor envia rajadas de ondas de rádio que oscilam numa frequência de 27.000.000 de ciclos por segundo (27 MHz). O carro monitora constantemente a frequência designada à procura de um sinal. Quando o receptor recebe as rajadas de rádio do transmissor, ele envia o sinal para um filtro, que bloqueia quaisquer outros sinais captados pela antena que estejam fora da frequência de 27 MHz. O sinal remanescente é convertido novamente em uma sequência de pulsos elétricos. Ela é enviada para o C.I. no circuito do carro, que decodifica o sinal e faz funcionar o motor elétrico apropriado.

O controle remoto do veículo possui uma antena transmissora, uma placa de circuito e dois *joysticks*, um responsável por movimentar o carro para frente e trás e, o outro, para girar as rodas dianteiras para a esquerda ou direita. A figura 2.5, apresenta foto do controle responsável por controlar o carro utilizado no projeto.

Figura 2.5 – Controle Remoto do Carro

2.3.4 – Câmera *Wireless*

Para captar as imagens no ambiente a ser monitorado, foi utilizada uma câmera de pequenas dimensões instalada no veículo. Ela é alimentada através de uma bateria de 9 Volts. Na figura 2.6, é exibida a pequena câmera juntamente com a sua fonte de alimentação. Através dela, pode-se ter a percepção do tamanho real da câmera.

Figura 2.6 – Microcâmera sem fio

A transmissão do vídeo pela câmera também é feita através da radiofrequência, utilizando-se a faixa de 1,2 GHz, de acordo com as informações fornecidas em sua caixa. Trata-se de um tipo de sinal elétrico originário do vídeo composto[1], mas modulado em alta frequência, com o objetivo de ser transmitido através de uma antena. Quando este sinal chega a antena transmissora, são geradas ondas eletromagnéticas que se propagam no ar que podem ser captadas por uma ou mais antenas receptoras. Quando uma antena capta essas ondas, são gerados sinais elétricos semelhantes aos que foram usados no transmissor, mas estes sinais são muito mais fracos que o original, e ainda chegam ao receptor misturados com sinais de diversos outros emissores. Através de um processo chamado *sintonização,* é possível separar o sinal desejado, e através da *amplificação* ter o seu nível aumentado. Finalmente, é usada a *demodulação* para que o sinal de radiofrequência seja novamente transformado em vídeo composto. A figura 2.7 apresenta o receptor, juntamente com a sua antena, utilizada na captação dos sinais de vídeo.

1 Vídeo composto é um dos tipos mais populares de conexão de vídeo. Ele utiliza o conector RCA e é usado pelos populares conectores “vídeo in” e “vídeo out”, encontrados em videocassetes, aparelhos de TV, aparelhos de DVD e projetores de vídeo. Nos computadores, algumas placas de vídeo têm esta saída.

Figura 2.7 – Receptor da Microcâmera

Além da antena, o receptor possui um botão giratório (*TUNE*) usado para encontrar a faixa de frequência exata na qual a câmera está transmitindo, uma entrada DC para alimentação que pode ser entre 9 a 12 Volts, um LED para indicar quando o receptor encontra-se ligado e duas saídas RCAs, uma para áudio e outra para vídeo. Como o computador utilizado no trabalho não possui estas entradas, é necessário a utilização de uma placa de captura de vídeo denominada *EASYCAP*. Para a ligação entre a placa e o receptor é necessário um cabo RCA comum. Para se conectar ao computador, a placa *EASYCAP* dispõe de uma saída USB. A placa possui 4 entradas de vídeo e uma entrada de áudio, todas no padrão RCA. Na figura 2.8, é apresentado a placa *EASYCAP*.

Figura 2.8 – Placa de captura EASYCAP

Para visualizar o áudio e vídeo captados pelo receptor, é utilizado um programa denominado *Multiviewer*, versão 2.0 e criada por Zhong Kai Ran. O aplicativo tem algumas

funções interessantes como, por exemplo, enviar um e-mail ao usuário ou disparar um alarme quando o programa detecta movimento ou perde o sinal do vídeo, gravar as imagens apenas quando o programa detecta movimento, configurar níveis de brilho, contraste, etc. Na figura 2.9, pode-se ver a interface do programa.

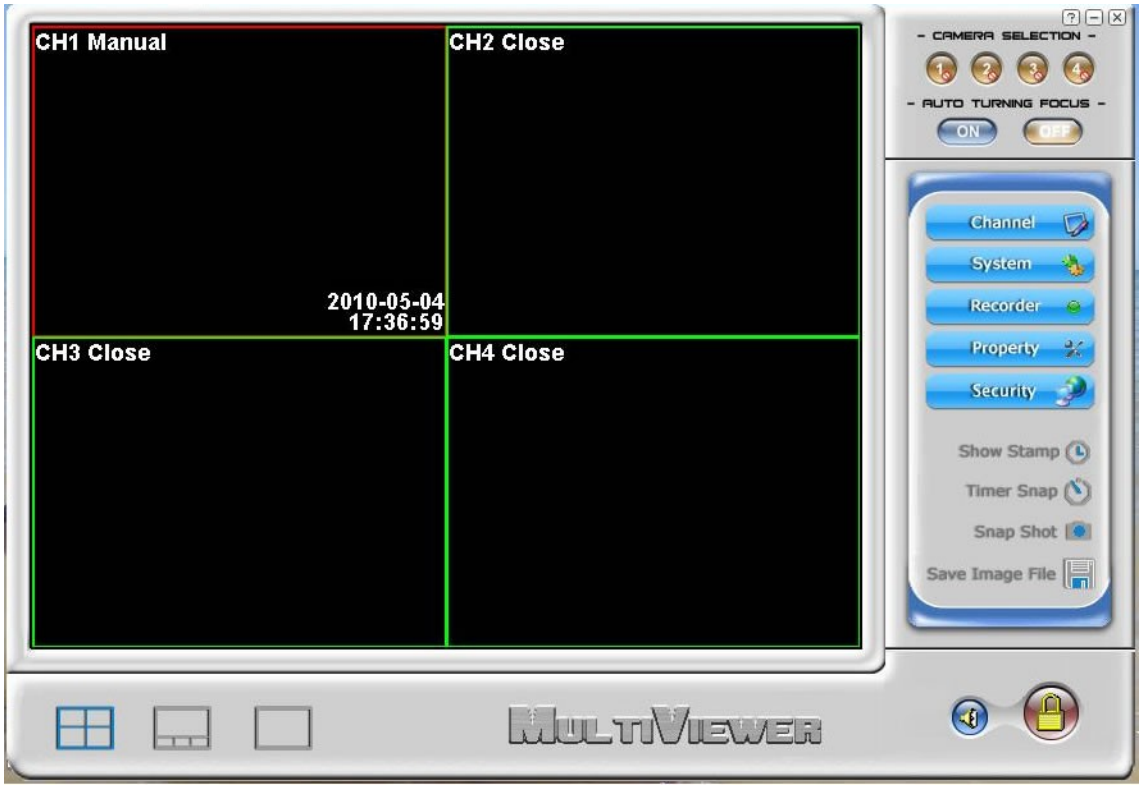


Figura 2.9 – Interface do Aplicativo MultiViewer

2.4 – CODIFICAÇÃO DE SINAL

Quando sinais digitais e analógicos são utilizados no transporte das informações pelo meio de transmissão, as propriedades dos sinais precisam, de alguma forma, representar os dados. O papel da codificação é justamente definir estas propriedades. A codificação é a representação do sinal digital transportado no sistema, através da amplitude e período dos sinais binários, cujas características físicas são selecionadas para possibilitar a otimização da performance de transmissão destes sinais digitais com relação ao do canal de transmissão (meio físico), bem como da performance do equipamento receptor. [TITTEL, 2002]

Em contrapartida, a decodificação, é usada pelo destinatário do sinal transmitido de modo que os padrões do sinal sejam convertidos novamente em sinais significativos.

Um dos modos mais simples de comunicar dados em uma rede de computadores, por exemplo, é utilizar um pequeno sinal elétrico para a representação dos dados. Assim, a tensão no meio que conecta dois dispositivos pode ser alterada de modo que um nível de tensão positiva (5v), representa o binário um (1) e um valor de tensão negativa (-5v), representa o binário zero (0). Transmitindo o binário 1, o dispositivo transmissor coloca uma tensão positiva no fio por um curto período de tempo e depois retorna ao nível 0 de tensão. O dispositivo receptor detecta a tensão positiva e registra a chegada do binário 1. Enviando o binário 0, o dispositivo transmissor coloca uma tensão negativa no fio por um período curto de tempo e, então ajusta o nível de tensão 0. O destinatário, recebendo a tensão negativa, registra a chegada do binário 0. [TITTEL, 2002]

A codificação deve proporcionar o sincronismo do receptor em relação à fase do sinal recebido, comparado ao sinal transmitido. Quando o sincronismo não é ideal, o sinal decodificado não detém as amplitudes originais do sinal transmitido, levando-se em consideração a ordem de chegada dos bits, ocasionando uma maior probabilidade de erro de bits recebidos.

O estudo sobre transmissão de sinais tem mostrado que, um sinal que oscila continuamente, viaja por uma distância maior do que outros sinais. Em sistemas de comunicação de longa distância, este sinal é chamado de portadora. Elas são, usualmente, senoidais e oscila mesmo quando não há nenhum sinal sendo transmitido. Para enviar dados na presença de um sinal contínuo, o transmissor modifica ligeiramente a portadora para representar a informação. Esta modificação é chamada de modulação e é utilizada em sinais de rádio, telefone e redes de computadores. [TITTEL, 2002]

De preferência, deve-se escolher uma codificação que possua uma estrutura, a qual possibilite a detecção de erro de bits transmitidos. Note que, o sinal decodificado deve possuir características físicas apropriadas para o meio físico que está sendo utilizado, por exemplo, meios físicos metálicos, ou meios físicos de fibras ópticas. Estas características físicas são únicas para cada tipo de meio físico, pois cada um deles possui diferentes comportamentos com relação à interferência, distorção, capacitância, e perda de amplitude. Neste projeto, o meio físico é o ar, transportando o dado através de ondas de rádio.

Um fator importante sobre a codificação utilizada no projeto, é que ela não deve conter componente DC (a média do nível do sinal deve ser 0), para que o demodulador no receptor interprete corretamente o dado recebido, pois a componente DC ocasiona uma modificação nas características elétricas do sinal no lado receptor, surgindo uma maior probabilidade de erro de decodificação. [FOROUZAN, 2006]

2.4.1 – Codificação Manchester

A codificação Manchester possui a característica de possibilitar a transmissão arbitrária de bits sem a ocorrência de longos períodos sem transição do estado da amplitude do bit, o que proporciona não perder o sincronismo do sinal de relógio, e a não ocorrência de erro de bit em baixas taxas de transmissão em enlaces com precária equalização. Trata-se, portanto de uma codificação auto-relógio. [FOROUZAN, 2006]

Os bits são normalmente referidos em grupos conhecidos como *bytes* ou octetos. Um *byte* ou octeto, é um grupo de oito bits e é normalmente representado no formato de dois caracteres hexadecimais (hex).Os valores hexa legais são de 0 a 9 e de A a F.

Na codificação Manchester, o dado é representado de acordo com o ponto no qual o sinal varia, ou seja, nas localizações onde o sinal altera de um valor positivo para zero representa o binário um (1) e, quando a tensão altera de zero para um valor positivo, o binário zero (0) é representado. Diz-se que o hardware usado nos esquemas de Codificação Manchester é sensível à borda do pulso de tensão e as mudanças ou transições são conhecidas como borda de subida ou descida. Quando a borda da frente sobe para uma tensão positiva, o binário 0 é codificado e, quando desce para a tensão zero, o binário 1 é codificado. Com este tipo de estrutura, cada período de bit é dividido em dois intervalos iguais, tendo uma transição no meio que facilita para o receptor sincronizar com o transmissor. Para assegurar que haja uma sincronização das janelas de tempo (*time slots*) quando da amostragem do sinal pelo receptor, a Codificação Manchester usa um preâmbulo para permitir a sincronização pelo receptor. O preâmbulo é composto por uns e zeros que se alternam, o qual é enviado antes do quadro de dados. Um padrão que alterna uns e zeros produz uma onda quadrada pela qual o receptor pode determinar o valor das janelas de tempo. O efeito combinado do preâmbulo e da transição no meio do período do bit elimina a necessidade de um relógio externo para sincronizar o transmissor e o receptor. [TITTEL, 2002]

A figura 2.10 demonstra o resultado da codificação Manchester para um determinado sinal.

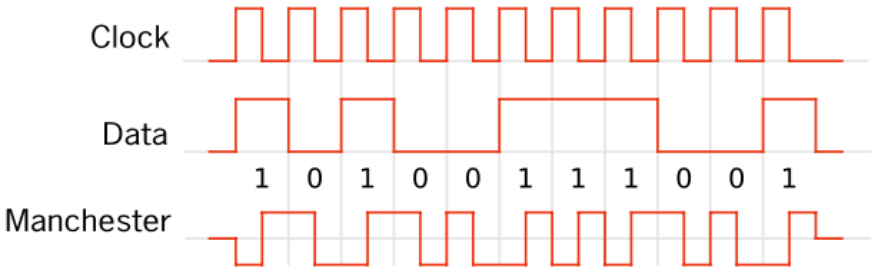


Figura 2.10 – Codificação Manchester do Sinal
Fonte: http://en.wikipedia.org/wiki/Manchester_code

A codificação Manchester não possui componente DC o que torna mais simples o processo de regeneração do sinal, bem como economia de energia. Ela proporciona uma maneira simples de codificação arbitrária de sequências binárias, não existindo longos períodos sem transição do sinal, possibilitando a não existência de perda de sincronismo, ou erros de bits transmitidos devido ao deslocamento da componente DC. [LIMA Jr, 2009]

Este tipo de codificação é empregado no circuito transmissor de temperatura sem fio e, a decodificação, no circuito receptor. Os detalhes de como foram criados a codificação e decodificação são explicados no tópico 3.3.

2.4.2 – Componente de Nível DC

A componente DC é uma energia extra residente na linha de sinal. Quando a ocorrência de símbolos uns (1s) e zeros (0s), em um sinal transmitido, é igual, não existe componente DC. Porém, tratando-se de longas sequências de 1s, ou 0s, surge uma componente DC, que causa distorção no sinal e pode ocasionar distorções nas informações recebidas, conforme ilustra a figura 2.11.

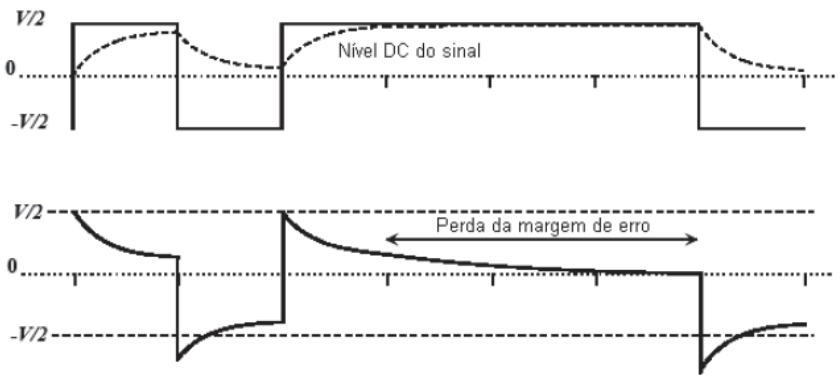


Figura 2.11 – Componente de nível DC.
Fonte: LIMA Jr, 2009

O fato acima comentado ocasiona grandes problemas em redes que utilizam acoplamento AC através de capacitores, transformadores e amplificadores AC, pois a resposta de frequência é prejudicada na parte da componente DC. O resultado disto é que o sinal se desloca para o nível zero (0), em longos períodos de 0s, o que reduz a margem de erro, e isto ocasiona o aumento da taxa de erro de bit, que pode ser evitado, através do uso do código Manchester, o qual é utilizado em Redes Locais de Computadores (LANs). [LIMA Jr, 2009]

A figura 2.12, exibe a forma dos pulsos Manchester (parte superior), e um exemplo de sequência de bits com codificação Manchester.

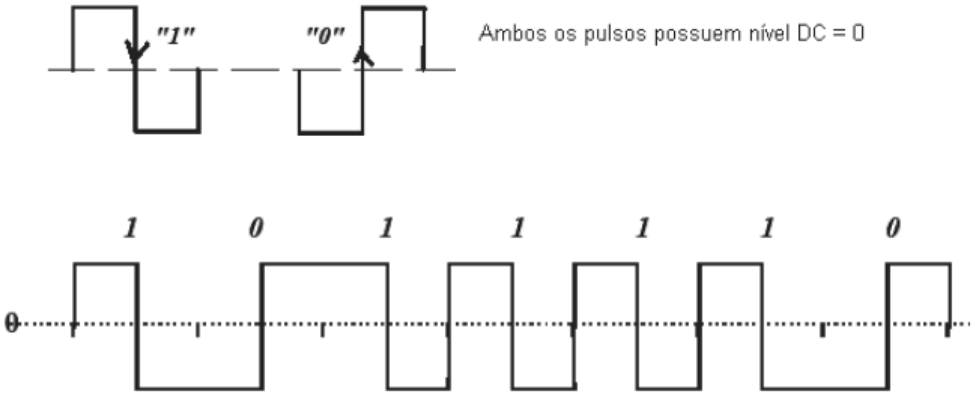


Figura 2.12 – Nível DC = 0 para pulsos Manchester.
Fonte: LIMA Jr, 2009

2.5 – MICROCONTROLADORES

O microcontrolador (MCU ou µC) é um componente eletrônico dotado de inteligência programável, sendo utilizado no controle de diversos periféricos como LEDs, *buzzers*, botões, diversos sensores (luminosidade, temperatura, umidade, pressão, etc), *displays*, entre outros. A sua lógica de operação é estruturada através de um programa que é gravado dentro do componente. [SOUZA, 2003]

O microcontrolador difere em muitos aspectos de um microprocessador (MPU ou µP). Enquanto o microprocessador precisa de componentes como memória, *timers* e interfaces de entrada e saída de dados para o seu funcionamento, o microcontrolador possui todos os recursos necessários para a sua utilização. Além destes elementos, pode-se citar, também, dispositivos periféricos como conversores analógico/digitais (ADC), conversores digitais/analógicos (DAC), comunicação serial, contadores, PWM (*Pulse Width Modulation*), etc. Devido a estes fatores, o microcontrolador é chamado de sistema computacional em um

único circuito integrado (*on-chip computer*). Outra diferença é que os microprocessadores são utilizados em aplicações onde são requeridos cálculos matemáticos complexos e com muita velocidade e, os microcontroladores são utilizados onde a velocidade de processamento não é tão alta e de forma dedicada, por exemplo em eletrodomésticos. As principais áreas de atuação do microcontrolador são a área automobilística, automação, robótica, entretenimento, médica, controle de tráfego e segurança.

Os microcontroladores operam utilizando uma frequência de *clock* muito baixa se comparados aos microprocessadores atuais, entretanto, são apropriados para uma vasta área de aplicações usuais.

O consumo do microcontrolador, em geral, é relativamente baixo, normalmente na casa dos *miliwatts* e muitos modelos possuem a habilidade de entrar em modo de espera (*Sleep*) aguardando por uma interrupção ou evento externo, como por exemplo o acionamento de uma tecla, ou um sinal que chega por uma interface de dados. O consumo destes microcontroladores em modo de espera pode chegar na casa dos *nanowatts*, tornando-os ideais para aplicações onde a exigência de baixo consumo de energia é um fator decisivo para o sucesso do projeto. [SOUZA, 2003]

Para o sistema, são utilizados os microcontroladores PIC12F675, PIC16F676 e PIC18F4550, todos fabricados pela Microchip. A partir deles, é possível a realização de diversas tarefas necessários ao funcionamento do sistema. Entre essas tarefas, pode-se destacar a conversão analógica/digital, transmissão e recepção de dados via comunicação serial, codificação e decodificação de dados, conexão ao computador por intermédio da porta USB, entre outros. No capítulo 3, são fornecidos mais detalhes sobre o modo de funcionamento dessas tarefas.

De acordo com a informação fornecida no tópico 2.2.1, o programa utilizado na criação e compilação do código necessário para o funcionamento dos microcontroladores é o C Compiler da empresa CCS. Após o código ser gerado, é necessário que o mesmo seja transferido do computador para o microcontrolador. Para isso, foi utilizado o dispositivo *PICBurner*, que é um gravador e tem a função de fazer a conexão entre o computador e o microcontrolador e inserir os dados na memória deste último. O gravador se conecta ao computador através da porta serial. Na figura 2.13 é exibida a foto do gravador.

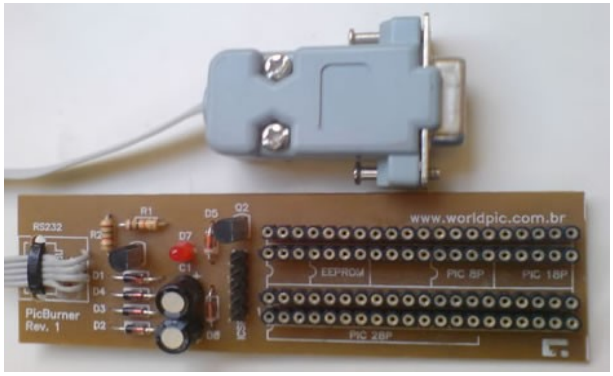


Figura 2.13 – Gravador *PICBurner*

Vale ressaltar que o gravador é apenas o meio de transmissão. Além dele, foi preciso o uso de um aplicativo capaz de enviar o código hexadecimal até a porta serial. O programa utilizado foi o *PICPgm Development Programmer* e sua interface é apresentada na figura 2.14:

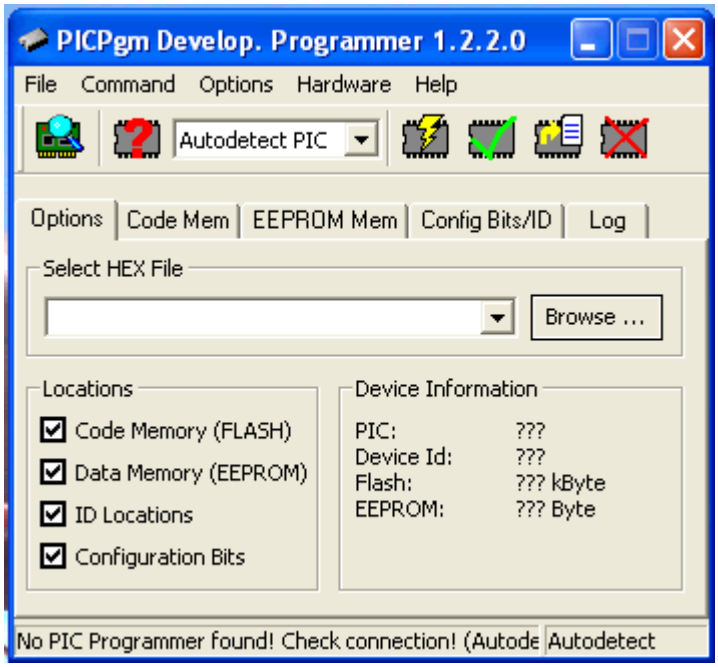


Figura 2.14 – Interface do *PICPgm*

2.6 – SERVOMOTORES

O servomotor é um dispositivo eletromecânico que, ao receber um sinal elétrico na sua entrada, pode ter o seu eixo posicionado numa determinada posição angular. Enquanto

existir o sinal aplicado na entrada, o servo mantém a posição angular do eixo. Quando este sinal é alterado, a posição angular do eixo também é modificado. Estes motores são amplamente utilizados na robótica devido ao seu baixo custo, facilidade de acionamento e precisão de posicionamento. Além disso, são bem pequenos, leves, compactos e possuem uma elevada potência em relação ao seu tamanho.

O projeto utiliza dois micros servos da marca *TowerPro*, modelos MG90 e MG90S. Eles tem a função de movimentar a câmera sendo que, o MG90S é utilizado para movimentação no sentido vertical e, o MG90, no sentido horizontal. A tabela 2.5 contém as especificações de ambos modelos, de acordo com o fabricante:

Tabela 2.5 – Características do Servomotor

SERVO	MG90	MG90S
Peso	14 gramas	13,4 gramas
Dimensão	23 x 12,2 x 29 mm	22,8 x 12,2 x 28,5mm
Torque máximo	2,2 Kg/cm (4,8V) 2,5 Kg/cm (6,0V)	1,8 Kg/cm (4,8V) 2,2 Kg/cm (6,0V)
Velocidade de operação	0,11 seg/60º (4,8V) 0,10 seg/60º (6,0V)	0,1 seg/60º (4,8V) 0,08 seg/60º (6,0V)
Voltagem de operação	4,8 a 6 V	4,8 a 6 V
Largura de Banda Morta	10 µs	5 µs

Fonte: http://www.towerpro.com.tw

A largura de banda “morta” previne o servo de estar em estado de movimento contínuo, quando pulsos muito pequenos ocorrem.

A figura 2.15 apresenta os dois servomotores utilizados neste projeto, mais especificamente, no circuito explicado no item 3.5.

Figura 2.15 – Servomotores utilizados no Projeto

O servomotor possui um sistema eletrônico de controle e um potenciômetro conectado ao eixo de saída. O sistema de controle é responsável por receber os sinais e energia do receptor e controlar o motor de acordo com o sinal recebido e a posição do potenciômetro. [SANTOS, 2007]

O potenciômetro permite ao circuito de controle, monitorar o ângulo do eixo do servo. Quando o eixo está no ângulo certo, o motor pára. Caso o circuito de controle detecte que o eixo não se encontra no ângulo correto, o motor é acionado até que o ângulo certo seja detectado. [SANTOS, 2007]

Nas engrenagens do servomotor, existe um limitador que atua no ângulo de giro do eixo, fazendo com que o mesmo varie de zero a cento e oitenta graus (0º a 180º). A figura 2.16 demonstra todos os componentes presentes em um servomotor.

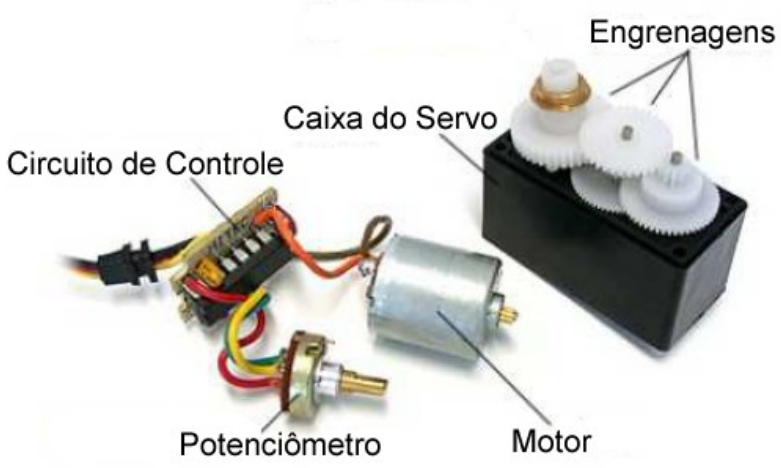


Figura 2.16 – Componentes que formam o servomotor
Fonte: SANTOS, 2007

Além dos componentes, o servomotor possui três fios responsáveis pela conexão do mesmo ao sistema, demonstrados na figura 2.17:

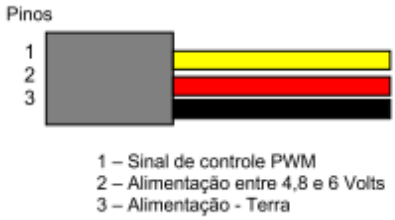
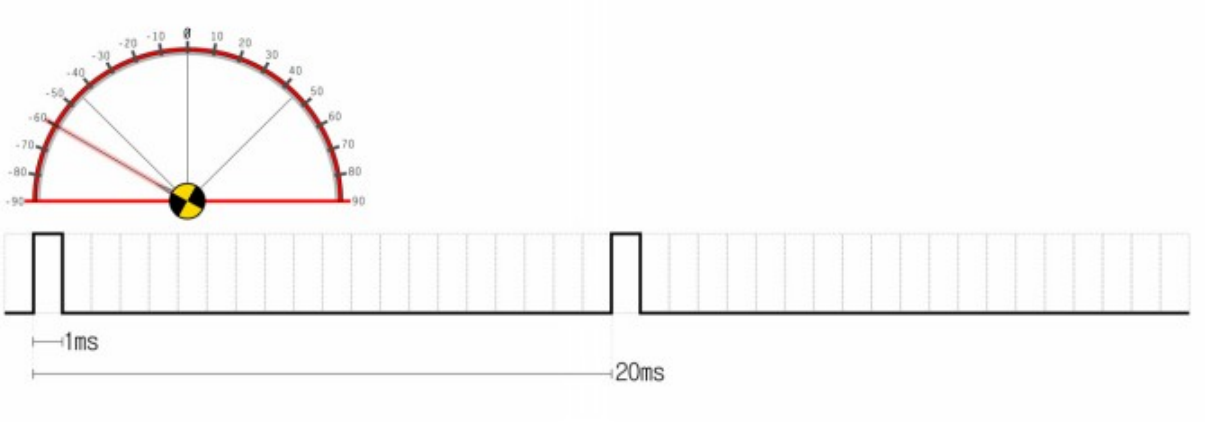


Figura 2.17 – Sinais no conector do servomotor

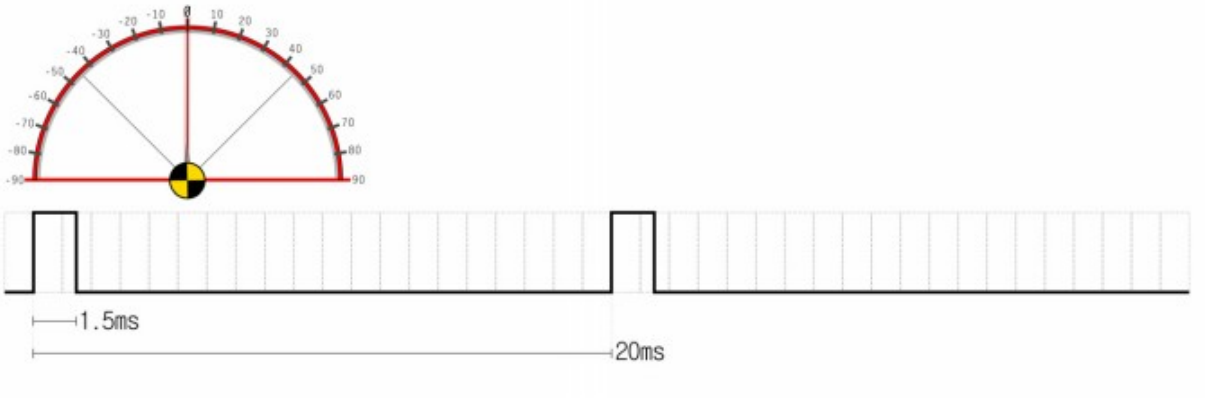
O pino um é responsável por receber o sinal de controle. Os pinos dois e três são destinados à alimentação do servo. O ângulo do eixo é determinado a partir da duração da

largura de pulso (PWM – *Pulse Width Modulation*) recebido pelo pino um. Este sinal pode ser de zero ou cinco Volts. O circuito de controle monitora o sinal em intervalos de vinte milissegundos (20 ms). A largura do impulso determina o ângulo de rotação do motor. Um impulso de um milissegundo e meio (1.5 ms), por exemplo, faz com que o motor gire o seu braço até a posição zero graus (também designada posição neutra). Se o impulso for de aproximadamente um milissegundo (1.0 ms), então o motor movimenta o eixo no sentido anti-horário cerca de sessenta graus (-60º). Se o impulso for de dois milissegundos (2.0 ms) aproximadamente, então, o eixo do motor percorre no sentido horário até a posição de 60 graus. Estes valores são apenas referenciais, uma vez que não são padronizados e, dependendo do fabricante, o motor utiliza outros valores de largura de pulso para que o eixo se movimente até determinada posição. Inclusive, pôde-se notar pequenas diferenças de posições do braço utilizando os mesmos valores de largura de pulso entre os dois servomotores do projeto, lembrando que ambos são da mesma marca.

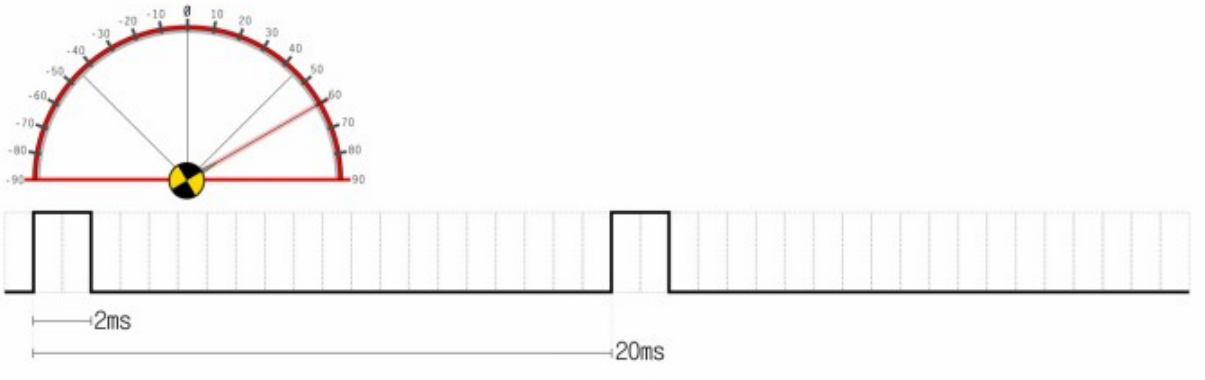
A figura 2.18 representa os tempos de cada pulso (em milissegundos) e sua respectiva posição do eixo do servo para um servomotor qualquer.



(a)



(b)



(c)

Figura 2.18 – Pulsos de controle de posição

CAPÍTULO 3 – DESENVOLVIMENTO DO PROJETO

3.1 – CIRCUITO FAROL

O circuito farol consiste em acionar automaticamente dois LEDs brancos de alto brilho quando o veículo encontra-se em ambientes com pouca ou nenhuma luminosidade. Assim como o acionamento, nos casos em que o carro sai de um local escuro para outro iluminado, os LEDs são desligados. Estes LEDs são posicionados juntamente a microcâmera, auxiliando o usuário na visibilidade tanto para controlar o veículo, quanto para monitorar o ambiente.

Para o circuito, os principais componentes utilizados são os LEDs, um LDR (do inglês *Light Dependent Resistor* ou *Resistor Variável Conforme Incidência De Luz em português*) e um C.I. LM741.

O LED (do inglês *Light-emitting Diode*) é um diodo que utiliza compostos semicondutores diferentes de germânio ou silício. Os diodos comuns apresentam uma certa resistência à passagem da corrente elétrica e dissipam a energia na forma de calor e os LEDs são produzidos com materiais que têm a capacidade de emitir luz além de calor. [PATZKO[2], 2006]

A aplicação dessas diferentes substâncias correspondem a utilização da teoria da eletroluminescência na prática. A teoria diz que alguns materiais têm capacidade de emitir luz quando conduzem uma corrente elétrica. Portanto, não é necessário o aquecimento de um material (incandescência) ou uma reação química (quimioluminescência) para que se crie uma fonte de luz.

Quando há a condução de uma corrente elétrica, os elétrons do composto ficam excitados e emitem sua energia na forma de fótons, ou seja, de luz. A frequência da luz emitida pelo elétron determina sua cor e depende da substância que é utilizada. Por exemplo, os LEDs vermelhos são fabricados utilizando-se o nitrato de gálio (GaN) e os LEDs azuis são produzidos com o fosfato de gálio (GaP). [PATZKO[2], 2006]

Uma das principais vantagens do LED, é a utilização de elementos eletroluminescentes na sua fabricação, pois, ao contrário das lâmpadas incandescentes ou fluorescentes, não se faz necessário o aquecimento de um filamento ou um gás para que

haja a emissão de luz. Com isso, percebe-se que o LED é muito mais eficiente, pois necessita de menos corrente elétrica para produzir a mesma quantidade de energia de uma lâmpada comum, possibilitando uma redução no consumo de energia, o que é essencial principalmente em equipamentos onde são utilizados pilhas e baterias. A figura 3.1 apresenta alguns LEDs com diferentes formas e cores.

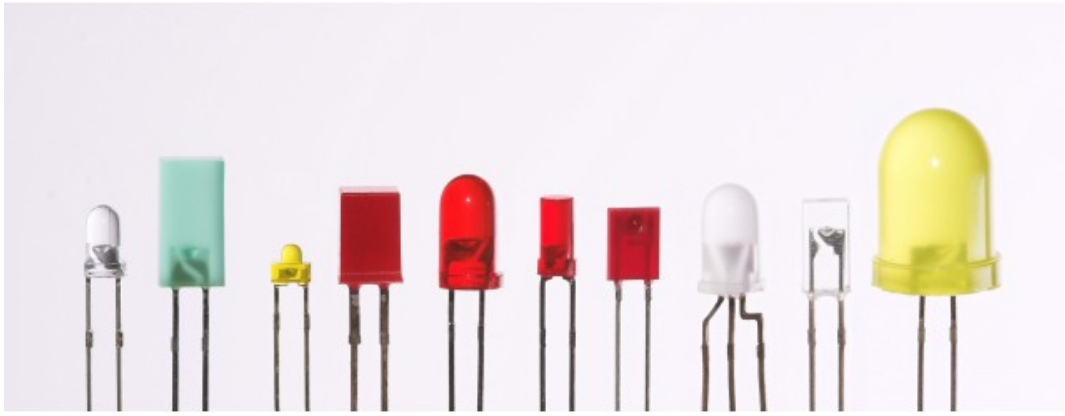


Figura 3.1 – LEDs diversos
Fonte: PATZKO[2] ,2006

Outras vantagens a serem destacadas, é a sua vida útil de aproximadamente 100.000 horas contra 1.000 horas das lâmpadas incandescentes e 10.000 horas das fluorescentes, resistência a choques e vibrações devido ao encapsulamento de plástico maciço e a sua rapidez no acendimento (o LED leva cerca de 0,01 segundo para ter brilho total). A figura 3.2 apresenta os dois LEDs utilizados neste circuito.

Figura 3.2 – Par de LEDs

Conforme o seu nome diz, o LDR é um tipo de resistor cuja resistência varia conforme a intensidade de radiação eletromagnética do espectro visível que incide sobre ele. Pode-se dizer que ele é um transdutor de entrada (sensor) que converte a luz em valores de resistência. A figura 3.3 apresenta uma foto do componente LDR.

Figura 3.3 – LDR

Os LDRs são compostos por sulfeto de cádmio (CdS), um material semicondutor, que é disposto num traçado onduloso na superfície do componente. Esse material tem a propriedade de diminuir sua resistência à passagem da corrente elétrica quando a luminosidade sobre ele aumenta, podendo-se chegar a uma mínima resistência de aproximadamente 100 ohms e, também, aumentar a resistência conforme diminui a luminosidade obtendo-se uma resistência máxima acima de 1M ohms. [PATZKO[1], 2006]

Através da figura 3.4, pode-se verificar o gráfico de resposta do LDR:

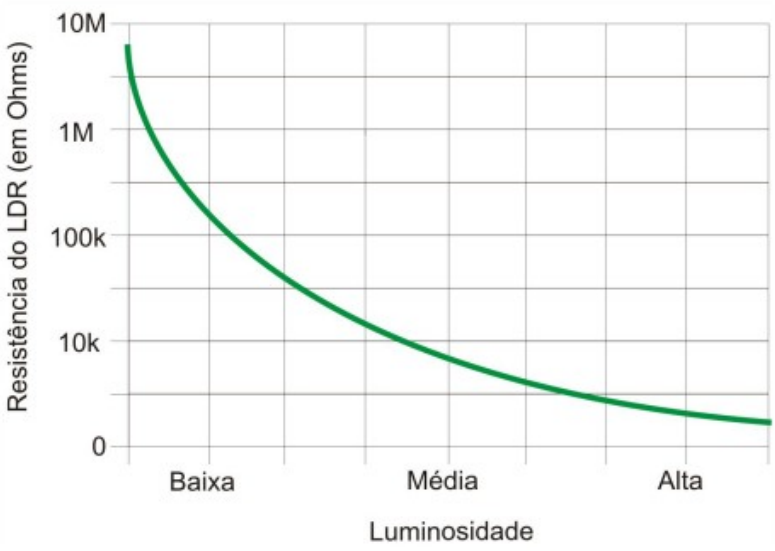


Figura 3.4 – Gráfico de resposta do LDR
Fonte: PATZKO[2] ,2006

O LDR presente no circuito foi utilizado em um divisor de tensão. Um divisor de tensão é composto por dois resistores ligados em série, conforme demonstra o esquema elétrico presente na figura 3.5. A tensão no ponto entre os dois resistores depende dos valores das suas resistências e da tensão de alimentação. Essa tensão pode ser calculada através da fórmula 3.1.

$$Vr = \frac{Vt \times R1}{R1 + R2} \tag{3.1}$$

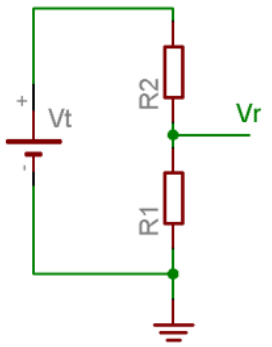


Figura 3.5 – Divisor de Tensão

Como o LDR é um sensor resistivo, pode-se substituir um dos resistores do divisor de tensão por este componente. Substituindo R2 por um LDR conforme é mostrado na figura 3.6, obtêm-se um circuito, cuja tensão aumenta de acordo com a luminosidade do ambiente.

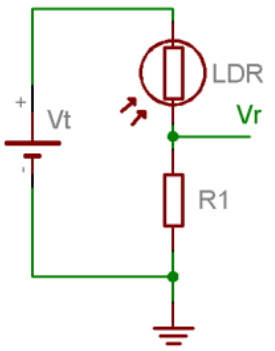


Figura 3.6 – Divisor de tensão com LDR

O C.I. 741 é um amplificador operacional (AmpOp) constituído por um bloco amplificador de tensão de alto ganho, baseado em transístores (bipolares ou fet) dotados de uma única saída e duas entradas, sendo uma inversora e uma não inversora. A figura 3.7 mostra a representação do C.I. por dentro, bem como, a sua pinagem.

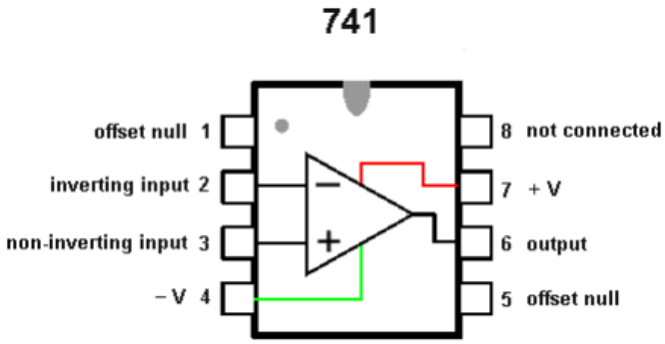


Figura 3.7 – Pinagem do CI 741

O C.I. 741 possui várias aplicações. Dentre elas, pode-se destacar:

1. Amplificador inversor;
1. Amplificador não-inversor;
2. Amplificador diferencial;
3. Comparador de tensão;
4. Oscilador;
5. Filtro ativo de frequência.

Neste circuito, ele é utilizado como um comparador de tensão. Um comparador de tensão é um amplificador operacional de alto ganho ligado de forma a comparar uma tensão de entrada com uma tensão de referência. O AmpOp compara a tensão das duas entradas, inversora (negativa) e não inversora (positiva). Quando o módulo da tensão da entrada positiva é superior à da entrada negativa, a saída do AmpOp tem a tensão de alimentação do circuito (VCC). Se a tensão da entrada negativa é superior à da entrada positiva, acontece o contrário, ou seja, na saída do AmpOp é encontrado uma tensão de zero Volts (GND). A figura 3.8 é o desenho esquemático do circuito desenvolvido:

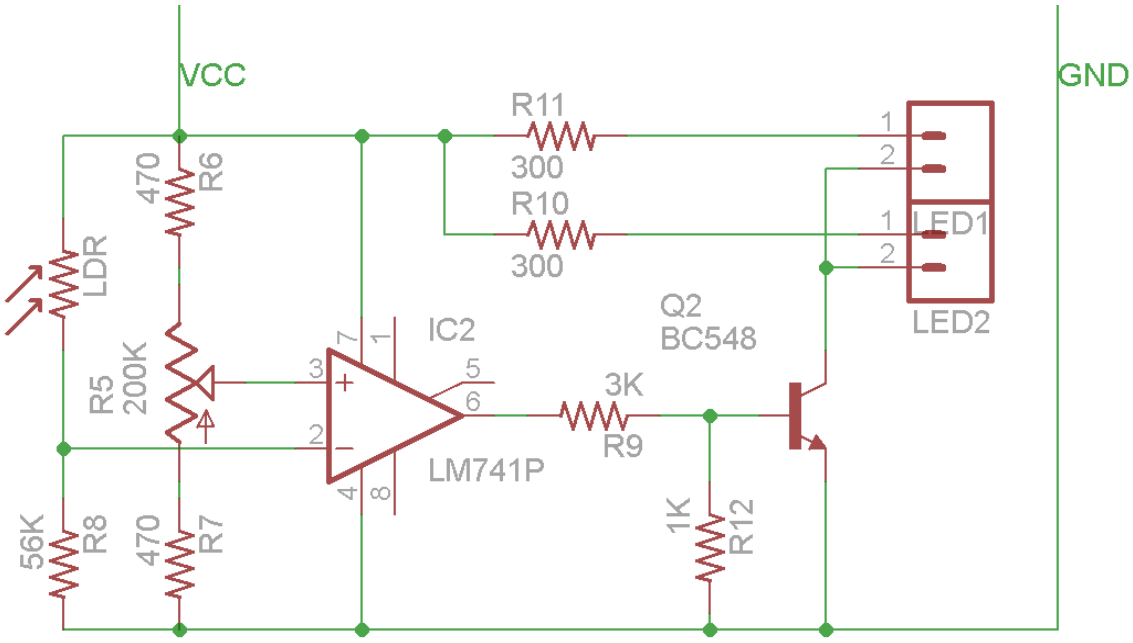


Figura 3.8 – Esquema do circuito farol

Quando há luz incidida o suficiente sobre o LDR, a resistência formada pelo divisor de tensão entre ele e o resistor R8 é baixa, fazendo com que a tensão na entrada inversora seja maior que na entrada não inversora e, com isso, a saída (pino 6) tem nível baixo (Vsaída=0). Quando há pouca ou nenhuma luz sobre o LDR, a tensão na entrada inversora é menor do que a tensão na entrada não inversora, tornando a saída do amplificador operacional alta e conduzindo a tensão que passa pelo resistor R9 que limita a corrente que flui para a base do transistor BC548. Este transistor funciona como uma chave, que fecha o circuito, permitindo que a corrente flua entre o coletor e o emissor, realizando, assim, a ativação dos LEDs. Para que a saída do C.I. não fique flutuante, foi adicionado um resistor de *Pull-down* (R12) entre ela e a base do transistor.

O resistor R5 é um potenciômetro e, através dele, pode-se ajustar a sensibilidade do circuito. Vale destacar que o amplificador operacional foi extremamente importante para o correto funcionamento do circuito, pois sem ele, os LEDs acenderiam aos poucos, conforme fosse baixando a luminosidade incidida no LDR. A figura 3.9 exibe uma foto da placa de circuito impresso construída.

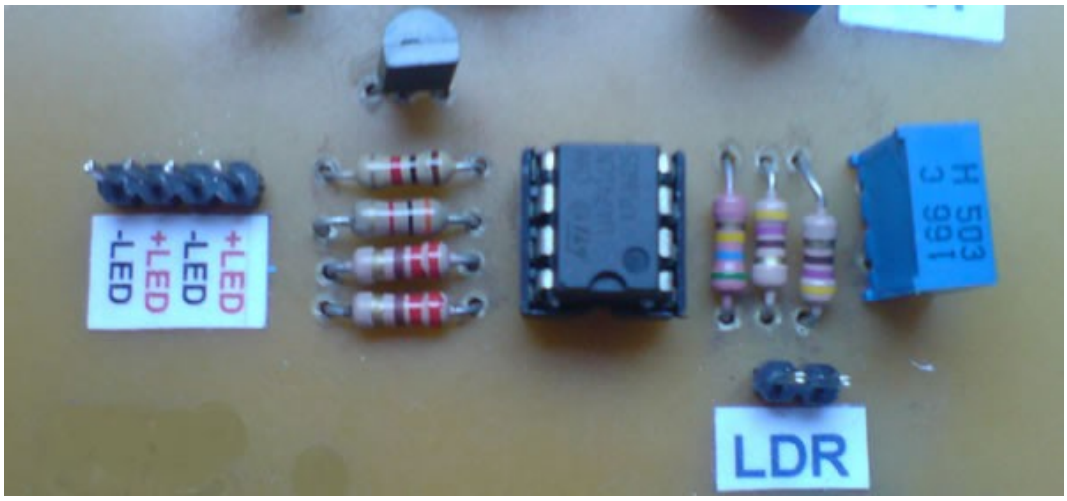


Figura 3.9 – Placa do Circuito Farol

3.2 – CIRCUITO USB

Este circuito é responsável por realizar a conexão entre o computador e os circuitos do projeto para que possam enviar ou receber dados entre eles. A placa se conecta ao computador através da porta USB graças ao microcontrolador PIC18F4550. Ele possui uma USB SIE (*Serial Interface Engine*) compatível com *low-speed* e *full-speed*, o que possibilita a rápida comunicação entre um *host* USB e o microcontrolador. O circuito também se conecta a placa responsável pelo controle do carro e da câmera através de um cabo com conectores *Headers* de 10 pinos, sendo que 8 pinos são responsáveis para ativar os pinos de movimento dos controles e os 2 pinos restantes são o positivo e o negativo da tensão da porta USB. Como a porta USB fornece uma tensão de 5 Volts, essa alimentação foi utilizada tanto neste circuito, quanto nos circuitos de controles, dispensando uma fonte externa para os mesmos. A figura 3.10 apresenta o encapsulamento e pinagem do microcontrolador PIC18F4550.

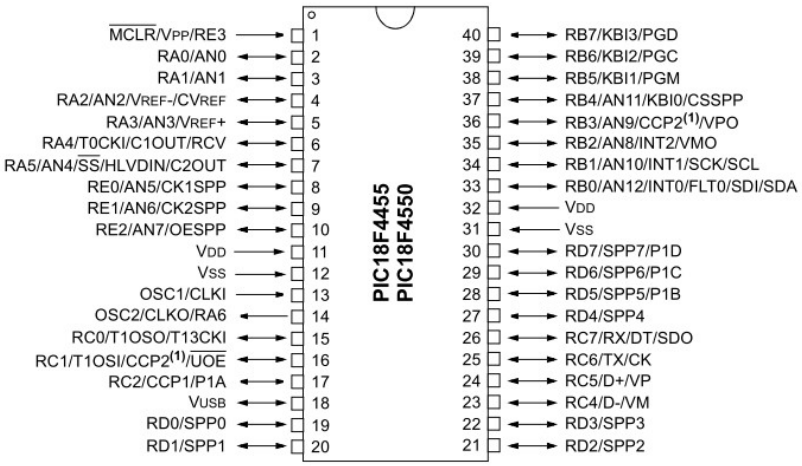


Figura 3.10 – Microcontrolador PIC18F4550
Fonte: Data Sheet – PIC18F4550, Microchip

Além do microcontrolador, outro componente importante neste circuito é um módulo receptor de radiofrequência, responsável por receber os dados emitidos pelo transmissor remoto instalado no circuito de medição de temperatura, enviando-os ao microcontrolador para que o mesmo decodifique estes dados e encaminhe-os ao computador logo em seguida. A utilização do módulo receptor é explicado com mais detalhas no item 3.3 deste trabalho, enquanto o tipo de codificação utilizada para a transmissão é explicado no item 2.4.2.

Os dados que são transmitidos pelo computador, são caracteres enviados um a um que chegam até o microcontrolador. O computador envia até 16 caracteres diferentes, sendo 8 caracteres utilizados para o controle do carro e os 8 restantes para o controle da câmera. No controle do carro, dependendo do caractere recebido pelo microcontrolador, coloca determinado pino em tensão alta ou baixa. No caso do pino em tensão alta, faz com que o respectivo movimento do carro seja acionado e, o pino em tensão baixa, cessa o movimento. A mesma explicação se aplica ao controle da câmera. O desenho esquemático do circuito é apresentado na figura 3.11.

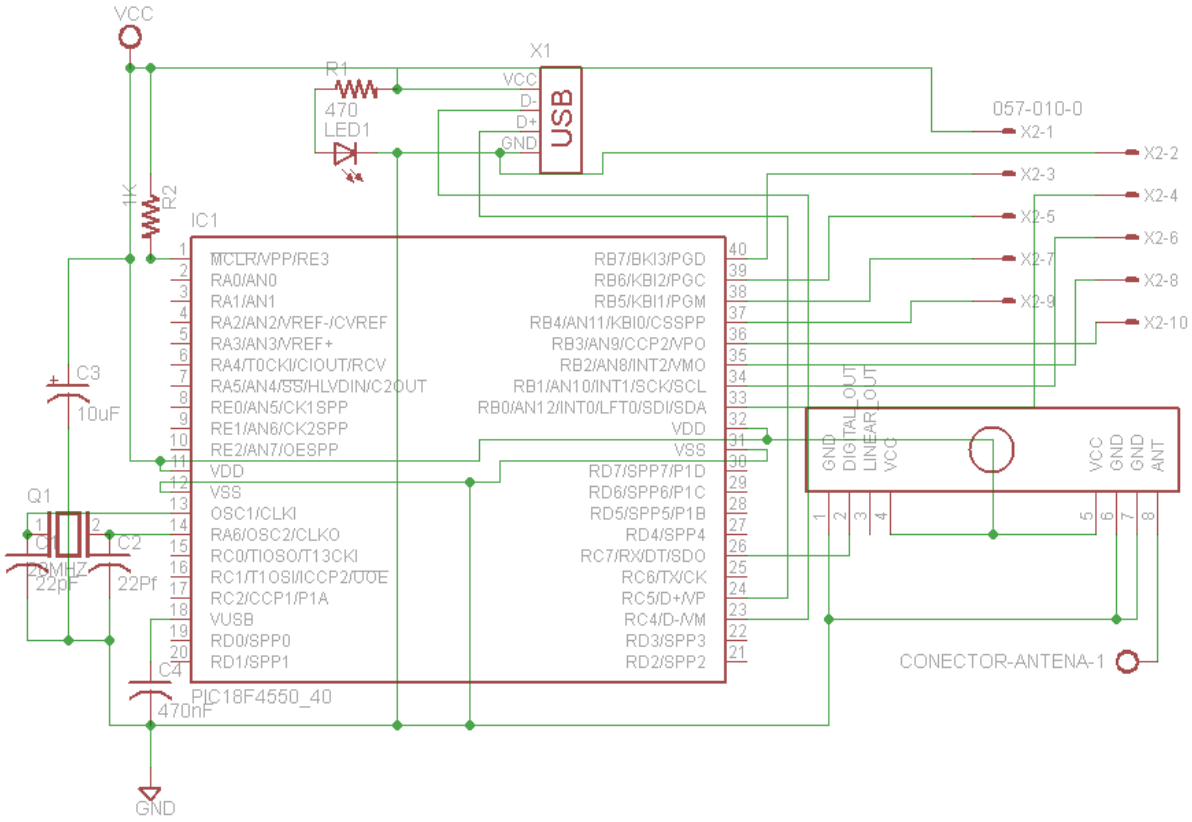


Figura 3.11 – Esquema elétrico do circuito USB

Apesar de o circuito utilizar a porta USB para se conectar ao computador, o microcontrolador se comporta como um dispositivo do tipo CDC (*Communications Device Class* - Classe 02h). Sendo assim, o sistema operacional o reconhece como se fosse um dispositivo que se comunica com o computador através de uma porta serial de comunicação (COM).

Assim como todos os dispositivos USB, o presente circuito necessita que um *driver* seja instalado ao computador para que funcione corretamente. Quando o circuito é conectado pela primeira vez no computador, o sistema operacional informa que um novo hardware foi detectado, conforme mostra a figura 3.12.

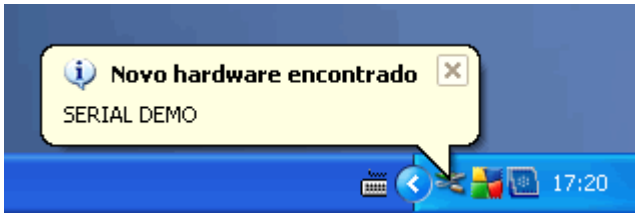


Figura 3.12 – Mensagem do Sistema Operacional

Em seguida, é iniciado o “Assistente para adicionar novo hardware” do *Windows já*

que o mesmo não dispõe do *driver* necessário para o funcionamento do dispositivo e solicita o local onde o arquivo pode ser encontrado.

Este *driver* é o arquivo *'cdc_NTXPVista.inf'* e está situado na pasta *'Drivers'* criada pela instalação do aplicativo C Compiler. Após a indicação do local, o *driver* é instalado conforme demonstra a figura 3.13.

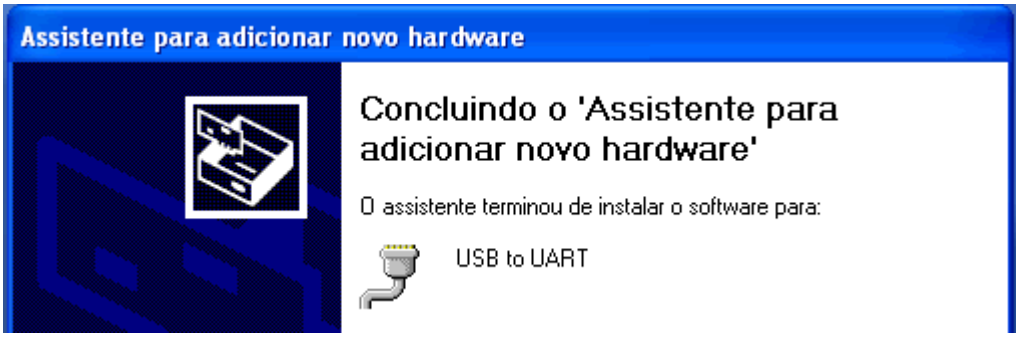


Figura 3.13 – Tela de instalação do driver

Como o dispositivo se comporta como se fosse um *hardware* conectado a porta serial do computador, o sistema operacional cria uma porta COM virtual, conforme demonstra a figura 3.14:

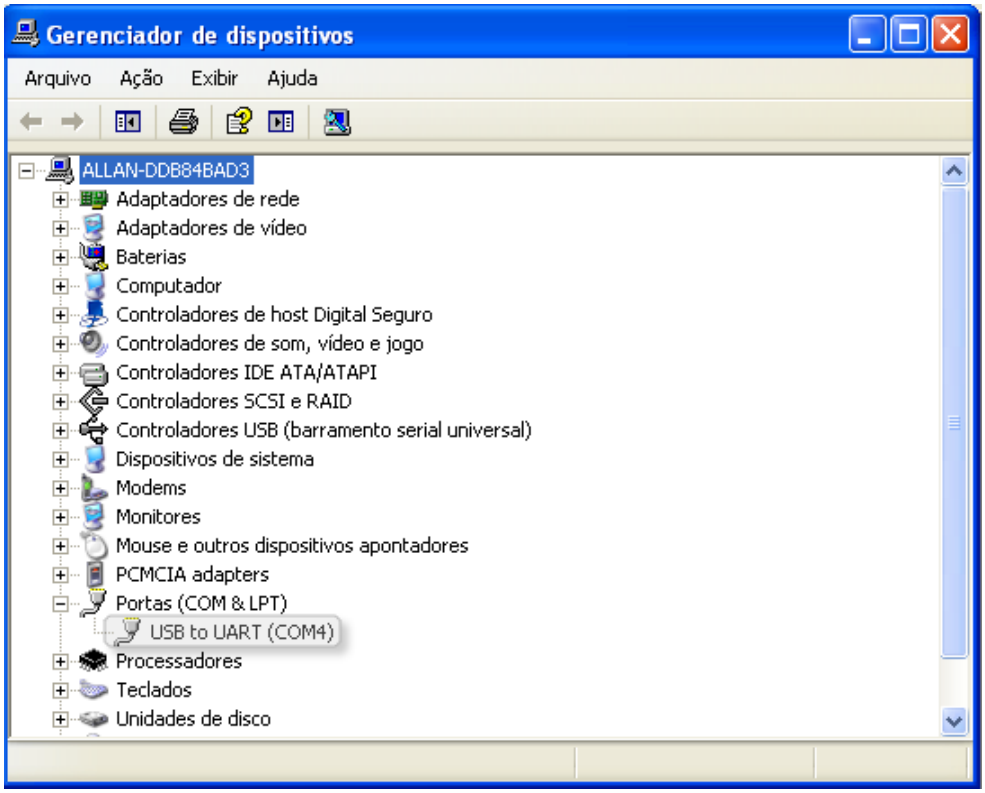


Figura 3.14 – Criação da porta COM virtual

Conforme consta na figura 3.14, o circuito está instalado em '*Portas (COM & LPT)*', como se estivesse conectado a porta COM4. A figura 3.15, exibe foto do circuito construído.

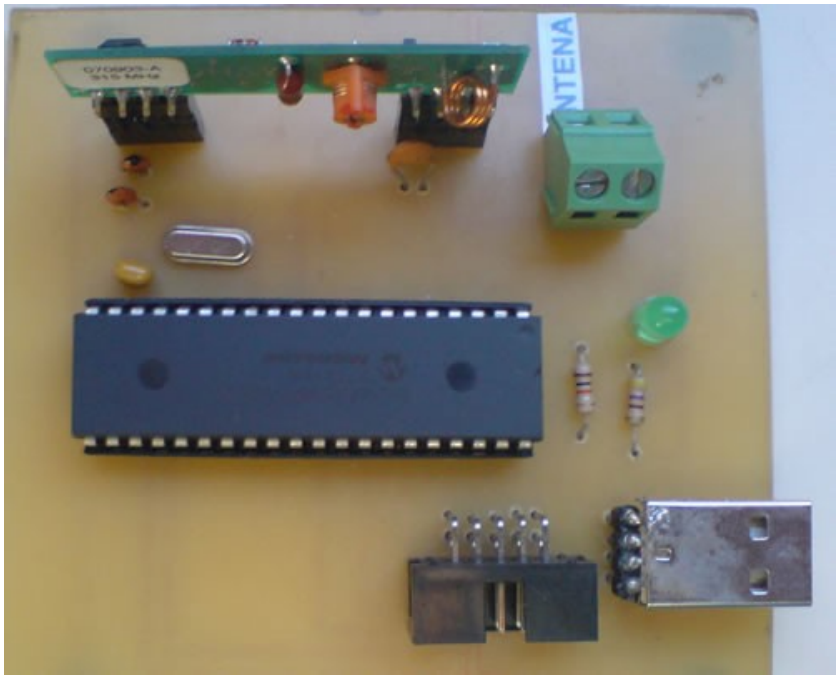


Figura 3.15 – Placa do Circuito USB

3.3 – CIRCUITO MEDIDOR DE TEMPERATURA SEM FIO

Esta parte do projeto é uma espécie de termômetro sem fio. Ele consiste de dois circuitos. O primeiro circuito está localizado dentro do veículo e tem a função de medir a temperatura do ambiente onde se encontra e enviar a informação através de um transmissor de radiofrequência. A figura 3.16 representa o desenho esquemático do circuito:

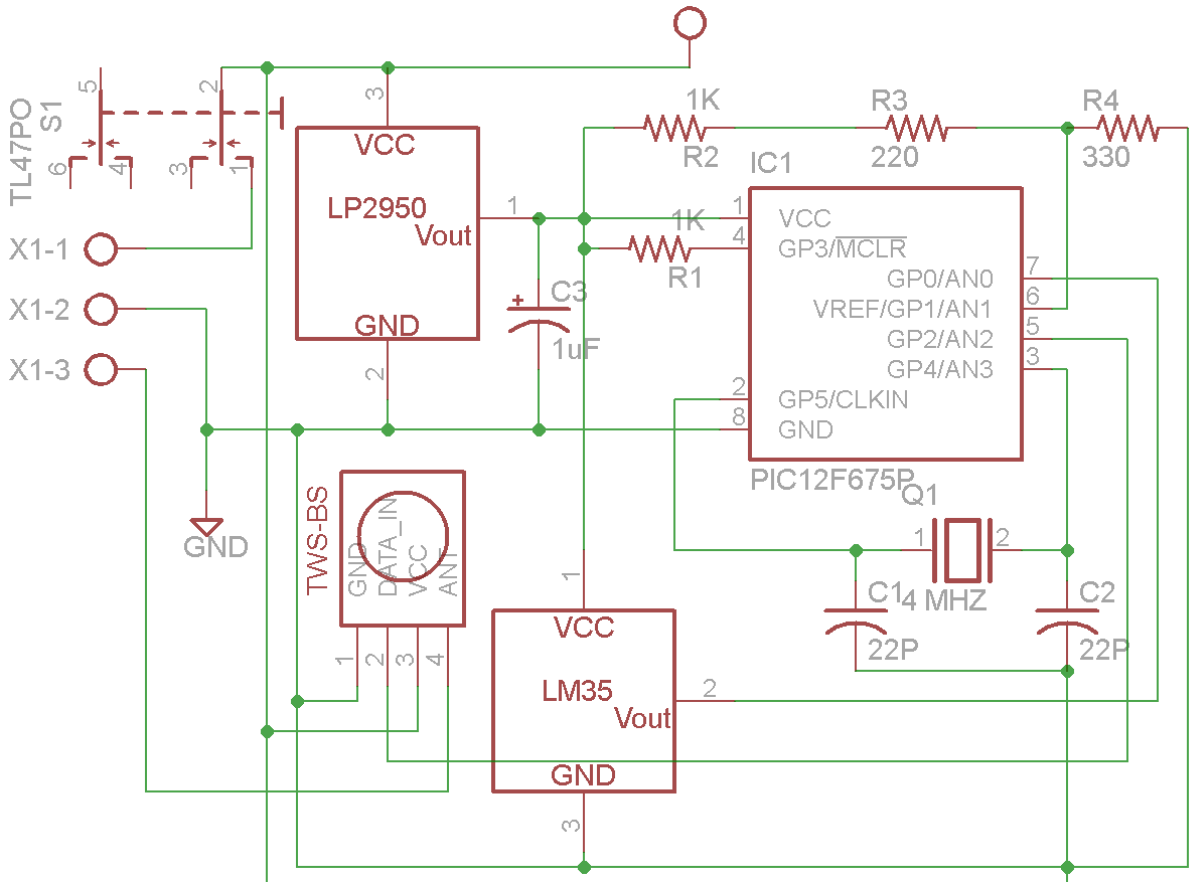


Figura 3.16 – Esquema elétrico do circuito leitor e transmissor de temperatura

O segundo circuito, responsável por receber as informações enviadas, é o circuito USB que se encontra conectado ao computador, cujos detalhes estão reproduzidos no item 3.2. Este circuito possui um receptor de radiofrequência que recebe as informações enviadas pelo transmissor e um microcontrolador que decodifica as informações recebidas e as envia ao computador.

Os módulos de transmissão e recepção são fabricados pela empresa *Wenshing*, utilizam a modulação ASK (*Amplitude Shift Keying* – Chaveamento de Amplitudes) e trabalham na frequência de 315 MHz. O modelo do transmissor é o TWS-BS-6 mostrado na figura 3.17, juntamente, com a sua pinagem. De acordo com o seu D*ata Sheet*, ele possui uma taxa de transmissão de 8000 bps e pode ser alimentado por uma tensão que vai de três (3) a doze (12) Volts.

Figura 3.17 – Módulo Transmissor TWS-BS-6
Fonte: Data Sheet – TWS-BS-6, Wenshing

Já o modelo de receptor usado é o RWS-374-3. Sua taxa de transferência é de 4800 bps e deve ser alimentado por uma tensão de cinco Volts. A figura 3.18 mostra a imagem do módulo receptor com os seus respectivos pinos.

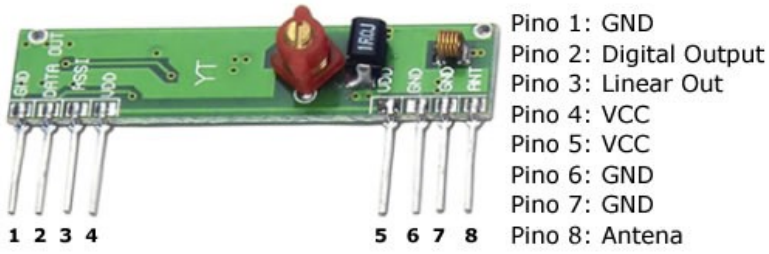


Figura 3.18 – Módulo Receptor RWS-374-3
Fonte: Data Sheet – RWS-374-3, Wenshing

A alimentação do primeiro circuito é feita através da própria bateria do veículo que é de 9,6 Volts com a adição de um C.I. regulador de tensão (LP2950) de 5 Volts, pois o microcontrolador não suporta tensão maior do que 5,5 Volts.

Para a medição da temperatura, é utilizado o circuito integrado LM35DZ. Ele é um sensor de temperatura cuja tensão de saída é linearmente proporcional a escala de temperatura Celsius. Cada incremento de 0,1°C representa um aumento de 1mV na tensão, ou seja, quando a temperatura medida é 24,8ºC, a tensão na saída do circuito integrado é 0,248V. O LM35DZ opera num intervalo de 0º a 100ºC e, de acordo com o seu D*ata Sheet*, é adequado para aplicações remotas e tem uma precisão de 0,5ºC quando a temperatura está acima de 25ºC, ou seja, no momento em que a temperatura ambiente é 26ºC, a resposta do sensor corresponde a um valor entre 25,5 e 26,5ºC. O encapsulamento do modelo utilizado é do tipo TO-92, cujo o mesmo é utilizado em pequenos transistores. Nas figuras 3.19 e 3.20, têm-se, respectivamente, uma ilustração do componente com os seus respectivos pinos e, o seu gráfico de resposta.

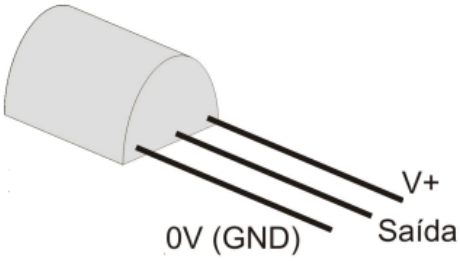


Figura 3.19 – Pinagem do LM35DZ

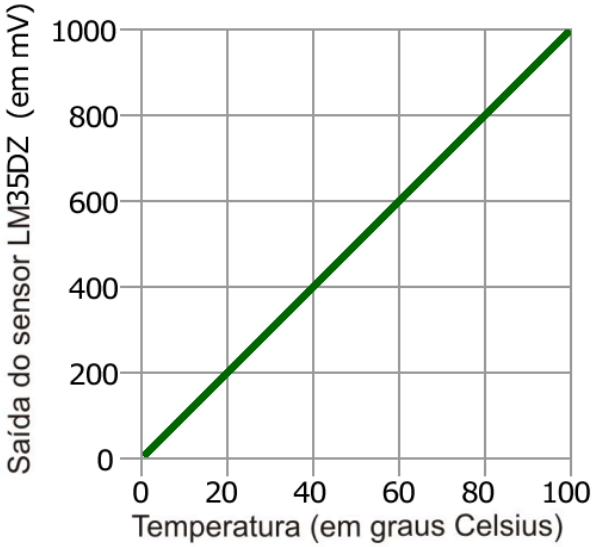


Figura 3.20 – Gráfico de resposta do LM35DZ

Além do sensor de temperatura, outro componente importante empregado neste circuito é o microcontrolador PIC12F675, sendo representado na figura 3.21 com os seus respectivos pinos.

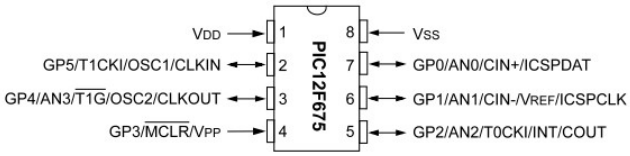


Figura 3.21 – Microcontrolador PIC12F675
Fonte: Data Sheet - PIC12F675

Este modelo é dotado de um conversor A/D interno, responsável por fazer a leitura da saída do LM35DZ e convertendo-a para um número binário, proporcional ao valor lido na entrada analógica e a tensão de referência. Essa tensão de referência pode ser uma tensão inserida no pino V_{REF} do microcontrolador ou a própria tensão utilizada na alimentação do

microcontrolador.

Para demonstrar o modo de funcionamento de um conversor A/D, é mostrado na figura 3.22, um gráfico contendo o exemplo de uma conversão feita por um modelo com resolução de 3 bits. Com essa resolução, o conversor é capaz de gerar apenas 8 valores binários distintos correspondentes à tensão em sua entrada, de 000 a 111, ou seja, de 0 a 7 se convertido para decimal.

Figura 3.22 – Exemplo de conversão A/D

O conversor analógico/digital do microcontrolador utilizado tem uma resolução de 10 bits, o que significa dizer que a tensão analógica é convertida para um número entre 0 e 1023.

No circuito, foi especificada uma tensão de referência de 1 Volt, resultante de um divisor de tensão, onde R_1 tem o valor de 300 Ohms e R_2 1,2 KOhms , sendo que, este último é uma soma de dois resistores em série, um de 1k Ohm e outro de 220 Ohms. Considerando que a tensão de entrada utilizada é a de 5 Volts, o valor da tensão de saída é obtido, substituindo os valores através da equação (3.1).

$$V_{Ref} = \frac{5 \times 300}{300 + (1220)} \approx 0{,}9869\, V \tag{3.2}$$

Com a resolução do conversor e a tensão de referência definidas, tem-se que o valor de cada bit é equivalente a

$$V_{Bit} = \frac{V_{Ref}}{1024 - 1} = \frac{0{,}9869}{1023} \approx 0{,}9647\, mV \tag{3.3}$$

A figura 3.23 apresenta trecho do código inserido na memória do PIC 12F675 responsável pela conversão A/D.

```
#device ADC=10
int16 dado;
(...)
dado=read_adc();
(...)
setup_ADC_ports(sAN0|VSS_VREF);
setup_adc(ADC_CLOCK_DIV_16);
```

Figura 3.23 – Código para Conversão A/D

A primeira linha configura a resolução do conversor como sendo de 10 bits. A segunda, é a declaração da variável que tem a função de armazenar o valor da leitura analógica através da função *'read_adc( )'*. As duas últimas linhas são configurações, sendo que a primeira especifica o pino AN0 como entrada analógica do conversor e a faixa de tensão lida nesta porta que vai da tensão do terra à tensão de referência, ou seja, de 0 a 1 Volt. A última expressão configura o tempo utilizado na conversão A/D. Segundo o D*ata Sheet*, para uma correta conversão, esse período precisa ser de no mínimo 1,6 µs (microssegundos). O termo *'ADC_CLOCK_DIV_16'* informa que o tempo usado na conversão é a frequência utilizada pelo microcontrolador, que é um cristal de 4MHz, dividido pelo decimal 16. Para calcular o período em segundos, é utilizado a equação

$$t = \frac{1}{f} \tag{3.4}$$

onde

$$t = Período\,(em\ segundos)$$
$$f = Frequência\,(em\ Hertz)$$

Inserindo o valor da frequência na fórmula e, considerando que a frequência deve ser dividida por 16, tem-se que

$$t = \frac{1}{\left(\frac{4\times10^{6}}{16}\right)} = \frac{1}{250\times10^{3}} = 0{,}000004\,s = 4\,\mu s \tag{3.5}$$

Sendo assim, percebe-se que o resultado ultrapassa o valor de tempo necessário à conversão e, portanto, satisfaz a condição imposta pelo Data Sheet.

Além da função de conversor analógico, o microcontrolador é utilizado para codificar os dados e enviá-los ao transmissor de radiofrequência através de uma comunicação serial assíncrona implementada por software, tendo em vista que o PIC12F675 não possui USART (*UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER*) interna. A codificação é necessária para que não exista componente DC na transmissão serial dos

bits. A codificação é feita utilizando a codificação Manchester pois, além de manter a componente DC em zero, a implementação de um código de detecção de erro é bem simples. O código presente na figura 3.24 demonstra como a codificação do dado é realizada.

```
  int nibenc1, nibenc2, nibenc3, nibenc4;
  int16 dado;
  int i, bit=15;
  int32 dadoencode;

    for(i=0;i<=15;i++)
    {
    dadoencode<<=2;

     if(bit_test(dado, bit))
     dadoencode=dadoencode|0b10;
     else
     dadoencode=dadoencode|0b01;
     bit--;
    }
nibenc1=make8(dadoencode, 3);
nibenc2=make8(dadoencode, 2);
nibenc3=make8(dadoencode, 1);
nibenc4=make8(dadoencode, 0);
```

Figura 3.24 – Código para Codificação

Como a informação a ser enviada é um valor de 10 bits e na linguagem C não existe variável exatamente deste tamanho, o dado precisou ser alocado a uma variável imediatamente maior, que neste caso, é a de 16 bits (variável dado). Sendo assim, os 6 bits adicionais são preenchidos automaticamente com zeros. Porém, conforme explicado no item 2.4.1, na codificação *Manchester* o bit 1 é uma transição do bit 1 para o 0 e o bit 0, uma transição do bit 0 para o 1, o que é correto dizer que neste método de codificação, 1 bit é transformado em 2 bits. Portanto, o dado de 16 bits precisa ser convertido para outro de 32 bits (variável dadoencode) antes de ser transmitido.

A conversão é feita da seguinte maneira: Os 16 bits da variável dado são verificados um a um, através da função *'bit_test( )'*, partindo do bit 15 até chegar ao bit 0. Caso o bit seja 1, é feita uma operação do tipo OR entre dadoencode e o binário 00000001, mas se o bit for 0, a operação OR é feita entre dadoencode e o binário 00000010. Após a operação, os bits do resultado são deslocados duas posições à esquerda (função <<2). Essas operações são repetidas em todos os bits da variável dado. Na figura 3.25, é mostrado um exemplo onde as operações citadas são executadas em apenas um bit.

```
dado = 1000001011101110
    BITS 15      10      5      0
bit = 15
bit_test(dado, 15) = 1
dadoencode =  00000000000000000000000000000000
              | OR
        0b10 =  00000000000000000000000000000010
              00000000000000000000000000000010
              <<2
dadoencode    00000000000000000000000000001000
```

Figura 3.25 – Exemplo de codificação

Após a conversão, o dado é fragmentado em 4 partes de 8 bits cada (variáveis $nibenc_{núm.}$), também chamadas de octetos, através da função *'make8( )'*. Essa separação é necessária para que o microcontrolador possa enviar o dado ao transmissor TWS-BS para que ele remeta-o remotamente. Antes do envio, é importante transmitir uma série de *strings* exibidas na figura 3.26. Essas *strings* são chamadas de preâmbulo e são mencionadas no item 2.4.1.

```
putc(0x55);
delay_us(1000);
putc(0x55);
delay_us(1000);
putc(0x55);
delay_us(1000);
putc(0x55);
delay_us(1000);
putc(0xff);
delay_us(1000);
putc(0x00);
delay_us(1000);
putc(0xfe);
delay_us(1000);
```

Figura 3.26 – Preambulo

O termo *'putc'* é o comando utilizado para enviar dados (em formato binário) através da comunicação serial do microcontrolador. O número 55 é transmitido quatro vezes em sequência. O termo '0x' que antecede o valor 55, indica que o mesmo está em base numérica hexadecimal. Este valor, ao ser convertido para base numérica binária, torna-se uma sequência de zeros e uns alternados (01010101). Eles têm a função de inicializar o demodulador do receptor. O dado seguinte, 0xff é o byte de sincronia e permite sincronizar o byte de início (*start byte*) que, no caso, é o hexadecimal 'fe' (11111110 em binário). Este último indica que, após o seu recebimento, as próximas sequências representam o dado transmitido. Além dos quatro octetos, o microcontrolador também envia as negações destes

fragmentos ou, conforme é chamado na aritmética binária, o complemento de um. Isto significa que o estado de cada bit é invertido. Por exemplo, caso o byte do dado enviado seja 10011101, o próximo byte a ser remetido é 01100010. Estes bytes extras são partes necessárias do código de detecção de erro desenvolvido e implementado neste projeto com o intuito de verificar a integridade do dado recebido pelo módulo receptor TWS-374 e que é repassado ao microcontrolador PIC18F4550. O *firmware* completo gravado na memória microcontrolador PIC12F675 é apresentado no Apêndice A.

A figura 3.27 mostra uma parcela do código responsável por receber os dados transmitidos.

```
if (dado==0xfe)
 {
 State=1;
 }
 else if(State==1)
 {
 dadoenc[State-1]=dado; //dadoenc[0]
 state=2;
 }
 else if(State==2 && dado==~dadoenc[0])
 {
 dadoenc[State-1]=dado; //dadoenc[1]
 State=3;
 }

(...)

dadoenc=make32(nibble1, nibble2, nibble3, nibble4);
```

Figura 3.27 – Código para Recepção dos Dados

O programa aguarda até que a informação recebida seja o byte de início (0xfe). Ao recebê-lo, o programa então, espera pela chegada do próximo byte, armazena-o e aguarda a chegada do terceiro byte. Quando este chega, o programa executa um teste condicional para verificar se o terceiro byte é a negação do segundo. Se o teste confirmar a veracidade da informação, ele é validado, e o programa espera a chegada dos dois próximos bytes para executar o mesmo teste e, enquanto a condição for atendida, serão armazenados todos os bytes recebidos. Caso o teste detecte um byte falso, todos os outros são descartados e o programa retorna ao começo onde aguarda a chegada do *byte* de início. Se os 4 bytes estiverem corretos, os mesmos são unidos através da função *'make32( )'*, formando novamente um dado de 32 bits para, em seguida, ser decodificado e convertido para o valor original de 16 bits. O código responsável pela decodificação é mostrado na figura 3.28.

```
int bit=31, i;
int16 dadodec=0;

for(i=0;i<=15;i++)
 {
  dadodec<<=1;
  if(bit_test(dadoenc, bit))
  bit_set(dadodec, 0);
  else
  bit_clear(dadodec, 0);
  bit-=2;
 }
```

Figura 3.28 – Código para Decodificação do Dado

A decodificação é feita verificando-se todos os bits ímpares da informação, iniciando-se do último até chegar ao primeiro bit ímpar, lembrando que os bits vão de zero (0) a trinta e um (31). Assim como na decodificação, o bit é checado através da função *'bit_test( )'*. Se o bit for igual a 1, a variável *dadodec* recebe o valor 1 mas se o bit fo 0, a variável recebe 0. Após o recebimento, os bits do elemento *dadodec* avançam uma posição à esquerda e o teste recomeça no próximo bit ímpar imediatamente inferior.

A partir da figura 3.29, pode-se analisar uma amostra da execução do código apresentado na figura 3.28.

```
dadoenc = 10000000000000000000000000000000
    BITS    31        25              15      10      5       0
bit = 31
bit_test(dadoenc, 31) = 1
dadoenc =   00000000000000000000000000000001
dadodec =                       0000000000000001
                                 <<1
dadodec =                       0000000000000010
```

Figura 3.29 – Exemplo de decodificação

Após a decodificação, o microcontrolador realiza o cálculo para achar o valor da temperatura através da fórmula

$$T_{TEMP} = 100\, V_{Bit}\, Num_{A/D} \tag{3.6}$$

onde V_{Bit} é o valor já informado na equação 3.3, $Num_{A/D}$ é o valor de 10 bits retornado pelo conversor analógico/digital e o valor 100 é necessário para se obter a temperatura em graus Celsius. Após o cálculo, o microcontrolador remete o valor da temperatura para o computador através da porta USB, sendo impresso no monitor pelo *software* desenvolvido neste projeto e que é falado no item 3.6. No Apêndice B, é exibido o *firmware* do microcontrolador PIC18F4550.

A figura 3.30 apresenta uma foto com o circuito transmissor. Já a foto do circuito receptor foi apresentada anteriormente, estando presente na figura 3.15.

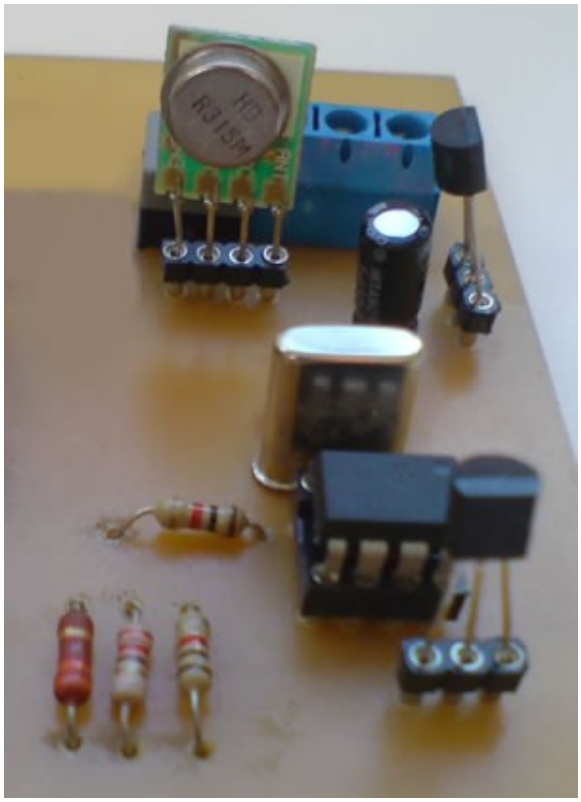


Figura 3.30 – Placa do Circuito Transmissor de Temperatura

3.4 – CIRCUITO DE CONTROLE DO CARRO

O circuito de controle para o carro é utilizado em conjunto com o próprio controle do veículo e, portanto, um está conectado ao outro. Ele foi construído com o intuito de que, o usuário, ao manusear o equipamento, possa controlar o carro diretamente pelo computador, dispensando a utilização dos *joysticks* do controle. Ele foi intitulado de circuito de ativação eletrônica. Ainda assim, o controle original é utilizado para transmitir sinais ao veículo.

Conforme explicitado no item 2.3.3, para que o controle ative uma direção, se faz necessário que um par de contatos elétricos se encontrem, fechando o circuito, conectado a um certo pino de um circuito integrado, enviando pulsos elétricos. Sendo assim, foi necessário desenvolver um método eletrônico para fazer com que estes pares se encostem quando necessário, substituindo o modo manual realizado através dos *joysticks* presentes no controle. Para essa tarefa, chegou-se a conclusão de adicionar uma espécie de chave

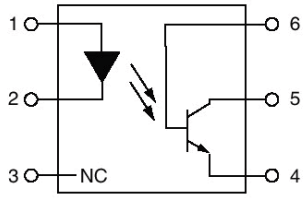
eletrônica entre os pares, abrindo-a sempre que houver a necessidade do encontro entre os pares, acionando a direção desejada. Para exercer a função de chave, a idéia inicial era utilizar relés. Os relés são chaves eletromecânicas utilizadas em circuitos de baixa tensão para acionar outro circuito, geralmente de alta tensão. Ele tem sua construção baseada em um contato metálico que se abre ou fecha sob a influência de um campo eletromagnético induzido numa bobina em seu interior. Assim, quando os contatos da bobina do relé são percorridos por uma corrente elétrica, ele atrai o contato metálico e abre ou fecha o contato, conforme o modelo de relé utilizado. Os relés têm o inconveniente de serem ruidosos.

Após algumas pesquisas, descobriu-se a existência de um componente denominado acoplador óptico, utilizado para possibilitar a transferência de um sinal de controle ou mesmo de um sinal que carrega uma informação, de um circuito para outro, sem a necessidade de acoplamento elétrico entre eles.

O sinal é transferido por um feixe de luz produzido por um emissor, geralmente um LED e, recebido por um sensor, que pode ser desde um fotodiodo até um foto-diac.

Como o contato entre os dois componentes inexiste, o isolamento entre eles é teoricamente infinito. Na prática, há um limite na tensão máxima que pode haver entre os dois elementos sem que haja centelhamento, tipicamente variando entre 2000 e 7000V.

No circuito, são utilizados quatro acopladores ópticos do modelo 4N25, que faz uso de um LED emissor de infravermelho e um fototransistor bipolar como sensor. Conforme consta no seu D*ata Sheet*, o C.I. 4N25 tem algumas aplicações como interruptor de circuitos, relé de estado sólido, interface e acoplamento de sistemas com diferentes potências e impedâncias e interface de entrada/saída. A figura 3.31 ilustra o acoplador óptico 4N25 e os seus respectivos pinos.



PINOS
1. ANODO
2. CATODO
3. NÃO CONECTADO
4. EMISSOR
5. COLETOR
6. BASE

Figura 3.31 – Acoplador Óptico 4N25

A figura 3.32 contém o desenho esquemático do circuito de ativação eletrônica do

controle.

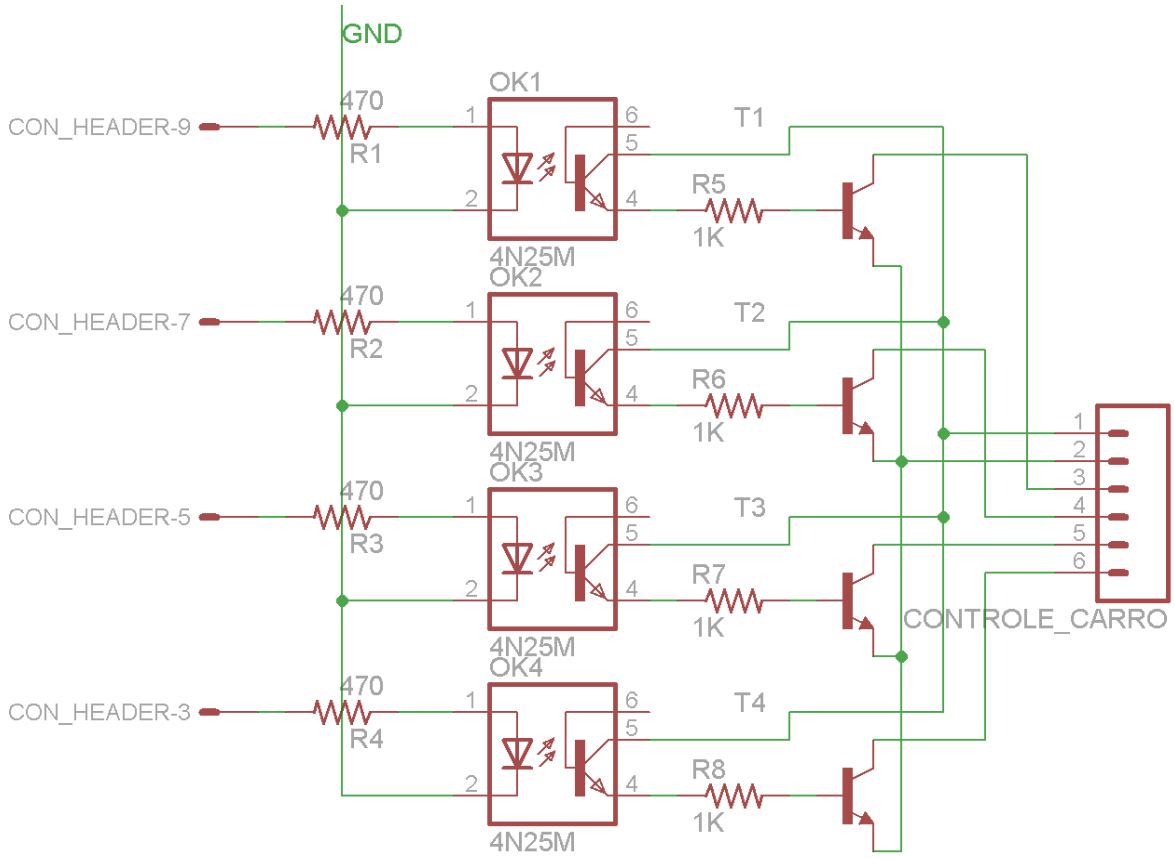


Figura 3.32 – Esquema elétrico do circuito de ativação eletrônica

Analisando o circuito esquemático da figura 3.32, nota-se que ao lado direito existe um conector de 6 pinos. Este conector é responsável por interligar o circuito ao próprio controle do veículo para que o mesmo receba os sinais da placa e ative as direções desejadas. Por isso, foi necessário abrir o controle e soldar alguns fios em sua placa de circuito. A foto do circuito com os fios soldados pode ser vista na figura 3.33.

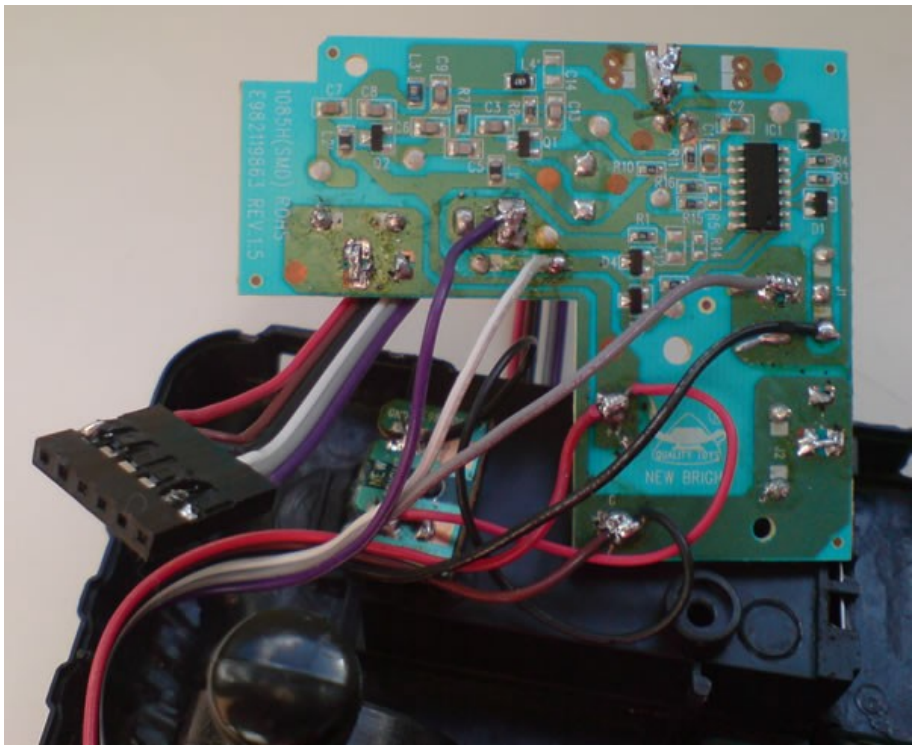


Figura 3.33 – Placa de Circuito do Controle do Carro com Fios Soldados

Foram conectados seis fios ao todo, sendo que dois deles foram ligados à alimentação do veículo, um fio ao VCC, outro ao GND. Os outros quatro restantes, foram soldados aos contatos elétricos que estão em aberto (sem conexão), responsáveis por enviar os sinais de controle ao veículo. Conforme citado no item 2.3.3, para que o veículo possa se movimentar para uma determinada direção, o seu contato elétrico deve ser encostado a outro de forma que o circuito seja fechado. Analisando o controle remoto, pode-se notar que o contato elétrico necessário para fechar o circuito em qualquer umas das quatro direções é o terra.

Conforme citado no item 3.2, o circuito de ativação eletrônica está conectado ao circuito USB através de um cabo com conectores *Headers*. O microcontrolador, ao receber o devido sinal enviado pelo computador, ativa um de seus quatro pinos responsáveis por enviar a corrente necessária para a ativação dos LEDs emissores, passando primeiramente pelos resistores R1, R2, R3 e R4, limitando a corrente para evitar a queima destes LEDs. Os coletores dos fototransistores presentes nos acopladores estão conectados ao VCC do próprio controle do carro. Já os emissores estão conectados à base de transistores BC548 presentes no circuito, passando primeiramente por resistores limitadores de corrente. Nos quatro transistores, cada coletor está ligado a um contato e os emissores estão ligados ao outro contato necessário para o fechamento do circuito que é o terra do circuito do próprio controle do carro. O LED, ao acender, faz com que o fototransistor responda, entrando em

condução, permitindo que a corrente transite entre o coletor e o emissor, ativando, também, o transistor BC548, fazendo com que a ligação entre os pares seja realizada. Caso cesse a corrente no pino, o LED se apaga, fazendo com que o fototransistor pare de conduzir.

A figura 3.34 mostra a placa de circuito impresso construída para este propósito. Esta placa foi construída juntamente com o circuito transmissor responsável por controlar os servomotores, pois ela também se conecta ao circuito USB através do mesmo cabo com conectores *Headers*. O apêndice E contém os desenhos esquemáticos dos três circuitos mencionados, demonstrando as conexões entre elas. O circuito transmissor é explicado com maiores detalhes no tópico 3.5 deste trabalho.

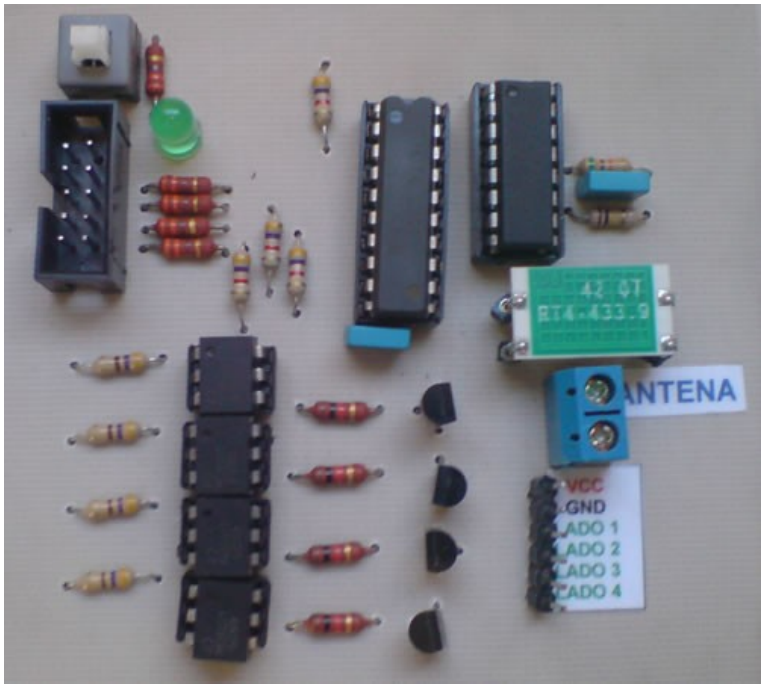


Figura 3.34 – Placa de Circuito de Ativação Eletrônica e de Controle dos Servos

3.5 – CIRCUITO PAN & TILT E LOCALIZADOR SONORO

Este circuito tem duas funções distintas:

a) controlar remotamente a microcâmera a partir de uma aplicação *Pan & Tilt;*

b) emitir sons por meio de um *buzzer* (buzina) presente no circuito.

A aplicação *Pan & Tilt* tem o intuito de movimentar a câmera, ampliando o campo visual do ambiente a ser monitorado. A função do *buzzer* é guiar o usuário, por intermédio da audição, até o veículo para evitar a perda do mesmo nos casos em que o usuário não

souber exatamente onde ele se encontra, devido a problemas que possam vir a ocorrer, como falta de bateria ou perda do sinal por parte da microcâmera ou do carro. Esta última aplicação foi chamada de localizador sonoro.

O termo *Pan & Tilt* tem origem nas palavras inglesas *Panorama* e *Tilt*. Ao traduzi-las de forma técnica, significa dizer que o sistema possui capacidade de movimentação panorâmica (giro no eixo horizontal) e a capacidade para inclinar-se (giro no eixo vertical). [SOARES, 2006]

Para realizar a tarefa de movimentação, foi preciso a utilização de dois motores. Um motor que realizasse o movimento no eixo horizontal e outro para o eixo vertical. Inicialmente, a intenção era utilizar motores de passo para a função, porém, após a realização de pesquisas, optou-se pelos servomotores por serem menores, mais leves e possuírem boa potência.

Conforme mencionado no item 2.4 do capítulo 2, os servomotores precisam receber um sinal de largura de pulso (PWM) com certa duração para se movimentarem. O fabricante dos servomotores utilizados no projeto não especificou os valores da duração da largura de pulso.

Como o circuito da aplicação *Pan & Tilt* está instalada no carro rádio controlado e, o circuito com a função de controlar a aplicação deve estar conectada ao computador, é necessário que haja uma comunicação sem fio entre o controle e o circuito a ser controlado. Portanto, são utilizados um módulo transmissor e um módulo receptor de sinais via rádio frequência.

O componente do circuito transmissor é o módulo híbrido RT4-433 fabricado pela empresa Telecontrolli. Para o seu funcionamento, ele deve ser alimentado por uma tensão entre 2 e 14 Volts. A figura 3.35 apresenta o módulo e a sua pinagem.

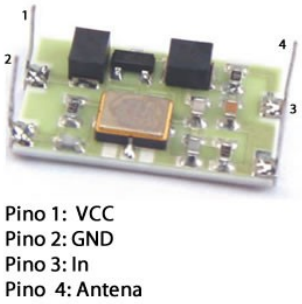
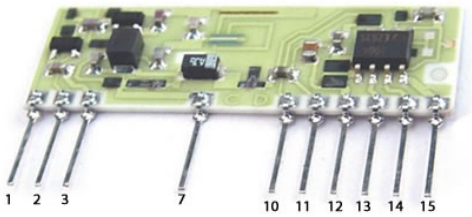


Figura 3.35 – Módulo Transmissor RT4
Fonte: Data Sheet – RT4, Telecontrolli

Para o circuito receptor, é utilizado o módulo RR3-433, também da Telecontrolli. Diferentemente do transmissor, este módulo deve ser alimentado por uma tensão entre 4,5 e 5,5 Volts. O número 433 em ambos os módulos significam que eles trabalham nesta faixa de frequência. Na figura 3.36, apresenta-se o módulo RR3 com a descrição dos seus pinos.



Pino 1: RF +VCC
Pino 2: RF -GND
Pino 3: In
Pino 7: RF -GND
Pino 10: AF +VCC
Pino 11: AF -GND
Pino 12: AF +VCC
Pino 13: Test Point
Pino 14: Out
Pino 15: AF +VCC

Figura 3.36 – Módulo Receptor RR3
Fonte: Data Sheet – RR3, Telecontrolli

O módulo RT4-433 apresenta uma largura de banda para a transferência de dados de 4 KHz, porém, ele deve operar com uma largura de banda de até 2 KHz (podendo ser menor), visto que essa é a largura de banda do módulo receptor RR3-433.

Para o correto funcionamento do circuito do transmissor, é necessário a utilização de um C.I codificador e, para o circuito receptor, um C.I. decodificador, de modo que se possa transmitir os dados digitalmente.

A codificação é feita pelo C.I. MC145026 e a decodificação, pelo C.I. MC145027, ambos da Motorola, cuja pinagem dos mesmos, encontram-se disponíveis na figura 3.37.

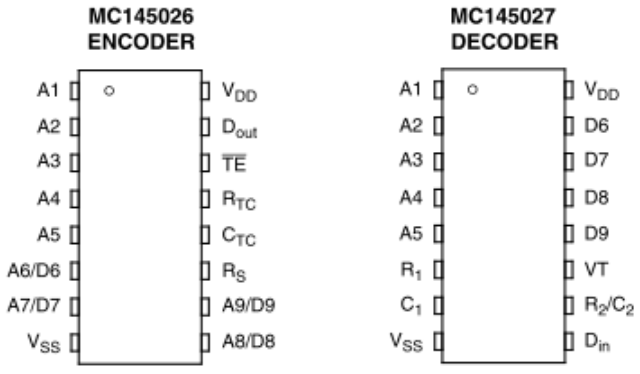


Figura 3.37 – Pinagem dos CIs MC145026 e MC145027
Fonte: Data Sheet – MC145026 – MC145027, Motorola

De acordo com o D*ata Sheet*, o MC145026 codifica nove linhas de informação (pinos A1 a A9/D9). Elas podem funcionar em modo trinário (alto, baixo ou aberto) ou binário (alto

ou baixo). Essa codificação é realizada através de pulsos. Um dígito lógico 0 (baixo) é codificado em dois pulsos curtos consecutivos, um lógico 1 (alto) como dois pulsos longos consecutivos e, em aberto (alta impedância), é codificado em um pulso longo seguido por um pulso curto.

Neste C.I., existem dois modos de operação para as linhas de informação: pode-se utilizar todos os 9 pinos para endereçamento ou, apenas os 5 primeiros, sendo os 4 restantes (pinos *A6/D6* a *A9/D9*) usados como entrada de dados. Este endereçamento é necessário e deve ser o mesmo tanto no C.I. codificador quanto no C.I. decodificador. Para o correto funcionamento do circuito, é preciso utilizar os pinos *A6/D6* a *A9/D9* como entrada de dados (sinais de controle), portanto, somente os 5 pinos iniciais (*A1* a *A5*) são responsáveis pelo endereço do codificador. Além do mais, o decodificador utilizado possui, apenas, os mesmos 5 pinos para endereçamento sendo que, os outros 4 pinos (D6 a D9), são saídas de dados. O C.I. codificador possui outro pino de entrada chamado de *TE* (*Transmit Enable* ou Transmissão Ativa). Quando ele é levado a nível baixo (conectado ao terra), a transmissão é iniciada. Antes da codificação, é realizada a multiplexação das informações (endereço e dados), pois as mesmas chegam ao codificador por meio das 9 entradas paralelas e as mesmas são enviadas serialmente como se fosse uma única palavra. A informação é transmitida duas vezes em sequência para garantir que a mesma seja recebida corretamente. O dado é emitido através do pino de saída pino D_{OUT} (pino 15) para o pino *DATA* do transmissor RF, sendo enviado pelo ar por meio da antena. Caso *TE* vá a nível alto e a segunda palavra tenha sido transmitida, o codificador é automaticamente desabilitado até que seja levado a nível baixo novamente.

O CI MC145027 realiza os processos de verificação da validade do endereço, a decodificação e demultiplexação[2] da palavra codificada recebida. A informação transmitida consiste de duas palavras idênticas que são interceptadas, primeiramente, pelo receptor RF por intermédio de sua antena, sendo transferida logo a seguir, da saída (pino *OUT*) do módulo até a linha D_{in} (pino 9), que é o pino de entrada de dados do decodificador. As palavras recebidas estão contidas dentro de um sinal de onda quadrada, na mesma frequência utilizada pelo transmissor.

Ao chegar ao decodificador, o dado é recebido pelo bloco denominado extrator de dados, responsável por separar os pulsos do endereço dos pulsos dos dados. Assim, o endereço extraído é comparado com o endereço configurado no decodificador. Caso os dois endereços sejam iguais, os quatro bits de dados são armazenados no decodificador. Ao

2 Seleciona qual(is) saída(s) deve(m) receber a informação presente na única entrada do decodificador.

receber a segunda palavra, o seu endereço também é verificado e, sendo o mesmo do primeiro endereço, então uma nova comparação é realizada, desta vez entre os quatro bits de dados da segunda sequência com os quatro bits de dados já armazenados da primeira sequência. Caso as palavras sejam idênticas, os dados são disponibilizados nos pinos de saída (pinos 12 a 15) do decodificador e a saída *VT* (*Valid Transmission*) vai a nível alto para indicar que um novo conjunto de dados acabou de ser disponibilizado na saída do decodificador. Essa saída *VT* permanece em nível alto até que um erro seja recebido ou até que nenhum sinal seja recebido por quatro períodos de dado. No circuito, esta saída está conectada a um LED para indicar visualmente quando a mesma está em nível alto ou baixo. Os dados, ficam disponíveis nas saídas paralelas do decodificador até que novos dados os substituam ou até quando for retirada a alimentação do decodificador.

Logo abaixo, seguem as figuras 3.38 e 3.39, ilustrando o esquema elétrico do circuito transmissor e do circuito receptor, respectivamente.

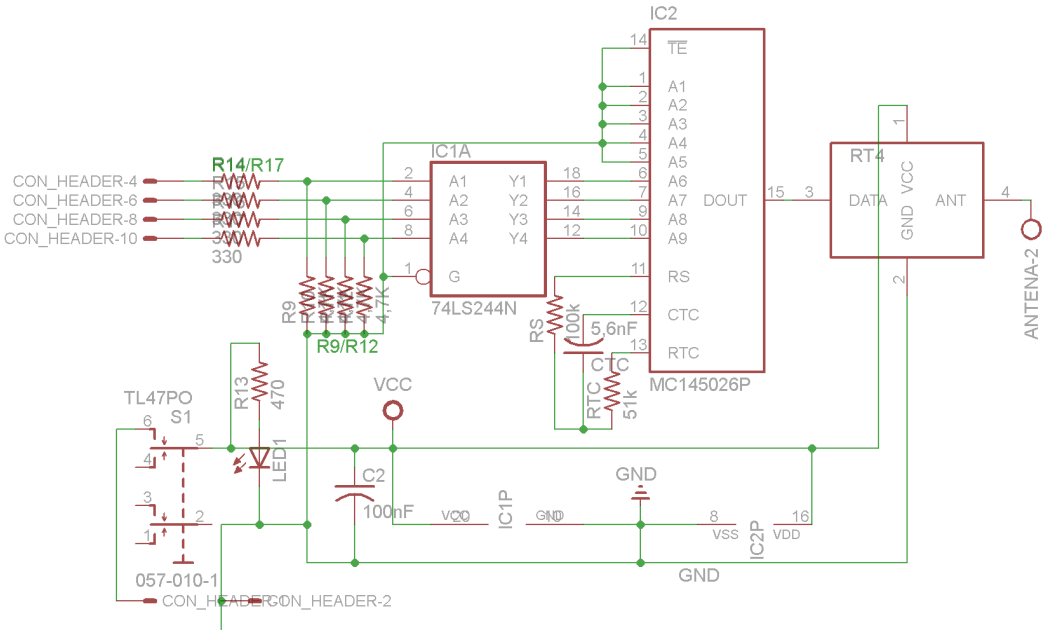


Figura 3.38 – Esquema Elétrico do Circuito do Transmissor

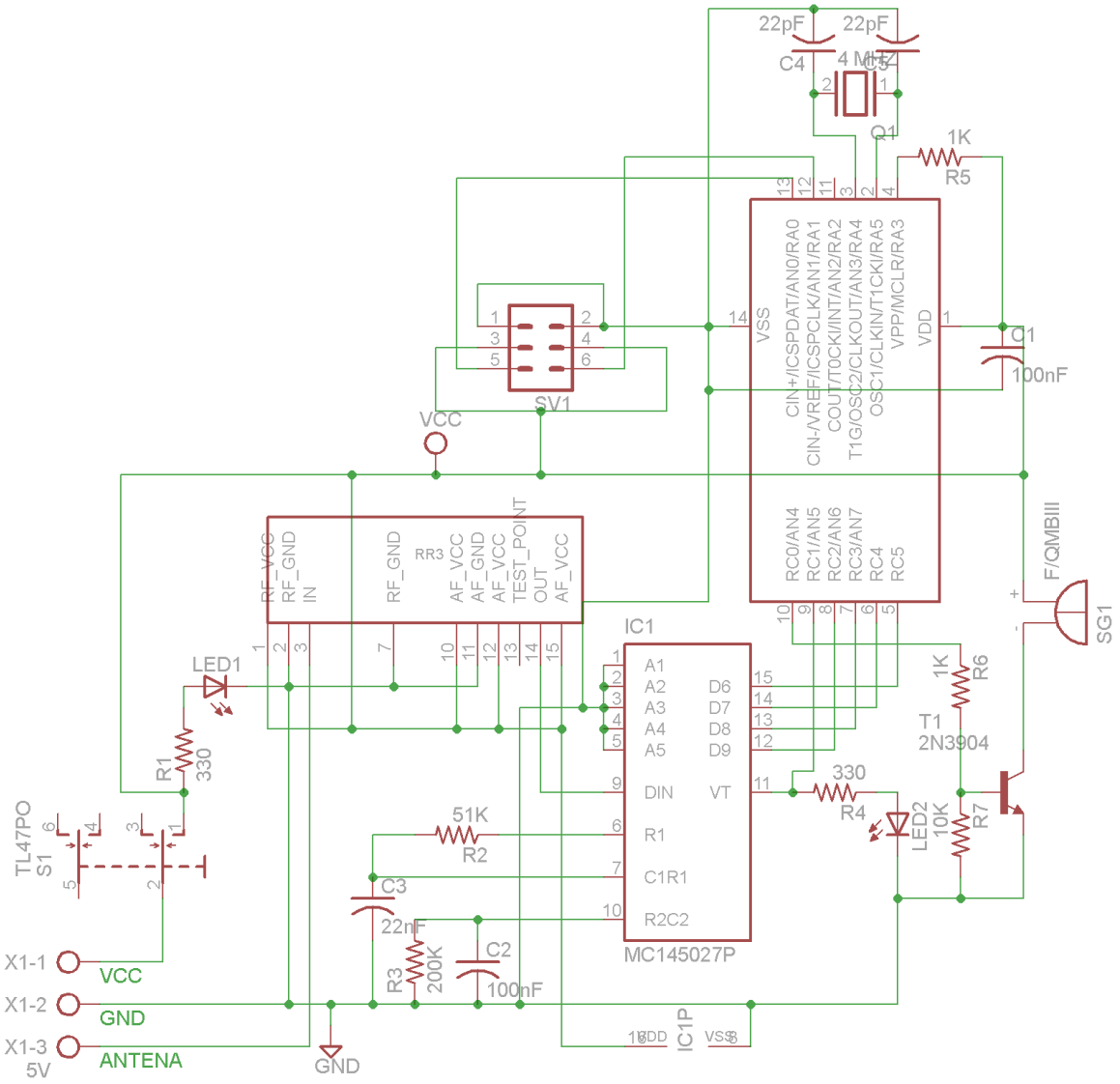


Figura 3.39 – Esquema Elétrico do Circuito do Receptor

Analisando os esquemas das figuras 3.38 e 3.39, pode-se verificar que os pinos *'A1'* a *'A5'* dos componentes MC145026 e MC145027 estão conectados ao terra do respectivo circuito, o que significa dizer que ambos possuem o mesmo endereço.

Além do endereço, o codificador e o decodificador devem oscilar na mesma frequência para que possam se comunicar corretamente. O C.I. MC145026 é dotado de um oscilador que opera a uma frequência baixa para obter um sinal de onda quadrada. Essa frequência é determinada por uma rede RC externa, composta pelos componentes R_S, C_{TC} e R_{TC}. Conforme mencionado anteriormente, a frequência deve ser igual ou menor que 2 KHz, uma vez que este é o valor da largura de banda do circuito receptor de RF e, por isso, o transmissor deve necessariamente utilizar este valor como parâmetro. Para calcular o valor

da frequência ou dos valores dos componentes da rede RC, deve-se utilizar a equação disponibilizada no D*ata Sheet*:

$$f_{OSC} \approx \frac{1}{2.3\, R_{TC} C_{TC}'} \tag{3.7}$$

onde

$$C_{TC}' = C_{TC} + 20\text{pF}$$
$$R_S \approx 2\, R_{TC}$$
$$R_S \geqslant 20\, k$$
$$R_{TC} \geqslant 10\, k$$

No MC152027, para que o extrator de dados obtenha a mesma frequência do codificador, são conectados um resistor e um capacitor às linhas *R1* e *C1* (pinos 6 e 7), respectivamente, responsáveis por determinar quando um pulso curto ou longo é recebido. Ao receber os pulsos do endereço e os pulsos de comandos ou dados, eles são identificados pela linha *R2/C2* (pino 10), onde um resistor e um capacitor são conectados em paralelo, utilizados para detectar o fim de uma palavra recebida e, também, o fim da transmissão.

Para descobrir os valores dos componentes citados acima, são utilizadas as fórmulas 3.8 e 3.9:

$$R_1 C_1 = 3{,}95\, R_{TC} C_{TC} \tag{3.8}$$

$$R_2 C_2 = 77\, R_{TC} C_{TC} \tag{3.9}$$

onde

$$R_1 \geqslant 10\, k\,\Omega$$
$$C_1 \geqslant 400\ pF$$
$$R_2 \geqslant 100\, k\,\Omega$$
$$C_2 \geqslant 700\ pF$$

No D*ata Sheet*, consta uma tabela com alguns exemplos de valores que podem ser utilizados tanto no MC145026, quanto no MC145027. A mesma, é reproduzida na tabela 2.6:

Tabela 3.1 – Exemplos de valores R/C

f_{osc} (kHz)	R_{TC}	$C_{TC'}$	R_S	R_1	C_1	R_2	C_2
362	10 k	120 pF	20 k	10 k	470 pF	100 k	910 pF
181	10 k	240 pF	20 k	10 k	910 pF	100 k	1800 pF
88.7	10 k	490 pF	20 k	10 k	2000 pF	100 k	3900 pF
42.6	10 k	1020 pF	20 k	10 k	3900 pF	100 k	7500 pF
21.5	10 k	2020 pF	20 k	10 k	8200 pF	100 k	0.015 µF
8.53	10 k	5100 pF	20 k	10 k	0.02 µF	200 k	0.02 µF
1.71	50 k	5100 pF	100 k	50 k	0.02 µF	200 k	0.1 µF

Fonte: Data Sheet – MC145026 – MC145027, Motorola

Consultando a tabela 2.6, consta apenas um exemplo que poderia ser utilizado com estes módulos RF que é a frequência de 1,71 KHz. Porém, esta faixa utiliza alguns componentes com valores não padronizados, o que dificulta o encontro dos mesmos no mercado. Sendo assim, estes componentes foram substituídos por outros com valores próximos. Substituindo estes valores na equação 3.10 tem-se que:

$$f_{OSC} \approx \frac{1}{2.3 \times 51.10^{3} \times 5,6.10^{-9}} \qquad f_{OSC} \approx 1,52\ KHz \qquad (3.10)$$

onde

$R_{TC} = 51\ K$

$C_{TC}' = 5600\ pF$

O valor da frequência encontrada é menor do que 2 KHz, portanto, os valores citados podem ser utilizados para os respectivos componentes.

O circuito transmissor está conectado ao circuito USB. Este último, tem a função de enviar os dados de comandos ao primeiro, bem como, prover alimentação ao mesmo, através da porta USB do computador. Os dados, ao serem enviados para o circuito transmissor chegam ao C.I. 74LS244 antes de serem repassados ao codificador. Este C.I. é um *buffer* não inversor de oito entradas e oito saídas *tri-state*. Ele é utilizado para proteção, deixando o sinal trafegar apenas em uma direção, não permitindo que ele retorne ao microcontrolador do circuito USB.

Os resistores *R14* a *R17* tem a função de limitar a corrente que chega ao codificador. Já os resistores *R9* a *R12* são resistores de *pull-down* para evitar que essas linhas fiquem em estado flutuante (alta impedância).

Analisando o circuito receptor, percebe-se que as 4 linhas de dados do decodificador

(pinos *D6* a *D9*), mais o pino de saída *VT*, estão diretamente conectadas a 5 pinos de entrada (pino *RC1* a *RC5*) do microcontrolador PIC16F676. Sua pinagem é apresentada na figura 3.40.

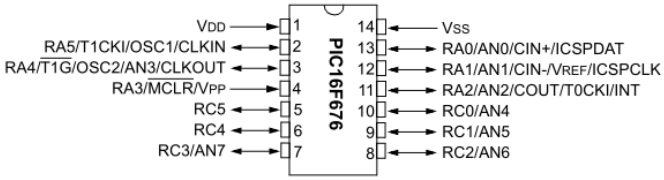


Figura 3.40 – Microcontrolador PIC16F676
Fonte: Data Sheet – PIC16F676, Microchip

Este microcontrolador tem duas funções:

a) fornecer os sinais de largura de pulso (PWM), responsáveis por movimentarem os servomotores;

b) acionar a buzina do localizador após determinado período de tempo.

Os sinais de largura de pulso são disponibilizados em dois pinos de saídas (pinos *RA0* e *RA1*) do microcontrolador. Cada saída, por sua vez, está conectada a entrada de sinal de um servomotor.

Das 4 entradas, 2 são responsáveis pelos movimentos de um servomotor e as outras 2, pelos movimentos do outro servomotor. Como exemplo, a figura 3.41 apresenta o código responsável por emitir os sinais de largura de pulso apenas para um dos servomotores já que ambos os códigos são semelhantes.

```
if ((input(pin_c5))&&(i>650))
{
output_high(pin_A0);
delay_us(i);
output_low(pin_A0);
delay_ms(120);
i=i-50;
}

if ((input(pin_c4))&&(i<2400))
{
output_high(pin_A0);
delay_us(i);
output_low(pin_A0);
delay_ms(120);
i=i+50;
}
```

Figura 3.41 – Trecho do Código de Controle dos Servos

No código, a variável *i* representa a largura do impulso. Quando o pino *C5* recebe um sinal alto do decodificador e a variável *i* está com valor acima de 650, faz com que o microcontrolador emita o sinal de PWM (de tamanho *i*) para o respectivo servomotor, fazendo com que ele gire em sentido horário. Após a emissão do sinal, a variável *i* é decrescida do valor 50. Essas operações se repetem e param somente, quando a entrada (pino C5) retorna ao nível baixo ou até esgotar o limite de giro do servo, neste caso, enquanto *i* for maior que 650. Apesar do sistema de controle do servomotor monitorar o sinal em intervalos de 20 milissegundos, no presente código, este intervalo é de 120 milissegundos para retardar a velocidade do servomotor, tendo em vista que utilizando o primeiro intervalo, o servo realiza a trajetória completa de 180° em menos de 1 segundo. Essa mesma explicação equivale para a segunda função condicional. As diferenças em relação ao anterior são o pino de entrada (C4), o limite da variável *i* como condição e o fato de que ela, desta vez, é acrescida do valor 50 e, por fim, o sentido que o servomotor executa.

O localizador sonoro presente no circuito tem o seu funcionamento relacionado diretamente com a conexão do pino *VT* do decodificador ao pino *RC1* do microcontrolador mencionada anteriormente. O que ocorre é que mesmo que nenhum sinal de comando seja acionado através do circuito transmissor, o codificador está sempre enviando dados já que a entrada *TE* está conectada permanentemente ao terra. Com o decodificador recebendo dados ininterruptamente, a saída *VT* se mantém sempre em nível alto. Caso o circuito transmissor seja desligado ou o circuito receptor saia da área de alcance do sinal, o decodificador deixa de receber os dados codificados, fazendo com que o pino *VT* vá para nível baixo. O microcontrolador, ao deixar de receber o sinal, inicia uma contagem que dura aproximadamente 5 segundos e, se ao final deste período, o decodificador não receber um dado válido, o pino *RC0* entra em estado alto, acionando o transistor que funciona como chave eletrônica, ativando, assim, a buzina. Se após o acionamento da mesma, o decodificador receber um dado genuíno, a buzina é automaticamente desativada e a contagem é zerada. A figura 3.42 mostra o código responsável pela ativação da buzina.

```
if(input(pin_c1))
 {
 j=0;
 output_low(pin_c0);
 }

 //Comando para ativar localizador sonoro
 if (input_C() == 0 && ++j>100000)
 {
 output_high(pin_c0);
 delay_ms(500);
 output_low(pin_c0);
 delay_ms(500);
 }
 }
```

Figura 3.42 – Código de Ativação da Buzina

A variável *j* é responsável pelo tempo de espera para a ativação da buzina seja ativada. Sempre que o decodificador recebe um dado válido e *VT* vá a nível alto, a variável *j* é zerada. Assim que *VT* e os pinos *RC2* a *RC5* vão a nível baixo, é iniciada uma função que verifica duas condições: a primeira certifica se todos os pinos *RC* estão em nível alto e, a segunda, se a variável *j* é maior que 100.000. Além das verificações, a variável *j* recebe o incremento de mais um ao seu valor e a função se repete enquanto uma das condições não for satisfeita. Essa repetição dura cerca de cinco segundos, até quando *j* receber o valor de 100001, tornando as duas condições positivas e iniciando, assim, as linhas de código presentes dentro da função. Essas linhas fazem com que a saída *RC0* vá a nível alto e nível baixo em intervalos de 0,5 segundo. Durante o período que a saída *RC0* permanece em nível alto, faz com que o transistor ligado a ela entre em condução, ativando, consequentemente, a buzina e desligando-a, quando o transistor para de conduzir. O código completo do *firmware* inserido no microcontrolador PIC16F676 é apresentado no Apêndice C.

Logo abaixo, segue a figura 3.43, contendo a imagem do circuito receptor presente no carro. A foto do circuito transmissor está presente na figura 3.34, já mostrada anteriormente.

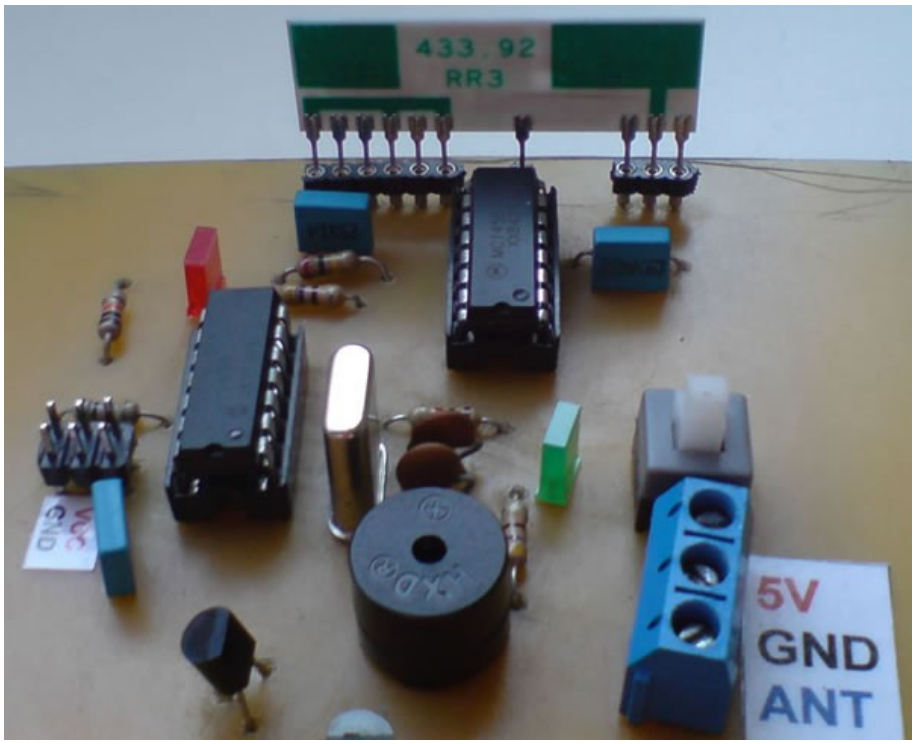


Figura 3.43 – Placa do Circuito Receptor e Buzina

3.6 – PROGRAMA

Para imprimir na tela, a temperatura enviada pelo microcontrolador PIC18F4550 e controlar o carro e a câmera através do teclado do computador, foi necessário o desenvolvimento de uma aplicação capaz de executar estas funções. A mesma foi desenvolvida utilizando a linguagem C++ através da ferramenta Borland C++ Builder versão 6.0.

O C++ Builder é um ambiente de desenvolvimento visual orientado a objetos conhecido como RAD, sigla para Rapid Application Development, ou Desenvolvimento Rápido de Aplicações se traduzido para o português. Ele permite a utilização de uma programação orientada a eventos que são ações do usuário. Essas ações, determinam o próximo procedimento a ser executado, dependendo da escolha e necessidade por parte do programador.

Os eventos podem ser um clique no botão do mouse, o pressionamento de uma tecla ou ao soltá-la, o redimensionamento da janela, a seleção de um item de uma lista, etc. Na programação orientada a eventos, os programas são diferentes da programação tradicional, onde o programa inicia-se no seu disparo, percorrendo todo o programa e se encerrando

após a execução da última linha do código. Neste estilo de programação, os programas aguardam o acontecimento de determinados eventos para que possam executar as ações pré-definidas no programa. Após processar a resposta a um determinado evento, o sistema volta ao estado de espera até que ocorra outro evento, sendo terminado apenas em resposta ao evento de encerramento ou caso de falha.

Conforme citado no item 3.2, o circuito USB se comunica com o computador como se fosse um dispositivo conectado a uma porta serial do computador. Sendo assim, o programa desenvolvido, além de ter a necessidade de ativar e desativar uma comunicação serial, também precisa enviar e receber dados por intermédio de uma porta COM.

Para a programação dessas funções, foi utilizada a API (*Application Programming Interface* ou Interface de Programação de Aplicativos) do *Windows*. Ela é mais conhecida como *WinAPI* e é uma interface de interação entre a aplicação e serviços do sistema operacional voltada para funções (conjunto de funções) de controle do sistema como, por exemplo, criação de janelas, criação de menus, envio de comunicação entre janelas ativas, manipulação de *threads*/processos, manipulação de arquivos, suporte gráfico para desenho, gerência de memória, etc.

A figura 3.44 apresenta o trecho do código responsável pela abertura da Porta Serial.

```
hCom = CreateFile(
    NomePorta,
    GENERIC_READ | GENERIC_WRITE,
    0,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
         );
```

Figura 3.44 – Código para Abertura da Porta COM

A função *CreateFile( )* cria ou abre um volume para o fluxo de dados entre o aplicativo e um dispositivo virtual ou físico que, no projeto, é a porta serial (virtual). Esta função retorna um identificador para se ter acesso ao dispositivo. Ela aceita vários parâmetros, sendo o primeiro, *NomePorta*, um ponteiro apontando para o nome da porta, por exemplo COM1, COM3, etc. O parâmetro *GENERIC_READ* | *GENERIC_WRITE* especifica o tipo de acesso a porta serial sendo, neste caso, o modo genérico de leitura e escrita. O terceiro parâmetro, quando igual a zero (0), indica que a porta serial não deve ser compartilhada com outro aplicativo, ou seja, enquanto o aplicativo está utilizando determinada porta, nenhum outro programa tem permissão para utilizá-la. O quarto parâmetro é um ponteiro e deve ser nulo.

O parâmetro *OPEN_EXISTING* indica que somente um dispositivo já existente pode abrir a porta. Os outros dois restantes devem ser nulos. Já a variável *hCom*, é um identificador responsável por armazenar todas as informações necessárias a respeito do dispositivo aberto, neste caso, a Porta Serial.

A função *CloseHandle( )* utiliza um indentificador como argumento para fechar um objeto aberto ou criado pela função *CreateFile( )*. Neste caso, o identificador é a variável *hCom* e o objeto, a porta serial. A figura 3.45 apresenta a linha utilizada no aplicativo para fechar a comunicação serial.

```
CloseHandle(hCom);
```

Figura 3.45 – Código de Fechamento da Porta COM

As funções *GetCommState( )* e S*etCommState( )* trabalham juntas e são responsáveis por, respectivamente, obter e alterar as configurações da Porta Serial encontradas na estrutura *DCB* (Device Control Block) que é utilizada para definir todos os parâmetros de comunicação.

A figura 3.46, contém a parte do código do programa responsável pelas configurações da Porta Serial.

```
if(!GetCommState(hCom, &dcb))
 {
  return false;
 }
dcb.BaudRate = StrToInt(Form1->ComboBoxBps->Text);
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

if( SetCommState(hCom, &dcb) == 0 )
 {
  return false;
 }
return true;
```

Figura 3.46 – Configuração da Porta COM

A função *GetCommState( )* obtêm os parâmetros atuais armazenados na UART. Ela recebe o identificador da porta serial (*hCom*) e o endereço da estrutura DCB, responsável por salvar as configurações da porta serial.

Para alterar as configurações, deve-se primeiramente definir os novos parâmetros. O parâmetro *BaudRate* recebe o valor da taxa de transmissão. O *ByteSize*, especifica o valor de bits transmitidos ou recebidos por byte. O parâmetro *Parity* especifica o esquema de

paridade e o *StopBits*, a quantidade de bits de parada (stop bits). No aplicativo desenvolvido, apenas a taxa de transmissão pode ser alterada através da interface do aplicativo, enquanto as outras três restantes foram pré-configuradas com um valor único já que as mesmas não tem influência no projeto em si.

Após as alterações, utiliza-se a função *SetCommState( )*, onde ela recebe um identificador do objeto, o qual é setado com os novos valores e um endereço de onde serão retirados os valores de configuração, ou seja, da estrutura *DCB*.

A função responsável por ler dados que chegam através da Porta Serial é denominada *ReadFile( )*. Para escrever dados, utiliza-se a função *WriteFile( )*. Para que ambas funcionem corretamente, é necessário que a função *CreateFile( )* tenha retornado um identificador válido associado a Porta Serial especificada.

A função *ReadFile( )* aceita 5 argumentos, sendo o primeiro, o identificador da Porta Serial (*hCom*). O segundo argumento é um *buffer*, em outras palavras, um lugar temporário na memória onde os dados são armazenados. No programa, ele é chamado de *BufferRead* e o seu tamanho é especificado no terceiro argumento (*LEN_BUFFER*).

O parâmetro *ReadBytes* , é uma variável do tipo ponteiro, onde a função *ReadFile( )* armazena a quantidade exata de bytes lidos. O último é um ponteiro não utilizado, portanto, o argumento é *NULL* (nulo). A figura 3.47 apresenta trecho do código responsável pela leitura de dados na porta serial.

```
const unsigned short LEN_BUFFER = 100;
DWORD ReadBytes;
char BufferRead[LEN_BUFFER];

(...)

if(ReadFile(hCom, BufferRead, LEN_BUFFER, &ReadBytes, NULL) != 0 )
  {
   if(ReadBytes > 0)
   {
    Form1->MemoRX->SetSelTextBuf(BufferRead);
    Form1->Memo1->Text = DateTimeToStr(Now());
    }
```

Figura 3.47 – Leitura de Dados na Porta COM

As três primeiras linhas são declarações das variáveis já explicadas acima. A cada chamada à função *ReadFile( )*, os dados lidos da Porta Serial são armazenados na variável *BufferRead*, e a quantidade total de bytes são armazenados na variável *ReadBytes*. Se *ReadFile( )* retornar um valor igual a zero (0), significa que houve erro. Caso o retorno seja

diferente de zero, é verificado se a variável *ReadBytes* é maior que zero (*ReadBytes* > *0*). Caso a condição seja verdadeira, significa dizer que a Porta Serial recebeu um dado sendo, o mesmo, impresso automaticamente na tela do aplicativo sem a necessidade de qualquer comando por parte do usuário, juntamente com a data e a hora do recebimento. É importante ressaltar que este dado recebido é o valor da temperatura medida transmitida pelo circuito explicado no item 3.3 e que as informações de data e hora são retiradas da configuração do sistema operacional do computador que está fazendo uso do programa, então é importante que o mesmo esteja com estes dados atualizados.

Assim como *ReadFile( )*, a função *WriteFile( )* aceita os mesmos 5 argumentos. A cada chamada da função, os dados armazenados no *buffer* são transmitidos para a porta serial. Caso ocorra algum erro após a chamada, o valor zero (0) é retornado. Diferentemente da leitura de dados recebidos, que é feita automaticamente, a aplicação envia dados a partir de eventos pré-estabelecidos em sua programação. Estes eventos dependem da ação do usuário e ocorrem ao pressionamento de uma tecla (função *onKeyDown*) ou, ao soltá-la (função *onKeyUp*), fazendo com que o programa verifique se a mesma corresponde a alguma tecla informada em suas linhas de código. As teclas pré-definidas são as setas, responsáveis pelo controle do carro e as teclas *'Z'*, *'X'*, *'S'* e *'C'* que controlam os movimentos da câmera. Caso a tecla pressionada corresponda com uma das citadas, o programa envia determinado dado para a porta serial. Ao soltar a tecla pressionada, outro dado é enviado.

A figura 3.48 apresenta as funções *onKeyDown* e *onKeyUp* presentes no programa e que são responsáveis por detectarem as ações do usuário ao pressionar ou soltar uma tecla.

```
void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key,
TShiftState Shift)
{
//Código
}

void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
TShiftState Shift)
{
//Código
}
```

Figura 3.48 – Funções de Teclas

Na figura 3.49, é apresentado o código responsável por enviar dados através da porta serial.

```
bool FlagEnviaDados;
DWORD WriteBytes;
int SizeString;
char BufferWrite[LEN_BUFFER];

(...)

if (Key == VK_LEFT)
{
ch=1;
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}

(...)

if(FlagEnviaDados == true)
  {
  WriteFile(hCom, BufferWrite, SizeString, &WriteBytes, NULL);
  FlagEnviaDados = false;
  }
```

Figura 3.49 – Envia Dados pela Porta COM

O termo *FlagEnviaDados* é uma variável booleana criada para avisar quando há algum dado pronto para ser transmitido. A variável *WriteBytes* é um ponteiro, responsável por retornar a quantidade de *bytes* escritos. A terceira variável (*SizeString*), informa à função *WriteFile( )* o tamanho do *buffer* de dados (*BufferWrite*).

A primeira expressão de condição verifica se a tecla 'seta esquerda' do teclado foi pressionada. Em caso positivo, a variável 'ch' recebe a *string* a ser enviada. Esta *string* é convertida para uma variável do tipo char e copiada em *BufferWrite e o seu* tamanho é calculado por *SizeString. A* variável *FlagEnviaDados* recebe o valor que indica a existência de dado pronto para ser enviado, tornando a segunda expressão de condição verdadeira, executando assim, a função que envia o dado. Após o envio, *FlagEnviaDados* recebe o seu valor inicial, indicando que não há nenhum dado a ser enviado.

A versão completa do código-fonte do aplicativo é apresentado no Apêndice D. A figura 3.50 exibe a interface da aplicação desenvolvida.

Figura 3.50 – Interface gráfica do programa desenvolvido

Percebe-se pela tela do aplicativo que há dois conjuntos de setas que indicam as direções de controle da câmera e do carro. Quando o usuário pressiona uma das teclas de controle, a respectiva seta altera a sua cor, indicando a direção acionada, retornando a cor original quando a tecla é solta.

3.7 – TESTES E RESULTADOS

Com a finalização das placas de circuitos e do programa detalhados no presente capítulo, foi iniciada a montagem final do protótipo. A figura 3.51 apresenta foto do veículo com os seus componentes acoplados ao mesmo.

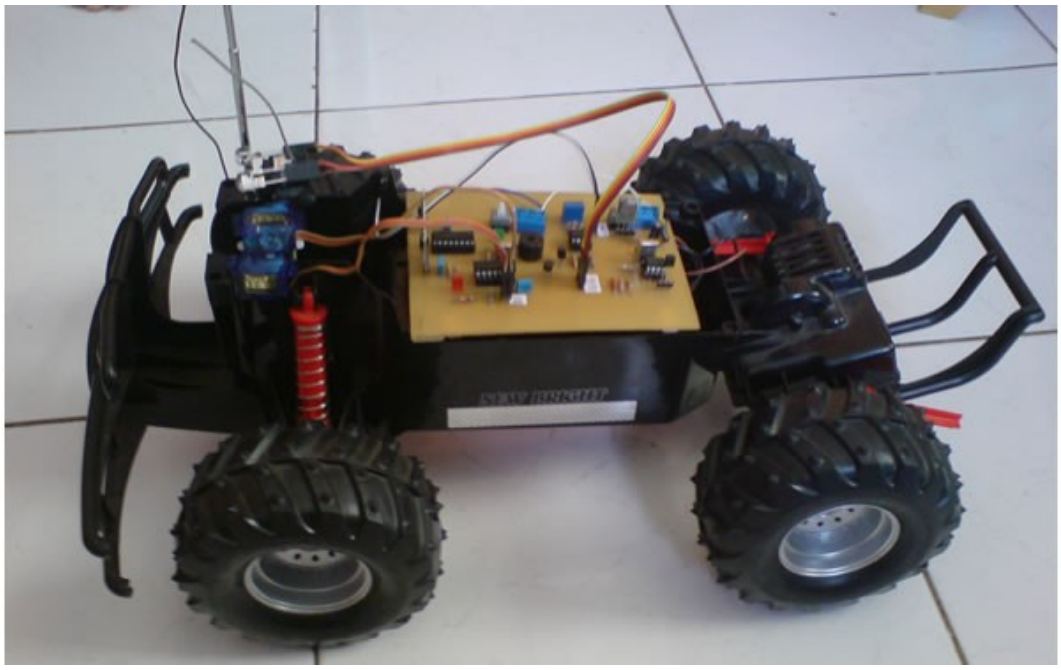


Figura 3.51 – Protótipo

A placa de circuito impresso presente no veículo está representada no Apêndice F através dos circuitos esquemáticos. Na figura 3.52, é exibida foto contendo as placas e dispositivos conectados ao computador utilizado no projeto

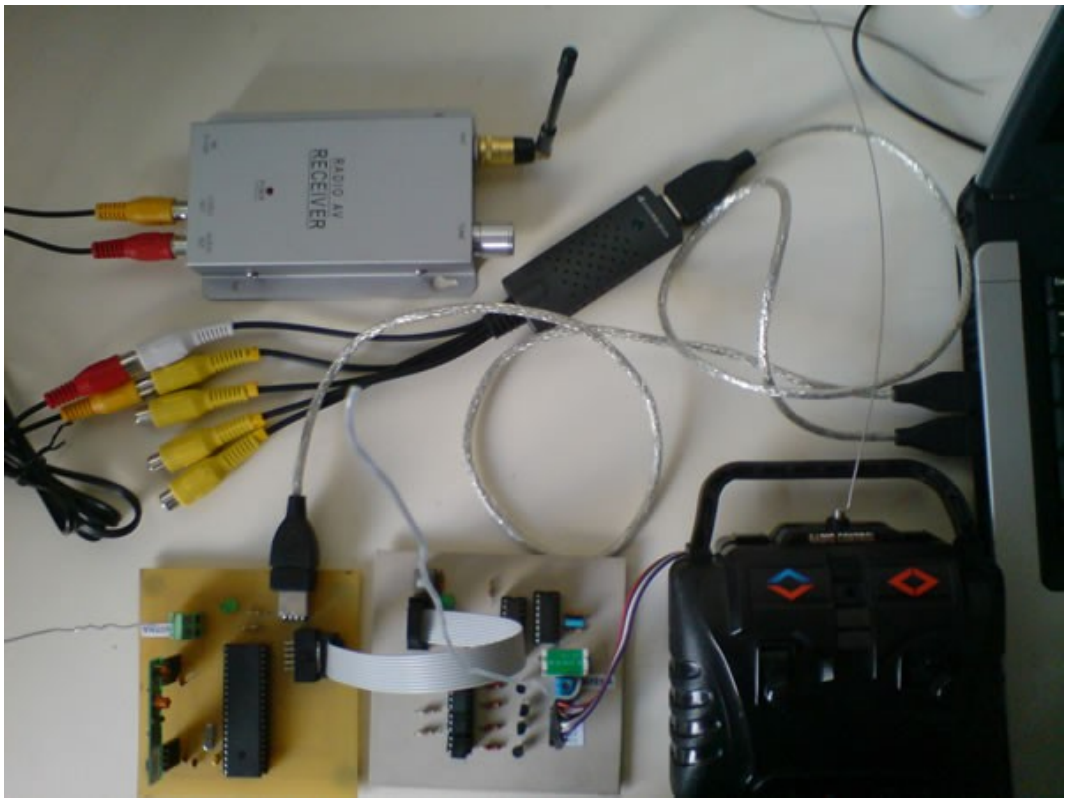


Figura 3.52 – Circuitos e dispositivos conectados ao computador

Após a montagem do protótipo e com as conexões efetuadas no computador, foram iniciados os testes para averiguar o comportamento do sistema.

Os testes consistiram em verificar a distância máxima de alcance de transmissão dos itens que se comunicam através da radiofrequência. Estes itens são o veículo, o circuito medidor de temperatura, a microcâmera e o circuito responsável pelo controle da mesma. Estes testes foram realizados no pilotis de um edifício residencial e o veículo sendo movimentado em linha reta.

Partindo do marco zero que é o veículo próximo ao computador até uma distância aproximada de 40 metros, todos os itens funcionaram perfeitamente. Ao ultrapassar essa distância, o receptor do veículo deixou de receber os sinais vindos do controle. Isso já era esperado, tendo em vista que o manual do veículo especifica essa como sendo a distância máxima de comunicação entre o controle e o veículo. Portanto, para a continuidade dos testes, foi necessário que uma pessoa transportasse o veículo de modo a verificar a comunicação dos outros itens a distâncias maiores que 40 metros. Com aproximadamente 60 metros, o circuito de controle dos servos não estava mais recebendo sinais do seu transmissor, inclusive, ativando o localizador sonoro presente no circuito, indicando que o seu funcionamento está de acordo com o planejado. A essa distância, a microcâmera e o circuito de temperatura ainda conseguiam transmitir para os seus respectivos receptores. O receptor do circuito medidor de temperatura parou de receber dados quando o seu transmissor alcançou cerca de 85 metros. Por fim, o computador parou de receber as imagens da microcâmera quando a mesma atingiu uma distância aproximada de 150 metros.

Após o término dos testes de alcance dos transmissores, foi executado um último teste para verificar o funcionamento do circuito farol, detalhado no item 3.1. O veículo era guiado, partindo de um lugar iluminado até outro local sem luminosidade com o intuito de que o circuito acionasse os LEDs. Após o acionamento, o veículo era novamente controlado, retornando ao primeiro ambiente de forma que o circuito desligasse os LEDs. O funcionamento ocorreu conforme o esperado. Porém, apesar dos LEDs proporcionarem uma luminosidade satisfatória tendo como referência o olho humano, eles não foram suficientes para atender a câmera perfeitamente. Ocorre que no escuro, a câmera requer uma iluminação mais forte do que o esperado. Com os LEDs utilizados, a câmera só consegue captar objetos e paredes quando estão a apenas alguns centímetros de distância. Para corrigir esse problema, seria necessário aumentar a quantidade de LEDs ou substituí-los por outros modelos mais fortes.

3.8 – CUSTO ESTIMADO DO PROJETO

A tabela 3.2 representa uma estimativa do investimento efetuado no projeto.

Tabela 3.2 – Valor Estimado do Projeto

ITENS	VALOR (Em Reais)
Notebook	R$ 1.200,00
Câmera sem fio	R$ 70,00
Carro de Controle Remoto	R$ 99,00
Servomotores	R$ 65,00
Microcontroladores	R$ 40,00
Módulos Receptores e Transmissores (2 Pares)	R$ 60,00
Placa de Captura de Áudio e Vídeo	R$ 30,00
Demais Componentes Eletrônicos	R$ 30,00
Gravador PICBurner	R$ 23,00
Total do Custo Estimado	R$ 1.617,00

Fonte: o Autor

Considerando os valores mostrados na tabela 3.2, pode-se perceber que o item Notebook possui preço bem elevado se comparado aos demais itens registrados na tabela. Como o autor do projeto já possuía este item, pode-se afirmar que o mesmo obteve um gasto estimado de aproximadamente R$ 417,00 (quatrocentos e dezessete Reais). Além disso, este item pode ser suprimido facilmente por outra pessoa que tenha interesse em desenvolver o projeto, já que o notebook pode ser substituído por qualquer outro computador e, atualmente, grande parte das residências possui pelo menos um computador.

CAPÍTULO 4 – CONCLUSÃO

Neste projeto, foi criado um sistema a partir de um veículo e uma microcâmera, capaz de percorrer e monitorar ambientes diversos. O sistema pode assistir aos usuários que necessitam inspecionar ambientes insalubres ou de difícil acesso. Pode, também, vistoriar ambientes comuns, auxiliando pessoas com dificuldades para se locomover.

O trabalho foi concluído e todos os objetivos traçados inicialmente foram alcançados com sucesso. O único problema existente, foi o fato de os LEDs utilizados como faróis não conseguirem fornecer a luminosidade suficiente para que a câmera pudesse obter uma visualização aceitável no escuro. Entretanto, este fator não prejudicou o modo de funcionamento do circuito explicado no item 3.1.

Um fator importante a ser destacado, é que todo o projeto foi elaborado utilizando-se componentes de baixo custo, sendo assim, um produto bastante viável economicamente.

O desenvolvimento ajudou a reforçar, na prática, assuntos diversos vistos em sala de aula e, também, foi possível adquirir novos conhecimentos. Só para citar alguns, o aprendizado do modo de funcionamento de diversos circuitos integrados, a programação de um aplicativo com interface gráfica usando a linguagem C++ e, também, fazendo uso de tecnologias que estão em evidência e são largamente utilizadas que são a transmissão de dados por intermédio de uma comunicação sem fio e a utilização da porta USB. A inclusão e o uso dessas novas tecnologias foram importantes, já que contribuíram para a criação de um sistema atual e não obsoleto, tornando-o mais flexível, pois além de dispensar a utilização de fios, permitiu a conexão do sistema a qualquer computador portátil dotado de porta USB e uma bateria, como os *notebooks* ou *netbooks*, tornando todo o sistema com mobilidade ainda maior.

Não houve muitas dificuldades durante o desenvolvimento do projeto. Apenas dois fatores demandaram bastante paciência e tempo:

a) A quantidade exaustiva de testes feitos com os servomotores, utilizando diversos valores de tempos em milissegundos (ms) para conseguir com que os mesmos se movimentassem a uma velocidade satisfatória, pois os valores normais fazem com que eles girem muito rapidamente, saindo de um extremo e chegando ao outro em menos de 1 segundo;

b) A confecção das placas de circuito impresso foram feitas manualmente, sendo

necessário a furação e a soldagem dos componentes um a um.

Este trabalho permite diversas inclusões e melhorias, assim, qualquer aluno poderia dar segmento ao seu próprio projeto a partir deste.

O projeto foi desenvolvido para pequenas distâncias, devido a limitações existentes entre os quatro transmissores de dados via radiofrequência presentes no sistema. Como eles possuem diferentes potências, no que se refere a área de alcance de transmissão do sinal, o sistema torna-se dependente do transmissor com menor alcance, tendo em vista que a perda de comunicação entre qualquer um dos transmissores com o seu respectivo receptor comprometerá o correto funcionamento da aplicação. Essa limitação, porém, não influi no trabalho em si já que ele é a demonstração de uma idéia e a substituição destes transmissores por outros mais potentes seria suficiente para eliminar o problema. Essa é uma sugestão para projetos futuros. Uma outra sugestão, seria elaborar estudos mais aprofundados visando o melhor tipo de antena a ser utilizado, bem como, o seu comprimento de forma a explorar a capacidade máxima provida pelos módulos transmissores presentes nesta aplicação. Outra melhoria que pode ser aplicada, seria regular a velocidade do veículo utilizado de modo a diminuí-la, tendo em vista que o mesmo possui uma velocidade alta para os fins deste projeto.

Além disso, para os alunos que tiverem interesse em dar continuidade a este trabalho, poderão incluir outros sensores de acordo com a necessidade do usuário ou do local a ser monitorado. Como exemplo, pode-se citar os sensores de umidade ou proximidade. A instalação de uma célula fotovoltaica, aproveitando uma fonte de energia luminosa para o recarregamento das baterias presentes no veículo e a inclusão de um dispositivo capaz de coletar materiais que devem ser analisados pelo próprio usuário são melhorias bem interessantes.

REFERÊNCIAS BIBLIOGRÁFICAS

1. AXELSON, Jan. **Serial Port Complet: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems** – 2. ed. Estados Unidos da América: Lakeview. Research, 2007.

2. AXELSON, Jan. **USB Complete – The Developer's Guide** - 4. ed, Estados Unidos da América: Lakeview. Research, 2009.

3. BRAIN, Marshall. **How the Radio Spectrum Works.** 01 de abril de 2000. . Disponível em: <http://electronics.howstuffworks.com/radio-spectrum.htm> Acesso em 15/12/2009.

4. FOROUZAN, Behrouz **A. Comunicação de Dados e Redes de Computadores**, Ed. São Paulo, Bookman, 2006.

5. KERNIGHAN, Brian W. e RITCHIE, Dennis. **The C Programming Language**. 2ª ed., New Jersey, Prentice Hall PTR, 1988.

6. LAYTON, Julia. **How Remote Controls Works.** 10 de novembro de 2005. Disponível em: <http://electronics.howstuffworks.com/remote-control.htm> Acesso em 18/12/2009.

7. LIMA Jr., Almir Wirth. **Rede de Computadores – Tecnologia e Convergência das Redes**. Editora Alta Books, 2009.

8. MORAZ, Eduardo. **Curso Essencial de Hardware**. São Paulo, Digeraty Books, 2006.

9. PATZKO[1], Luís Fernando. **Aplicações e Funcionamento de Sensores**. 18 de dezembro de 2006. Disponível em: <http://www.maxwellbohr.com.br>. Acesso em 12/02/2010.

10. PATZKO[2], Luís Fernando. **Montagem da Barra de LEDS**. 18 de dezembro de 2006. Disponível em: <http://www.maxwellbohr.com.br>. Acesso em 12/02/2010.

11. SAIBAMAIS.ORG. Disponível em: <http://www.saibamais.org/Personal-Computer> Acesso em 05/03/2009.

12. SANTOS, André. **Servomotores**. Agosto de 2007. Disponível em: <http://www.sumoderobos.org>. Acesso em 05/08/2009.

13. SOARES, Márcio José. Controle Pan&Tilt Mecatrônica Fácil para Câmera de Segurança. **Revista Mecatrônica Fácil 31**, 2006.

14. SOUZA, David J,. **Desbravando o PIC**. Editora Érica, 2003.

15. STROUSTRUP, Bjarne. A History of C++: 1979-1991. **ACM SIGPLAN Notices**, vol. 28, março de 1993, pp.271-297.

16. TITTEL, Ed. **Redes de Computadores** – Coleção Shaum, Editora Bookman, 2002.

APÊNDICE A – *FIRMWARE* GRAVADO NO PIC12F675

```
#include <12F675.h>
#device ADC=10
#fuses XT,NOWDT,NOPROTECT,PUT,BROWNOUT
#use delay(clock=4000000)
#use rs232(baud=1200, xmit=PIN_A2, parity=N, bits=8)

int nibenc1, nibenc2, nibenc3, nibenc4;
int16 dado;

void temperatura()
{
dado=read_adc();
}

 void encodado()
 {
 int i, bit=15;
 int32 dadoencode;
 nibenc1=0;
 nibenc2=0;
 nibenc3=0;
 nibenc4=0;
 delay_ms(500);

   for(i=0;i<=15;i++)
   {
   dadoencode<<=2;

     if(bit_test(dado, bit))
     dadoencode=dadoencode|0b10;
     else
     dadoencode=dadoencode|0b01;
     bit--;
   }
 nibenc1=make8(dadoencode, 3);
 nibenc2=make8(dadoencode, 2);
 nibenc3=make8(dadoencode, 1);
 nibenc4=make8(dadoencode, 0);
 }

 void senddado()
 {
 putc(0x55);
 delay_us(1000);
 putc(0x55);
 delay_us(1000);
```

```
  putc(0x55);
  delay_us(1000);
  putc(0x55);
  delay_us(1000);
  putc(0xff);
  delay_us(1000);
  putc(0x00);
  delay_us(1000);
  putc(0xfe);
  delay_us(1000);
  putc(nibenc1);
  delay_us(1000);
  putc(~nibenc1);
  delay_us(1000);
  putc(nibenc2);
  delay_us(1000);
  putc(~nibenc2);
  delay_us(1000);
  putc(nibenc3);
  delay_us(1000);
  putc(~nibenc3);
  delay_us(1000);
  putc(nibenc4);
  delay_us(1000);
  putc(~nibenc4);
  delay_us(1000);
  }

void main()
{
setup_ADC_ports(sAN0|VSS_VREF);
setup_adc(ADC_CLOCK_DIV_16);
delay_ms(1000);

  while(1)
  {
  temperatura();
  encodado();
  senddado();
  delay_ms(1000);
  }
}
```

APÊNDICE B – *FIRMWARE* GRAVADO NO PIC18F4550

```
#include <18F4550.h>
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
#use delay(clock=48000000) //configura o cristal de 20MHz para operar
a 48MHz
#include <usb_cdc.h>

#use rs232(baud=1200, rcv=PIN_C7, parity=N, bits=8, ERRORS)
int dado, State=0;
char  dado_usb;
float temp;

int16 decodado(byte nibble1, byte nibble2, byte nibble3, byte nibble4)
{
int bit=31, i;
int16 dadodec=0;
int32 dadoenc;
dadoenc=make32(nibble1, nibble2, nibble3, nibble4);

  for(i=0;i<=15;i++)
  {
    dadodec<<=1;
    if(bit_test(dadoenc, bit))
    bit_set(dadodec, 0);
    else
    bit_clear(dadodec, 0);
    bit-=2;
  }
    return (dadodec);
}

void rda_isr()
{
int dadoenc[8], encodado1, encodado2, encodado3, encodado4;
int16 dadodecode;

dado=getc();

if (dado==0xfe)  // É o byte de start
   {
   State=1;
   }
  else if(State==1)
  {
  dadoenc[State-1]=dado; //dadoenc[0]
  state=2;
  }
```

```
else if(State==2 && dado==~dadoenc[0])
{
dadoenc[State-1]=dado; //dadoenc[1]
State=3;
}
else if(State==3)
{
dadoenc[State-1]=dado; //dadoenc[2]
State=4;
}
else if(State==4  && dado==~dadoenc[2])
{
dadoenc[State-1]=dado; //dadoenc[3]
State=5;
}
else if(State==5)
{
dadoenc[State-1]=dado; //dadoenc[4]
state=6;
}
else if(State==6 && dado==~dadoenc[4])
{
dadoenc[State-1]=dado; //dadoenc[5]
State=7;
}
else if(State==7)
{
dadoenc[State-1]=dado; //dadoenc[6]
State=8;
}
else if(State==8  && dado==~dadoenc[6])
{
dadoenc[State-1]=dado; //dadoenc[7]
State=9;
}
else
{
State=0;
}

 if (State==9) //Após todos os bytes serem recebidos
    {
     State=0;
    encodado1=dadoenc[0];
    encodado2=dadoenc[2];
    encodado3=dadoenc[4];
    encodado4=dadoenc[6];
    dadodecode=decodado(encodado1, encodado2, encodado3, encodado4);
    temp= (float)(dadodecode*1.02*100)/1023;
```

```
    printf(usb_cdc_putc,"\r\n%3.1f", temp);
    }

  }

void rda_isr_usb()
{

dado_usb=usb_cdc_getc();

//Controle do carro
if(dado_usb=='1') //Esquerda
output_high(pin_b5);

if(dado_usb=='2') //Direita
output_high(pin_b7);

if(dado_usb=='3') //Frente
output_high(pin_b4);

if(dado_usb=='4') //Trás
output_high(pin_b6);

if(dado_usb=='5') //Esquerda
output_low(pin_b5);

if(dado_usb=='6') //Direita
output_low(pin_b7);

if(dado_usb=='7') //Frente
output_low(pin_b4);

if(dado_usb=='8') //Trás
output_low(pin_b6);

//Controle da câmera
if(dado_usb=='A')
output_high(pin_b3);

if(dado_usb=='S')
output_high(pin_b0);

if(dado_usb=='D')
output_high(pin_b2);

if(dado_usb=='F')
output_high(pin_b1); //b0

if(dado_usb=='G')
output_low(pin_b3);

if(dado_usb=='H')
output_low(pin_b0);
```

```
if(dado_usb=='J')
output_low(pin_b2);

if(dado_usb=='K')
output_low(pin_b1); //b0
}

void main() {
usb_init_cs();
usb_init();
usb_task();

while(!usb_cdc_connected()) {}
do {

  if (usb_enumerated()) {

if (usb_cdc_kbhit())
{
rda_isr_usb();
}

if (kbhit())
{
rda_isr();
}

}} while (TRUE);
}
```

APÊNDICE C – *FIRMWARE* GRAVADO NO PIC16F676

```
#include <16f676.h>
  #fuses XT,NOWDT,NOPROTECT,PUT,BROWNOUT
  #use delay(clock=4000000)

 int x;
  int16 i, k;
  int32 j;
void main()
{

   i=1500;
   k=1200;
j=0;


   for (x=0;x<20;x++)
    {
   //Comando para centralizar servo 1
    output_high(pin_A0);
    delay_us(1500);
    output_low(pin_a0);
    delay_us(18500);

   //Comando para centralizar servo 2
    output_high(pin_A1);
    delay_us(1200);
    output_low(pin_a1);
    delay_us(18800);
    }

   while(1)
   {
    if ((input(pin_c5))&&(i>650))
    {
    output_high(pin_A0);
    delay_us(i); //650
    output_low(pin_a0);
    delay_ms(120);
    i=i-50;
    }

    if ((input(pin_c4))&&(i<2400))
    {
    output_high(pin_a0);
    delay_us(i);
    output_low(pin_a0);
```

```
   delay_ms(120);
   i=i+50;
   }
   if ((input(pin_c3))&&(k>450))
   {
   output_high(pin_a1);
   delay_us(k);
   output_low(pin_a1);
   delay_ms(120);
   k=k-50;
   }

   if ((input(pin_c2))&&(k<2300))
   {
   output_high(pin_a1);
   delay_us(k);
   output_low(pin_a1);
   delay_ms(120);
   k=k+50;
   }

   if(input(pin_c1))
   {
   j=0;
   output_low(pin_c0);
   }

  //Comando para ativar localizador sonoro
   if (input_C() == 0 && ++j>100000)
   {
   output_high(pin_c0);
   delay_ms(500);
   output_low(pin_c0);
   delay_ms(500);
   }
  }
}
```

APÊNDICE D – CÓDIGO FONTE DO PROGRAMA APRESENTADO NO ITEM 3.6

```
//---------------------------------------------------------------------
---------
#include <vcl.h>
#pragma hdrstop
#include "Program.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
//---------------------------------------------------------------------
---------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
TMultLinha *MultLinha;

const unsigned short LEN_BUFFER = 100;//Define o tamanho do buffer de
recepção de dados.
HANDLE hCom; //identificador da porta serial.
DCB dcb;
COMMTIMEOUTS CommTimeouts;
DWORD WriteBytes;//Retorna a quantidade de bytes escritos.
DWORD ReadBytes;//Retorna a quantidade de bytes lidos.
char BufferRead[LEN_BUFFER]; //Armazena a string a ser lida.
bool AbrirPorta(char *NomePorta);
bool ConfiguraControle(void);
bool ConfiguraTimeOuts(void);
bool FlagConectado = false; //Indica se a porta já está aberta.
bool FlagEnviaDados;

int SizeString; //Calcula o tamanho da string
char BufferWrite[LEN_BUFFER]; //Armazena a string a ser enviada
AnsiString ch;
int i, j;

__fastcall     TMultLinha::TMultLinha(bool     CreateSuspended)     :
TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------
------------
void __fastcall TMultLinha::Execute()
{
  while( !Terminated ) //loop infinito. Vida programa.
  {
  if(FlagConectado == true)
  {
```

```
   if(ReadFile( hCom, BufferRead, LEN_BUFFER, &ReadBytes, NULL) != 0 )
    {
     if(ReadBytes > 0)
     {
      BufferRead[ReadBytes] = NULL;
            Form1->MemoRX->SetSelTextBuf(BufferRead);//Imprime o valor
recebido
      Form1->Memo1->Text = DateTimeToStr(Now());//Imprime data e hora do
recebimento
     }
    }
    if(FlagEnviaDados == true)
    {
    Sleep(10);
    WriteFile(hCom, BufferWrite, SizeString, &WriteBytes, NULL);
    FlagEnviaDados = false;
    }
   else
    {
       Sleep(5); //Processa algo. Necessário para que o programa não
use 100% da CPU.
    }
  }
  }
}
//---------------------------------------------------------------------
---------
bool AbrirPorta(char *NomePorta)
{
 hCom = CreateFile(
       NomePorta,
            GENERIC_READ | GENERIC_WRITE, //Modo Genérico de Leitura e
Escrita
       0, // Acesso Exclusivo ao Dispositivos
       NULL, // sem atributos de segurança
       OPEN_EXISTING, // Dispositivo que já existe
       0, // I/O sem ovelap
       NULL // hTemplate deve ser NULL para comm
     );
 if(hCom == INVALID_HANDLE_VALUE)
 {
  return false;
 }
 return true;
}
//---------------------------------------------------------------------
---------
bool ConfiguraControle(void)
{
 if(!GetCommState(hCom, &dcb))
```

```
  {
    return false;
  }
  dcb.BaudRate = StrToInt(Form1->ComboBoxBps->Text);
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
  dcb.StopBits = ONESTOPBIT;

  if( SetCommState(hCom, &dcb) == 0 )
  {
    return false;
  }
  return true;
}
//---------------------------------------------------------------------
---------
bool ConfiguraTimeOuts(void)
{
  if( GetCommTimeouts(hCom, &CommTimeouts) == 0 )
  {
    return false;
  }

  CommTimeouts.ReadIntervalTimeout = 2;
  CommTimeouts.ReadTotalTimeoutMultiplier = 0;
  CommTimeouts.ReadTotalTimeoutConstant = 2;
  CommTimeouts.WriteTotalTimeoutMultiplier = 5;
  CommTimeouts.WriteTotalTimeoutConstant = 5;

  if( SetCommTimeouts(hCom, &CommTimeouts) == 0 )
  {
    return false;
  }
  return true;
}

//---------------------------------------------------------------------
---------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
ComboBoxCOM->ItemIndex = 0; //Seleciona a porta COM1.
ComboBoxBps->ItemIndex = 2; //Seleciona o item 1200 BAUDS.
FechaPorta->Enabled = false;
}
//---------------------------------------------------------------------
---------
void __fastcall TForm1::AbrePortaClick(TObject *Sender)
{
bool Sucesso;
```

```
  if(AbrirPorta(ComboBoxCOM->Text.c_str()) == true)
  {
    FlagConectado = true;
    Sucesso = ConfiguraControle();
    Sucesso = ConfiguraTimeOuts();
    if(Sucesso == false)
    {
      FlagConectado = false;
      CloseHandle(hCom);
    }else{
      FrmAviso2->ShowModal();
      AbrePorta->Enabled = false;
      FechaPorta->Enabled = true;
      ComboBoxCOM->Enabled = false;
      ComboBoxBps->Enabled = false;
      LedVerde->Visible = true;
      LedVermelho->Visible = false;
      Form1->StatusBar1->Panels->Items[0]->Text = "Conectado";
      MultLinha->Resume(); //Inicia processo.
      }
  }else{
    MessageBeep(0);
    FlagConectado = false;
    FrmAviso->ShowModal();
  }
}
//---------------------------------------------------------------------
---------
void __fastcall TForm1::FechaPortaClick(TObject *Sender)
{
  FechaPorta->Enabled = false;
  AbrePorta->Enabled = true;
  ComboBoxCOM->Enabled = true;
  ComboBoxBps->Enabled = true;
  FlagConectado = false;
  CloseHandle(hCom);
  FrmAviso3->ShowModal();
  Form1->StatusBar1->Panels->Items[0]->Text = "Desconectado";
  LedVerde->Visible = false;
  LedVermelho->Visible = true;
  MultLinha->Suspend();
}
//---------------------------------------------------------------------
---------
void __fastcall TForm1::FormCloseQuery(TObject *Sender, bool &CanClose)
{
MultLinha->Terminate();
MultLinha = NULL;
delete MultLinha;
```

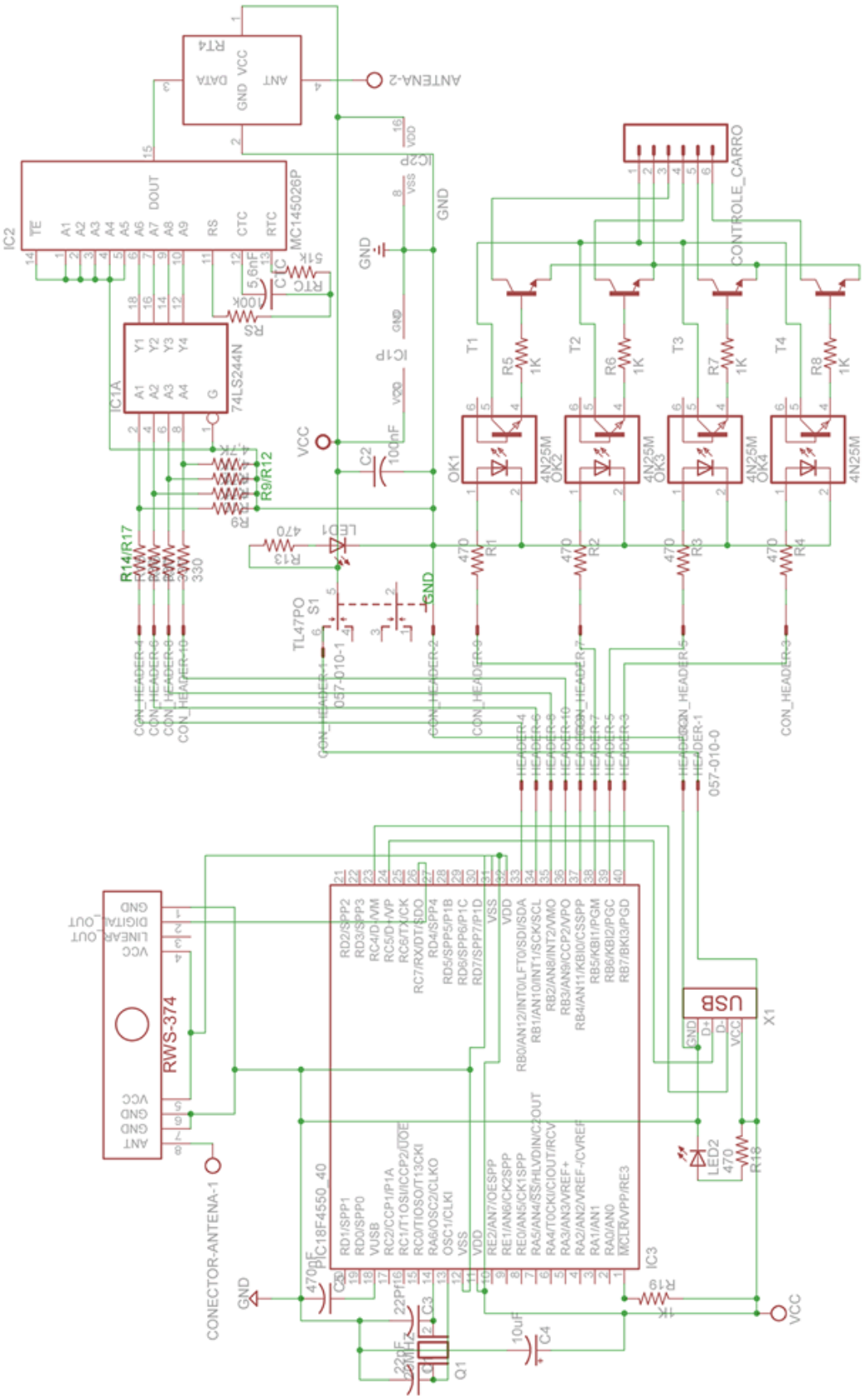
```
if( FlagConectado )
CloseHandle(hCom);
}
//---------------------------------------------------------------------
---------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 memset( BufferRead, 0, LEN_BUFFER);
 MultLinha = new TMultLinha(true); //Aloca memória para o objeto.
 MultLinha->Priority = tpNormal; //Define a prioridade.
}
//---------------------------------------------------------------------
---------
void __fastcall TForm1::FormKeyDown(TObject *Sender, WORD &Key,
   TShiftState Shift)
{
if (FlagConectado == true)
{
if (Key == VK_LEFT)
{
if (i==0)
FlagEnviaDados = true;
Esquerda2->Visible = true;
Esquerda1->Visible = false;
ch='1';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
i++;
}
if (Key == VK_RIGHT)
{
if (i==0)
FlagEnviaDados = true;
Direita2->Visible = true;
Direita1->Visible = false;
ch='2';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
i++;
}
if (Key == VK_UP)
{
if (j==0)
FlagEnviaDados = true;
Cima2->Visible = true;
Cima1->Visible = false;
ch='3';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
j++;
}
if (Key == VK_DOWN)
{
```

```
if (j==0)
FlagEnviaDados = true;
Baixo2->Visible = true;
Baixo1->Visible = false;
ch='4';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
j++;
}
if (Char(Key) == 'Z')
{
if (i==0)
FlagEnviaDados = true;
ch='A';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
Esquerda4->Visible = true;
Esquerda3->Visible = false;
i++;
}
if (Char(Key) == 'X')
{
if (j==0)
FlagEnviaDados = true;
ch='S';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
Baixo4->Visible = true;
Baixo3->Visible = false;
j++;
}
if (Char(Key) == 'C')
{
if (i==0)
FlagEnviaDados = true;
ch='D';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
Direita4->Visible = true;
Direita3->Visible = false;
i++;
}
if (Char(Key) == 'S')
{
if (j==0)
FlagEnviaDados = true;
ch='F';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
Cima4->Visible = true;
Cima3->Visible = false;
j++;
}
```

```
}
}
//---------------------------------------------------------------------
---------

void __fastcall TForm1::FormKeyUp(TObject *Sender, WORD &Key,
      TShiftState Shift)
{
if (FlagConectado == true)
{
i=0;
j=0;
if (Key == VK_LEFT)
{
Esquerda1->Visible = true;
Esquerda2->Visible = false;
ch='5';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Key == VK_RIGHT)
{
Direita1->Visible = true;
Direita2->Visible = false;
ch='6';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Key == VK_UP)
{
Cima1->Visible = true;
Cima2->Visible = false;
ch='7';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Key == VK_DOWN)
{
Baixo1->Visible = true;
Baixo2->Visible = false;
ch='8';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Char(Key) == 'Z')
{
Esquerda4->Visible = false;
Esquerda3->Visible = true;
ch='G';
```

```
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Char(Key) == 'X')
{
Baixo4->Visible = false;
Baixo3->Visible = true;
ch='H';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Char(Key) == 'C')
{
Direita4->Visible = false;
Direita3->Visible = true;
ch='J';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;
}
if (Char(Key) == 'S')
{
Cima4->Visible = false;
Cima3->Visible = true;
ch='K';
strcpy(BufferWrite, ch.c_str());
SizeString= strlen(BufferWrite);
FlagEnviaDados = true;

}
}
}

//---------------------------------------------------------------------
---------
```

APÊNDICE E – CIRCUITOS CONECTADOS DIRETAMENTE AO COMPUTADOR



APÊNDICE F – REPRESENTAÇÃO DA PLACA ACOPLADA AO VEÍCULO