Pontifícia Universidade Católica do Paraná

CCET – Centro de Ciências Exatas e Tecnológicas

Engenharia de Computação

Alex Douglas Fukahori

Arthur Teixeira Brita

Felipe Cornehl

Hélio Pasko Rompkoyski

Projeto Integrado SmartCar

Documentação referente ao Projeto Integrado
desenvolvido no 3º Período do Curso de
Engenharia de Computação da Pontifícia
Universidade Católica do Paraná.
Orientador: Afonso Ferreira Miguel e Gil Marcos Jess.

Curitiba, Junho de 2009

Sumário

1. Introdução

A cada dia, a quantidade de carros está aumentando gradativamente em cidades grandes. Conseqüentemente, os números de acidentes causados pelas colisões nas ruas aumentam também. Pensando nesse problema foi desenvolvido o SmartCar que é um carro que evita colisões através de sensores Infra-Vermelhos, controlado através da porta serial (RS-232) do computador.

1.1 Objetivo

O SmartCar tem como objetivo evitar batidas, possibilitando no futuro ser implementado em automóveis de grande porte, assim, aumentando a segurança do motorista e dos passageiros.

1.2 Descrição do projeto

No projeto foi adquirido um carrinho de controle simples, e acima dele foram embarcados os módulos e os circuitos necessários para fazer o carrinho obedecer aos controles via porta Serial (RS-232) de um computador e parar o carro ao encontrar um obstáculo em frete do mesmo.

Como principal objetivo foi parar o carrinho ao ver um obstáculo em frente dele, para isso, utilizamos um sensor de infravermelho por reflexão, assim, possibilitando a identificação do obstáculo e após a identificação é invertido a corrente do carrinho através de uma ponte H.

1.2.1 Fluxograma do funcionamento

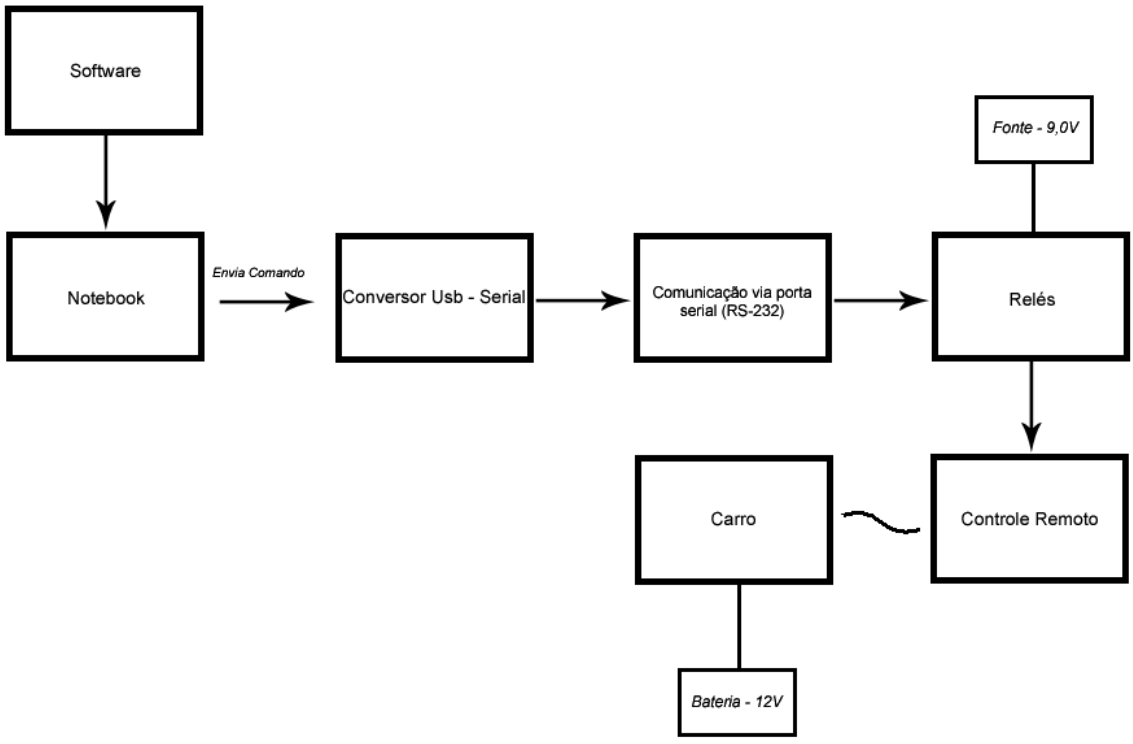

Figura 1.0 – Fluxograma de funcionamento do controle

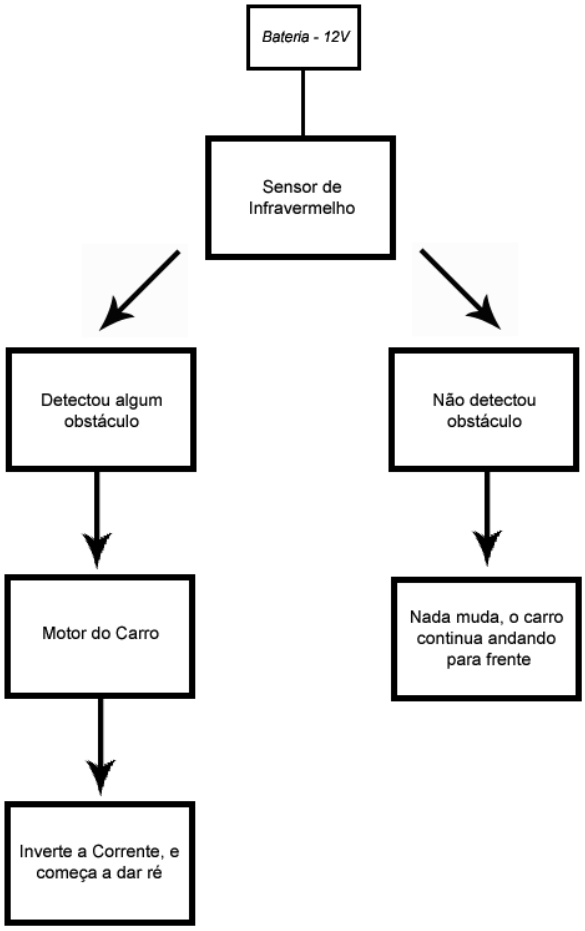

Figura 1.2 – Fluxograma de funcionamento do controle

1.3 Problemas encontrados

- Como inicialmente foi adquirido um carrinho simples, por questão de custos, ele era bem frágil, por esse motivo logo na primeira semana do início do projeto o carrinho foi danificado e irrecuperável;

- Ao iniciar a programação com a classe CSerial foi percebido uma certa dificuldade ao tentar enviar um comando aos circuitos responsáveis na identificação dos comandos;

- Houve certa dificuldade à realizar a programação do software, pois nos computadores do laboratório de engenharia de computação não possuí o Microsoft Visual Studio 2008, e o CSerial só funciona no Visual Studio 2008;

- Como em Notebook's atuais não possuem a porta Serial (RS-232), nos impossibilitou de testar os circuitos apartir do Notebook;

- Por motivos de falta de conhecimento e habilidades na soldagem de componentes, tivemos certas placas com problemas de soldas frias;

- Houve problemas com o sensor Infravermelho durante todo o projeto;

- Ao tentar produzir uma placa de circuito impresso tivemos problemas no manuseio da prensa, pois hora não passava o toner para a placa e hora borrava o toner na placa.

1.4 Métodos de resoluções utilizados

- Após danificar o carrinho adquirimos um carrinho melhor e mais forte;

- A dificuldade de envio do comando aos circuitos era a falta de alguns de limitadores de escopo, que o Visual Studio não estava reconhecendo, esse problema só percebemos após prestar mais atenção;

- O problema de falta do Visual Studio 2008 nos laboratórios foi resolvido, só após, carregar sempre um Notebook na hora da realização do projeto;

- Por causa da falta de porta Serial em Notebooks, tivemos que adquirir um adaptador USB para Serial, assim convertendo uma porta USB em uma porta Serial;

- As soldas frias resolveram-se após verificar melhor as placas e ressoldando-os;

- Foi montado 5 tipos de sensores infravermelho, mas nenhum funcionou corretamente, logo, até o prazo final da entrega do projeto não foi possível resolver esse problema;

- O problema com o manuseio da prensa foi resolvido utilizando um ferro de passar roupa no lugar da prensa, pois assim, podemos controlar a força e verificar melhor á cada intervalo de tempo se o toner passou à placa ou não.

2. Módulos desenvolvidos

2.1 Módulo para comunicação via Porta Serial (RS-232)

Inicialmente foi montado um conversor RS-232 - TTL para possibilitar a comunicação entre o computador e o controle do carrinho. O principal componente utilizado pelo circuito é o circuito integrado (veja o tópico 6) MAX232 que é responsável por converter o sinal recebido de uma porta Serial do computador em um sinal TTL ou CMOS (veja o tópico 6).

Lista de componentes utilizados para o conversor:

1x CI LM7805

1x Capacitor Eletrolítico de 100uF/16V

4x Capacitor Eletrolítico de 1uF/16V

2x Resistor de 1Kohms 1/4W

1x Transistor BC548

1x Conector RS-232 macho

1x CI MAX232

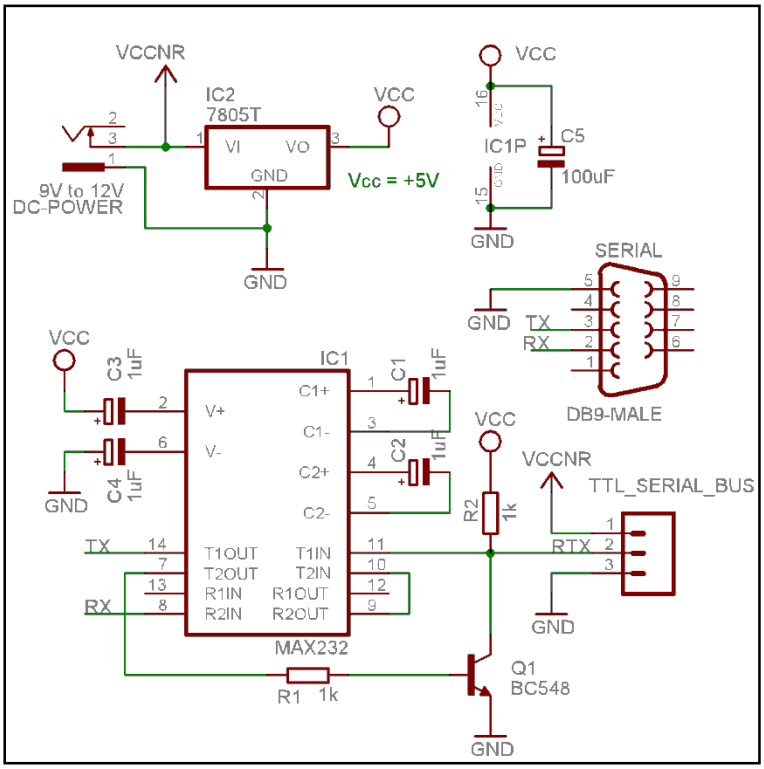

Figura 2.1- Esquemático do conversor RS232 e TTL.

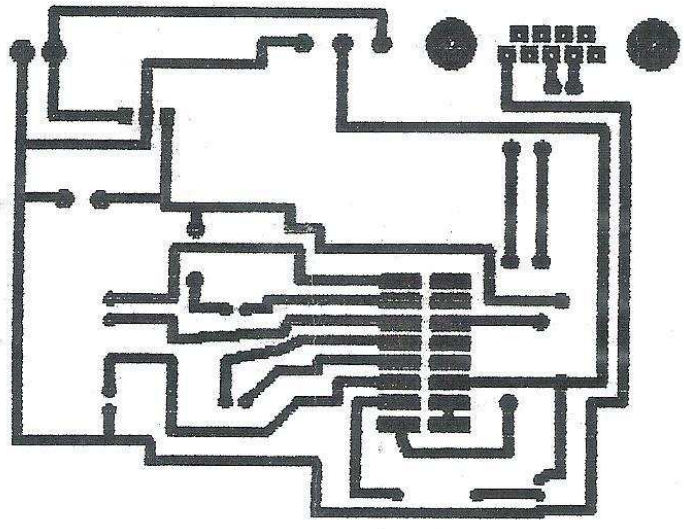

Figura 2.2 – Desenho do circuito Impresso para o Conversor RS232 e TTL.

2.2 Módulo para controlar os Relés

Esse módulo é responsável por processar a comunicação da linha TTL_SERIAL_BUS do módulo anterior, assim, reconhecendo um comando específico dos dispositivos que estará na entrada e saída, onde na saída digitais ligamos ou desligamos os relés (veja o tópico 6) que estarão conectados no controle remoto do carrinho para acionar cada botões de movimento, já nas entradas digitais podemos

verificar quais são os relés ligado e quais estarão desligado. Para isso foi programado um PIC16F675 que processa a comunicação da linha TTL_SERIAL_BUS do módulo anterior.

Lista de materiais necessários:

1x CI 78L05

1x Capacitor Eletrolítico de 100uF/16V

1x Capacitor Eletrolítico de 10uF/16V

1x PIC16F675

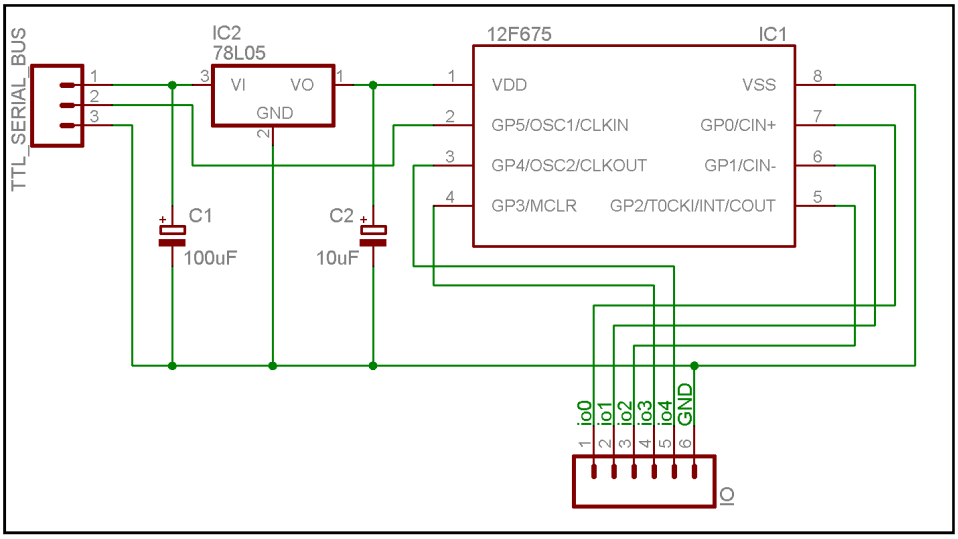

Figura 2.3 – Esquemático do módulo responsável por controlar os Relés.

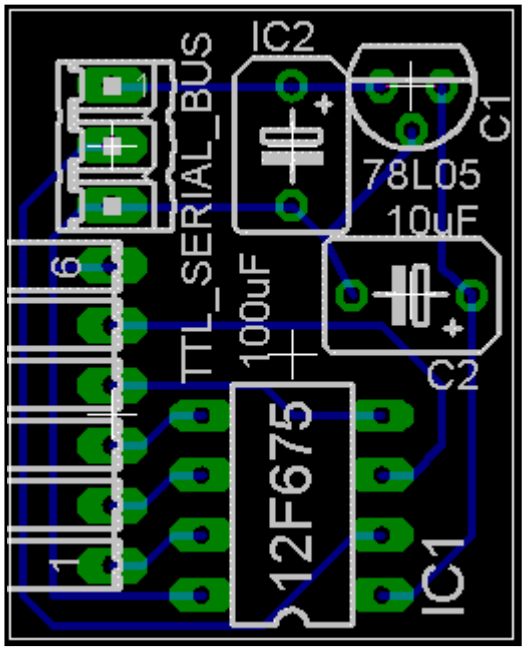

Figura 2.4 – Desenho do circuito impresso do módulo para controlador dos relés.

2.3 Etapa de potência para os relés

A etapa de potência garante que o componente ligado nele seja alimentado corretamente, evitando assim a falta de corrente (veja o tópico 6) ou tensão (veja o tópico 6). No caso desse projeto o componente que precisava ser colocado na etapa de potência foi os relés, como dito no tópico do módulo anterior, eles serão acionados para controlar o carrinho.

Lista de componentes necessária para a montagem da etapa de potência

4x Resistores de 470ohms

4x Relés de 5v

4x Transistores BC548

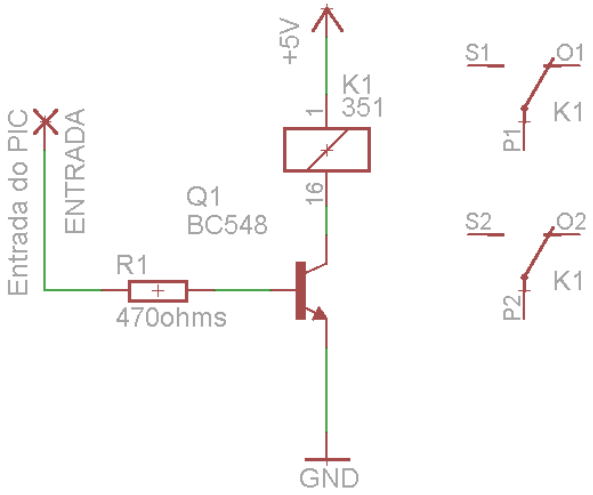

Figura 2.5 – Esquemático do circuito da etapa de potência de cada relé.

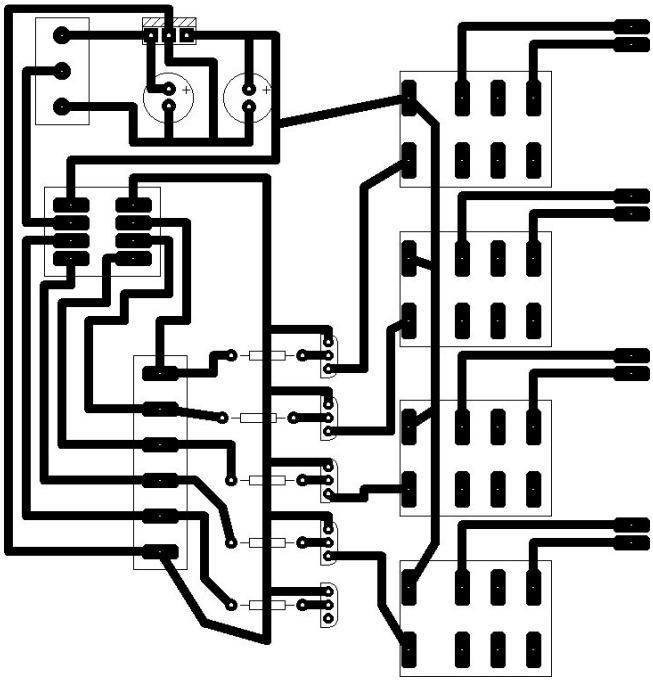

Figura 2.6 – Desenho do circuito impresso das etapas de potências dos relés, juntamente com o Módulo do PIC16F675.

2.4 Sensor de infravermelho

O sensor infravermelho é responsável por reconhecer obstáculos através de reflexão da luz infravermelha que é emitido por um fotodiodo (veja o tópico 6), e recebido por um fototransistor (veja o tópico 6). Para o sensor foi programado um PIC12F675, assim, possibilitando que o controlador PIC reconheça cada fototransistor e acione o relé ligado nele.

Lista de componentes necessários:

1x Relé de 5V

2x Transistor BC548

3x Diodo 1N4007

1x PIC12F675

1x Capacitor 470uF

2x Resistor 1Kohms 1/4W

2x Resistor 100ohms 1/4W

2x Resistor 4,7Kohms 1/4W

2x TIL31

2x TIL78

1x Regulador de tensão LM7805

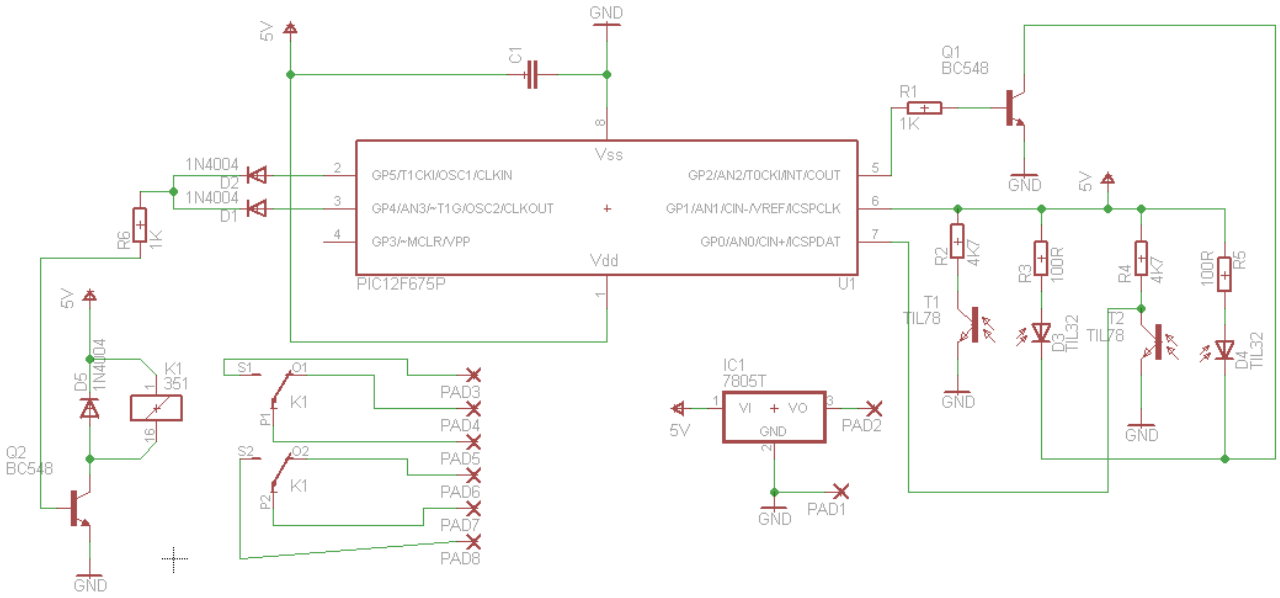

Figura 2.7 – Esquemático do circuito do sensor Infravermelho.

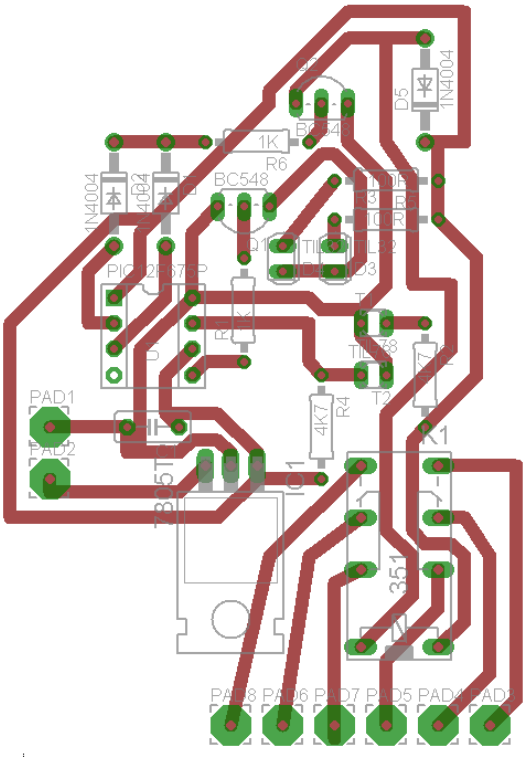

Figura 2.8 – Desenho do circuito impresso do sensor de Infravermelho.

2.5 Ponte H

A ponte H é necessário para quando o sensor Infravermelho encontrar o obstáculo, fazer a inversão da corrente, assim, possibilitando com que o carrinho automaticamente dê ré e evitar a colisão com o obstáculo.

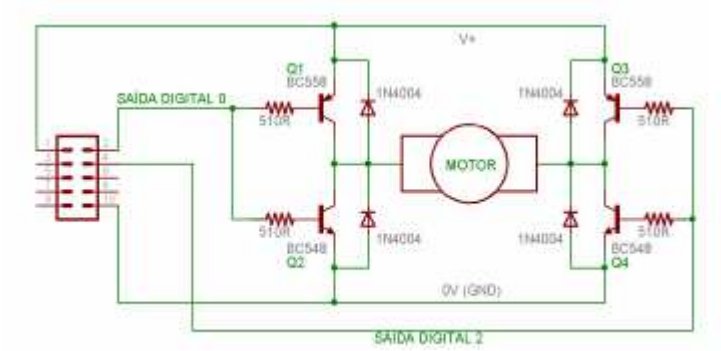

Figura 2.9 – Desenho esquemático do circuito Ponte H.

3. Software

3.1 Funcionamento

O software basicamente utiliza a classe CSerial para enviar aos circuitos montados as informações que os mesmos devem receber.

A classe do software foi adaptada para o Visual Studio 2008, pois até então, funcionava apenas com o Visual 6.0.

3.2 Código

Código “Serial.h”:

```
// Serial.h

#include <windows.h>

#ifndef __SERIAL_H__
#define __SERIAL_H__

#define FC_DTRDSR       0x01
#define FC_RTSCTS       0x02
#define FC_XONXOFF      0x04
#define ASCII_BEL       0x07
#define ASCII_BS        0x08
#define ASCII_LF        0x0A
#define ASCII_CR        0x0D
#define ASCII_XON       0x11
#define ASCII_XOFF      0x13

class CSerial
{

public:
      CSerial();
      ~CSerial();

      BOOL Open( int nPort, int nBaud );
      BOOL Close( void );

      int ReadData( void *, int );
      int SendData( const char *, int );
      int ReadDataWaiting( void );

      BOOL IsOpened( void ){ return( m_bOpened ); }
      BOOL GetCTS();

protected:
      BOOL WriteCommByte( unsigned char );

      HANDLE m_hIDComDev;
      OVERLAPPED m_OverlappedRead, m_OverlappedWrite;
      BOOL m_bOpened;

};

#endif
```

Código “Serial.cpp”:

```
// Serial.cpp
#include "stdafx.h"
#include <iostream>
#include "Serial.h"
```

```
CSerial::CSerial()
{

	memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
	memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );
	m_hIDComDev = NULL;
	m_bOpened = FALSE;

}

CSerial::~CSerial()
{

	Close();

}

BOOL CSerial::Open( int nPort, int nBaud )
{

	if( m_bOpened ) return( TRUE );

	char szPort[15];
	char szComParams[50];
	DCB dcb;

	sprintf( szPort, "COM%d", nPort );
	m_hIDComDev = CreateFileA( szPort, GENERIC_READ | GENERIC_WRITE,
0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
NULL );
	if( m_hIDComDev == NULL ) return( FALSE );

	memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
	memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );

	COMMTIMEOUTS CommTimeOuts;
	CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
	CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
	CommTimeOuts.ReadTotalTimeoutConstant = 0;
	CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
	CommTimeOuts.WriteTotalTimeoutConstant = 5000;
	SetCommTimeouts( m_hIDComDev, &CommTimeOuts );

	sprintf( szComParams, "COM%d:%d,n,8,1", nPort, nBaud );

	m_OverlappedRead.hEvent = CreateEvent( NULL, TRUE, FALSE,
NULL );
	m_OverlappedWrite.hEvent = CreateEvent( NULL, TRUE, FALSE,
NULL );

	dcb.DCBlength = sizeof( DCB );
	GetCommState( m_hIDComDev, &dcb );
	dcb.BaudRate = nBaud;
	dcb.ByteSize = 8;
	unsigned char ucSet;
	ucSet = (unsigned char) ( ( FC_RTSCTS & FC_DTRDSR ) != 0 );
	ucSet = (unsigned char) ( ( FC_RTSCTS & FC_RTSCTS ) != 0 );
	ucSet = (unsigned char) ( ( FC_RTSCTS & FC_XONXOFF ) != 0 );
	if( !SetCommState( m_hIDComDev, &dcb ) ||
```

```
            !SetupComm( m_hIDComDev, 10000, 10000 ) ||
            m_OverlappedRead.hEvent == NULL ||
            m_OverlappedWrite.hEvent == NULL ){
            DWORD dwError = GetLastError();
            if( m_OverlappedRead.hEvent != NULL )
CloseHandle( m_OverlappedRead.hEvent );
            if( m_OverlappedWrite.hEvent != NULL )
CloseHandle( m_OverlappedWrite.hEvent );
            CloseHandle( m_hIDComDev );
            return( FALSE );
            }

      m_bOpened = TRUE;

      return( m_bOpened );

}

BOOL CSerial::Close( void )
{

      if( !m_bOpened || m_hIDComDev == NULL ) return( TRUE );

      if( m_OverlappedRead.hEvent != NULL )
CloseHandle( m_OverlappedRead.hEvent );
      if( m_OverlappedWrite.hEvent != NULL )
CloseHandle( m_OverlappedWrite.hEvent );
      CloseHandle( m_hIDComDev );
      m_bOpened = FALSE;
      m_hIDComDev = NULL;

      return( TRUE );

}

BOOL CSerial::WriteCommByte( unsigned char ucByte )
{
      BOOL bWriteStat;
      DWORD dwBytesWritten;

      bWriteStat = WriteFile( m_hIDComDev, (LPSTR) &ucByte, 1,
&dwBytesWritten, &m_OverlappedWrite );
      if( !bWriteStat && ( GetLastError() == ERROR_IO_PENDING ) ){
            if( WaitForSingleObject( m_OverlappedWrite.hEvent, 1000 ) )
dwBytesWritten = 0;
            else{
                  GetOverlappedResult( m_hIDComDev, &m_OverlappedWrite,
&dwBytesWritten, FALSE );
                  m_OverlappedWrite.Offset += dwBytesWritten;
                  }
            }

      return( TRUE );

}

int CSerial::SendData( const char *buffer, int size )
{
      //TRACE1(buffer);
      if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );
```

```
	DWORD dwBytesWritten = 0;
	int i;
	for( i=0; i<size; i++ ){
		WriteCommByte( buffer[i] );
		dwBytesWritten++;
		}

	return( (int) dwBytesWritten );

}

int CSerial::ReadDataWaiting( void )
{

	if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

	DWORD dwErrorFlags;
	COMSTAT ComStat;

	ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );

	return( (int) ComStat.cbInQue );

}

int CSerial::ReadData( void *buffer, int limit )
{

	if( !m_bOpened || m_hIDComDev == NULL ) return( 0 );

	BOOL bReadStatus;
	DWORD dwBytesRead, dwErrorFlags;
	COMSTAT ComStat;

	ClearCommError( m_hIDComDev, &dwErrorFlags, &ComStat );
	if( !ComStat.cbInQue ) return( 0 );

	dwBytesRead = (DWORD) ComStat.cbInQue;
	if( limit < (int) dwBytesRead ) dwBytesRead = (DWORD) limit;

	bReadStatus = ReadFile( m_hIDComDev, buffer, dwBytesRead,
&dwBytesRead, &m_OverlappedRead );
	if( !bReadStatus ){
		if( GetLastError() == ERROR_IO_PENDING ){
			WaitForSingleObject( m_OverlappedRead.hEvent, 2000 );
			return( (int) dwBytesRead );
			}
		return( 0 );
		}

	return( (int) dwBytesRead );

}

BOOL CSerial::GetCTS()
{
	DWORD lpModemStat;
	if( !m_bOpened || m_hIDComDev == NULL ) return FALSE;
	if(!GetCommModemStatus(m_hIDComDev,&lpModemStat))
		return FALSE;
	return (lpModemStat & MS_CTS_ON)?TRUE:FALSE;
```

```
}
```

Código “Interface.cpp”:

```
// interface.cpp : main project file.

#include "stdafx.h"
#include "Form1.h"

using namespace interface1;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
	// Enabling Windows XP visual effects before any controls are
created
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	// Create the main window and run it
	Application::Run(gcnew Form1());
	return 0;
}
```

Código Form1.h:

```
#pragma once
#include "serial.h"
#include <iostream>

namespace interface1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	/// <summary>
	/// Summary for Form1
	///
	/// WARNING: If you change the name of this class, you will need
to change the
	///          'Resource File Name' property for the managed
resource compiler tool
	///          associated with all .resx files this class depends
on.  Otherwise,
	///          the designers will not be able to interact properly
with localized
	///          resources associated with this form.
	/// </summary>
	public ref class Form1 : public System::Windows::Forms::Form
	{
	public:
		Form1(void)
		{
			InitializeComponent();
			//
```

```
                //TODO: Add the constructor code here
                //
            }

        protected:
            /// <summary>
            /// Clean up any resources being used.
            /// </summary>
            ~Form1()
            {
                if (components)
                {
                    delete components;
                }
            }
        private: System::Windows::Forms::Button^  button1;
        private: System::Windows::Forms::Button^  button2;
        private: System::Windows::Forms::Button^  button3;
        private: System::Windows::Forms::Button^  button4;
        private: System::Windows::Forms::Button^  button5;
        private: System::Windows::Forms::Button^  button6;
        private: System::Windows::Forms::Button^  button7;
        private: System::Windows::Forms::Button^  button8;




        protected:

        private:
            /// <summary>
            /// Required designer variable.
            /// </summary>
            System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            void InitializeComponent(void)
            {
                this->button1 = (gcnew
System::Windows::Forms::Button());
                this->button2 = (gcnew
System::Windows::Forms::Button());
                this->button3 = (gcnew
System::Windows::Forms::Button());
                this->button4 = (gcnew
System::Windows::Forms::Button());
                this->button5 = (gcnew
System::Windows::Forms::Button());
                this->button6 = (gcnew
System::Windows::Forms::Button());
                this->button7 = (gcnew
System::Windows::Forms::Button());
                this->button8 = (gcnew
System::Windows::Forms::Button());
                this->SuspendLayout();
                //
```

```
                    // button1
                    //
                    this->button1->AccessibleRole =
System::Windows::Forms::AccessibleRole::Sound;
                    this->button1->Cursor =
System::Windows::Forms::Cursors::Hand;
                    this->button1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 8.25F,
System::Drawing::FontStyle::Bold,  System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button1->Location = System::Drawing::Point(120,
32);
                    this->button1->Name = L"button1";
                    this->button1->Size = System::Drawing::Size(83, 69);
                    this->button1->TabIndex = 0;
                    this->button1->Text = L"Cima";
                    this->button1->UseVisualStyleBackColor = true;
                    this->button1->Click += gcnew
System::EventHandler(this, &Form1::button1_Click);
                    //
                    // button2
                    //
                    this->button2->AccessibleRole =
System::Windows::Forms::AccessibleRole::Sound;
                    this->button2->Cursor =
System::Windows::Forms::Cursors::Hand;
                    this->button2->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 8.25F,
System::Drawing::FontStyle::Bold,  System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button2->Location = System::Drawing::Point(120,
107);
                    this->button2->Name = L"button2";
                    this->button2->Size = System::Drawing::Size(83, 69);
                    this->button2->TabIndex = 1;
                    this->button2->Text = L"Baixo";
                    this->button2->UseVisualStyleBackColor = true;
                    this->button2->Click += gcnew
System::EventHandler(this, &Form1::button2_Click);
                    //
                    // button3
                    //
                    this->button3->AccessibleRole =
System::Windows::Forms::AccessibleRole::Sound;
                    this->button3->Cursor =
System::Windows::Forms::Cursors::Hand;
                    this->button3->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 8.25F,
System::Drawing::FontStyle::Bold,  System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button3->ImageKey = L"(none)";
                    this->button3->Location = System::Drawing::Point(31,
107);
                    this->button3->Name = L"button3";
                    this->button3->Size = System::Drawing::Size(83, 69);
                    this->button3->TabIndex = 2;
                    this->button3->Text = L"Esquerda";
                    this->button3->UseVisualStyleBackColor = true;
                    this->button3->Click += gcnew
System::EventHandler(this, &Form1::button3_Click);
                    //
```

```
                    // button4
                    //
                    this->button4->AccessibleRole =
System::Windows::Forms::AccessibleRole::Sound;
                    this->button4->Cursor =
System::Windows::Forms::Cursors::Hand;
                    this->button4->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 8.25F,
System::Drawing::FontStyle::Bold,  System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button4->Location = System::Drawing::Point(209,
107);
                    this->button4->Name = L"button4";
                    this->button4->Size = System::Drawing::Size(83, 69);
                    this->button4->TabIndex = 3;
                    this->button4->Text = L"Direita";
                    this->button4->UseVisualStyleBackColor = true;
                    this->button4->Click += gcnew
System::EventHandler(this, &Form1::button4_Click);
                    //
                    // button5
                    //
                    this->button5->AccessibleRole =
System::Windows::Forms::AccessibleRole::Sound;
                    this->button5->Cursor =
System::Windows::Forms::Cursors::Hand;
                    this->button5->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 6.75F,
System::Drawing::FontStyle::Italic,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button5->ForeColor =
System::Drawing::Color::Red;
                    this->button5->Location = System::Drawing::Point(120,
2);
                    this->button5->Name = L"button5";
                    this->button5->Size = System::Drawing::Size(83, 24);
                    this->button5->TabIndex = 4;
                    this->button5->Text = L"Anula Cima";
                    this->button5->UseVisualStyleBackColor = true;
                    this->button5->Click += gcnew
System::EventHandler(this, &Form1::button5_Click);
                    //
                    // button6
                    //
                    this->button6->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 6.75F,
System::Drawing::FontStyle::Italic,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button6->ForeColor =
System::Drawing::Color::Red;
                    this->button6->Location = System::Drawing::Point(31,
182);
                    this->button6->Name = L"button6";
                    this->button6->Size = System::Drawing::Size(83, 24);
                    this->button6->TabIndex = 5;
                    this->button6->Text = L"Anula Esquerda";
                    this->button6->UseVisualStyleBackColor = true;
                    this->button6->Click += gcnew
System::EventHandler(this, &Form1::button6_Click);
```

```
                    //
                    // button7
                    //
                    this->button7->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 6.75F,
System::Drawing::FontStyle::Italic,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button7->ForeColor =
System::Drawing::Color::Red;
                    this->button7->Location = System::Drawing::Point(120,
182);
                    this->button7->Name = L"button7";
                    this->button7->Size = System::Drawing::Size(83, 24);
                    this->button7->TabIndex = 6;
                    this->button7->Text = L"Anula Baixo";
                    this->button7->UseVisualStyleBackColor = true;
                    this->button7->Click += gcnew
System::EventHandler(this, &Form1::button7_Click);
                    //
                    // button8
                    //
                    this->button8->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 6.75F,
System::Drawing::FontStyle::Italic,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->button8->ForeColor =
System::Drawing::Color::Red;
                    this->button8->Location = System::Drawing::Point(209,
182);
                    this->button8->Name = L"button8";
                    this->button8->Size = System::Drawing::Size(83, 24);
                    this->button8->TabIndex = 7;
                    this->button8->Text = L"Anula Direita";
                    this->button8->UseVisualStyleBackColor = true;
                    this->button8->Click += gcnew
System::EventHandler(this, &Form1::button8_Click);
                    //
                    // Form1
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);
                    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(322, 225);
                    this->Controls->Add(this->button8);
                    this->Controls->Add(this->button7);
                    this->Controls->Add(this->button6);
                    this->Controls->Add(this->button5);
                    this->Controls->Add(this->button4);
                    this->Controls->Add(this->button3);
                    this->Controls->Add(this->button2);
                    this->Controls->Add(this->button1);
                    this->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 8.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                          static_cast<System::Byte>(0)));
                    this->Name = L"Form1";
                    this->ShowIcon = false;
                    this->Text = L"Smart Car";
```

```
                this->ResumeLayout(false);

            }
#pragma endregion
      private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e)
                  {
                        CSerial Comunica;
                        Comunica.Open(1,1200);

                        char comando1[200];
                        int tam=0;
                        if(Comunica.IsOpened())
                        {
                              sprintf(comando1, "led.io0.set(1)\r");
                              //Sleep(100);
                              Comunica.SendData(comando1,
strlen(comando1));
                        }
                  }
      private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e)
                  {
                        CSerial Comunica;
                        Comunica.Open(1,1200);

                        char comando2[200];
                        int tam=0;
                        if(Comunica.IsOpened())
                        {
                              sprintf(comando2, "led.io1.set(1)\r");
                              //Sleep(100);
                              Comunica.SendData(comando2,
strlen(comando2));
                        }
                  }
      private: System::Void button3_Click(System::Object^  sender,
System::EventArgs^  e)
                  {
                        CSerial Comunica;
                        Comunica.Open(1,1200);

                        char comando3[200];
                        int tam=0;
                        if(Comunica.IsOpened())
                        {
                              sprintf(comando3, "led.io2.set(1)\r");
                              //Sleep(100);
                              Comunica.SendData(comando3,
strlen(comando3));
                        }
                  }
      private: System::Void button4_Click(System::Object^  sender,
System::EventArgs^  e)
       {
            CSerial Comunica;
            Comunica.Open(1,1200);

            char comando4[200];
            int tam=0;
            if(Comunica.IsOpened())
```

```
            {
                  sprintf(comando4, "led.io4.set(1)\r");
                  //Sleep(100);
                  Comunica.SendData(comando4, strlen(comando4));
            }
       }
      private: System::Void button5_Click(System::Object^  sender,
System::EventArgs^  e)
       {
            CSerial Comunica;
            Comunica.Open(1,1200);

            char comando5[200];
            int tam=0;
            if(Comunica.IsOpened())
            {
                  sprintf(comando5, "led.io0.set(0)\r");
                  //Sleep(100);
                  Comunica.SendData(comando5, strlen(comando5));
            }
       }
       private: System::Void button6_Click(System::Object^  sender,
System::EventArgs^  e)
      {
             CSerial Comunica;
            Comunica.Open(1,1200);

            char comando6[200];
            int tam=0;
            if(Comunica.IsOpened())
            {
                  sprintf(comando6, "led.io2.set(0)\r");
                  //Sleep(100);
                  Comunica.SendData(comando6, strlen(comando6));
            }
       }

      private: System::Void button7_Click(System::Object^  sender,
System::EventArgs^  e)
      {
            CSerial Comunica;
            Comunica.Open(1,1200);

            char comando7[200];
            int tam=0;
            if(Comunica.IsOpened())
            {
                  sprintf(comando7, "led.io1.set(0)\r");
                  //Sleep(100);
                  Comunica.SendData(comando7, strlen(comando7));
            }
       }
      private: System::Void button8_Click(System::Object^  sender,
System::EventArgs^  e)
       {
            CSerial Comunica;
            Comunica.Open(1,1200);

            char comando8[200];
            int tam=0;
            if(Comunica.IsOpened())
```

```
            {
                  sprintf(comando8, "led.io4.set(0)\r");
                  //Sleep(100);
                  Comunica.SendData(comando8, strlen(comando8));
            }
       }
   private: System::Void button9_Click(System::Object^  sender,
System::EventArgs^  e)
      {
            CSerial Comunica;
            Comunica.Open(1,1200);
            char comando9[200];
            if(Comunica.GetCTS())
            {
                  sprintf(comando9, "led.io0.set(0)\r");
                  //Sleep(100);
                  Comunica.SendData(comando9, strlen(comando9));
            }
      }
};
};
```

Código “AssemblyInfo.cpp”:

```
#include "stdafx.h"

using namespace System;
using namespace System::Reflection;
using namespace System::Runtime::CompilerServices;
using namespace System::Runtime::InteropServices;
using namespace System::Security::Permissions;

//
// General Information about an assembly is controlled through the
following
// set of attributes. Change these attribute values to modify the
information
// associated with an assembly.
//
[assembly:AssemblyTitleAttribute("interface")];
[assembly:AssemblyDescriptionAttribute("")];
[assembly:AssemblyConfigurationAttribute("")];
[assembly:AssemblyCompanyAttribute("")];
[assembly:AssemblyProductAttribute("interface")];
[assembly:AssemblyCopyrightAttribute("Copyright (c)  2009")];
[assembly:AssemblyTrademarkAttribute("")];
[assembly:AssemblyCultureAttribute("")];

//
// Version information for an assembly consists of the following four
values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the value or you can default the Revision and
Build Numbers
// by using the '*' as shown below:
```

```
[assembly:AssemblyVersionAttribute("1.0.*")];

[assembly:ComVisible(false)];

[assembly:CLSCompliantAttribute(true)];

[assembly:SecurityPermission(SecurityAction::RequestMinimum,
UnmanagedCode = true)];
```

3.3 Telas

Como o Software simplesmente envia os comando a direção e aceleração, a interface do mesmo é bem simples como podem ver na imagem abaixo:

Imagem 3.0 – Interface do Software.

4. Conclusão

Infelizmente até o prazo de entrega marcado não foi possível terminar as metas impostas, pois houve inúmeros problemas, principalmente na hora do desenvolvimento do sensor Infravermelho. Mas até a data foi possível fazer o controle do carrinho através do computador e em ambientes sem interferência de luzes, às vezes, o sensor funcionava corretamente, mas o funcionamento do mesmo, não foi possível demonstrar durante a apresentação do projeto.

5. Galeria de Fotos

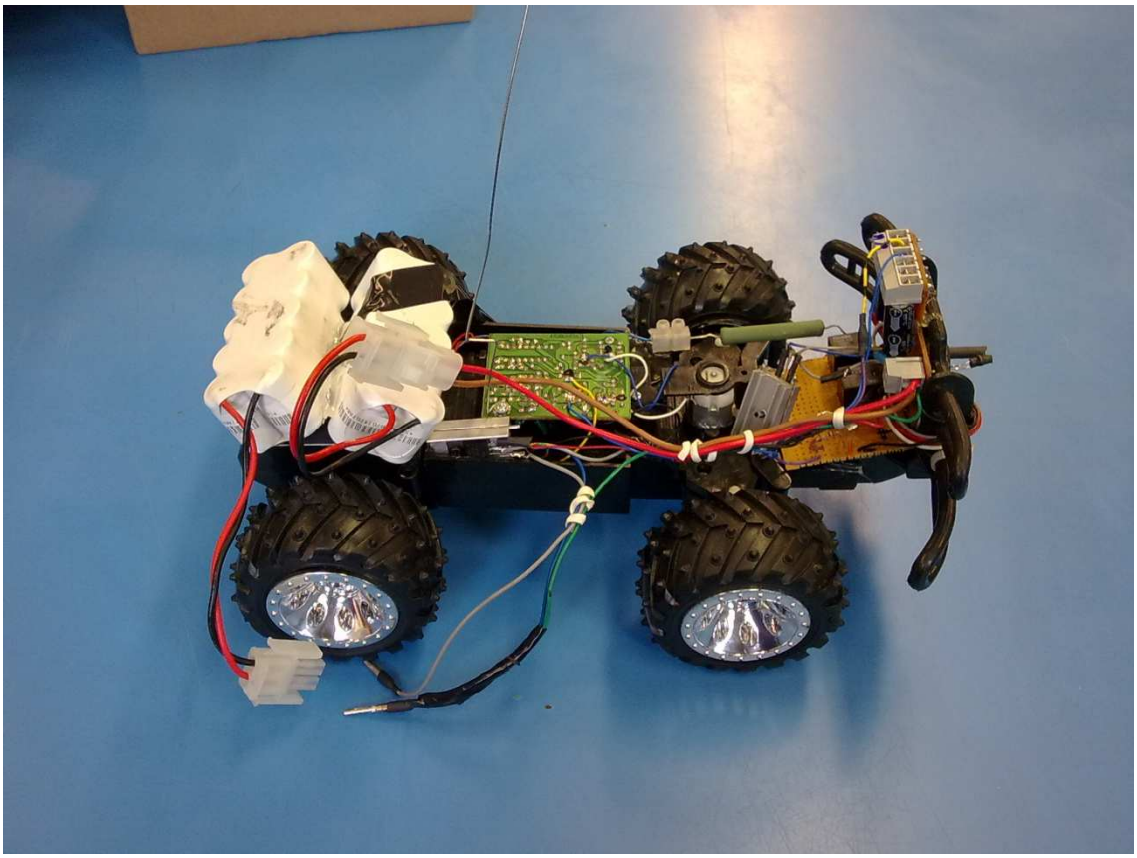

Figura 5.0 – Foto mostrando o carrinho em com todos os componentes embarcado.

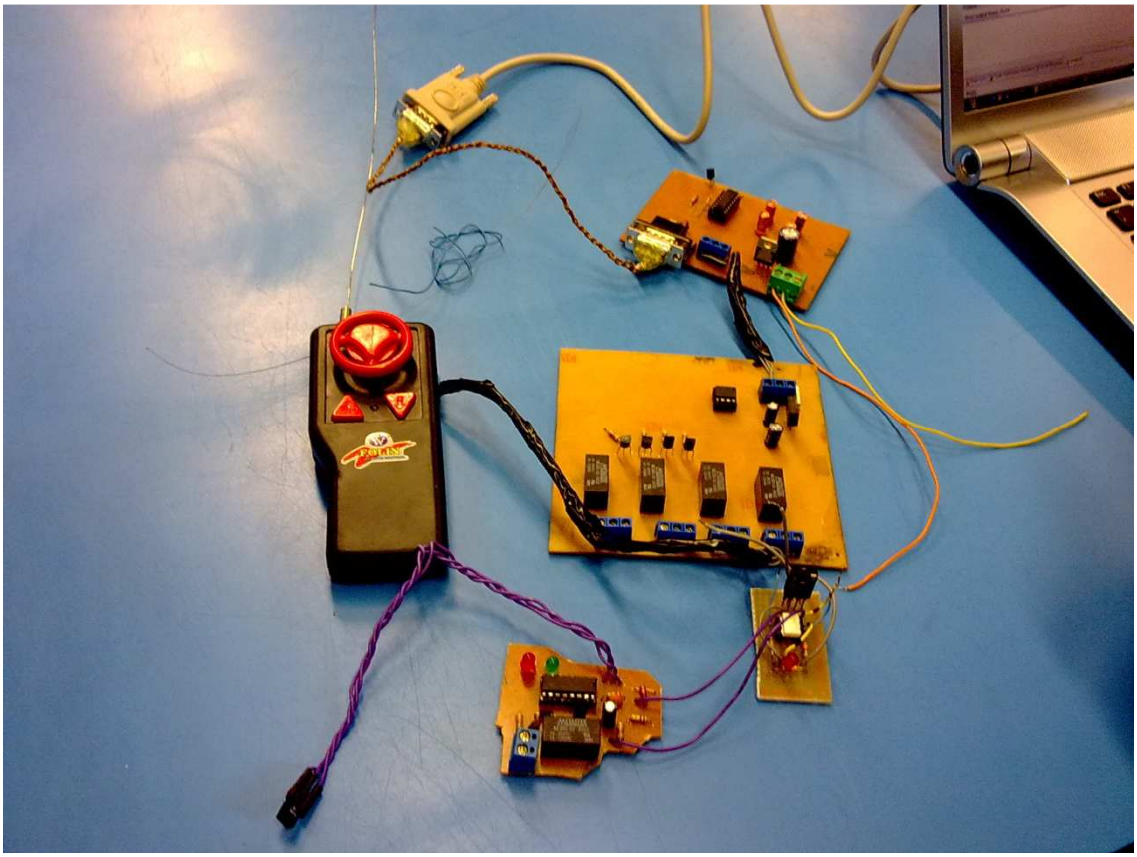

Figura 5.1 – Foto mostrando o módulo de controle via porta Serial.

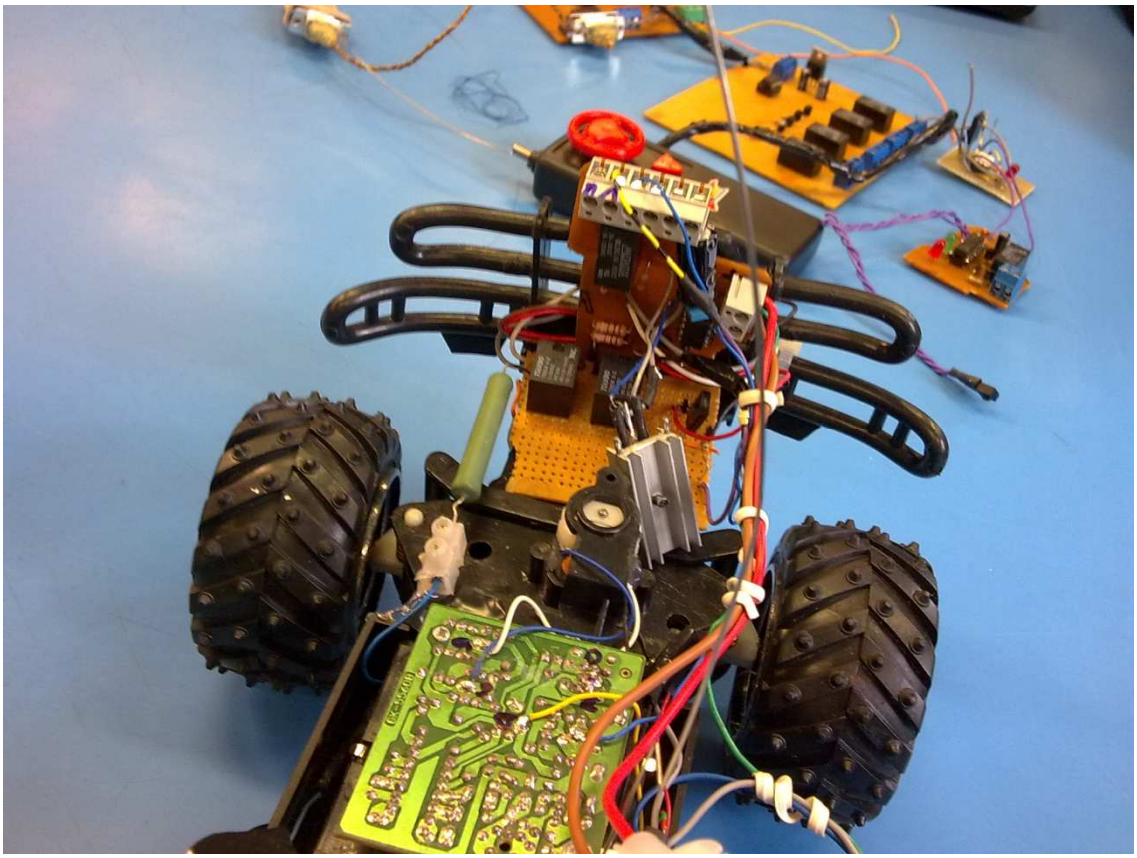

Figura 5.2 – Detalhes das placas embarcadas.

6. O que é?

6.1 Corrente elétrica

A corrente elétrica é o movimento ordenado de partículas eletricamente carregadas. Vamos explicar a corrente elétrica a partir de um condutor metálico (um fio elétrico, por exemplo). Dentro desses condutores há muitos elétrons livres descrevendo um movimento caótico, sem direção determinada. Ao aplicar-se uma diferença de potencial entre dois pontos do metal (ligando as pontas do fio a uma bateria, por exemplo), estabelece-se um campo elétrico interno e os elétrons passam a se movimentar numa certa ordem, constituindo assim a corrente elétrica.

A corrente elétrica é definida como a razão entre a quantidade de carga que atravessa certa secção transversal (corte feito ao longo da menor dimensão de um corpo) do condutor num intervalo de tempo. A unidade de medida é o *Coulomb por segundo* (C/s), chamado de *Ampère* (A) no SI em homenagem ao físico e matemático francês André-Marie Ampère (1775-1836).

Fonte: UFPA.

6.2 Diodo

Um diodo é o tipo mais simples de semicondutor. De modo geral, um semicondutor é um material com capacidade variável de conduzir corrente elétrica. A maioria dos semicondutores é feita de um condutor pobre que

teve impurezas (átomos de outro material) adicionadas a ele. O processo de adição de impurezas é chamado de dopagem**.**

Um semicondutor com elétrons extras é chamado material tipo-N, já que tem partículas extras carregadas negativamente. No material tipo-N, elétrons livres se movem da área carregada negativamente para uma área carregada positivamente.

Um semicondutor com elétrons em buraco extras é chamado material tipo-P, já que ele efetivamente tem partículas extras carregadas positivamente. Os elétrons podem pular de buraco em buraco, movendo-se de uma área carregada negativamente para uma área carregada positivamente. Como resultado, os próprios buracos parecem se mover de uma área carregada positivamente para uma área carregada negativamente.

Um diodo é composto por uma seção de material tipo-N ligado a uma seção de material tipo-P, com eletrodos em cada extremidade. Essa combinação conduz eletricidade apenas em um sentido. Quando nenhuma tensão é aplicada ao diodo, os elétrons do material tipo-N preenchem os buracos do material tipo-P ao longo da junção entre as camadas, formando uma zona vazia. Em uma zona vazia, o material semicondutor volta ao seu estado isolante original - todos os buracos estão preenchidos, de modo que não haja elétrons livres ou espaços vazios para elétrons, e assim a corrente não pode fluir.

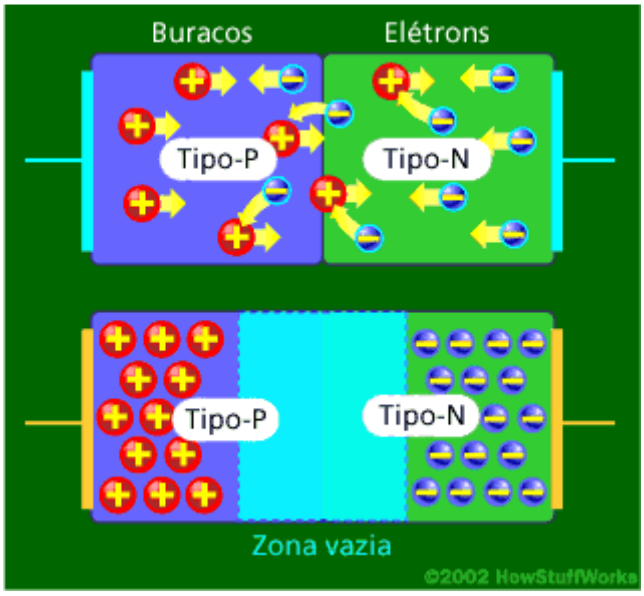

Figura 6.0 – Por dentro de um diodo.

Para se livrar da zona vazia, você precisa que elétrons se movam da área tipo-N para a área tipo-P e que buracos se movam no sentido inverso. Para fazer isto, você conecta o lado tipo-N do diodo ao terminal negativo do circuito e o lado tipo-P ao terminal positivo. Os elétrons livres no material tipo-N são repelidos pelo eletrodo negativo e atraídos para o eletrodo positivo. Os buracos no material tipo-P se movem no sentido contrário. Quando a diferença de potencial entre os eletrodos é alta o suficiente, os elétrons na zona vazia são retirados de seus buracos e começam a se mover livremente de novo. A zona vazia desaparece e a carga se move através do diodo.

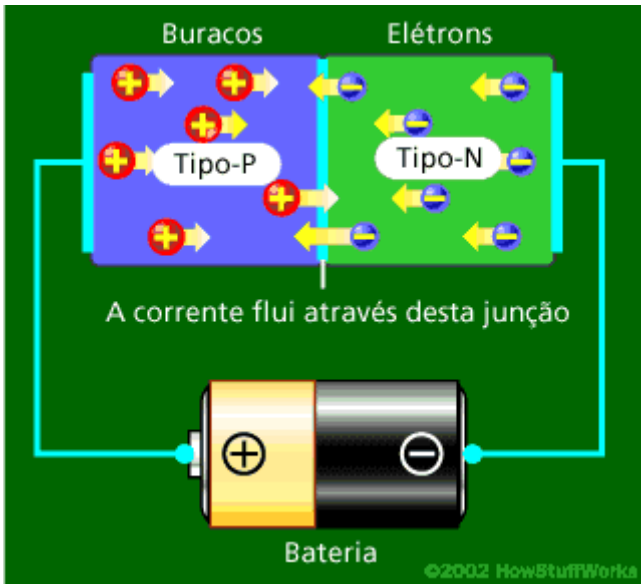

Figura 6.1 – Energia Elétrica fluindo pelo diodo.

Se você tentar mover a corrente no sentido oposto, com o lado tipo-P conectado ao terminal negativo do circuito e o lado tipo-N conectado ao pólo positivo, a corrente não fluirá. Os elétrons negativos no material tipo-N são atraídos para o eletrodo positivo. Os buracos positivos no material tipo-P são atraídos para o eletrodo negativo. Nenhuma corrente flui através da junção porque os buracos e os elétrons estão cada um se movendo no sentido errado. A zona vazia então aumenta.

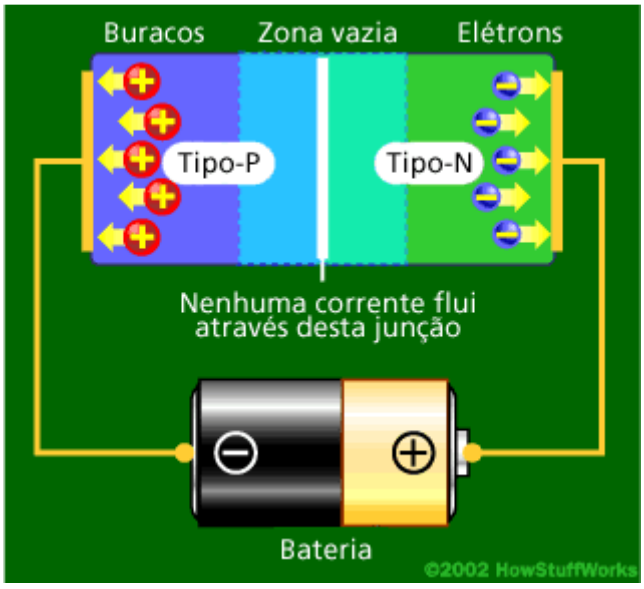

Figura 6.2 – Diodo não fluindo corrente elétrica.

Figura 6.3 – Diodos existentes no mercado.

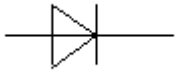

Figura 6.4 - Desenho esquemático de um diodo.

Fonte: HowStuffWorks

6.3 Fotodiodo

O fotodiodo é um diodo de junção construído de forma especial, de modo a possibilitar a utilização da luz como fator determinante no controle da corrente elétrica. É um dispositivo de junção P e N semicondutor cuja região de operação é limitada pela região de polarização reversa e caracteriza-se por ser sensível à luz. A aplicação de luz à junção resultará em uma transferência de energia das ondas luminosas incidentes (na forma de fótons) para a estrutura atômica, resultando em um aumento do número de portadores minoritários e um aumento do nível da corrente reversa. A corrente negra é a corrente que existirá sem nenhuma iluminação aplicada. A corrente retornará a zero somente se for aplicada uma polarização positiva igual à Vo.

Em resumo, podemos dizer então que um fotodiodo é um dispositivo que converte a luz recebida em uma determinada quantidade de corrente elétrica. A corrente reversa e o fluxo luminoso variam quase que linearmente, ou seja, um aumento na intensidade luminosa resultará em um aumento semelhante na corrente reversa. Podemos admitir que a corrente reversa seja essencialmente nula na ausência de luz incidente. Como os tempos de subida e de queda (parâmetros de mudança de estado) são da ordem de nanossegundos, o dispositivo pode ser usado na aplicação de contagem ou comutação de alta velocidade. O germânio é mais adequado para luz incidente na região infravermelha, já que abrange um espectro mais amplo de comprimentos de onda do que o silício, apesar de sua corrente negra ser maior. O nível de corrente gerada pela luz incidente sobre um fotodiodo não é suficiente para que ele possa ser usado em um controle direto, sendo necessário para isto que haja um estágio de amplificação.

O fotodiodo pode ser aplicado no foco automático de filmadora, na unidade ótica do CD Player e em sistema contador de pulso. Outra aplicação muito usada na rede de iluminação pública é o sensor crepuscular.

Nos sistemas de iluminação publica é importante saber em que altura é que está suficientemente escuro, para ativar as luzes. Este controle não pode ser efetuado de forma eficaz utilizando temporizadores, uma vez que em dias de chuva ou nevoeiro intenso pode ser necessário ativar o sistema de iluminação por razões de segurança.

Além disso, o horário do próprio nascer e pôr do Sol não é constante, muda todos os dias. Pelas razões apontadas, a solução que reúne maior consenso é aquela que utiliza sensores de luz ambientes também conhecidos como crepusculares.

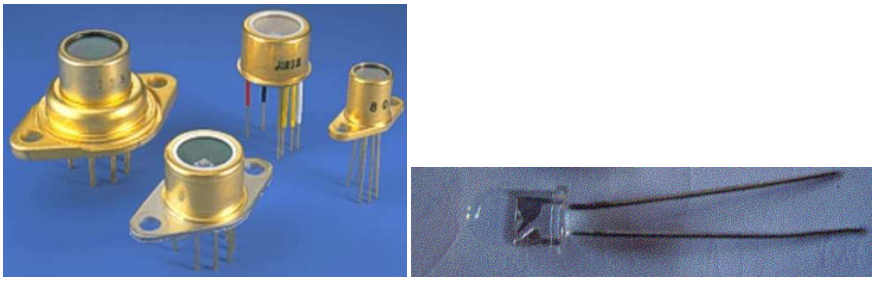

Figura 6.5 – Imagem de alguns fotodiodos existentes no mercado.

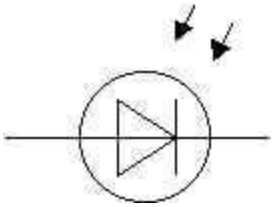

Figura 6.6 - Imagem do símbolo de um Fotodiodo.

Fonte: Wikipédia, CEFET - SC.

6.4 Fototransistor

Obs.: Antes de ler sobre o fototransistor recomenda-se a leitura sobre transistores.

O fototransistor é mais um dispositivo que funciona baseado no fenômeno da fotocondutividade. Ele pode, ao mesmo tempo, detectar a incidência de luz e fornecer um ganho dentro de um único componente. Como o transistor convencional, o fototransistor é uma combinação de dois diodos de junção, porém, associado ao efeito transistor aparece o efeito fotoelétrico. Em geral, possui apenas dois terminais acessíveis, o coletor e o emissor, sendo a base incluída apenas para eventual polarização ou controle elétrico.

Como nas outras células fotocondutivas, a incidência de luz (fótons) provoca o surgimento de lacunas na vizinhança da junção base-coletor. Esta tensão conduzirá as lacunas para o emissor, enquanto os elétrons passam do emissor para a base. Isso provocará um aumento da corrente de base, o que por conseqüência implicará numa variação da corrente de coletor “b” vezes maior lembrando que, para Ib (corrente “b”) sendo a corrente da base e Ic (corrente “c”) a do coletor, temos a relação Ic = a.Ib, onde “a” é o ganho do transistor (valor fornecido pelo fabricante), sendo essa variação proporcional à intensidade da luz incidente.

Como a base está normalmente desconectada, a corrente que circula por ela dependerá apenas do fluxo luminoso incidente. Assim, na ausência de luz, a corrente de base será zero e o fototransistor estará cortado, resultando na tensão do coletor igual à tensão de polarização Vcc. Quando há luz incidindo, a tensão no coletor irá diminuir devido ao aumento da corrente.

O fototransistor possui diversas aplicações, sendo mais encontrado em aplicações on-off, onde a não linearidade do transistor não é um problema. A aplicação

mais usual é a de um interruptor. Enquanto não á luz incidindo no fototransistor, não haverá uma corrente no emissor, e a tensão de saída será zero, estando ele em corte.

Com a incidência de luz, teremos uma corrente no emissor, provocando uma tensão igual à IeRe. Tais como os transistores bipolares, os fototransistores estão sujeitos às variações de temperatura. Com o aumento da temperatura em torno de 8 a 10 graus Celsius, a corrente Iceo (corrente que circula no componente enquanto não existe incidência de luz) dobrará. Para elevadas temperaturas, essa corrente terá um valor significativo em relação à corrente total.

Figura 6.7 – Imagem de alguns fototransistores existentes no mercado.

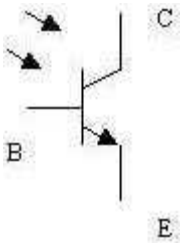

Figura 6.8 - Imagem do símbolo de um fototransistor.

Fonte: Wikipedia, Cefet-SC.

6.5 LED

LED’s é uma abreviação de “Light Emitting Diode” ou em português seria um Diodo Emissor de Luz, ele nada mais é do que um semicondutor que ao ser energizado ele emite uma luz. Ele é uma junção de semicondutores do tipo P e N, onde, o P é o positivo ou cátodo (falta de elétrons) e o N é o negativo ou o ânodo (excesso de elétrons), para mais detalhes sobre semicondutores a página onde está sendo explicado sobre os transistores nesse documento.

A cor do LED depende do cristal e da impureza de dopagem com que o componente é fabricado. O LED que utiliza o arseneto de gálio emite radiações infravermelhas. Dopando-se com fósforo, a emissão pode ser vermelha ou amarela, de acordo com a concentração. Utilizando-se fosfeto de gálio com dopagem de nitrogênio, a luz emitida pode ser verde ou amarela. Hoje em dia, com o uso de outros materiais, consegue-se fabricar LED’s que emitem luz azul, violeta e até ultravioleta. Existem também os LED’s brancos, mas esses são geralmente LED’s emissores de cor azul, revestidos com uma camada de fósforo do mesmo tipo usado nas lâmpadas fluorescentes, que absorve a luz azul e emite a luz branca. Com o barateamento do preço, seu alto rendimento e sua grande durabilidade, esses LED’s tornaram-se ótimos

substitutos para as lâmpadas comuns, e devem substituí-las a médio ou longo prazo. Existem também os LED’s chamados RGB, e que são formados por três "chips", um vermelho (R de red), um verde (G de green) e um azul (B de blue), esses LED’s podem ser utilizados juntamente com um microcontrolador para variar as cores do modo que quiser.

Em geral, os leds operam com nível de tensão de 1,6 a 3,3V, sendo compatíveis com os circuitos de estado sólido. É interessante notar que a tensão é dependente do comprimento da onda emitida. Assim, os leds infravermelhos geralmente funcionam com menos de 1,5V, os vermelhos com 1,7V, os amarelos com 1,7V ou 2.0V, os verdes entre 2.0V e 3.0V, enquanto os leds azuis, violeta e ultravioleta geralmente precisam de mais de 3V, já a corrente consumida normalmente é de 20mA, mas dependendo o tipo de LED esse valor pode variar.O tempo de vida útil dele é de aproximadamente 100mil horas.

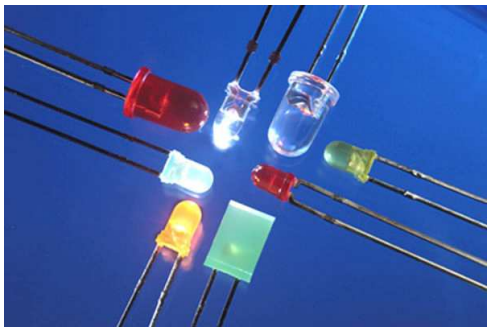

Figura 6.9 – Imagem de alguns LED’s existentes no mercado.

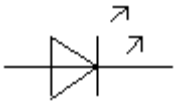

Figura 6.10 – Imagem do desenho esquemático de um LED.

Fonte: Wikipédia, HowStuffWorks.

6.6 Relé

Os relés são componentes eletromecânicos capazes de controlar circuitos externos de grandes correntes a partir de pequenas correntes ou tensões, ou seja, podemos acionar um relé com uma pilha e controlar um dispositivo (Motor, lâmpada e etc...) que precisa ser ligado em tensões mais altas (110 ou 220 volts, por exemplo).

O funcionamento dos relés é bem simples, quando uma corrente circula por uma bobina que se localiza dentro do relé, esta cria um campo magnético que atrai um ou vários contatos fechando ou abrindo circuitos que estiver ligado no relé. Quando tirar a corrente da bobina o campo magnético acaba, fazendo com que os contatos voltem para a posição original.

Os relés podem ter diversas configurações quanto aos seus contatos: podem ter contatos NA (normalmente aberto), NF (normalmente fechado) ou ambos, neste caso com um contato comum ou central (C). Os contatos NA são os que estão abertos

enquanto a bobina não está energizada e que fecham, quando a bobina recebe corrente. Os NF se abrem quando a bobina recebe corrente, ao contrário dos NA. O contato C é o comum, ou seja, é ele que se move quando a corrente passa ou deixa de passar na bobina.

A principal vantagem dos Relés é que o circuito de carga está completamente isolado do circuito de controle, assim, podendo inclusive trabalhar com tensões diferentes entre o controle e a carga.

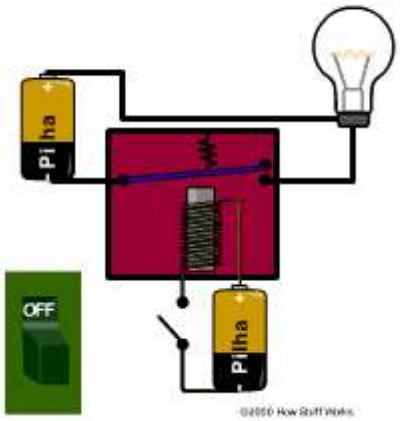

Figura 6.11 – Figura mostrando o estado de um relé desligado.

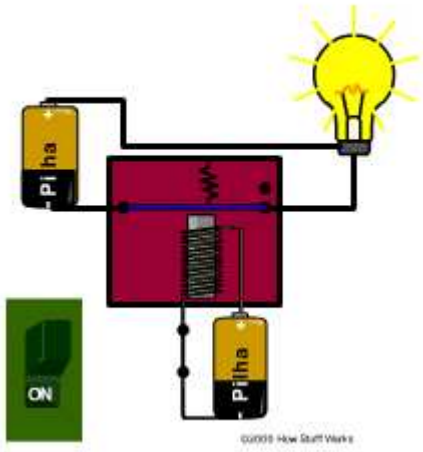

Figura 6.12 - Figura mostrando o estado de um relé ligado.

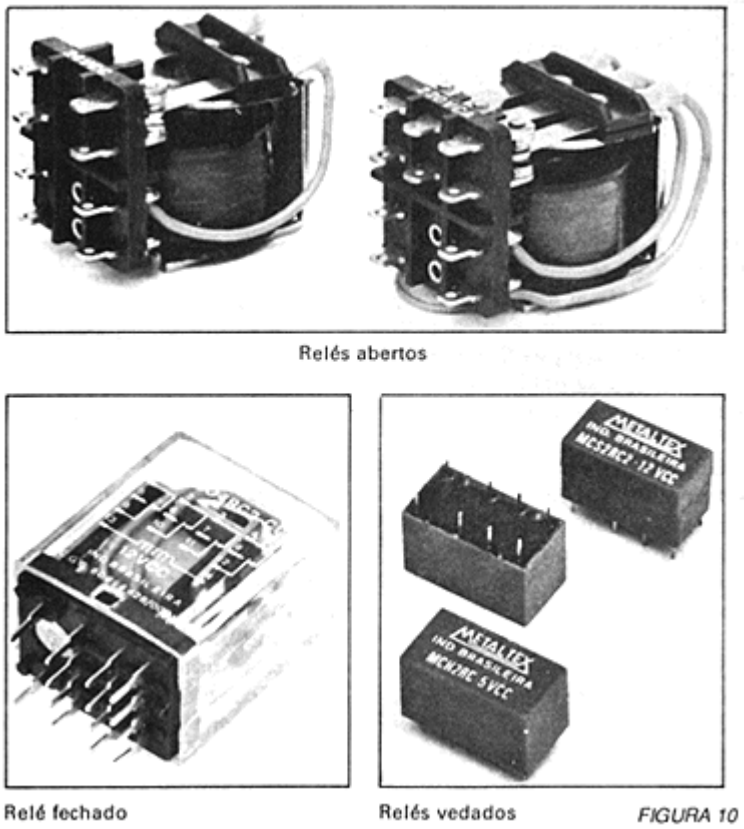

Figura 6.13 – Imagem de alguns relés existentes no mercado.

Fonte: Wikipédia, Angelfire e HowStuffWorks.

6.7 Resistor

Os resistores são componentes responsáveis por transformar energias elétricas em energia térmica através do efeito Joule. Ele é fabricado com matérias resistivo, como carbono, por exemplo. Um resistor tem umas faixas coloridas que podem mostrar os valores da resistividade e a sua tolerância desse resistor, alguns resistores são longos e finos, com o material resistivo colocado ao centro, e um terminal de metal ligado em cada extremidade. Este tipo de encapsulamento é chamado de encapsulamento axial. Resistores usados em computadores e outros dispositivos são tipicamente muito menores, freqüentemente são utilizadas tecnologia de montagem superficial (Surface-mount technology), ou SMT, esse tipo de resistor não possui terminais, já os resistores de maiores potências são produzidos mais robustos para dissipar calor de maneira mais eficiente, mas eles seguem basicamente a mesma estrutura.

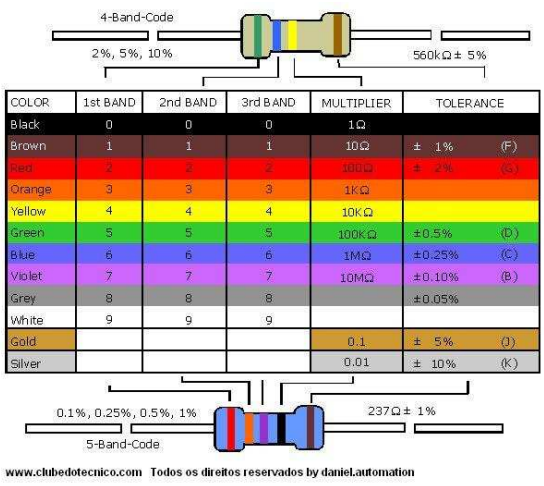

Figura 6.14 – Leitura das faixas do resistor.

Figura 6.15 – Imagem de um resistor SMD (cima) e um resistor de carbono (baixo).

Fonte: Wikipédia, HowStuffWorks.

6.8 Circuitos Integrados

Um circuito integrado, também conhecido por chip, é um dispositivo microeletrônico que consiste de muitos transistores e outros componentes interligados capazes de desempenhar muitas funções. Suas dimensões são extremamente reduzidas, os componentes são formados em pastilhas de material semicondutor.

A importância da integração está no baixo custo e alto desempenho, além do tamanho reduzido dos circuitos aliado à alta confiabilidade e estabilidade de funcionamento. Uma vez que os componentes são formados ao invés de montados, a resistência mecânica destes permitiu montagens cada vez mais robustas a choques e impactos mecânicos, permitindo a concepção de portabilidade dos dispositivos eletrônicos.

No circuito integrado completo ficam presentes os transístores, condutores de interligação, componentes de polarização, e as camadas e regiões isolantes ou condutoras obedecendo ao seu projeto de arquitetura.

No processo de formação do chip, é fundamental que todos os componentes sejam implantados nas regiões apropriadas da pastilha. É necessário que a isolação seja perfeita, quando for o caso. Isto é obtida por um processo chamado difusão, que se dá entre os componentes formados e as camadas com o material dopado com fósforo, e separadas por um material dopado com boro, e assim por diante.

Após sucessivas interconexões, por boro e fósforo, os componentes formados ainda são interconectados externamente por uma camada extremamente fina de alumínio, depositada sobre a superfície e isolada por uma camada de dióxido de silício.

Figura 6.16 – Imagem de um Circuito Integrado.

Fonte: Wikipédia, Guia do Hardware.

6.9 Transistor

O primeiro projeto surgiu em 16 de Dezembro de 47, onde era usado um pequeno bloco de germânio (que na época era junto com o silício o semicondutor mais pesquisado) e três filamentos de ouro. Um filamento era o pólo positivo, o outro o pólo negativo, enquanto o terceiro tinha a função de controle. Tendo apenas uma carga elétrica no pólo positivo, nada acontecia, o germânio atuava como um isolante, bloqueando a corrente. Porém, quando certa tensão elétrica era aplicada usando o filamento de controle, um fenômeno acontecia e a carga elétrica passava a fluir para o pólo negativo. Haviam criado um dispositivo que substituía a válvula, sem possuir partes móveis, ao mesmo tempo, muito mais rápidos. Este primeiro transistor era relativamente grande, mas não demorou muito para que este modelo inicial fosse aperfeiçoado.

Durante a década de 50, o transistor foi gradualmente dominando a indústria, substituindo rapidamente as problemáticas válvulas. Os modelos foram diminuindo de tamanho, caindo de preço e tornando-se mais rápidos. Alguns transistores da época podiam operar a até 100 MHz. Claro que esta era a freqüência que podia ser alcançada por um transistor sozinho, nos computadores da época, a freqüência de operação era muito menor, já que em cada ciclo de processamento o sinal precisa passar por vários transistores.

Mas, o grande salto foi à substituição do germânio pelo silício. Isto permitiu miniaturizar ainda mais os transistores e baixar seu custo de produção. Os primeiros transistores de junção comerciais foram produzidos partir de 1960 pela Crystalonics. A idéia do uso do silício para construir transistores é que adicionando certas substâncias em pequenas quantidades é possível alterar as propriedades elétricas do silício. As primeiras experiências usavam fósforo e boro, que transformavam o silício em condutor por cargas negativas ou condutoras por cargas positivas, dependendo de qual dos dois materiais fosse usado. Estas substâncias adicionadas ao silício são chamadas de impurezas, e o silício "contaminado" por elas é chamado de silício dopado.

O funcionamento e um transístor são bastante simples, quase elementar. É como naquele velho ditado "as melhores invenções são as mais simples". As válvulas eram muito mais complexas que os transistores e mesmo assim foram rapidamente substituídas por eles. Um transistor é composto basicamente de três filamentos, chamados de base, emissor e coletor. O emissor é o pólo positivo, o coletor o pólo negativo, enquanto a base é quem controla o estado do transistor, que como vimos, pode estar ligado ou desligado. Quando o transistor está desligado, não existe carga elétrica na base, por isso, não existe corrente elétrica entre o emissor e o coletor (temos então um bit 0). Quanto é aplicado certa tensão na base, o circuito é fechado e é estabelecida a corrente entre o emissor e o receptor (um bit 1).

Método de fabricação do transistor

Os materiais utilizados atualmente na fabricação do transistor são o Silício (Si), o Gálio (Ga) e alguns óxidos. Na natureza, o silício é um material isolante elétrico, devido à conformação das ligações eletrônicas de seus átomos, gerando uma rede eletrônica altamente estável.

O silício é purificado e passa por um processo que forma uma estrutura cristalina em seus átomos. O material é cortado em finos discos, que a seguir vão para um processo chamado de dopagem, onde são introduzidas quantidades rigorosamente controladas de materiais selecionados (conhecidos como impurezas) que transformam a estrutura eletrônica, introduzindo-se entre as ligações dos átomos de silício, recebe ou doa elétrons dos átomos, gerando o silício P ou N, conforme ele seja positivo (tenha falta de elétrons) ou negativo (tenha excesso de elétrons). Se a impureza tiver um elétron a mais, um elétron fica sobrando na estrutura cristalina. Se tiver um elétron a menos, fica faltando um elétron, o que produz uma lacuna (que funciona como se fosse um buraco móvel na estrutura cristalina). Como resultado, temos ao fim desse processo um semicondutor.

O transistor é montado juntando uma camada P, uma N e outra P, criando-se um transistor do tipo PNP. O transistor do tipo NPN é obtido de modo similar. A camada do centro é denominada base, e as outras duas são o emissor e o coletor. No símbolo do

componente, o emissor é indicado por uma seta, que aponta para dentro do transistor se o componente for PNP, ou para fora se for NPN.

Figura 6.17 – Imagem de alguns transistores existentes no mercado.

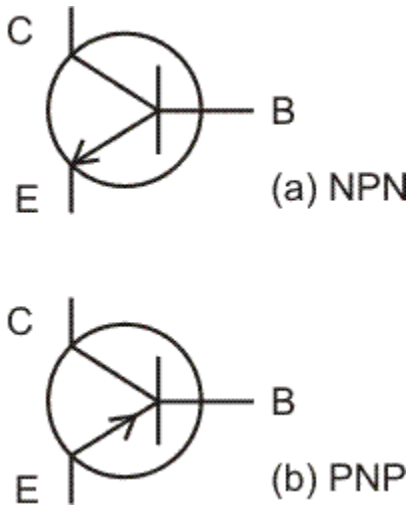

Figura 6.18 - Imagem do símbolo de um transistor do tipo PNP e outro do NPN.

Fonte: Wikipédia, Guia do Hardware.