UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MICHAEL DE OLIVEIRA RESENDE

DESENVOLVIMENTO DE UM SISTEMA SUPERVISÓRIO COM ACESSO REMOTO PARA O CONTROLE DE UM PROCESSO DE ACIONAMENTO ELÉTRICO UTILIZANDO INVERSOR DE FREQUÊNCIA.

VIÇOSA
2014

MICHAEL DE OLIVEIRA RESENDE

DESENVOLVIMENTO DE UM SISTEMA SUPERVISÓRIO COM ACESSO REMOTO PARA O CONTROLE DE UM PROCESSO DE ACIONAMENTO ELÉTRICO UTILIZANDO INVERSOR DE FREQUÊNCIA.

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.
Orientador: Prof. Dr. André Gomes Torres.

VIÇOSA
2014

MICHAEL DE OLIVEIRA RESENDE

DESENVOLVIMENTO DE UM SISTEMA SUPERVISÓRIO COM ACESSO REMOTO PARA O CONTROLE DE UM PROCESSO DE ACIONAMENTO ELÉTRICO UTILIZANDO INVERSOR DE FREQUÊNCIA.

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovada em 03 de fevereiro de 2014.

COMISSÃO EXAMINADORA

Prof. Dr. André Gomes Torres - Orientador
Universidade Federal de Viçosa

Prof. Dr. Denílson Eduardo Rodrigues - Membro
Universidade Federal de Viçosa

Prof. Dra. Ketia Soares Moreira - Membro
Universidade Federal de Viçosa

“Lute com determinação, abrace a vida com paixão, perca com classe e vença com ousadia, porque o mundo pertence a quem se atreve e a vida é muito para ser insignificante.”

(Charles Chaplin)

Aos meus pais e a todos aqueles que contribuíram para minha formação, não só acadêmica; na descoberta contínua da vida, minha eterna gratidão e a certeza que a busca continua.

Agradecimentos

Nos grandes desafios da vida, a única forma de se alcançar o êxito é tendo fé em Deus e perseverança. Agradeço a Ele por me dar forças e sabedoria para essa caminhada rumo à vitória.

Agradeço aos meus amados pais, Gilma e Eli, pelo apoio, dedicação e força, principalmente nos momentos em que a minha força não parecia suficiente para superar as dificuldades. E aos meus irmãos David e Belíria, pelo incentivo e torcida incondicional. Eu sou o reflexo do amor e confiança depositados por vocês.

Agradeço a minha namorada Helen pelo companheirismo e compreensão. E aos amigos da Engenharia Elétrica, por sempre estarem ao meu lado nos momentos bons e ruins. Sem o apoio de todos, eu não teria alcançado a vitória.

Agradeço aos colegas de trabalho, professores e técnicos do DEL, pelo apoio e compreensão para que eu pudesse desenvolver esse trabalho.

Agradeço ao professor Dr. André pela dedicação e orientação no desenvolvimento dessa monografia.

Resumo

Os sistemas supervisórios no controle de processos são ferramentas muito importantes, pois permitem que sejam monitoradas informações de um processo produtivo ou instalação física. Essas informações referentes ao processo são coletadas através de equipamentos de aquisição de dados e, em seguida, manipuladas, analisadas, armazenadas e apresentadas ao usuário através de uma interface gráfica amigável. O referido trabalho tem como principal objetivo a criação de um sistema supervisório na plataforma Microsoft Visual C, usando a linguagem C#, para monitorar um sistema de acionamento elétrico de um inversor de frequência. O programa foi desenvolvido utilizando o software Microsoft Visual C# 2010 Express, que é uma versão gratuita da plataforma de desenvolvimento. Este é o responsável por mostrar os dados recebidos do valor da corrente do motor em gráfico de evolução e a velocidade do motor em formato de *Gauge* e ainda, enviar os valores de ki, kp e kd para o microcontrolador calcular o valor de um controle PID internamente. Também sendo responsável por guardar os valores lidos em arquivos de formato *.txt, os quais, podem ser utilizados por outros programas. Com a utilização de um Microcontrolador PIC, da família 18F4550, foi feita a interface entre o computador e o inversor de frequência. O PIC usa uma comunicação USB com o computador e uma comunicação analógica/digital com o inversor de frequência. O sistema supervisório conta com acesso remoto. Para isso, é usado um protocolo de controle de rede TCP/IP.

Para o uso do protocolo TCP/IP será usado o conceito de *socket* que é constituído por uma porta de comunicação e um endereço de IP, que identifica o computador e o programa a ser utilizado. Logo, com o sistema desenvolvido, acionamentos utilizando Inversores de frequência CFW11 da WEG podem utilizar o sistema de monitoramento e controle sendo possível o seu monitoramento remotamente. O sistema apresentou uma ótima estabilidade nas comunicações feitas, com o sistema de supervisão servidor foi possível o monitoramento de vários sistemas clientes ao mesmo tempo.

Abstract

The supervisor process in the process control are really important devices, because installations of a process and physical installations can be monitored by them. These information about the process are collect by the data acquisition equipments and after that manipulated and analyzed, stored and presented to the user through out a friendly graphic interface. The aim of this project is to build a supervisor system in the Macrosoft Visual platform C, making using of a C# language to monitor the electric drivers and a frequency inverter. The program was developed by using the Macrosoft Visual C# 2010 Express software, which is a free version of the development platform. This one is responsible to show the received data in evolution graphic of the motor current as time passes by, the velocity of the motor in Gauge format and it still sends the KI, KP, and Kd values to microcontroller to calculate the value of a inner PID control. It is also responsible to keep the read data from the format *txt, that can be used by other programs. With the use of a microcontroller PIC, from the family 18F4550, was taken the interface between the computer and the frequency inverter. The PIC uses a USB communication with the computer and a analog/ digital with the frequency inverter. The supervisor systems have a remote access, for this, a TCP/IP control protocol it is used. To use the TCP/IP protocol is used the concept consisting of a gateway and an IP address, that can identify the computer and program to be used. Having the system developed the actuation using frequency CFW11 inverters can make use of the monitoring control system being also possible to control it remotely. The system showed good stability for the communications made with the server system of supervision was possible to monitor multiple client systems simultaneously.

Sumário

Lista de Figuras

1 Introdução

A humanidade nem sempre teve conhecimento suficiente para obter energia a partir da matéria. No início, a energia era adquirida pelo trabalho humano ou por trabalho de animais. Essa situação mudou apenas com o surgimento da máquina a vapor, no século XVIII, onde foi possível transformar a energia da matéria em trabalho. Porém, o homem apenas teve a sua condição de trabalho mudada, passando do trabalho puramente braçal ao trabalho mental. Nessa nova condição, o homem teve que concentrar seus esforços no controle desta nova fonte de energia, exigindo dele então muita intuição e experiência, além de constantemente estar expondo-o ao perigo devido a condições de segurança precárias. No começo desse tipo de controle intuitivo, a demanda de produção era baixa, o que justificava a eficiência desse método. Porém, com o passar dos anos e grande aumento da demanda, o homem sentiu-se obrigado a desenvolver técnicas e equipamentos que seriam capazes de substituí-lo nesta nova tarefa, libertando-o de grande parte deste esforço braçal e mental **[01]**.

Com essa necessidade, surgiram os componentes da automação, sendo que qualquer sistema que dependa de um computador ou equipamento programável, que substitua o trabalhador de tarefas repetitivas e vise soluções rápidas e econômicas, tendo como resultado atingir os objetivos das industriais para que foram projetados, podem ser chamados de sistemas automatizados **[03]**.

Com o passar do tempo, os sistemas a serem automatizados apresentavam problemas cada vez mais complexos. Com isso, os componentes de um sistema de automação tiveram que evoluir para atender essa nova demanda, desde os primeiros sistemas baseados em controle automático mecanizado, até sistemas baseados nas tecnologias atuais como a microeletrônica. A atuação da automação passou a não ser apenas no chão de fábrica, mas uma ferramenta capaz de interligar os vários ambientes de uma empresa, sendo usados como indicativos de produção e até para tomada de decisões em sistemas de gerência **[02]**.

1.1 Sistemas Supervisórios no Controle de Processos.

1.1.1 Introdução

Os processos industriais modernos apresentam a necessidade de serem controlados para que se possa obter um maior desempenho, e para esse controle ser eficiente, é necessário um monitoramento adequado. Com isso, temos o conceito importante de supervisão de processos, que ao longo dos anos e com a evolução crescente da tecnologia se tornaram cada vez mais rápidos e eficientes. Com a alta tecnologia de processamento de informações moderna, podemos classificar determinado processo como sendo supervisionado em tempo real. Essa definição de tempo real pode ser entendida de maneira fácil, sendo que todas as variáveis supervisionadas no processo são mostradas instantaneamente ao operador, a medida que variam no processo real **[04]**.

Nos processos industriais, os sistemas supervisórios vêm atender a diversas necessidades e exigências das aplicações. Na indústria, existe a necessidade de uma centralização de informações, para que se possa ter o maior número de informações com o menor tempo possível. Com o uso de Sistemas Supervisórios é possível diminuir o tamanho dos painéis de controle e deixar a interface homem/máquina mais eficiente. São baseados em computadores executando *softwares* específicos de supervisão de processos industriais. Com esse tipo de *softwares*, é possível uma visualização gráfica com informações do processo por cores e animações, tornando assim, uma interface mais amigável ao operador. Além dessas vantagens, os projetistas podem contar com diversos tipos de comunicação com as mais variadas marcas e modelos de equipamentos disponíveis no mercado **[03] [06]**.

Para desenvolver um sistema supervisório de boa qualidade e que seja aplicável, é necessário seguir algumas etapas de planejamento, dentre as quais podemos salientar **[06]**:

• Entendimento do processo a ser automatizado: Nessa etapa, é necessário reunir o máximo possível de informações sobre o processo, tendo orientação dos operadores no caso da planta já existir ou com especialistas de planejamento. E também ter conhecimento de quais variáveis são importantes para gerência e demais estrutura administrativa.

• Tomada de dados: Determinar quais são as variáveis essenciais do processo, para que não sobrecarregue o sistema computacional com excesso de informações desnecessárias.

• Planejamento do banco de dados: Nesse ponto é necessário designar as variáveis do sistema supervisório, desenvolver um sistema de nome das variáveis e usar pastas de arquivos para organizar variáveis.

• Planejamento dos alarmes: Devem-se definir as condições de acionamento dos alarmes, a forma na qual ele vai ser apresentado ao operador e se necessário enviar mensagens com ações a serem executadas, esses alarmes tem como principais funções as de atentar o operador para mudanças no processo, sinalizar alguma situação e dar ideia da condição geral do sistema.

• Planejamento da hierarquia de navegação entre telas: Consiste na navegação entre as telas que fornecem progressivamente detalhes das plantas à medida que se navega no aplicativo.

• Desenho das telas: Nessa etapa, deve ser determinados, símbolos e cores, nome dos botões, posição dos botões na tela, sendo necessário para facilitar o entendimento do processo.

• Gráficos de tendência: Apresentam como as variáveis mudam ao longo do tempo.

• Acesso e segurança: Necessário para restringir o acesso de pessoas sem autorização ao sistema.

• Padrão industrial: Essa etapa é necessária para que o sistema supervisório desenvolvido tenha comunicação com outros aplicativos do sistema operacional utilizado.

A tela de um sistema supervisório em uma sala de controle e supervisão pode ser vista na figura 1.

Figura 1 - Sala de supervisão e controle de processos.

(Fonte : [11]).

1.1.2 Sistemas SCADA.

Os sistemas supervisórios modernos têm várias ferramentas, além de ser possível visualizar na tela tudo o que está ocorrendo no processo industrial, também é tarefa do sistema supervisório controlá-lo, esse tipo de supervisório é denomina SCADA (*Supervisory Control And Data Aquisition*), as variáveis retiradas da planta são processadas por uma unidade de processamento e após a efetivação do processamento teremos o *feedback* do que é necessário ser reajustado no processo, esse sinal é recebido por um acionamento que sinalizará para o atuador que está na planta, realizar sua tarefa **[02] [04]**.

Os primeiros sistemas SCADA desenvolvidos davam aos operadores informações periódicas do estado corrente do processo industrial, supervisionando sinais representativos de medidas e estados de dispositivos, através de painéis de lâmpadas e indicadores, sem que houvesse qualquer interface aplicacional com o operador **[01][10][12]**. Uma figura de um esquema de supervisório usado antigamente pode ser visto na figura 2.

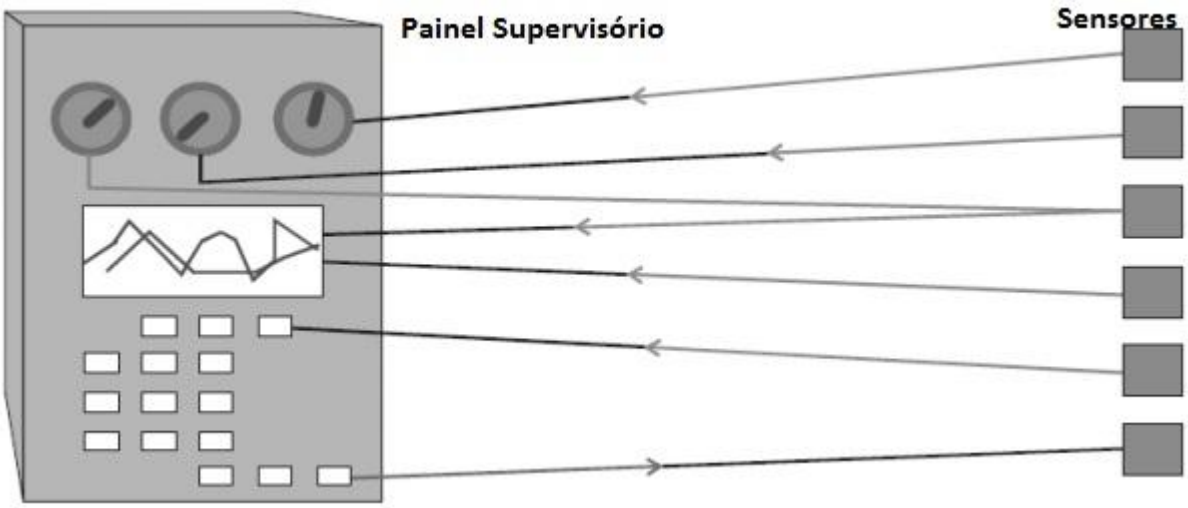

Figura 2 - Sistema supervisório antigo.

(adaptado de [12]).

Nos tempos modernos, os sistemas de automação industrial fazem uso de tecnologias de computação e comunicação para automatizar o controle e supervisão dos processos, efetuando coleta de dados em ambientes complexos, geralmente separados geograficamente, e apresentando de modo amigável para o operador, com recursos gráficos avançados através de uma Interface Homem Máquina (IHM) **[03]**.

Os sistemas SCADA identificam os *tags*, que são variáveis numéricas ou alfanuméricas envolvidas na aplicação desenvolvida, podendo executar operações matemáticas, lógicas, com vetores ou *strings* entre outras ou representar pontos de entrada/saída de dados do processo que está sendo controlado. Neste caso, corresponde às variáveis do processo real, como temperatura, nível, vazão entre outros, se comportando como a ligação entre o controlador e o sistema. É com base nos valores das *tags* que os dados coletados são apresentados ao usuário **[03][05]**.

As condições de alarmes do processo são verificadas pelos sistemas SCADA, essas situações são identificadas quando o valor da *tag* atinge uma faixa ou condição pré-estabelecida, sendo possível programar a gravação de registros em Bancos de Dados, ativação de som, mensagem, e-mail, celular, para alertar a pessoa responsável pelo processo **[12] [05]**. A arquitetura básica de um sistema SCADA, usando como elemento de captação de dados e controle um CLP pode ser vista na figura 3, onde são apresentados seus principais componentes.

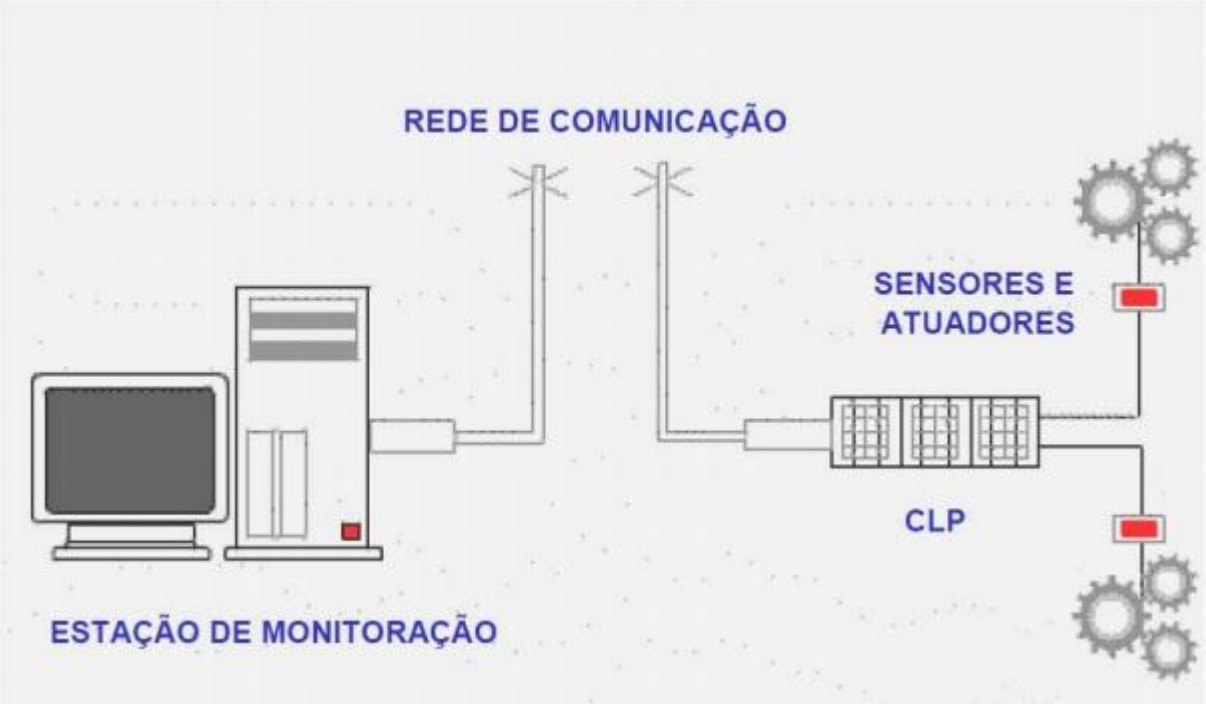

Figura 3 - Componentes Físicos de um sistema SCADA.

(Fonte: [13]).

1.2 Microcontroladores.

1.2.1 Introdução

A sigla em inglês *PIC* significa (*Programmable Integrated Controller***),** ou seja, em português um controlador integrado programável. Segundo PEREIRA (2003) **[07]**, na memória interna dos microcontroladores podem ser armazenados dois tipos de informações: instruções de controle, que corresponde ao programa que se executa, e instruções de dados, para manipular os dados como variáveis e constantes. Eles podem ser programados nas linguagens computacionais C, Basic e Assembler, que vai depender do compilador a ser usado.

Os PIC são constituídos de vários periféricos integrados em um único chip, dentre eles temos a memória de programa, memória de dados, portas de entrada e/ou saída paralela, temporizadores (*timers*), contadores, comunicação serial, *PWM's*, conversores analógico-digitais, entre outros recursos **[15]**.

1.2.2 PIC 18F4550

Os microcontroladores PIC18F4550 pertence a família de PIC18F da *Microchip*®. E está entre os de tecnologia mais avançada da *Microchip*. Este microcontrolador é muito usado devido a sua versatilidade e a funcionalidade de seus recursos. As principais características do PIC18F4550 estão listadas abaixo:

- Memória de programa *Flash* de 32k *bytes*.
- Memória de dados RAM de 2048 *bytes*.
- Memória EEPROM de 256 *bytes*.
- 35 Pinos de I/O. 8
- Comunicação serial EUSART.
- 13 Canais de entrada analógica de 10bits.
- 02 Comparadores.
- Canal de comunicação *I2C Master*.
- Canal de comunicação SPI.
- USB 2.0 *Full Speed*.
- Oscilador interno selecionável de 31KHz a 8MHZ.
- Entrega ou drena até 25 mA por pino.
- Três interrupções externas.
- Quatro *Timers* internos (TMR0, TMR1, TMR2, TMR3).
- Dois módulos - capture/compare/PWM.
- *Master Synchronous Serial Port* (MSSP).
- Disponibilidade em padrão DIP 40 pinos.

- Frequência de operação até 48 MHz.

O PIC18F4550 possui gerenciamento de *clock* especial, pois, no módulo USB interno, ele necessita de uma frequência 48 MHz, esta frequência é obtida através de um PLL interno (*Phase Locked Loop*) que multiplica o valor do oscilador e posteriormente o repassa aos blocos de USB e CPU, cada qual com sua frequência de trabalho independente. Essa conversão é necessária para que o módulo USB trabalhe no modo *Full Speed* (12 Mb/s) **[08]**.

Os periféricos do PIC18F4550 usados nesse trabalho serão descritos brevemente a seguir, levantando suas principais características.

- Módulo de conversão Analógica para Digital de 10-Bits.

O módulo conversor analógico-digital (A / D) tem 10 entradas que podem ser configuradas como analógicas para os dispositivos 40 pinos. Este módulo permite a conversão de um sinal de entrada analógico para um digital de 10 bits correspondentes número **[08]**.

- Portas de I/O(Entrada /saída).

Estes pinos geralmente são conhecidos por agrupamentos (portos). As I/O estão agrupadas em "Ports". No PIC 18F4550 temos o Port A com 6 postas, o Port B, Port C e Port D com oito portas, e o Port E três portas. Cada porta pode funcionar como entrada ou saída ou ambos, de acordo com o que for configurado no programa. Algumas portas podem ser configuras como Digitais ou como Analógicas. Quando configurada como digitais, os sinais são níveis de tensão de zero e cinco volts. Quando um pino é setado, ele envia um sinal de cinco volts, e quando é resetado, este nível de tensão baixa para zero volt. Porém, essas portas tem limite de acionamentos, não podendo liberar correntes acima do máximo especificado na folha de dados do componente **[08]**. Maiores informações podem sobre esse microcontrolador, pode ser encontrada na sua folha de dados e em outras várias bibliografias como **[08], [15], [16], [17].**

1.2.3 Módulo USB

O padrão USB foi criado para ser um padrão industrial de extensão para a arquitetura atual dos PC’s, suporta transferência de dados entre computadores e periféricos e é usado em

aplicações visando o aumento da produtividade. Várias são os motivos que determinam a definição da arquitetura USB para efetuar a comunicação entre PIC e PC:

- Facilidade na adição de periféricos ao PC;
- Baixo custo, suportando taxas de transferência de até 12 Mbps;
- Conecta a várias configurações de PC;
- Previsão de um padrão de interface capaz de espalhar-se rapidamente entre novos produtos;
- Possibilidade de criação de novas classes de dispositivos capazes de aumentar a capacidade dos computadores pessoais **[18]**.

A família de PIC 18F4550 contém um mecanismo de interface serial USB (*Serial Interface Engine - SIE*) compatível com modo de alta velocidade (*full-speed*) e com o de baixa velocidade (*low-speed*) que permite rápida comunicação entre qualquer USB anfitrião *(host)* e o microcontrolador PIC. O SIE pode ser interfaceado diretamente ao USB, utilizando um transceptor interno, ou pode ser conectado através de um transceptor externo. Um regulador interno de 3.3 V está também disponível para energizar o transceptor interno em aplicações de 5 V **[08]**. Maiores informações sobre o padrão USB pode ser encontrado em **[08]** e **[09].**

1.2.4 Conversor Digital/Analógico Malha R-2R

No presente trabalho será necessária a interface entre o PIC e o Inversor de Frequencia da WEG, para isso será usadas as portas analógicas do inversor, como o 18F4550 não possui um conversor digital para analógico, foi contruido um periférico externo para fazer essa conversão. O periférico é explicado a seguir.

Segundo TOCCI (2007) **[19]** a Conversão Digital para Analógica é o processo onde o código digital é convertido em tensão/corrente que é proporcional ao valor digital. No caso desse trabalho as entradas digitais provenientes do microcontrolador são convertidas em um valor de tensão analógica, na faixa de 0 a 5 V. O circuito usado para conversão D/A externamente no PIC é chamado de “rede R-2R”.

A resolução do conversor foi adotada como sendo igual ao do microcontrolador PIC18F4550, que é de 8 bits, portanto, a resolução adotada para a conversão D/A, possibilita $2^8 = 256$ valores de tensão possíveis.

1.3 O Ambiente de Desenvolvimento Microsoft Visual Studio 2010 Express

O *Microsoft Visual Studio* é um pacote de programas da *Microsoft* para desenvolvimento de *software* especialmente dedicado ao *.NET Framework* e às linguagens *Visual Basic (VB), C, C++, C# e J#.* A versão utilizada no projeto é a *Visual Studio 2010 Express* que é uma versão distribuída gratuitamente pela *Microsoft*. O *Visual Studio* também é um produto de desenvolvimento na área *web*, usando a plataforma do *ASP.NET*. Sendo a linguagem mais usada nessa plataforma a *VB. NET e o C#* **[21]**.

O *.NET Framework* é um ambiente de tempo de execução gerenciado, que proporciona uma variedade de serviços para os diversos aplicativos em execução. Ele consiste em dois principais componentes, que são: o *Common Language Runtime* (CLR), que é o mecanismo de execução e manipulação dos aplicativos que estão em execução e a biblioteca de classes do *.NET Framework*, que fornece uma biblioteca de código testado e, reutilizável que os desenvolvedores podem chamar a partir dos seus próprios aplicativos. Alguns dos serviços que o *.NET Framework* oferece aos aplicativos em execução incluem **[30]**:

- Gerenciar memória. Em várias linguagens de programação, os programadores são os responsáveis por alocar e liberar memória e a manipular por tempo de vida do objeto. Em aplicativos *.NET Framework*, o *CLR* fornece esses serviços em nome do aplicativo.

- Um sistema do tipo comum. Nas linguagens de programação tradicionais, os tipos básicos são definidos pelo compilador, o que de certa forma complica a interoperabilidade entre linguagens. *No .NET Framework*, os tipos básicos são definidos pelo tipo do sistema *.NET Framework* e são de uso comum em todas as linguagens que direcionam o *.NET Framework.*

- Uma biblioteca abrangente de classe. Ao invés de escrever uma grande quantidade de código a fim de lidar com operações de programação comuns em baixo nível, os programadores podem usar facilmente uma biblioteca de tipos a partir da Biblioteca de Classes do *.NET Framework*.

- Compatibilidade de versão. Com raras exceções, os aplicativos que são desenvolvidos com o uso de uma versão específica do *.NET Framework* podem executar sem alteração em uma versão posterior.

- Execução lado a lado. O *.NET Framework* ajuda a resolver conflitos de versão permitindo que várias versões do *common language runtime* existam no mesmo computador. Isso significa que várias versões de aplicativos também podem coexistir, e que um aplicativo pode executar na versão do *.NET Framework* com a qual foi compilada.

1.3.1 Introdução

O *Microsoft Visual Studio 2010 Express* é um ambiente de desenvonvimento completo, sendo, um Ambiente de Desenvolvimento Integrado (IDE em inglês). A IDE é o *Framework* de desenho no qual é possível a criação de aplicativos visuais. No projeto foi usada a ferramenta *Visual C#,* que está contida dentro das ferramentas do *Visual Studio 2010 Express*, essa linguagem foi escolhida por ser uma das mais populares dentre as outras **[21]**.

O *Visual **C#*** é uma linguagem de programação orientada a objetos, foi desenvolvida pela *Microsoft* e faz parte da plataforma .NET. A linguagem C#, foi baseada na linguagem C++ e tem muitos elementos da linguagem Pascal e Java **[21][22]**.

No princípio a plataforma .NET teve suas bibliotecas criadas em *Simple Managed C (SMC)*, portanto, isso restringiu o desenvolvimento na plataforma. No começo de 1999 liderada por Anders Hejlsberg, foi montada uma equipe de desenvolvedores, com o intuito de criar uma nova linguagem para a plataforma .NET. Esta nova linguagem foi desenvolvida para desatrelar a plataforma .NET de outras linguagens, pois o código das linguagens já existentes restringia o desenvolvimento da plataforma **[23][24]**.

Primeiramente, a nova linguagem foi criada com o nome de *Cool*. No momento do lançamento da plataforma .NET, mudaram o nome da linguagem para C#. A criação da

linguagem C# estimulou o desenvolvimento do .NET, pois a plataforma não necessitava mais se moldar a nenhum código de alguma outra linguagem já existente. O C# foi criado especificamente para .NET, sendo que muitas outras linguagens tem suporte ao C#. Algumas destas linguagens são VB.NET, C++ e J#. Embora a linguagem C# seja considerada muito semelhante ao Java, existem também varias diferenças **[23][24]**. A página inicial do *Microsoft Visual Studio C# 2010 Express* é apresentada na figura 4.

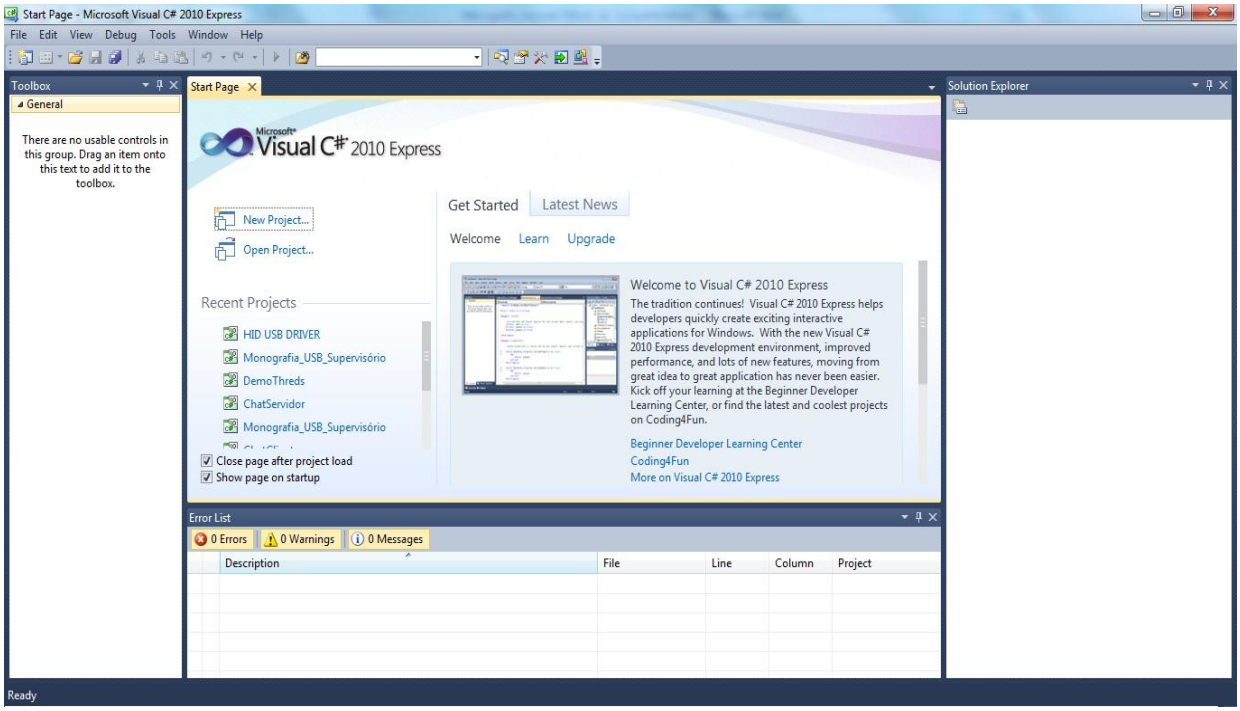

Figura 4 - Página Inicial *Microsoft Visual C# 2010 Express*.

1.3.2 Classe USBHidPort

Essa classe USB apresenta dispositivos que normalmente apresentam facilidades para os operadores se comunicarem com os computadores. A instalação é bem simples, pois, já existem protocolos de comunicação previamente definidos para estes equipamentos. São exemplos de dispositivos *HID* os mouses, teclados, controladores de jogos, entre outros. Na maioria dos casos, implementar estes dispositivos não apresentam grandes dificuldades, pois os protocolos de comunicação, normalmente já estão disponíveis prontos e já estão instalados nos sistemas operacionais, não necessitando assim de *drivers* para comunicação **[02]**.

Nesse projeto foi utilizada a biblioteca do Visual C# da classe *USBHidPort*, a biblioteca para comunicação é a *UsbLibrary.dll*, que está disponível em <*www.codeproject.com/KB/cs/USB_HID/usb_hid.zip*> que foi acessado no dia 28 de dezembro de 2013.

1.3.3 Controles de Rede (TCP/IP)

Como se sabe, a Internet é uma rede pública de comunicação de dados de alcance mundial, com controle descentralizado e que utiliza o conjunto de protocolos TCP/IP como base para a estrutura de comunicação e seus serviços de rede. Isto se deve ao fato de que a arquitetura TCP/IP fornece não somente os protocolos que habilitam a comunicação de dados entre redes, mas também define uma série de aplicações que contribuem para a eficiência e sucesso da arquitetura. Entre os serviços mais conhecidos da Internet estão o correio-eletrônico (protocolos SMTP, POP3), a transferência de arquivos (FTP), o compartilhamento de arquivos (NFS), a emulação remota de terminal (Telnet), o acesso à informação hipermídia (HTTP), conhecido como WWW (*World Wide Web*). O protocolo TCP/IP é um acrônimo para o termo *Transmission Control Protocol/Internet Protocol Suite*, ou seja é um conjunto de protocolos, onde dois dos mais importantes (o IP e o TCP) deram seus nomes à arquitetura. O protocolo IP, base da estrutura de comunicação da *Internet* é um protocolo baseado no paradigma de chaveamento de pacotes (*packet-switching*) **[26]**.

Os protocolos TCP/IP podem ser utilizados sobre qualquer estrutura de rede, seja ela simples como uma ligação ponto-a-ponto ou uma rede de pacotes complexa. Como exemplo, pode-se empregar estruturas de rede como *Ethernet*, *Token-Ring*, FDDI, PPP, ATM, X.25, *Frame-Relay*, barramentos SCSI, enlaces de satélite, ligações telefônicas discadas e várias outras como meio de comunicação do protocolo TCP/IP. A arquitetura TCP/IP, assim como OSI realiza a divisão de funções do sistema de comunicação em estruturas de camadas. Em TCP/IP as camadas são: Aplicação, Transporte, Inter-Rede, Rede **[26]**. A figura 5 apresenta as camadas do protocolo.

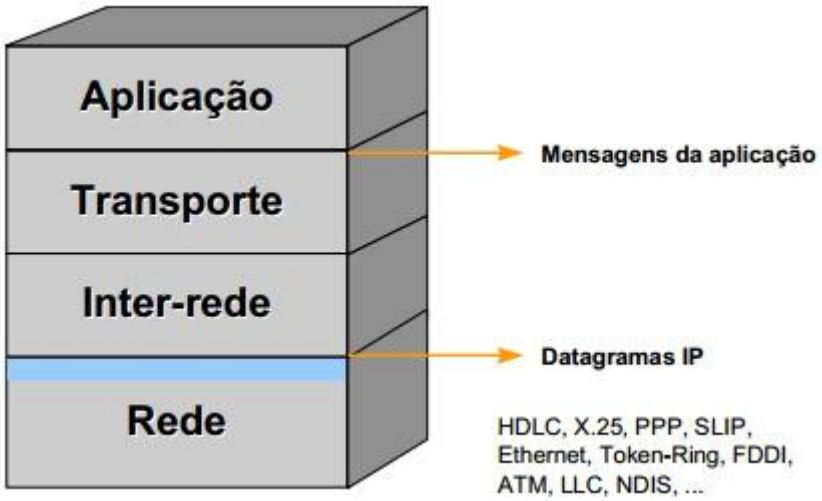

Figura 5 - Camadas protocolo TCP/IP.

(Fonte : [26]).

Na linguagem de programação C#, temos os controles responsáveis por criar a comunicação via rede, os quais criam um IP e uma porta de identificação do programa, que são os denominados *Sockets*. O *namespace System.Net.Sockets* fornece uma implementação gerenciada da interface do *Windows Sockets (Winsock)* para aplicativos que necessitam controlar o acesso à rede. O *TcpClient, TcpListener,* e *UdpClient* são classes que encapsulam os detalhes da criação de conexões TCP e UDP com a Internet **[25]**.

A Classe *The Socket* proporciona um conjunto poderoso de métodos e propriedades para comunicações de rede. A classe *Socket* permite transferência de dados tanto síncrona quanto assíncrona usando qualquer um dos protocolos de comunicação na enumeração *ProtocolType.* Essa classe segue o padrão de nomenclatura do *.NET Framework* para métodos assíncronos. **[25]**.

1.4 Objetivo

O trabalho tem como objetivo desenvolver um *software* supervisório, para monitorar um sistema de acionamento por Inversor de Frequência da WEG. Para isso, serão utilizadas técnicas de programação na linguagem C#, para criação de uma interface gráfica. E também será usado um microcontrolador para fazer a comunicação entre o inversor de frequência e o computador.

Por outro lado, o objetivo específico do supervisório desenvolvido é ter acesso remoto via rede *Ethernet*, pelo protocolo TCP/IP, para isso será utilizada uma interface *bridge*, que servirá como uma ponte de ligação à interface servidora, com isso, a interface servidor possibilitará a conexão de um número muito maior de clientes conectados a ele contando também com recursos de visualização gráficos.

2 *Materiais e Métodos*

O desenvolvimento desse trabalho foi realizado nos laboratórios de eletrônica do Departamento de Engenharia Elétrica da Universidade Federal de Viçosa (DEL-UFV), que consistiram no estudo das linguagens de programação C e C#, para criação do sistema de aquisição de dados e de controle do processo em questão. Para isso, foram desenvolvidos os programas do microcontrolador na linguagem C, no compilador CCS PCWHD versão demonstração **[31]**, para a aquisição e transmissão de dados para o computador via USB, e na linguagem C# foram criados os programas, de recepção via USB e transmissão/recepção via rede, ou seja, o programa *bridge*, para fazer a conexão entre o servidor e o sistema. Também foi criado o programa servidor de recepção/transmissão via rede, ou seja, o programa servidor. O esquema do sistema desenvolvido é apresentado na figura 6.

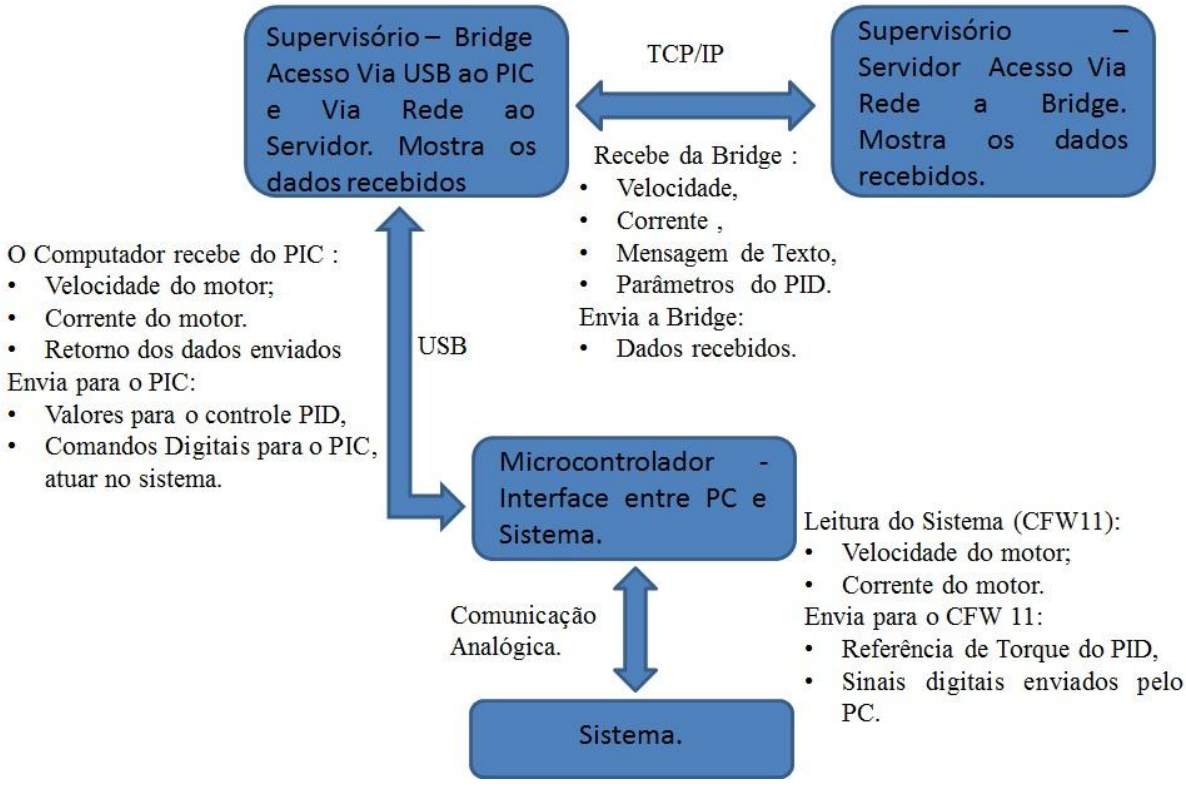

Figura 6 - Diagrama de Blocos do Sistema.

2.1 Criação do Programa do Microcontrolador

Nessa etapa escolheu-se o PIC 18F4550, devido aos diversos fatores apresentados anteriormente, principalmente pelo fato desse componente já possuir um módulo USB integrado e possuir portas de I/O suficientes para o desenvolvimento do trabalho.

O microcontrolador foi utilizado para fazer a interface entre o computador e o Inversor de Frequência CFW 11 da *WEG*, esse inversor possui vários tipos de comunicação, porém, foram utilizadas suas portas digitais e suas portas analógicas para o controle e supervisão do motor acionado por ele. Essa comunicação analógica e digital foi usada devido a não disponibilidade no laboratório, de módulos de expansão necessários para outros tipos de comunicação.

No inversor de frequência, temos duas portas analógicas que enviam uma tensão de 0 a 10 v, que são proporcionais a velocidade e a corrente do motor por ele acionado. Nesse equipamento também, temos uma porta de entrada analógica, que é usada para enviar uma referencia de velocidade para o inversor, para isso, será usado um sinal analógico criado pelo PIC através do conversor digital/analógico externo, explicado anteriormente. Também no CFW 11 temos portas digitais, que são ligadas as saídas digitais do PIC, e são responsáveis por dar a partida e a pausa no acionamento do motor.

As saídas analógicas do inversor são ligadas as portas do PIC, configuradas como conversores A/D. Com isso é possível monitorar a corrente e a velocidade do motor e fazer o seu controle de velocidade, a partir de um controle PID.

O módulo USB do programa do 18F4550 será responsável pelo envio das leituras feitas pelas suas portas analógicas para que, essas informações sejam mostradas pelo sistema supervisório e possam ser monitoradas por um operador. No processo de envio também será enviado um *byte* de *status*, que retornará se os comandos enviados pelo USB, do computador para o PIC, foram executados por ele.

A recepção do módulo USB no microcontrolador serve para que este receba os comandos de controle para serem enviadas as portas digitais do inversor, e assim, ligar e desligar o motor acionado por ele. Essa também é responsável, por receber os valores dos parâmetros do controle PID, enviados pela interface de controle no computador. O cálculo do

controle PID é feito internamente no PIC, pois, se for necessário que o inversor funcione sem supervisão é possível apenas retirando o cabo USB do computador.

Um esquema que representa o fluxograma do programa do microcontrolador é apresentado na figura 7, onde algumas etapas foram explicadas anteriormente.

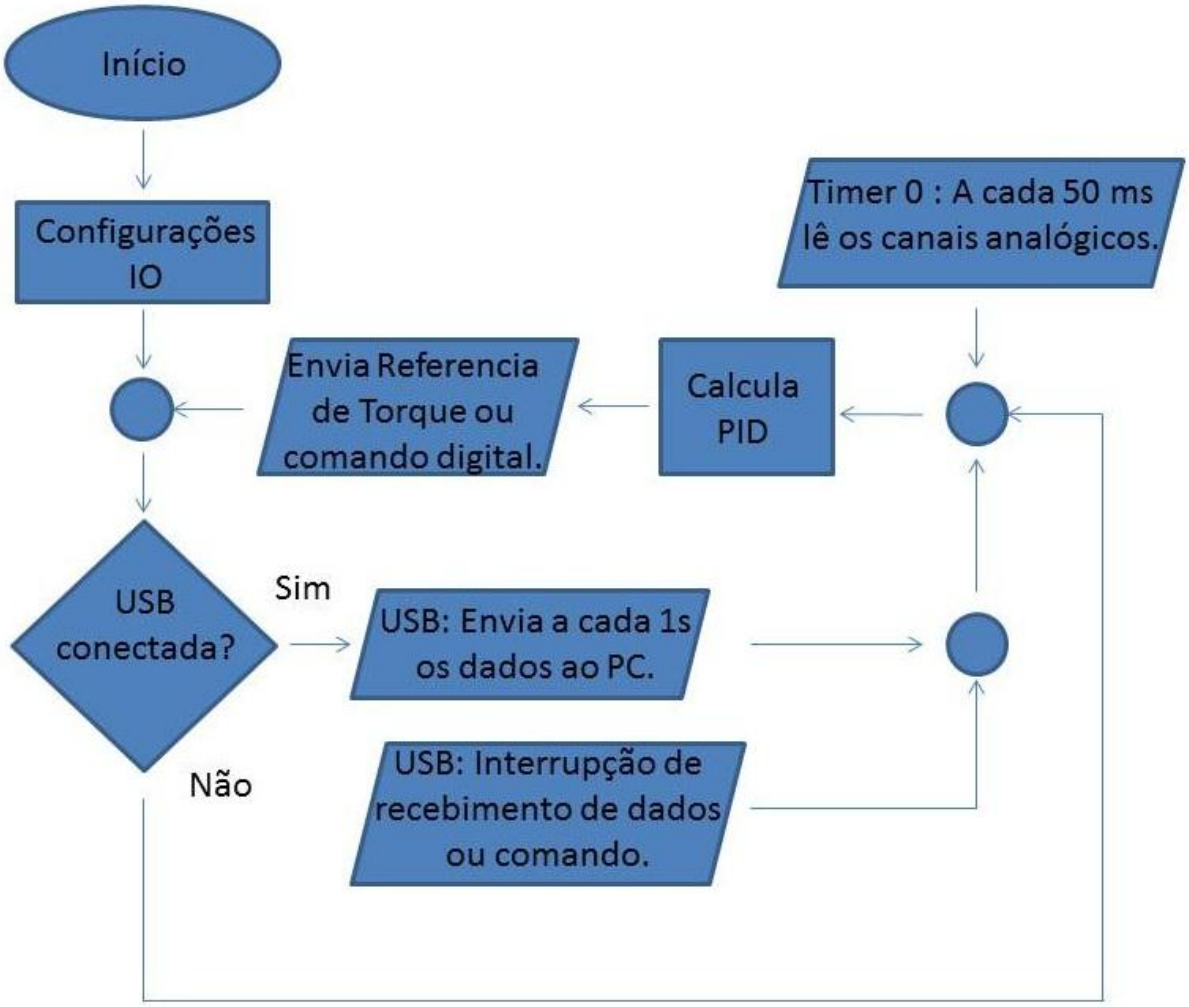

Figura 7 - Fluxograma do Programa do PIC.

No início do programa as variáveis são todas iniciadas com valores nulos, com isso, enquanto os parâmetros do PID, não forem passados pelo computador o cálculo do PID retornará uma referência nula de torque. E nas configurações é configurado e acionado o *Timer0* que faz a leitura dos canais A/D a cada 50ms.

2.2 Criação da Interface Bridge

A interface gráfica *bridge* como dito anteriormente foi criada no *Microsoft Visual C# 2010 Express*. Essa plataforma foi escolhida devido as suas várias ferramentas e algumas vantagens em relação a outras linguagens de criação de sistemas gráficos. Dentre as vantagens podemos destacar:

- A versão *Express* é distribuída gratuitamente pela *Microsoft*;
- A versão gratuita possui recursos de criação de interfaces gráficas, bem como todas as ferramentas necessárias para desenvolvimento desse trabalho;
- A linguagem é uma das mais populares, com isso é possível encontrar muito material para pesquisa.

Para a criação do supervisório, foi necessário fazer a inserção de algumas bibliotecas para que se pudesse fazer o uso dos controles USB, controles para mostrar os dados no formato de *Gauges* e de gráficos de tendências. Essas bibliotecas estão disponíveis na *internet* e o local de *download* é dado por **[27], [28] e [29]**. Para que estas ferramentas de controle das bibliotecas estejam disponíveis ao programa é necessário fazer com que o programa reconheça as bibliotecas. Para isso na janela do *Windows form*, no canto esquerdo da tela na aba *Toolbox*, clicando com o botão direito do mouse no item *General* dessa aba, como é mostrado na figura 8, aparece a tela de escolha das bibliotecas como mostrado na figura 9. Na janela exibida, deve-se clicar no botão *Browser*, e buscar um dos arquivos das bibliotecas baixadas por **[27], [28], [29]** e adicionar o componente à lista de itens da aba .NET *Framework* Componentes. Depois dessa inserção os controles estão disponíveis para serem usados no formulário.

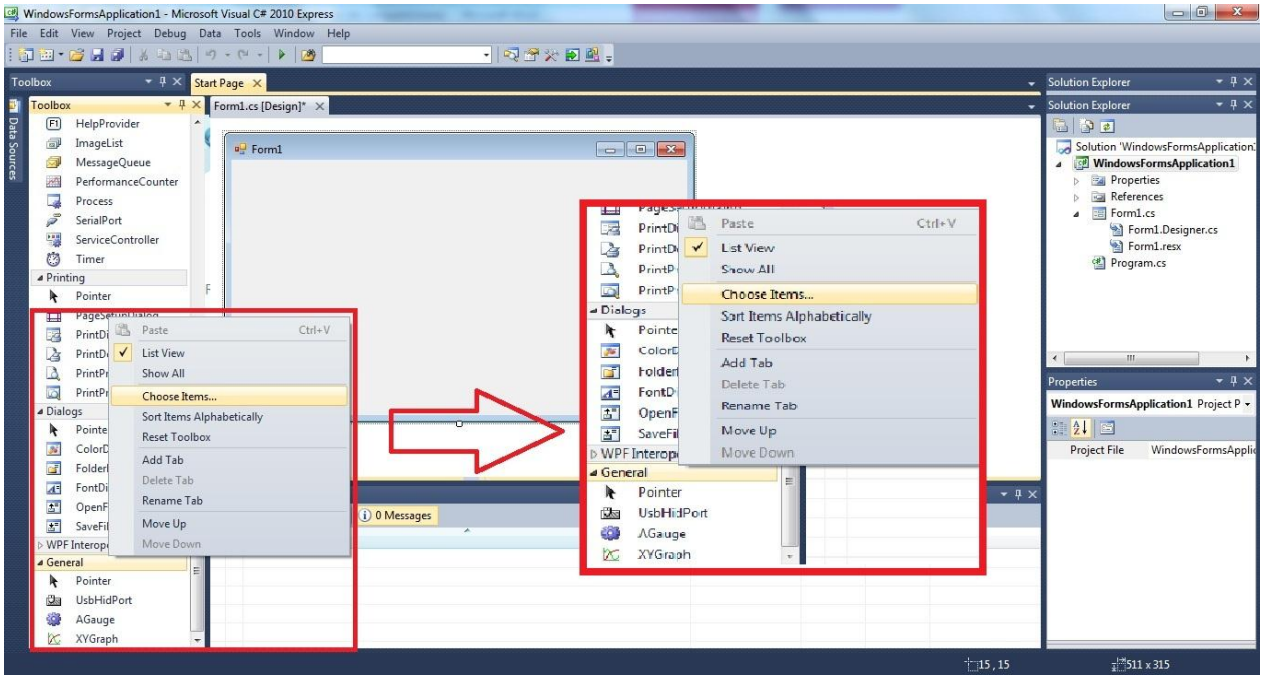

Figura 8 - Inserindo Bibliotecas de controle.

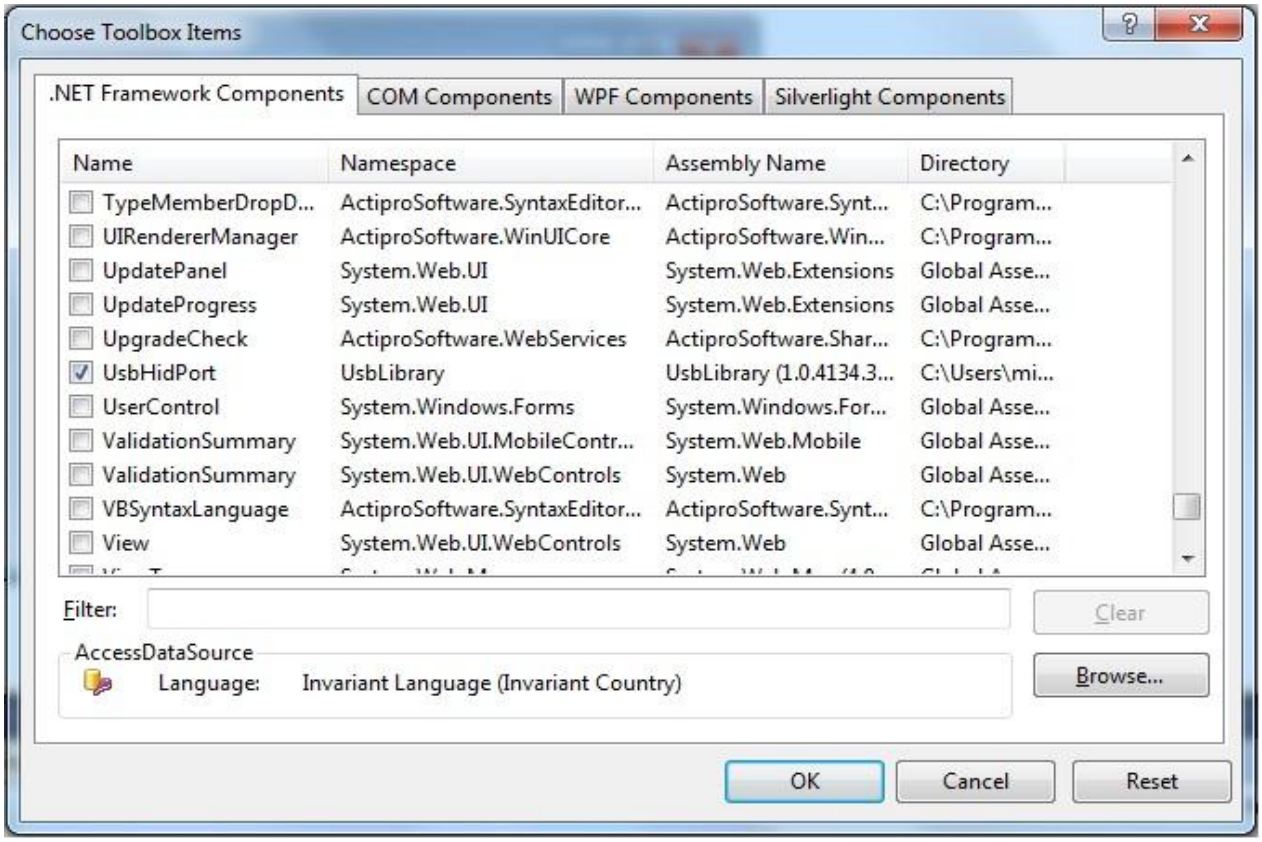

Figura 9 - Aba para Inserção das bibliotecas de controle.

Com os controles adicionados, foi possível criar um novo projeto do *Windows form Aplication*, no qual se podem colocar todos os comandos necessários para o projeto. Esses

comandos são botões, *gauges*, gráficos, *textbox* entre outros, um formulário em branco pode ser vista na figura 10 e alguns comandos disponíveis são apresentados na figura 11.

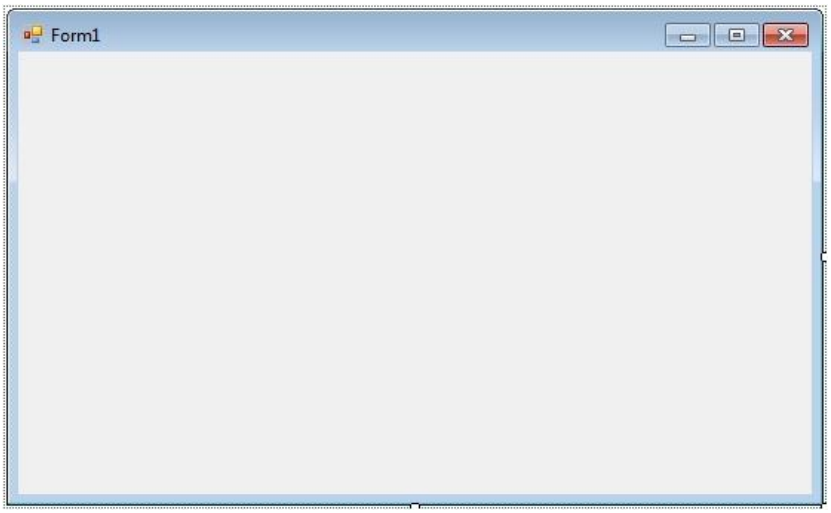

Figura 10 - Formulário em branco.

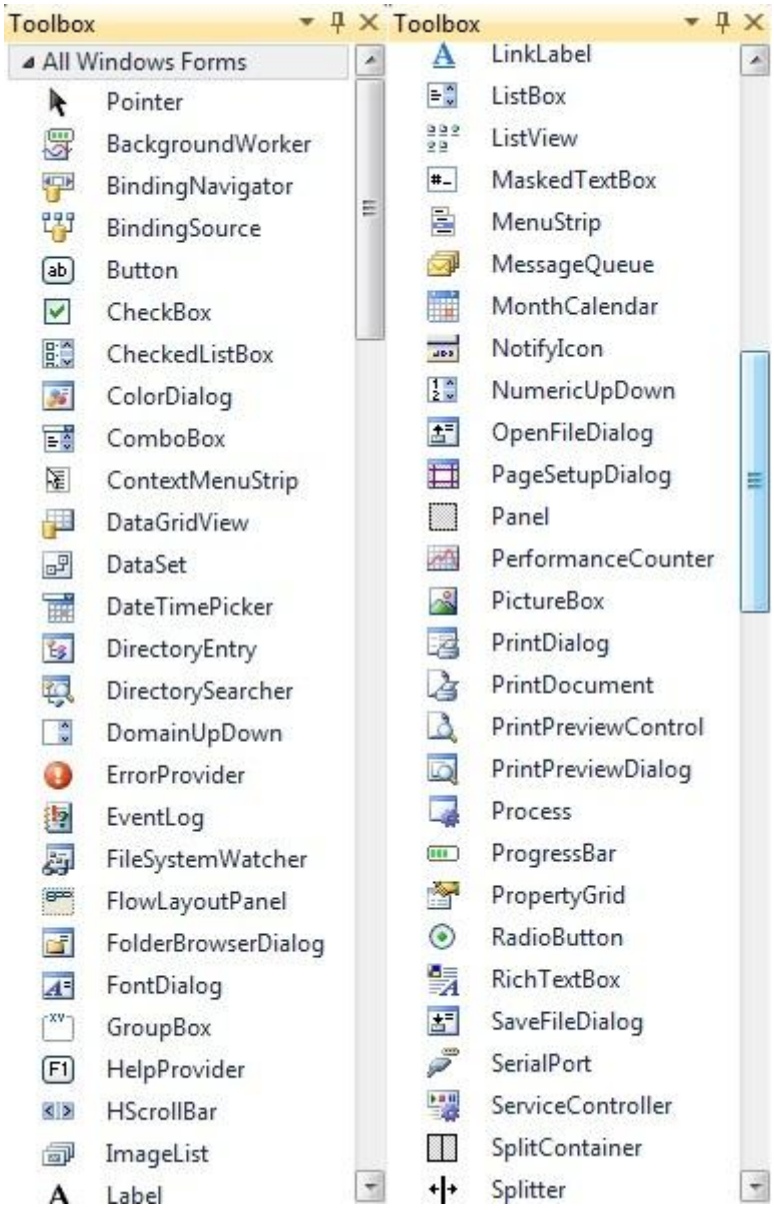

Figura 11 - Alguns controles do Formulário.

Para mostrar a velocidade do motor usou a visualização por *Gauges*, que são escalas que usam ponteiros para mostrar o valor das variáveis, como pode ser visto na figura 12 (a) e para mostrar o valor da corrente foi utilizado um gráfico de tendências como o mostrado na figura 12 (b).

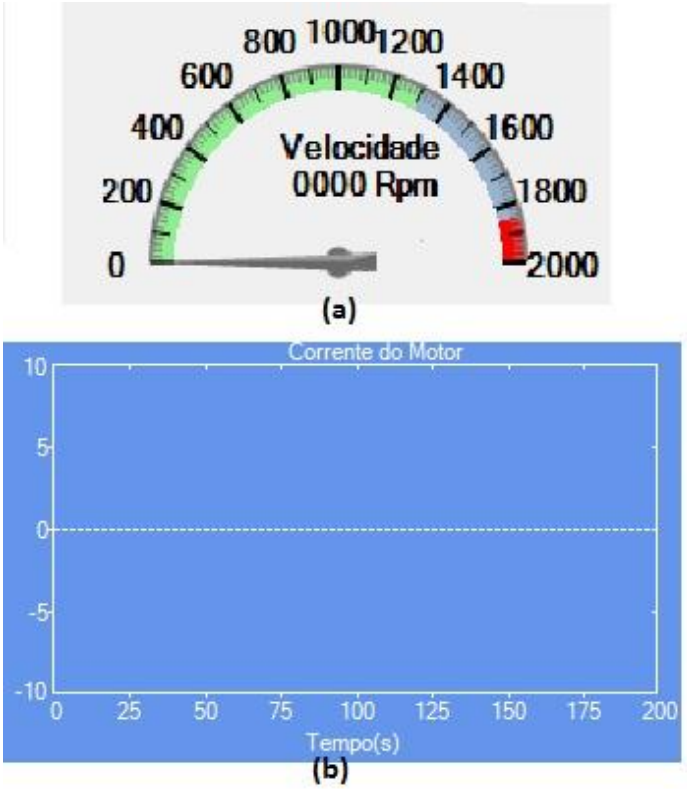

Figura 12 - (a) Representação por Gauge, (b) Gráfico.

Com o intuito de fazer um banco de dados para uso em futuras aplicações, o aplicativo conta com uma ferramenta em que é possível salvar os dados referentes às correntes e velocidade do motor. Como já mencionado anteriormente, os dados são armazenados em arquivos do formato**.txt*. Sendo que os valores ficam armazenados no mesmo arquivo **.txt*. E também para fazer uma monitoração correta dos dados, é gravada a data com dia, mês e ano e também o horário da aquisição com hora, minuto e segundo. Na figura 13 se pode ver um exemplo de como os dados são gravados nos arquivos **.txt* .

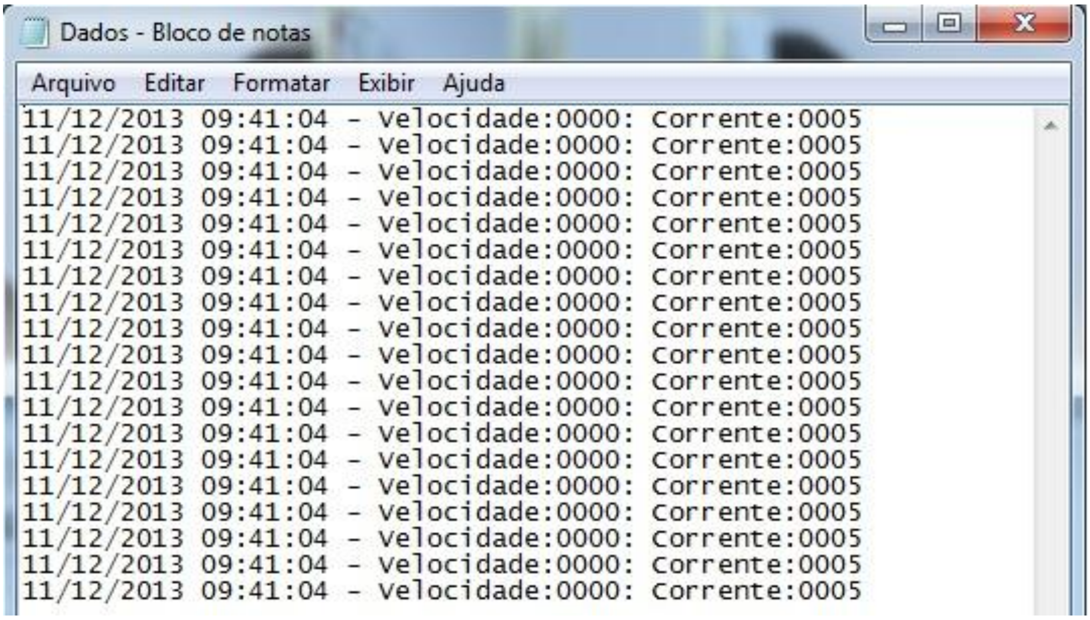

Figura 13 - Salvamento dos dados.

E depois da inserção da biblioteca USB, foi possível criar a classe responsável por receber e enviar via USB.

Para a comunicação via rede, foi usado o *namespace System.Net.Sockets* que fornece uma implementação gerenciada da interface do *Windows Sockets (Winsock)*, pois, o aplicativo necessita desse controle de rede. As classes *TcpClient*, *TcpListener* foram usadas para a criação de conexões TCP com a *Internet*, sendo assim foram feitas as funções de recepção de dados via rede e de envio de dados via rede, que serão explicadas adiante.

2.3 Criação da Interface Servidor

O supervisório servidor criado permite o acesso remoto do sistema monitorado pelo supervisório *bridge*, ou seja, aquele ligado diretamente via USB ao microcontrolador que capta as informações do inversor. O *software* servidor desenvolvido trata as conexões, armazenando-as em uma *hash table*. O sistema servidor é capaz de aceitar tantos clientes quanto forem permitidos pela *hash table*, e que também irá acompanhar todos os dados enviados e recebidos.

O aplicativo servidor é um pouco mais complexo do que a aplicação *bridge*, porque ele precisa manter informações sobre todos os clientes conectados, aguardar os dados de cada um e enviar mensagens de entrada para todos.

3 Resultados e Discussões

3.1 Programa e Hardware do Microcontrolador

O programa do microcontrolador desenvolvido tem como principais funções a leitura via entrada analógica, envio/recebimento via USB, processamento dos dados recebidos do supervisório e envio de comandos digitais, tanto para serem convertidos pela malha R-2R externa ou para informarem um comando digital diretamente ao CFW 11.

A primeira parte do programa do microcontrolador é onde são feitas as configurações dos *Hardwares* do PIC, através das *Fuses* e outras definições. Na primeira parte também, são incluídas as bibliotecas do compilador e as externas ao compilador com as funções necessárias no programa. Abaixo é apresentada essa parte do programa do microcontrolador com alguns comentários:

```
#include <18F4550.h> // Inclusão biblioteca do PIC utilizado
#device ADC=10; // Configuração da quantidade de bits utilizados no conversor A/D.
#fuses HSPLL, NOWDT, NOPROTECT, NOLVP, NODEBUG, USBDIV, PLL5, CPUDIV1, VREGEN.
#use delay (clock=48000000) //Define frequência de oscilação que é possível através do PLL interno do PIC com um Crystal externo de 20MHz (PLL5).

//Configura a Transmissão de dados :
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

//Define ao firmware que será utilizado o código de manipulação HID.
#DEFINE USB_HID_DEVICE  TRUE
/*Habilitação do endpoint de transmissão, aloca espaço na memória para os bytes de trasmissão*/
#define USB_EP1_TX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_TX_SIZE    5        //aloca 5 bytes para transmissão USB
/*Define a habilitação do endpoint de recepção e aloca espaço na memória para os bytes de recepção*/
#define USB_EP1_RX_ENABLE  USB_ENABLE_INTERRUPT
#define USB_EP1_RX_SIZE    8        //aloca 8 bytes para recepção da USB
// Inclusão das bibliotecas do CCS e outras externas
#include <pic18_usb.h>          //Microchip 18Fxx5x para o usb.c
#include <usb_desc_hid.h>  //Configuração da USB e Descritor do Dispositivo
#include <usb.c>  /*Manipula as configurações USB e obtém o relatório do descritor*/
#include <math.h>       //permite o uso de funções matemáticas
#include <stdlib.h>  // Uso do Atoi32 para conversão do Char recebido em Int32
```

Com as configurações feitas, temos que declarar as variáveis para que elas sejam utilizadas no programa. Temos que definir os *buffers* de recepção e envio de dados do microcontrolador, as variáveis que receberão os valores do conversor A/D e as variáveis que receberão os valores dos parâmetros enviados pelo supervisório via USB. As variáveis declaradas com alguns comentários podem ser vistas a seguir:

```
int32 t =0,g=0,h=0, v=0;// Recebe os valores dos parâmetros
int status = 0b00000000;// Envia o status dos comandos enviados ao PIC
float resultado_1=0; //Usado para multiplicar o int32 e transformar em float o parâmetro
//recebido
float m=0,n=0,o=0;//Recebem os parâmetros Kp, Ki,Kd em formato float

#bit s_L1 = status.0      //Bit de estado para o comando de ligar
#bit s_L2 = status.1      //Bit de estado para o comando de desligar
#bit s_L3 = status.2      //Bit de estado para o recebimento do valor da Vel. Ref.
#bit s_L4 = status.3      //Bit de estado para o recebimento do valor de Kp
#bit s_L5 = status.4      //Bit de estado para o recebimento do valor de Ki
#bit s_L6=  status.5      //Bit de estado para o recebimento do valor de Kd

int8 in_data[8];       //Buffer para os dados recebidos pela USB

char s[4]; char s_2[4]; char s_3[4]; char s_4[4]; char s_5[4]; char s_6[4]; /*Variáveis para
a recepção dos valores no formato char enviado pelo supervisório no computador.*/
////////////////////////////////////////////////////////////////
// Variáveis para a conversão analógica digital
int32 valor_adc01;
int32 valor_adc03;
////////////////////////////////////////////////////////////////
int8 out_data[20];      //Buffer para os dados enviados
int8 send_timer=0;      //Variável de controle de tempo
int8 send_adc_Timer=0; //Variável de controle de tempo para ADC
```

Outra parte importante do *software* do microcontrolador é configuração das portas analógicas, as quais fazem as leituras das tensões que são enviadas pelo inversor de frequência. Essa configuração é feita de maneira simples, primeiro é configurado o número de *bits*, como feito acima e depois às outras configurações adicionais. Essas configurações são mostradas a seguir:

```
setup_adc_ports(AN0_TO_AN3);    //Configura Portas 0 e 3 para ADC
setup_adc(ADC_CLOCK_INTERNAL);   //Configura clock interno para ADC
```

Para o envio de dados pela USB, é necessário formatar os dados, pois, a comunicação USB pelo modo HID são enviados pacotes de 8 *bits*. Os dados enviados ao computador é o resultado da conversão analógica/digital (A/D) dos canais 0 e 1 que monitoram as saídas analógicas do CFW11, que fornecem valores de tensão que são proporcionais aos valores de velocidade e corrente do motor acionado. Ambos os conversores A/D tem resolução de 10 *bits.* Outro dado que é enviado ao computador são os estados dos comandos digitais enviados (ligado/desligado) e o estado do recebimento dos valores dos parâmetros enviados pelo computador ao microcontrolador, com isso, o programa do computador tem a certeza que o *PIC* recebeu os comandos enviados por ele.

Para o estado dos valores e comandos recebidos, foi utilizado um pacote de 8 *bits* para o envio de todas as informações, sendo que cada um *bit* dos 8 funcionavam como confirmação da chegada de cada uma das instrução no *PIC*, sendo um dado ou comando. Porém para a conversão A/D, foi necessária a utilização de dois pacotes de 8 *bits* para o envio de resultados. Com isso, o número da conversão A/D (Valores de 0 a 1023) são quebrados em dois pacotes e enviado de 8 em 8 *bits*, ficando a cargo do aplicativo o tratamento dessas dados, juntando os pacotes e reagrupando o número.

Para dividir o resultado da conversão A/D em dois pacotes de 8 bits, foi utilizado o operador de deslocamento de *bits*. Como pode ser visto nas linhas de código abaixo, onde são armazenados os valores no *Buffer* de envio:

```
//Valores a serem armazenados nas posições 0 a 10 do vetor out_data
   out_data[0]= valor_adc01>>8;   //Armazena a parte alta da variável valor_adc01
   out_data[1]=(int) valor_adc01;   //Armazena a parte baixa da variável valor_adc01
   out_data[2]= valor_adc03>>8;   //Armazena a parte alta da variável valor_adc03
   out_data[3]=(int) valor_adc03; //Armazena a parte baixa da variável valor_adc03
   out_data[4]= status;      //Armazena o status dos comandos e parâmetros recebidos.
```

A função de envio de dados foi configurada para enviar os dados lidos pelo conversor A/D e dos estados dos comandos a cada 1s, portanto, foram configuradas as interrupções de configurações como se pode ver abaixo:

```
  while (TRUE) {
    usb_task();        //Verifica tarefas USB
    usb_debug_task();      //Verifica tarefas USB - Debug
    if (usb_enumerated())    //Se dispositivo conectado corretamente à USB
       {
      if (!send_timer)    //Verifica se variável = 0
         {
         send_timer=250;           //Retorna o valor 250 à variável send_timer
         send_adc_timer++;          //Incrementa a variável send_adc_timer
         if (send_adc_timer>=5)     //Envia os valores ADC em aprox. 1 segundo
           {
            set_adc_channel(0);      //Seleciona o canal analógico 0
            delay_us(1000);            //Atraso de 100us
     valor_adc01 = read_adc();/*Lê o canal 0 e armazena o valor em  valor_adc01*/
            set_adc_channel(3);       //Seleciona o canal analógico 3
            delay_us(1000);            //Atraso de 100us
     valor_adc03 = read_adc();  /*Lê o canal 3 e armazena o valor em valor_adc03*/
            send_adc_timer=0;          //zera a variável send_adc_timer
            }

  //Valores a serem armazenados nas posições 0 a 10 do vetor out_data
  out_data[0]= valor_adc01>>8;   //Armazena a parte alta da variável valor_adc01
  out_data[1]=(int) valor_adc01;   //Armazena a parte baixa da variável valor_adc01
  out_data[2]= valor_adc03>>8;   //Armazena a parte alta da variável valor_adc03
  out_data[3]=(int) valor_adc03; //Armazena a parte baixa da variável valor_adc03
  out_data[4]= status;      //Armazena o status dos comandos e parâmetros recebidos.

////////////////////////////////////////////////////////////////////////////
//Envia o vetor out_data à USB
////////////////////////////////////////////////////////////////////////////
  if (usb_put_packet(1, out_data, 5, USB_DTS_TOGGLE));
    s_L4 =0; s_L5 =0; s_L6 =0;}
```

Na parte de recebimento de dados pela USB, foi criada uma função que analisa qual dado está presente no *Buffer* de entrada, para saber se é um comando ou qual parâmetro do PID está presente na entrada. Para isso, no envio dos dados pelo programa do computador o primeiro pacote de 8 *bits* enviado é uma caractere, para que o microcontrolador possa reconhecer qual parâmetro é o enviado e salvar na variável correta. Também com essa função o PIC identifica qual comando chegou ao microcontrolador e uma porta do PIC é *setada* ou *resetada*, de acordo com o comando enviado pelo computador. A parte do código que mostra essa função e alguns dos testes pra identificar qual comando foi enviado e qual variável foi

recebida e fazer a conversão para o tipo *Float*, para ser usada no processamento interno do controle PID esta mostrada abaixo:

```
if (usb_kbhit(1))       //verifica se existe algum dado vindo da USB
    {
    //Recebe 8 bytes e armazena na variável in_data
    usb_get_packet(1, in_data, 8);

    //Verifica se os índices 0 e 1 da variável in_data formam a string 'LD'
    if (in_data[0] == 'B' && in_data[1] == 'T')
      {
       switch(in_data[2])   //Verifica o índice 2 da variável in_data
            {
            case '1': Output_toggle(L1);   //Caso seja 1
                    s_L1 = ~s_L1;       //altera estado da porta RD0
                    break;
            case '2': Output_toggle(L2);   //Caso seja 2
                    s_L2 = ~s_L2;      //altera estado da porta RD1
                    break;
            }
      }

       if (in_data[0] == 'P')//Teste do primeiro character para identificar dados.
      {
       s[0] = in_data[1];// Salva dados em um vetor char
       s[1] = in_data[2]; // Salva dados em um vetor char
       s[2] = in_data[3]; // Salva dados em um vetor char
       s[3] = in_data[4]; // Salva dados em um vetor char
       s_2[0] = in_data[5]; // Salva dados em um vetor char
       s_2[1] = in_data[6]; // Salva dados em um vetor char
       s_2[2] = in_data[7]; // Salva dados em um vetor char
       s_2[3] = 0x00; // zera a ultima posição do vetor

       resultado_1 = 0.0001;// Valor que parametriza com valor enviado pelo Computador
       t = atoi32(s);/* Comando da biblioteca stdlib.h, para transformar o vetor de Char
em uma variável int32*/
       m = resultado_1*t;//Transforma o valor int32 recebido em Float.
       s_L4 = 1;    //Sinaliza a chegada do valor do parâmetro P, para ser enviado ao PC.
      }
```

A placa de circuito impresso feita para o microcontrolador, foi desenvolvida na versão gratuita do *software EAGLE 6.4* e pode ser vista na figura 14, foi feita uma placa genérica, com apenas barras de pinos para ligadas aos pinos do PIC, pois com isso vários sistemas diferentes podem ser acionados com essa montagem. A figura 15 apresenta o diagrama

elétrico de todo o sistema desenvolvido e a figura 16 apresenta a placa feita e montada em uma fenolite.

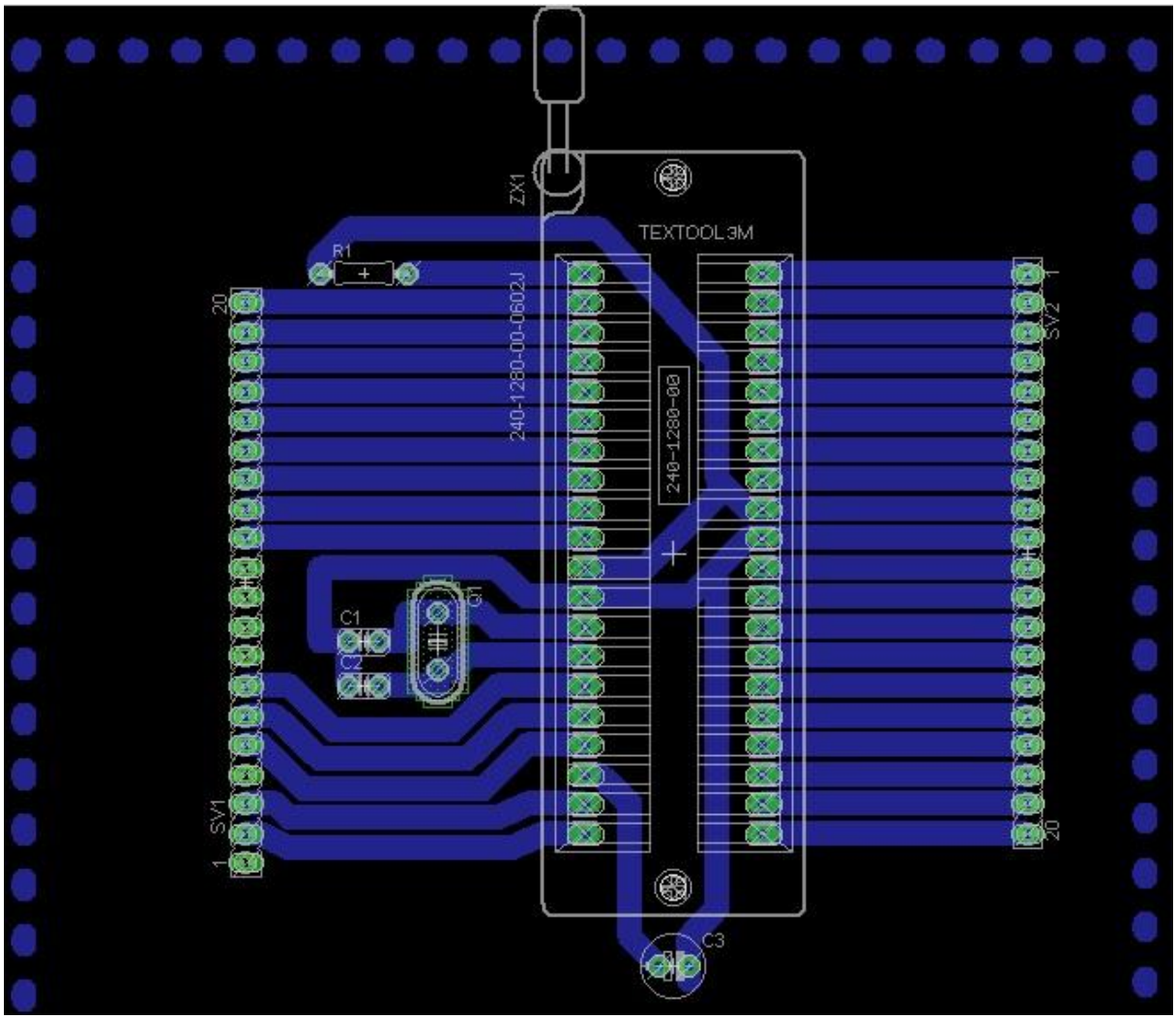

Figura 14 - Placa do 18F4550 no Eagle.

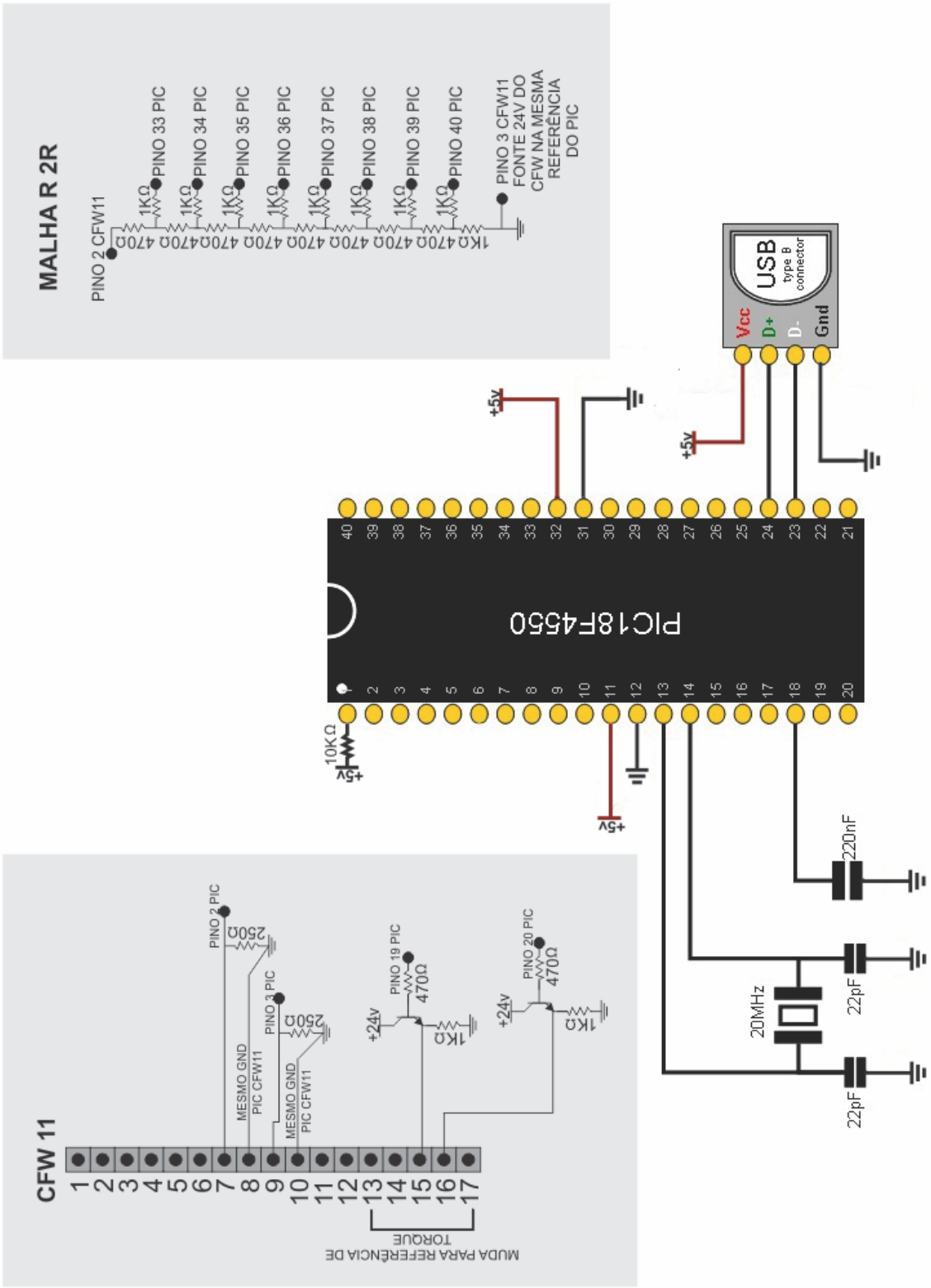

Figura 15 – Diagrama Elétrico do Sistema.

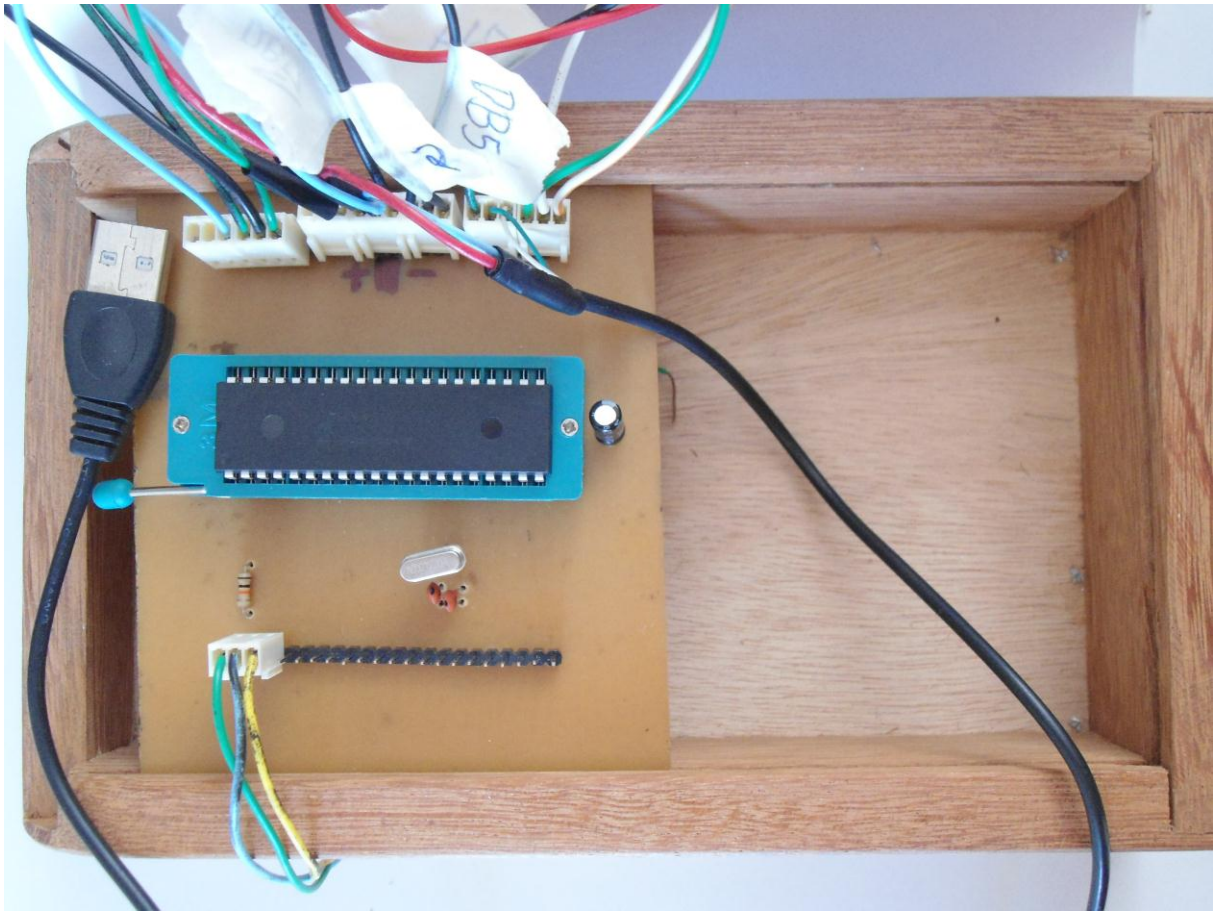

Figura 16 - Placa com microcontrolador.

3.2 Interface Bridge do computador

A interface *bridge* foi desenvolvida no IDE do *Microsoft Visual C# 2010 Express*, e tem como principais funções:

- Comunicação USB envia/recebe dado;
- Visualização dos dados em gráficos e *Gauges*;
- Salvamento dos dados em arquivos *.txt;
- Comunicação Via rede *Ethernet* (TCP/IP).

A seguir na figura 17 é apresenta a interface desenvolvida na íntegra, onde seus componentes principais serão abordados com maior detalhe ao decorrer desse item.

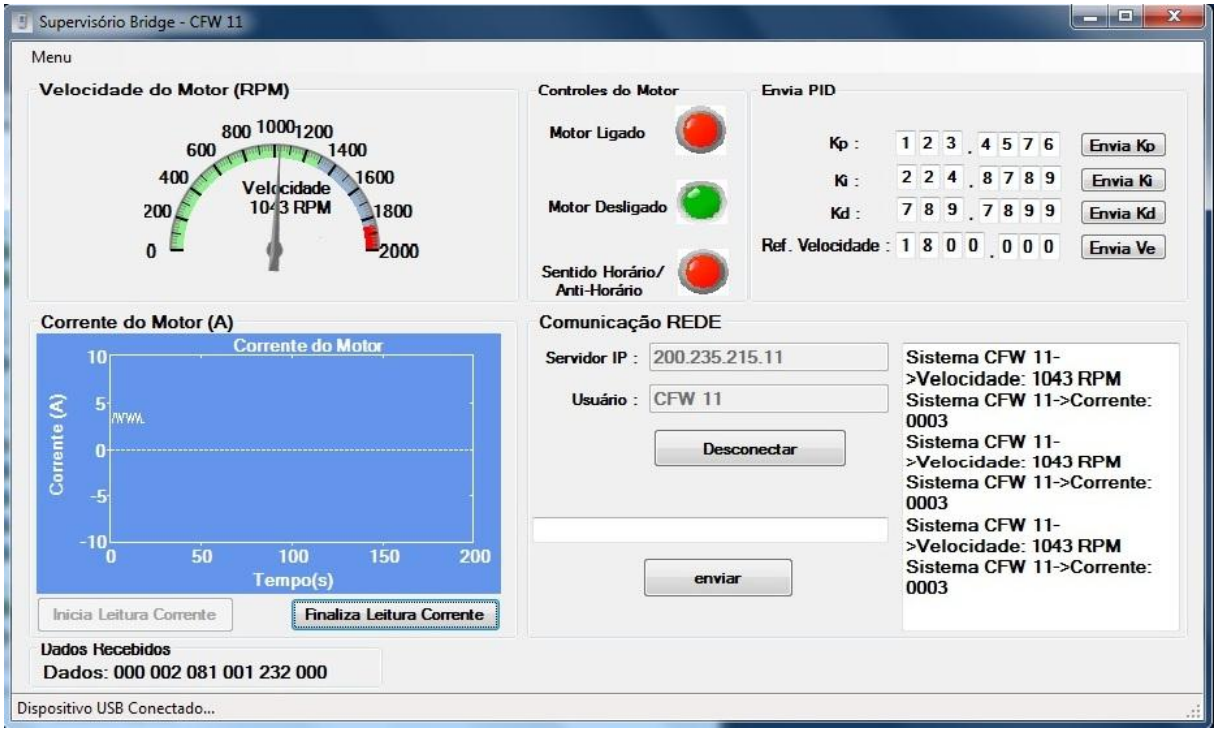

Figura 17 - Interface Bridge com todos os controles.

3.2.1 Comunicação USB e Inserção dos Componentes Visuais

Depois de incluído o controle para comunicação USB no modo HID, como explicado no item 2.2. Criação da Interface Bridge, no *Toolbox*, podemos inseri-lo no formulário para fazer uso dessa classe na interface desenvolvida.

A primeira ação do usuário, quando utiliza o aplicativo, é a verificação da correta conexão do dispositivo ao computador. Para isso, foi feito um *menu* que possibilite ao usuário, conectar-se e desconectar-se do microcontrolador quando necessário, conforme apresentado na figura 18.

Figura 18 - Menu do software.

Na opção do *menu* Conectar Via USB é onde é verificada a conexão com o dispositivo USB. A seguir é apresentado o programa em C# que executa essa função:

```
private void conectarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                USBCom.ProductId = 32;
                USBCom.VendorId = 1121;
                USBCom.CheckDevicePresent();
                button1.Enabled = true;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
```

Nessa função é verificado se o dispositivo com as identificações *ProductID* e *VentorID* são, respectivamente, 32 e 1121. Esses valores são definidos na biblioteca USB <*usb_desc_hid*> inserida no código do PIC no programa CCS. Em seguida, no método *CheckDevicePresent*, será verificada a inserção desse dispositivo com as características (*ProductID* e *VentorID*) anteriores no computador. Todas essas ações são executadas dentro do tratamento de exceção *try.catch*, para verificar possíveis problemas.

Para obter informação sobre essa conexão, foi utilizada uma barra de *status* no rodapé da interface, para isso, foi utilizado o componente do *Toobox statusStrip1*. Na figura 19, é apresentada essa barra de verificação do *status* do dispositivo USB.

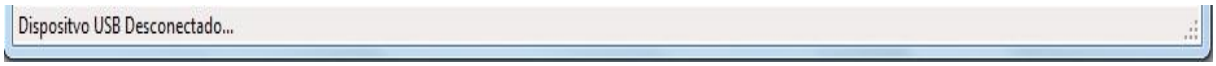

Figura 19 - Barra de Status do dispositivo.

O código para que assim que o aplicativo for executado, sem a conexão USB, exibir a mensagem de que o Dispositivo está desconectado é apresentado abaixo:

```
private void Form1_Load(object sender, EventArgs e)
        {
            toolStripStatusLabel1.Text = "Dispositvo USB Desconectado...";
        }
```

Para que quando a conexão for estabelecida com o PIC, a barra de *status* apresente a mensagem de conexão, é utilizado o evento do objeto USB (*OnSpecifiedDeviceArrived*). O código abaixo mostra o tratamento desse evento:

```
private void USBCom_OnSpecifiedDeviceArrived(object sender, EventArgs e)
        {
            toolStripStatusLabel1.Text = "Dispositivo USB Conectado...";
        }
```

Após a conexão do dispositivo temos que fazer o tratamento dos dados provenientes do microcontrolador de acordo com o protocolo definido anteriormente. Os dados recebidos pela USB são tratados por uma *thread* secundária, não estando na linha de comando principal do código. A criação e inicialização do componente principal da classe USB são definidas no formulário pelo código abaixo:

```
   public Form1()
        {
            InitializeComponent();
            USBCom.OnDataRecieved += new
UsbLibrary.DataRecievedEventHandler(USBCom_OnDataRecieved);

        }
```

Depois de inserido o evento de tratamento de recepção pela USB, temos a criação do código de recepção e tratamento dos dados que chegam pela USB. De acordo com o protocolo definido na programação do 18F4550, se tem um total de 5 *bytes* que estarão disponíveis na variável *args* do evento de recepção do *USBLibrary.DataRecievedEventArgs.* Para serem processados e enviados aos componentes que possibilitam a visualização desses valores no sistema supervisório. Com isso, cada dado recebido é armazenado na variável *mydata* que é do tipo *byte*. Na recepção é verificado a quantidade de caracteres que chegam, para serem convertidos num padrão de 3 caracteres, acrescentando-se zeros aos valores recebidos caso necessário. A variável que recebe esse número é a *bfrecebe*.

Ao final do método *USBCom_OnDataReceived*, o métodos *BeginInvoke* executa o *delegate* de forma assíncrona no *Thread* principal. O código utilizado nesse método é apresentado a seguir:

```
   private void USBCom_OnDataRecieved(object sender,
UsbLibrary.DataRecievedEventArgs args)
        {
            string bfRecebe = "Dados: ";

            foreach (byte mydata in args.data)
            {
                if (mydata.ToString().Length == 1)
                    bfRecebe += "00";

                if (mydata.ToString().Length == 2)
                    bfRecebe += "0";

                bfRecebe += mydata.ToString() + " ";
            }

            this.BeginInvoke(new Fdelegate(recebe_usb), new object[] { bfRecebe });

        }
```

Depois da finalização do método anterior é necessário a criação de outro método (*recebe_usb(string a)*), que será responsável pelo tratamento dos dados recebidos a serem visualizados na *thread* principal. Ela é chamada no método *BeginInvoke*.

Como dito anteriormente, foram usados dois tipos de formas de visualizar os valores de corrente e velocidade do motor, serão usados representações em Gráficos e *Guages* como mostrado na figura 20, que apresenta essa representação no supervisório criado.

Portanto para que esses controles sejam usados, devem ser incluídos da mesma forma como o controle de comunicação USB no modo HID, como explicado no item 2.2. Depois da inserção no *Toolbox* dessas ferramentas, podemos inseri-los no formulário para fazer uso dessas classes na interface desenvolvida. Ambos com suas respectivas bibliotecas **[28]** e **[29].** Outra função que já está contida no C#, mas que é utilizada pela visualização em gráficos é o *Timer*, que se encontra na Aba *Toolbox*.

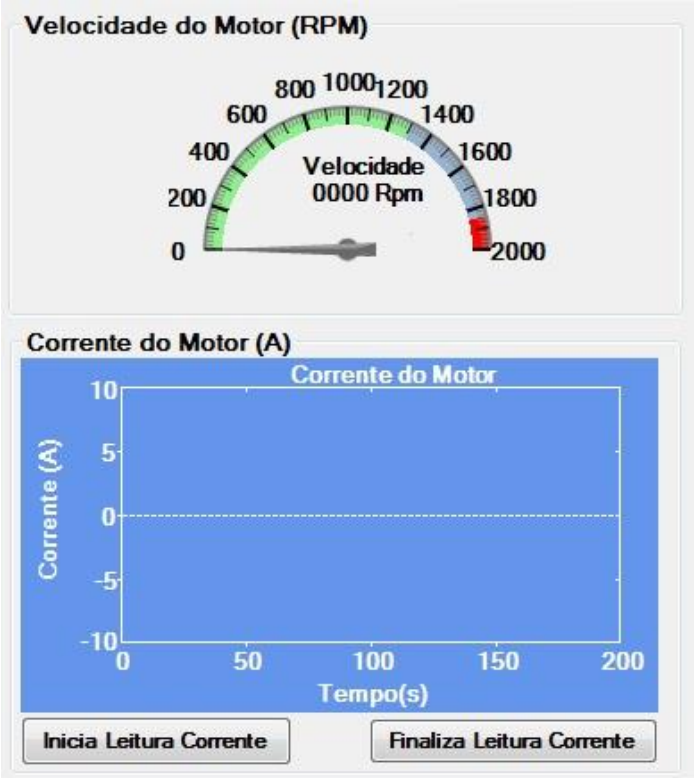

Figura 20 - Ferramentas de Visualização utilizados.

Para que os *Gauges* e Gráficos exibam corretamente os valores, foi utilizado o método *Split*, que quebra uma cadeia de caracteres em certos marcadores, no caso do trabalho a cada três caracteres. Para a gravação desses dados separados foi utilizado uma matriz de *string* com

o nome *txtsplit*. Nessa matriz não será utilizado o índice 1, pois, esse é utilizado pela comunicação USB como controle. Assim para a leitura do canal analógico 0 do PIC, temos os dados armazenados no índices 2 e 3 da matriz e para o canal 1 são usados os índices 4 e 5 e o índice 6 guarda o *byte* que retorna o estado de alguns comandos e variáveis enviadas ao 18F4550.

Pela seção anterior, foi exposto que para enviar os dados da conversão analógica/digital para o computador, foi utilizado dois pacotes de *bytes*, portanto o aplicativo deve ser capaz de reagrupa-los. Para juntar novamente esses dados em um único número, foram declaradas duas variáveis do tipo *int, adc1 e v1*, que armazenam os valores da conversão A/D do canal 0. A variável *adc1*, fará a conversão do índice 2 da matriz de *string* no tipo *int*, pois, essa é uma matriz do tipo *string*, depois é feito o deslocamento à esquerda dos 8 posições para que os valores do índice 3 da *txtsplit* seja gravado no local onde estavam os 8 *bits* deslocados da variável *adc1*. A variável *v1* armazena o valor da conversão de valores de 10 *bits* (0 a 1023) em seus valores correspondentes em Velocidade (*0 a 1800 rpm*) que é representado pela tensão na porta analógica, depois o valor dessa variável é atribuída a propriedade *value* do *aGauge* e depois convertida em *string* para ser exibida por essa classe em formato texto. O código que mostra o deslocamento de *bits* da variável e o reagrupamento e a atribuição na propriedade do *aGauge* é mostrado a seguir:

```
public void recebe_usb(string a)
        {

            string[] txtSplit;
            int adc1, v1, adc3;

            txtSplit = a.Split(' ');
            adc1 = (int)Convert.ToInt32(txtSplit[2]) << 8;
            adc1 += Convert.ToInt32(txtSplit[3]);

            v1 = (adc1 *1800) / 1023;//Modificar para a conta da velocidade

            aGauge1.Value = v1;
            aGauge1.Cap_Idx = 1;
            aGauge1.CapText = v1.ToString() + "RPM";

            adc3 = (int)Convert.ToInt32(txtSplit[4]) << 8;
            adc3 += Convert.ToInt32(txtSplit[5]);
            corrente = (adc3*8)/1023;// converter para o valor da corrente
        }
```

Para o canal 1, foram declaradas mais duas variáveis do tipo *int adc3* e *corrente* e o processo de deslocamento e reconstrução do valor de 10 *bits* é o mesmo usado no canal 0. Para a visualização em gráfico é necessário à inserção da diretiva para exibição de gráfico e no evento de carregamento do formulário é necessário, acrescentar a configuração do gráfico de acordo com o código abaixo:

```
using System.Drawing.Drawing2D;// diretiva para exibição de gráfico

private void Form1_Load(object sender, EventArgs e)
        {
            xyGraph1.AddGraph("XtraTitle", DashStyle.Solid, Color.White, 1, false);
        }
```

Para a exibição dos valores no gráfico, foi criado duas variáveis do tipo *float*, *eixo_x* e *eixo_y* iniciando-as com valor zero. O controle *Timer* é utilizado para controlar a variação do tempo de amostragem do gráfico e servir de contador de tempo para o *eixo_x*. Ainda na visualização do gráfico, foram utilizados dois botões para iniciar e parar a exibição da curva da corrente no gráfico, apresentado na interface do supervisório. Abaixo segue o código para os botões e visualização dos gráficos com os comandos do *Timer*:

```
private void button1_Click(object sender, EventArgs e)
        {

            timer1.Enabled = true;
            button1.Enabled = false;
            button2.Enabled = true;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            timer1.Enabled = false;
            button1.Enabled = true;
            button2.Enabled = false;
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            eixo_x++;
            eixo_y = corrente ;

            TempList.Add(DateTime.Now + " - " + txtList);
            qtde_data++;

            if (eixo_x >= 120)
            {
                timer1.Enabled = false;
                button1.Enabled = true;
                button2.Enabled = false;
            }
            xyGraph1.AddValue(0, eixo_x, eixo_y);
            xyGraph1.DrawAll();
        }
```

Outra função desempenhada pela USB é o envio de dados e comandos ao microcontrolador pela interface vista pela figura 21, onde é possível ver os botões de envio de comandos e a caixa onde temos as caixas para digitar os valores dos parâmetros para o cálculo do controle PID.

Figura 21 - Controles e Dados enviados na USB.

O envio dos comandos dos botões é feito através de caracteres que identificam qual botão foi acionado, quando esses comandos chegam ao microcontrolador eles *setam bits* do *byte* de posição 6 do *buffer* de envio do PIC, com isso podem ser identificados no computador que foram recebidos com sucesso no microcontrolador e assim alterar a figura do *pictureBox*, o código que representa o envio desses comandos e a recepção da confirmação é visto abaixo:

```
        // Envio dos comandos
         private void pictureBox1_Click(object sender, EventArgs e)
         {
             BufferOut[0] = 0x00;
             BufferOut[1] = (byte)'B';    // Carrega Caracteres para comando
             BufferOut[2] = (byte)'T';
             BufferOut[3] = (byte)'1';

             for (clr_buffer = 4; clr_buffer < 32; clr_buffer++)
BufferOut[clr_buffer] = 0;

             Envia_USB();

         }

         private void pictureBox2_Click(object sender, EventArgs e)
         {
             BufferOut[0] = 0x00;
             BufferOut[1] = (byte)'B';    // Carrega Caracteres para comando
             BufferOut[2] = (byte)'T';
             BufferOut[3] = (byte)'2';

             for (clr_buffer = 4; clr_buffer < 32; clr_buffer++)
BufferOut[clr_buffer] = 0;

             Envia_USB();
         }

// Recepção dos bits de confirmação

int cal = Convert.ToInt32(txtSplit[6]) & 1;

             if (cal == 1)
             {
                 pictureBox1.Image =
Monografia_USB_Supervisório.Properties.Resources.LED_ON;
             }
             else
                 pictureBox1.Image =
Monografia_USB_Supervisório.Properties.Resources.LED_OFF;

             cal = Convert.ToInt32(txtSplit[6]) & 2;

             if (cal == 2)
             {
                 pictureBox3.Image =
Monografia_USB_Supervisório.Properties.Resources.LED_OFF;
             }
             else
                 pictureBox3.Image =
Monografia_USB_Supervisório.Properties.Resources.LED_ON;
```

Como pode ser visto no código acima, depois que a figura que representa o comando desejado é clicado, três caracteres são carregados no *BufferOut*, para serem enviados o método *Envia_USB()*, que primeiramente testa se o dispositivo a receber os dados está conectado, se satisfeito é enviado os dados pelo método *SendData* da classe *USBCom*. O código desse método é apresentado a seguir:

```
private void Envia_USB()
        {
            try
            {
                if (this.USBCom.SpecifiedDevice != null)
                {

                    USBCom.SpecifiedDevice.SendData(BufferOut);
                }

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
```

Para o envio dos parâmetros do PID, foi feita primeiro a leitura dos dados digitados no *textbox* e convertidos para o tipo *Char*, para serem enviados em um *Array* de *bytes* de *char*, para que pudessem ser recebidos pelo microcontrolador. Também os dados foram convertidos no tipo *float* para que pudessem ser enviados para a comunicação Via rede. E um parâmetro só é enviado quando o PIC sinaliza para o computador que o primeiro parâmetro chegou com sucesso, para isso os botões de envio são desativados até receber a confirmação da chegada dos dados. O código para essa função é executado sempre que o botão de enviar o parâmetro é pressionado, sendo lido gravado em um *Buffer* e enviado se o dispositivo estiver conectado. O código que mostra essa leitura dos dados para o parâmetro P do controle e seu envio e retorno é apresentado a seguir, para os outros parâmetros foi utilizado à mesma ideia:

```
private void button3_Click(object sender, EventArgs e)
        {
            try
            {//lendo os parâmetros KP
                tentativa[0] = float.Parse(textBox1.Text);
                tentativa[1] = float.Parse(textBox2.Text);
                tentativa[2] = float.Parse(textBox3.Text);
                tentativa[3] = float.Parse(textBox4.Text);
                tentativa[4] = float.Parse(textBox5.Text);
                tentativa[5] = float.Parse(textBox6.Text);
                tentativa[6] = float.Parse(textBox7.Text);
            }

            catch (Exception)
            {
                MessageBox.Show("Esta entrada não é valida", "ATENÇÃO ERRO!!!",
MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

                textBox1.Text = ""; textBox1.Focus();
                textBox2.Text = ""; textBox2.Focus();
            try
            {
                //Valores de KP para enviar via USB
                tenta[0] = Char.Parse(textBox1.Text);
                tenta[1] = Char.Parse(textBox2.Text);
                tenta[2] = Char.Parse(textBox3.Text);
                tenta[3] = Char.Parse(textBox4.Text);
                tenta_1[0] = Char.Parse(textBox5.Text);
                tenta_1[1] = Char.Parse(textBox6.Text);
                tenta_1[2] = Char.Parse(textBox7.Text);

            }

            catch (Exception)
            {
                MessageBox.Show("Esta entrada não é valida", "ATENÇÃO ERRO!!!",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

                textBox1.Text = "";
                textBox1.Focus();
                textBox2.Text = "";
                textBox2.Focus();
                textBox3.Text = "";
                textBox3.Focus();

            }

            /* Carrega valores de KP para ser enviada ao PIC*/
            enviadados[0] = 0x00;
            enviadados[1] = (byte)'P';
            enviadados[2] = (byte)tenta[0];
            enviadados[3] = (byte)tenta[1];
            enviadados[4] = (byte)tenta[2];
            enviadados[5] = (byte)tenta[3];
            enviadados[6] = (byte)tenta_1[0];
            enviadados[7] = (byte)tenta_1[1];
            enviadados[8] = (byte)tenta_1[2];

            /*Fim do carregamento de KP para o pic*/
```

```
/* CALCUL DO P, I, D para ser enviado pela comunicação via REDE ETHERNET, para ser
mostrada no servidor*/

            P = tentativa[0] * 100 + tentativa[1] * 10 + tentativa[2] * 1 +
tentativa[3] / 10 + tentativa[4] / 100 + tentativa[5] / 1000 + tentativa[6] /
10000;


            if (this.USBCom.SpecifiedDevice != null)
            {
                button4.Enabled = false;
                button5.Enabled = false;
                button6.Enabled = false;

                USBCom.SpecifiedDevice.SendData(enviadados);
                for (int i = 0; i < enviadados.Length; ++i) enviadados[i] = 0;


            }
            else
            {
                MessageBox.Show("USB desconectada!! ");
            }

        }

//recepção da confirmação de recebimento do parâmetro método a seguir
 public void recebe_usb(string a)
        {
cal = Convert.ToInt32(txtSplit[6]) & 8;

            if (cal == 8)
            {
                button4.Enabled = true;
                button5.Enabled = true;
                button6.Enabled = true;

            }
```

3.2.2 Salvamento e Abertura de Conteúdo

Nesse aplicativo foi criada uma lista temporária com os dados para posterior armazenamento em arquivo. Os valores das correntes e da velocidade do motor serão os valores salvos. Primeiramente é necessário a declaração de criação da lista e a declaração da variável do tipo *string txtList*. Os valores a serem atribuídos à variável *txtList* são as variáveis *va1* e *corrente* no método *recebe_USB* e já na atribuição faremos a formatação da *string* para que os valores sempre tenho 4 digitos para a representação, se não possuir será preenchido com zero.

Os dados devem ser passados a lista a cada evento do *Timer1*. Os controles de abrir e salvar são inseridos no *menu* apresentado na figura 18, os controles (*saveFileDialog*) e

(*openFileDialog*) foram incluídos ao formulário e formatados para salvar e recuperar dados num arquivo **.txt* igual ao apresentado na figura 13. A inclusão da diretiva de controle de entrada e saída para a classe *System* é necessária, para ser possível a utilização dos métodos *StreamReader* para abrir e *TextWriter* para salvar.

```
private void abrirToolStripMenuItem_Click(object sender, EventArgs e)
        {
            StreamReader Arq;
            string Recebe = string.Empty;

            eixo_x = 0;

            xyGraph1.ClearGraph(0);

            try
            {
                if (openFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    Arq = new StreamReader(openFileDialog1.FileName);

                    while (!Arq.EndOfStream)
                    {
                        Recebe = Arq.ReadLine().Substring(35, 4);
                        eixo_y = float.Parse(Recebe);

                        xyGraph1.AddValue(0, eixo_x, eixo_y);
                        xyGraph1.DrawAll();
                        eixo_x++;
                    }
                }
            }
            catch (Exception erro)
            {
                MessageBox.Show(erro.ToString());
            }

        }
```

3.2.3 Comunicação Via Rede (TCP/IP)

A criação da aplicação *bridge* é mais simples, já que tudo o que ela tem a fazer é tentar se conectar ao servidor, pedir um nome de usuário, e começar a receber e enviar dados e, finalmente, desliga quando solicitado. A parte da interface responsável por esse controle de rede é apresentada na figura 22.

Figura 22 - Interface para comunicação via Rede.

Nessa configuração são usados alguns *Namespaces* e algumas variáveis especiais são declaradas para serem usadas nessa classe que possibilita a conexão via rede, os *Namespaces* usados e as variáveis são apresentados a seguir:

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;

//As variáveis usadas são:
// ////////////////////Variáveis da rede//////////////////////////

        // Trata o nome do usuário
        private string NomeUsuario = "Desconhecido";
        private StreamWriter stwEnviador;
        private StreamReader strReceptor;
        private TcpClient tcpServidor;
        // Necessário para atualizar o formulário com mensagens da outra thread
        private delegate void AtualizaLogCallBack(string strMensagem);
        // Necessário para definir o formulário para o estado "disconnected" de
outra thread
        private delegate void FechaConexaoCallBack(string strMotivo);
        private Thread mensagemThread;
        private IPAddress enderecoIP;
        private bool Conectado;

        // ////////////////////Fim das variáveis da rede//////////////////
```

Quando o botão conectar for pressionado o sistema tem que se conectar ao servidor, portanto a função do botão conectar pode ser descrito a seguir:

```
private void button4_Click(object sender, EventArgs e)
        {
            // se não esta conectando aguarda a conexão
            if (Conectado == false)
            {
                // Inicializa a conexão
                InicializaConexao();
            }
            else // Se esta conectado entao desconecta
            {
                FechaConexao("Desconectado a pedido do usuário.");
            }
        }
```

O método chamado quando se clica no botão conectar é o método *InicializaConexão()*, que tenta conectar o cliente ao servidor e é dado pelo código a seguir:

```
  private void InicializaConexao()
        {
            try
            {
                // Trata o endereço IP informado em um objeto IPAdress
                enderecoIP = IPAddress.Parse(txtServidorIP.Text);
                // Inicia uma nova conexão TCP com o servidor chat
                tcpServidor = new TcpClient();
                tcpServidor.Connect(enderecoIP, 2502);

                // AJuda a verificar se estamos conectados ou não
                Conectado = true;

                // Prepara o formulário
                NomeUsuario = txtUsuario.Text;

                // Desabilita e habilita os campos apropriados
                txtServidorIP.Enabled = false;
                txtUsuario.Enabled = false;
                txtMensagem.Enabled = true;
                btnEnviar.Enabled = true;
                btnConectar.Text = "Desconectado";

                // Envia o nome do usuário ao servidor
                stwEnviador = new StreamWriter(tcpServidor.GetStream());
                stwEnviador.WriteLine(txtUsuario.Text);
                stwEnviador.Flush();

                //Inicia a thread para receber mensagens e nova comunicação
                mensagemThread = new Thread(new ThreadStart(RecebeMensagens));
                mensagemThread.Start();
            }
            catch (Exception ex)
            {
                MessageBox.Show("Erro : " + ex.Message, "Erro na conexão com
servidor", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
```

No código acima, o endereço IP é analisado a partir do *TextBox* em um objeto *IPAddress*, e então é aberta uma conexão TCP para esse endereço. A porta é 2502, mas pode ser usada qualquer porta disponível. Em seguida, são preparados os controles no formulário, desativando alguns e habilitando outros.

Assim que clicado é alterado a legenda do botão conectar para exibir a mensagem ”Desconectado”. Através de um *Stream*, informamos ao servidor o nome de usuário, e imediatamente depois que começamos uma nova *thread* que chama o método *RecebeMensagens()* para receber as mensagens. Colocando isso em uma *thread* separada, pois, enquanto ele está escutando as mensagens do servidor mantem a conexão ativa. O código do método *RecebeMensagens()* é apresentado a seguir:

```
private void RecebeMensagens()
        {
            // recebe a resposta do servidor
            strReceptor = new StreamReader(tcpServidor.GetStream());
            string ConResposta = strReceptor.ReadLine();
            // Se o primeiro caracater da resposta é 1 a conexão foi feita com
sucesso
            if (ConResposta[0] == '1')
            {
                // Atualiza o formulário para informar que esta conectado
                this.Invoke(new AtualizaLogCallBack(this.AtualizaLog), new object[]
{ "Conectado com sucesso!" });
            }
            else // Se o primeiro caractere não for 1 a conexão falhou
            {
                string Motivo = "Não Conectado: ";
                // Extrai o motivo da mensagem resposta. O motivo começa no 3o
caractere
                Motivo += ConResposta.Substring(2, ConResposta.Length - 2);
                // Atualiza o formulário como o motivo da falha na conexão
                this.Invoke(new FechaConexaoCallBack(this.FechaConexao), new
object[] { Motivo });
                // Sai do método
                return;
            }

            // Enquanto estiver conectado le as linhas que estão chegando do
servidor
            while (Conectado)
            {
                // exibe mensagems no Textbox
                this.Invoke(new AtualizaLogCallBack(this.AtualizaLog), new object[]
{ strReceptor.ReadLine() });
            }
        }
```

Com isso, criamos um novo leitor de *stream* que conecta ao cliente TCP. Ele vai ouvir os dados recebidos. A primeira linha vinda do servidor contém a resposta nos dizendo se estamos ou não conectados com êxito. Existem dois motivos para a conexão falhar: usar um nome de usuário já em uso ou usar Administrador como nome de usuário.

O primeiro caractere da resposta dada pelo servidor nos diz o seguinte:

- 1 , se conexão foi bem sucedida;
- 0 , se conexão falhou.

Quando ocorre a falha na conexão, o motivo da falha inicia no terceiro caractere da mensagem, já que o primeiro é o número, e o segundo é um caractere de *pipe*.

O método *this.Invoke(...)* chama o formulário para se atualizar. Não se podem atualizar diretamente os elementos do formulário nos mesmos a partir deste método porque ele esta em uma *thread* distinta e operações entre *threads* são ilegais. Finalmente, o *loop while* continua chamando o método *strReceptor.ReadLine()* que verifica os dados recebidos do servidor. Em seguida, temos o método que continuamos chamando *this.Invoke(...),* tudo que ele faz é atualizar o *TextBox txtLog* com a mensagem mais recente. O código da rotina atualiza *log* e *keypress* são apresentados a seguir:

```
private void AtualizaLog(string strMensagem)
        {
            // Anexa texto ao final de cada linha
            txtLog.AppendText(strMensagem + "\r\n");
        }

        private void txtMensagem_KeyPress(object sender, KeyPressEventArgs e)
        {
            // Se pressionou a tecla Enter
            if (e.KeyChar == (char)13)
            {
                EnviaMensagem();
            }
        }
```

Para enviar um dado usamos o evento do *Timer1* neste caso chamamos a rotina *EnviaMensagem().* Essa rotina apenas verifica o dado, e então escreve o dado para a conexão TCP através do objeto *StreamWriter*. A chamda ao *métodoFlush()* garante que os dados estão sendo enviadas de imediato.

Para fechar a conexão chamamos o método *FechaConexao* é chamado, e neste código temos que o formulário está sendo levado de volta para o estado sem conexão, e a conexão TCP e os *streams* estão sendo fechados. O código desse método é apresentado abaixo:

```
private void FechaConexao(string Motivo)
        {
            // Mostra o motivo porque a conexão encerrou
            txtLog.AppendText(Motivo + "\r\n");
            // Habilita e desabilita os controles apropriados no formulario
            txtServidorIP.Enabled = true;
            txtUsuario.Enabled = true;
            txtMensagem.Enabled = false;
            btnEnviar.Enabled = false;
            btnConectar.Text = "Conectado";

            // Fecha os objetos
            Conectado = false;
            stwEnviador.Close();
            strReceptor.Close();
            tcpServidor.Close();
        }
```

Com isso finalizamos as funções principais do controle de rede do aplicativo *bridge* desenvolvidas.

3.3 Interface Servidor do computador

O aplicativo Servidor será um pouco mais complexo do que a aplicação *bridge*, porque ele precisa manter informações sobre todos os clientes conectados, aguardar os dados de cada um e enviar mensagens de entrada para todos. A interface do *software* Servidor é apresentada na figura 23.

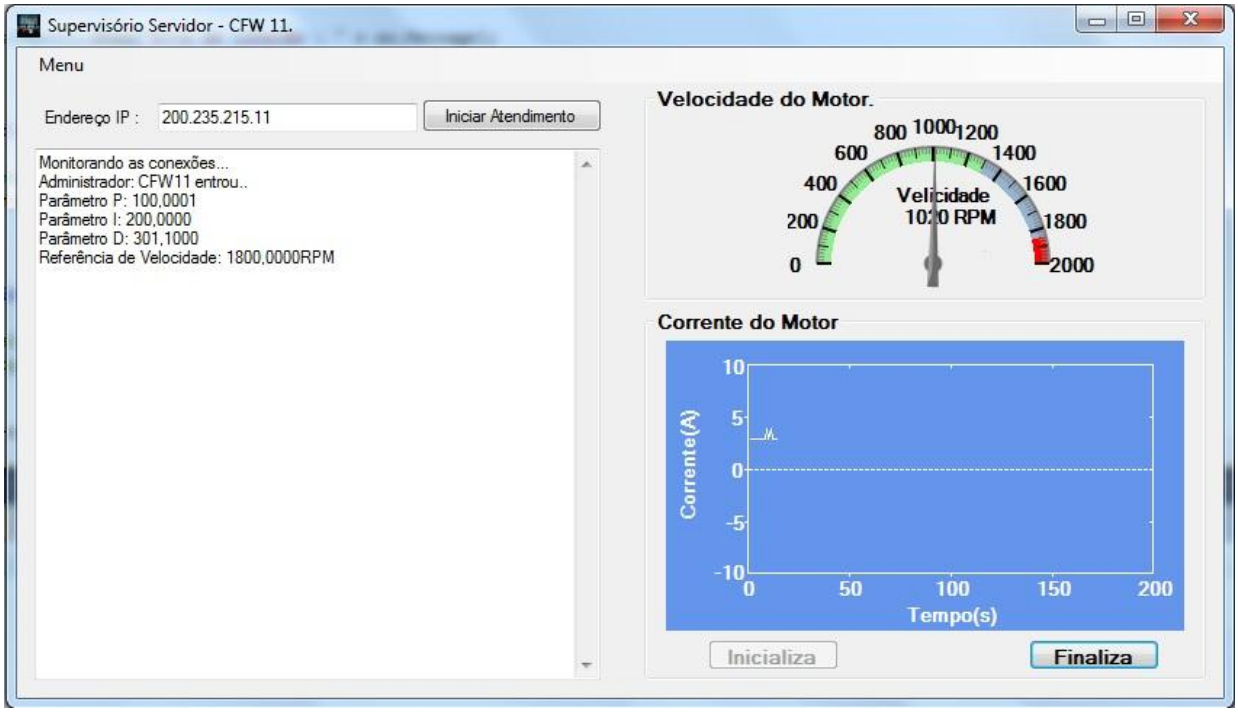

Figura 23 - Interface do Programa Servidor.

Nessa aplicação serão usados alguns *Namespaces* utilizados também na interface *bridge*. Foi necessário declarar um *delegate* que será usado para atualizar o *TextBox* - *txtLog* da outra *thread*. Essas primeiras configurações são vistas a seguir:

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Threading;
using System.Collections;

// Este delegate é necessário para especificar os parametros que estamos pasando
com o nosso evento

private delegate void AtualizaStatusCallback(string strMensagem);
// Este delegate é necessário para especificar os parametros que estamos pasando
com o nosso evento
    public delegate void StatusChangedEventHandler(object sender,
StatusChangedEventArgs e);
```

Quando o botão de Iniciar atendimento é pressionado a seguinte função é chamada:

```
private void btnAtender_Click(object sender, System.EventArgs e)
        {
            if (txtIP.Text==string.Empty)
            {
                MessageBox.Show("Informe o endereço IP.");
                txtIP.Focus();
                return;
            }

            try
            {

                // Analisa o endereço IP do servidor informado no textbox
                IPAddress enderecoIP = IPAddress.Parse(txtIP.Text);

                // Cria uma nova instância do objeto ChatServidor
                ChatServidor mainServidor = new ChatServidor(enderecoIP);

                // Vincula o tratamento de evento StatusChanged a
mainServer_StatusChanged
                ChatServidor.StatusChanged += new
StatusChangedEventHandler(mainServidor_StatusChanged);

                // Inicia o atendimento das conexões
                mainServidor.IniciaAtendimento();

                // Mostra que nos iniciamos o atendimento para conexões
                txtLog.AppendText("Monitorando as conexões...\r\n");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Erro de conexão : " + ex.Message);
            }
        }
```

Outra classe importante de ser apresentada é a que manipula eventos para o evento *StatusChanged*. Ele vai nos informar quando um cliente se conecta, uma nova mensagem foi recebida, um cliente foi desconectado, entre outras coisas. O seu código é apresentado a seguir:

```
public void mainServidor_StatusChanged(object sender, StatusChangedEventArgs e)
        {
            // Chama o método que atualiza o formulário
            this.Invoke(new AtualizaStatusCallback(this.AtualizaStatus), new
object[] { e.EventMessage });
        }

        private void AtualizaStatus(string strMensagem)
        {
            // Atualiza o logo com mensagens
            txtLog.AppendText(strMensagem + "\r\n");
        }
```

Finalmente, o método *IniciaAtendimento()* diz ao objeto Servidor para iniciar a ouvir as conexões de entrada.

Aqui também é utilizado um manipulador de eventos que já foi usado anteriormente, e o outro é o método *AtualizaStatus()* que é chamado quando uma atualização precisa ser feita para o formulário, como visto acima. Isso é necessário, pois se usou o *Invoke()* e o *delegado* para fazer uma chamada na *thread* cruzada

A seguir é apresentada uma classe para o inicialização do atendimento do cliente e a classe qua mantém a conexão ativa, são feitos alguns comentários:

```
    public void IniciaAtendimento()
        {
            try
            {

                // Pega o IP do primeiro dispostivo da rede
                IPAddress ipaLocal = enderecoIP;

                // Cria um objeto TCP listener usando o IP do servidor e porta
definidas
                tlsCliente = new TcpListener(ipaLocal, 2502);

                // Inicia o TCP listener e escuta as conexões
                tlsCliente.Start();

                // O laço While verifica se o servidor esta rodando antes de
checar as conexões
                ServRodando = true;

                // Inicia uma nova tread que hospeda o listener
                thrListener = new Thread(MantemAtendimento);
                thrListener.Start();
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }
       EventMsg = strEventMsg;
    }
  }

private void MantemAtendimento()
        {
            // Enquanto o servidor estiver rodando
            while (ServRodando == true)
            {
                // Aceita uma conexão pendente
                tcpCliente = tlsCliente.AcceptTcpClient();
                // Cria uma nova instância da conexão
                Conexao newConnection = new Conexao(tcpCliente);
            }
        }
```

4 Conclusões

Com o desenvolvimento do trabalho, é possível notar sua flexibilidade e funcionalidade em relação à aquisição e transmissão de dados, e ainda podem ser acrescentadas muitas outras funcionalidades, para melhorar o sistema em termos de recursos. O domínio da tecnologia de automação através da ligação entre microcontroladores programáveis e computador foi realizado de maneira eficiente, tanto em termos de velocidade quanto em termos de qualidade.

Outro fato importante no desenvolvimento do trabalho foi à utilização de versões gratuitas dos *softwares* para o desenvolvimento de todo o projeto, fazendo com que o desenvolvimento de todos os *softwares* seja custoso apenas no ponto de vista do trabalho humano não tendo custo nos IDE's utilizados.

A comunicação entre o computador e o microcontrolador foi muito bem sucedida, sendo uma comunicação robusta e de alta confiabilidade, como foi utilizado um componente HID, isso possibilidade a conectividade do dispositivo com qualquer tipo de computador sem a necessidade da inserção de *drivers* adicionais.

A comunicação via rede, possibilitou o acesso as informações do sistema de maneira remota via rede *Ethernet*, com isso é possível a centralização de informações em um único local sem a necessidade da sala de controle estar perto do processo. Outro ponto importante é a conectividade do servidor com vários clientes, possibilitando à conectividade de vários sistemas a mesma sala de controle.

Para trabalhos futuros é interessante o estudo de outras ferramentas que possam ser utilizadas no processo para tornar o monitoramento mais completo, e até mesmo o uso de microcontroladores mais velozes.

Referências Bibliográficas

CASOS

[01] OLIVEIRA, A. L. de Lima. (1999). **Instrumentação - Fundamentos de Controle de Processos** . Progama de Certificação de Pessoal de Manutenção. Trabalho realizado em parceria SENAI/CST(Companhia Siderúrgica Tubarão). Espírito Santo – 1999. 12,15

[02] SOARES, H. N. (2013). **Desenvolvimento de Protótipo de Veículo Controlado por Realidade Virtual.** Monografia de Graduação. Engenharia de Controle e Automação - Universidade Federal de Ouro Preto – 2013. 12,15,23

[03] CASILLO, D. (2011) **Automação e Controle - Sistemas Supervisórios.** Notas de Aula – Universidade Federal Rural do Semi-Árido - 2011. 12,13,16

[04] LOPES, M. A. M. (2009). **A importância dos Sistemas Supervisórios no Controle de Processos Industriais.** Monografia de Graduação. Engenharia de Controle e Automação - Universidade Federal de Ouro Preto - 2009. 13,15

[05] FUGITA, M. (2010). **Integração das Funções de Supervisório e Gerenciamento de Rede Via SNMP**. Tese de Doutorado. Universidade Federal de Itajubá, Maio - 2010. 16

[06] BRANDÃO, A. S. (2013). **Interface Homem Máquina**. Notas de Aula disciplina ELT431. Engenharia Elétrica - Universidade Federal de Viçosa - 2013. 13

[07] PEREIRA, F. (2003). **Microcontrolador PIC - Programção em C**. Editora Érica, 3 ª Edição, São Paulo - 2003. 17

[08] Datasheet PIC 18F4550. Disponível em : <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>**.** Acesso em 11 nov. 2013 19,20

[09] SANTOS, de Sá L. S. (2009). **Sistema de Comunicação USB com Microcontrolador.** Monografia de Graduação. Engenharia da Computação - Universidade de Pernambuco, Escola Politécnica de Pernambuco – 2009. 20

[10] RAYSARO, M. C. (2012). **Sistema Open-Source de Supervisão Controle e Aquisição de Dados.** Monografia de Graduação. Sistema de Informação - Universidade de Cuiabá - Faculdade de Tecnologia – 2012. 15

[11] Figura : Sala de Monitoramento e Controle. Fonte : <http://pt.wikipedia.org/wiki/Ficheiro:Leitstand_2.jpg>. Acesso em 27 dez. 2013. 15

[12] SILVA, H. A. **Supervisão de Sistemas.** Notas de Aula do curso de Tecnologia em Automação Industrial, Instituto Federal de Educação, Ciência e Tecnologia Rio Grande do Norte - Campus Natal - Central – Disponível em: <http://www.dca.ufrn.br/~humberto/Supervisao%20de%20Sistemas/Aula02%20-%20SCADA%20(Introducao).pdf> Acesso dia 27 dez. 2013. 15,16

[13] Figura : Componentes físicos de um sistema SCADA. Fonte : < http://paginas.fe.up.pt/~ee94082/SCADA.htm>. Acesso em 27 dez. 2013. 17

[14] FILHO, F. de M. L. (2010). **Controle Inteligente para Automação Predial.** Projeto de Graduação. Engenharia Elétrica - Universidade de Brasília – 2010.

[15] SOUZA, D. J. (2000). **Desbravando o PIC**, Editora Érica Ltda, 2 ª edição, São Paulo, 2000. 17,19

[16] CARVALHO, B. E. B. de. (2010). **Instrumentação Eletrônica para captação e Transmissão de Dados Referentes a Avaliação de um Trator Agrícola de Pneus.** Monografia de Graduação. Engenharia Elétrica - Universidade Federal de Viçosa - 2010. 19

[17] PEREIRA, H. A. (2006). **Desenvolvimento de um Sistema de Aquisição e Monitoramento Utilizando Microcontroladores.** Monografia de Graduação. Engenharia Elétrica - Universidade Federal de Viçosa, Dezembro 2006. 19

[18] UNIVERSAL SERIAL BUS. Disponível em : < http://homepages.dcc.ufmg.br/~adrianoc/usb/> Acessado em 27 dez. 2013. 20

[19] TOCCI, R. J. , WIDMER, N. S., MOSS, G. L. (2007). **Sistemas Digitais princípios e aplicações - Capítulo 11**, Editora Person Prentice Hall, 10 ª edição, 2007. 20

[20] LAGES, W. F. **Conversores A/D e D/A.** Notas de Aula da disciplina ELE00002 -Sistemas de Automação - Programa de Pós-Graudação em Engenharia Elétrica. Universidade Federal do Rio Grande do Sul, Escola de Engenharia.

[21]VISUAL STUDIO. Produtos Visual Studio Express 2010 - Microsoft Visual Studio 2010 Express, Disponível em <http://www.microsoft.com/visualstudio/ptb/products/visual-studio-express-products>. Acessado em 28 dez. 2013. 21,22

[22] CORTEZ, P. , QUINTELA, H. (2008). **Exemplos Práticos de Programação Visual em C#**. Departamento de Sistemas de Informação. Escola de Engenharia da Universidade do Minho. Guimarães - Portugal, 2008. 22

[23]LUZ, C. E. S. (2013). **Criação de Sistemas Supervisórios em Microsoft Visual C# 2010 Express**, Editora Érica, 2ª edição, São Paulo - 2013. 22,23

[24] Página da internet acecssada no dia 28 de dezembro de 2013: <http://www.infoescola.com/informatica/c-sharp/> 22,23

[25] Página da internet acessada no dia 28 de dezembro de 2013: <http://msdn.microsoft.com/pt-br/library/System.Net.Sockets(v=vs.110).aspx>. 25

[26] Apostila de "Internet e Arquitetura TCP/IP", volume 1, 2ª edição - Curso de Redes de Computadores. PUC RIO/ CCE. Disponível em : <http://grsecurity.com.br/apostilas/TCP/tcp-apostila.pdf> acessado dia 28 dez. de 2013. 24,25

[27] Biblioteca para comunicação USB, para linguagem C#.(UsbLibrary.dll) .Disponível em : <http://www.codeproject.com/KB/cs/USB_HID/usb_hid.zip.> acessado dia 28 dez. de 2013 30

[28] Biblioteca para o controle do componente aGauge, para linguagem C# (aGauge.dll).Disponível em : <http://www.codeproject.com/KB/miscctrl/A_Gauge.aspx> acessado dia 28 dez. de 2013 30

[29] Biblioteca para o controle do componente XYGraph, para linguagem C# (XYGraph.dll).Disponível em : <http://www.componentxtra.com/html/XYGraph.htm> acessado dia 28 dez. de 2013 30

[30] Página da internet acessada no dia 28 de dezembro de 2013: <http://msdn.microsoft.com/pt-br/library/hh425099%28v=vs.110%29.aspx>. 21

[31] Versão de demostração do CCS PCWHD, <www.ccsinfo.com/ccsfreedemo.php> 27