DESENVOLVIMENTO DE UMA INTERFACE DE CONTROLE MÓVEL PARA UTILIZAÇÃO EM CÉLULAS DE SOLDAGEM MIG/MAG

THIAGO ALMEIDA SIQUEIRA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

DESENVOLVIMENTO DE UMA INTERFACE DE CONTROLE MÓVEL PARA UTILIZAÇÃO EM CÉLULAS DE SOLDAGEM MIG/MAG

THIAGO ALMEIDA SIQUEIRA

ORIENTADOR: SADEK CRISOSTOMO ABSI ALFARO

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

PUBLICAÇÃO: ENM.DM – 28A/09

BRASÍLIA/DF: OUTUBRO - 2009

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

DESENVOLVIMENTO DE INTERFACE DE CONTROLE MÓVEL PARA UTILIZAÇÃO EM CÉLULAS DE SOLDAGEM MIG/MAG

THIAGO ALMEIDA SIQUEIRA

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECTRÔNICOS.

APROVADA POR:

_____

Prof. SADEK CRISOSTOMO ABSI ALFARO

_____

Prof. CARLOS HUMBERTO LLANOS QUINTERO

_____

Prof. TEODIANO FREIRE BASTOS FILHO

BRASÍLIA/DF, 16 DE OUTUBRO DE 2009.

FICHA CATALOGRÁFICA

SIQUEIRA, THIAGO ALMEIDA

Desenvolvimento de uma interface de controle móvel para utilização em células de soldagem MIG/MAG. [Distrito Federal] 2009.

xxviii, 101p., 297mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2009).

Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Mecânica.

1. Soldagem	2. Teleoperação
3. Comunicação Sem fio	4. PDA
I. ENM/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Siqueira, T.A. (2009). Desenvolvimento de uma interface de controle móvel para utilização em células de soldagem MIG/MAG. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-28A/2009, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 101p.

CESSÃO DE DIREITOS

AUTOR: Thiago Almeida Siqueira

TÍTULO: Desenvolvimento de uma interface de controle móvel para utilização em células de soldagem MIG/MAG.

GRAU: Mestre ANO: 2009

_____

Thiago Almeida Siqueira

SMPW QD. 20 CONJ 4 LT 8 CS A

71.745-004 Brasília – DF – Brasil.

AGRADECIMENTOS

Agradeço aos meus pais e a Aline pelo apoio e incentivo.

Aos meus colegas Marrocos, Eber, Ronald, Gerardo pela inestimável ajuda.

Aos professores Sadek, Milton, Llanos e Teodiano pela oportunidade.

Dedico este trabalho à minha família e a todos que apoiaram.

RESUMO

DESENVOLVIMENTO DE UMA INTERFACE DE CONTROLE MÓVEL PARA UTILIZAÇÃO EM CÉLULAS DE SOLDAGEM MIG/MAG

Autor: Thiago Almeida Siqueira
Programa de Pós-Graduação em Sistemas Mecatrônicos
Brasília, agosto 2009.

Com o aumento na demanda por produtos customizados e aumento na velocidade de fabricação, os conceitos de produção ágil passam a dominar os processos de fabricação, impondo à indústria um controle refinado de todos os equipamentos dentro das células de produção. Este trabalho propõe o desenvolvimento de uma nova interface de controle móvel, para utilização em células de soldagem MIG/MAG, capaz de controlar, remotamente, o funcionamento e a aquisição de dados do processo de soldagem MIG/MAG. Inicialmente, foi realizado o mapeamento do protocolo de comunicação da porta serial na fonte de soldagem, visando à obtenção de uma arquitetura para controlar os parâmetros dessa fonte, via porta serial. A partir deste mapeamento, foi possível identificar limitações de controle da fonte que inviabilizavam essa arquitetura, o que exigiu o desenvolvimento de uma nova arquitetura baseada no controle da fonte por meio de um módulo de interface para robôs. Para tanto, foi desenvolvido um *software* servidor capaz de transferir as informações de um dispositivo de aquisição de dados e de uma porta serial para um PDA, viabilizando a transferência dos dados entre o computador móvel, o módulo robótico de controle da fonte de soldagem e uma mesa posicionadora da tocha. Com essa nova arquitetura foi possível controlar os parâmetros de soldagem, tais como a potência e a velocidade de soldagem, bem como a aquisição instantânea dos valores reais de corrente, de tensão e da velocidade do arame. Durante a execução dos testes todos os parâmetros de controle e aquisição foram transferidos com sucesso entre o computador de mão e a fonte de soldagem sem nenhuma interferência perceptível do arco elétrico na comunicação sem fio.

ABSTRACT

DEVELOPMENT OF A MOBILE CONTROL INTERFACE FOR MIG/MAG WELDING CELLS

Author: Thiago Almeida Siqueira
Programa de Pós-Graduação em Sistemas Mecatrônicos
Brasilia, August 2009.

With the increased demand for customized products and the manufacturing speed, concepts of agile production start to dominate the manufacturing process, requiring the industry fine-grained control of all equipment within the production cells. This paper proposes the development of a new mobile interface control, for use in MIG/MAG welding cells, capable of remotely control the operation and data acquisition. Initially, the communication protocol mapping of a serial port on the welding power source was undertaken in order to obtain an architecture to control the welding parameters via serial port. This mapping allowed to identify limitations in the welding power source control becoming this architecture unfeasible, which required a new architecture that was developed based on the control of the power source through an interface module for robots. Consequently, a server software capable of transferring information from a data acquisition device and a serial port to a PDA was developed, enabling data transfer among the mobile computer, the robotic control module of the welding power source and the torch positioning table. With this new architecture was possible to control the welding parameters such as welding power and speed, as well as the instant acquisition of the actual values of current, voltage and wire speed. During the tests all parameters acquisition and control have been successfully transferred between the PDA and the welding power source without any noticeable interference of the electric arc in wireless communication.

SUMÁRIO

1 INTRODUÇÃO 1

1.1 Objetivos 2

1.1.1 Geral 2

1.1.2 Específico 2

2 REVISÃO BIBLIOGRÁFICA 3

2.1 Soldagem 3

2.1.1 Soldagem a arco 3

2.1.2 Processo de soldagem GMAW (MIG/MAG) 4

2.1.2.1 Gases de proteção 5

2.1.2.2 Modos de transferência metálica 5

2.1.2.3 Vantagens e desvantagens 5

2.2 FONTES DE SOLDAGEM 7

2.2.1 Fonte de corrente constante (CC) 8

2.2.2 Fontes de tensão constante 10

2.2.3 Fontes de energia de corrente constante (CC) e tensão constante (TC) combinada 10

2.2.4 Corrente pulsada 11

2.2.5 Inversoras 11

2.2.6 Soldagem Automatizada 11

2.3 TELEOPERAÇÃO / TELEROBÓTICA / TELEMANIPULAÇÃO 12

2.3.1 Interfaces 13

2.3.1.1 Interface em programação e APIs 14

2.3.1.2 Interface física 14

2.3.1.3 Interface (do usuário) de tela pequena 14
2.3.2 Comunicação 17
2.3.2.1 Comunicação Digital (Porta Serial) 17
2.3.2.2 Porta serial 17
2.3.2.3 Protocolos e formatos 18
2.3.2.4 Portas COM 23
2.3.3 Aquisição de dados e controle (DAQ) 25
2.3.3.1 Fundamentos da aquisição de dados 25
2.3.3.2 Transdutores e sensores 26
2.3.3.3 Fios de campo e cabeamento de comunicação 27
2.3.3.4 Condicionamento de sinal 27
3 METODOLOGIA 30
3.1 Comunicação 30
3.1.1 Criação de uma API para o controle via Porta Serial 30
3.1.1.1 Engenharia reversa do protocolo 30
3.1.1.2 Descobrindo o *baud rate* 31
3.1.1.3 Descobrindo a porta COM 32
3.1.1.4 Identificação da estrutura das mensagens 32
3.1.2 Controle da fonte Fronius TPS 5000 utilizando a Interface ROB 5000 e a placa de aquisição de dados NI USB-6009 34
3.1.2.1 Regulação da tensão 35
3.1.2.2 Entrada analógica: 35
3.1.2.3 Saída analógica 36
3.1.2.4 Entrada Digital 37
3.1.2.5 Saída Digital 37
3.1.2.6 Confecção da placa 37

3.2 Software de controle 40

3.2.1 Servidor 40

3.2.2 Cliente 41

4 RESULTADOS E DISCUSSÃO 43

4.1 Placa reguladora de tensão 43

4.2 Comunicação 47

4.2.1 Controle da mesa posicionadora 48

4.2.2 Interferência eletromagnética do arco na solda 49

4.3 Softwares desenvolvidos 49

4.3.1 Servidor 50

4.3.2 Cliente 50

4.3.3 Emcapsulamento dos comandos e dados dentro da comunicação 50

5 CONCLUSÃO 54

5.1 Projetos futuros 54

6 REFERÊNCIAS BIBLIOGRÁFICAS 55

APÊNDICE A – PLACA REGULADORA DE TENSÃO 58

Trilhas 58

Componentes 59

APÊNDICE B - CÓDIGO FONTE DO SERVIDOR 60

CurvaCalibracao.cs 60

DAQ.cs 61

Mesa.cs 65

Welding*Socket*.cs 68

Serializador.cs 71

APÊNDICE C - CÓDIGO FONTE DO CLIENTE 74

Persistencia.cs 74
Grafico.cs 75
Grafico.cs 75
Configuracao.cs 78
Configuracao.cs 78
Welding*Socket*.cs 80
ANEXO A – PINAGEM DA INTERFACE ROB 5000 83
ANEXO B – NI DAQ 6009 84
Terminais analógicos 84
Terminais digitais 84

LISTA DE TABELAS

Tabela 1 - Padrões de resolução de pequenas telas (Kortum, 2008). 16

Tabela 2 - Principais portas COM e seus respectivos endereços de memória (Axelson, 2007). 24

Tabela 3 - Conversão de tensões para os tipos de sinais. 35

LISTA DE FIGURAS

Figura 1 - GMAW (a) esquema do processo MIG/MAG ; (b) Região da poça de fusão. Imagem traduzida (Kou, 2003). 6

Figura 2 - Classificação das fontes de energia para soldagem. Imagem traduzida (K.L. Moore D.S. Naidu, 2003). 8

Figura 3 - Curva volt-ampere típica para: (a) fontes de soldagem de corrente constante e (b) fontes de soldagem de tensão constante. Imagem traduzida (K.L. Moore D.S. Naidu, 2003). 9

Figura 4 - Curva volt-ampere típica para fontes de energia CC e VC conbinada. Imagem traduzida (K.L. Moore D.S. Naidu, 2003). 10

Figura 5 - Transmissões síncronas incluem a linha de clock, enquanto a transmissões assíncronas requerem que cada computador tenha o seu clock (Axelson, 2007). 19

Figura 6 - Diagrama funcional de um sistema de aquisição de dados baseado em PC (Kirianaki, 2002). 26

Figura 7 - Software espião de porta serial. 32

Figura 8 - Cabo serial com sinal do TX bifurcado. 33

Figura 9 - Foto do cabo serial com sinal do TX bifurcado. 33

Figura 10 - Monitoramento dos *bytes* pelo programa *Docklight* 34

Figura 11 - Conversão da tensão para a entrada analógica. 35

Figura 12 - Conversão da tensão para a saída analógica. 36

Figura 13 - Conversão da tensão para a entrada digital. 37

Figura 14 - Placa reguladora de tensão (parte digital). 38

Figura 15- Placa reguladora de tensão (parte analógica). 39

Figura 16 - Corrosão da placa reguladora de sinais 40

Figura 17 - Tela do software de controle do servidor 41

Figura 18 - Tela inicial do software de controle do Cliente. 42

Figura 19 - Tela de configuração do software de controle do Cliente. 42

Figura 20 - Placa reguladora de tensão. 44

Figura 21 - Curva de calibração do parâmetro potência. Valores enviados ao dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 41). 45

Figura 22 - Curva de calibração do parâmetro corrente. Valores recebidos do dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 18). 45

Figura 23 - Curva de calibração do parâmetro tensão. Valores recebidos do dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 17). 46

Figura 24 - Curva de calibração do parâmetro velocidade do arame. Valores recebidos do dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 17). 46

Figura 25 - Arquitetura da interface de controle móvel proposta no trabalho (Siqueira, 2006). 47

Figura 26 - Arquitetura da interface de controle móvel proposta para a célula de soldagem. 48

Figura 27 - Controladora da mesa posicionadora. 49

Figura 28 - Principais classes e suas distribuições dentro da comunicação. 50

Figura 29 - Encapsulamento e serialização dos comandos de controle. 51

Figura 30 - Encapsulamento e serialização dos dados da fonte de soldagem. 51

Figura 31 - Diagrama de Classe do software servidor 52

Figura 32 - Diagrama de Classe do *software* cliente 53

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

API ..... Application Programming Interface (ou Interface de Programação de Aplicativos)

ASCII ..... Acrônimo para American Standard Code for Information Interchange, que em português significa "Código Padrão Americano para o Intercâmbio de Informação"

Bit ..... É a menor unidade de informação que pode ser armazenada ou transmitida

Byte ..... É um número binário de oito algarismos

Checksum ..... É um código usado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo

Clock ..... Sinal digital alternado utilizado para sincronizar circuitos digitais

COM ..... Porta de comunicação de entrada/saída presente na maioria dos PCs

CRC ..... Cyclic redundancy check, ou verificação de redundância cíclica é um código detector de erros

DAQ ..... Data acquisition (Aquisição de dados)

Driver ..... É um software utilizado para que o sistema operacional do computador se comunique com um dispositivo de hardware

Firmware ..... Conjunto de instruções operacionais programadas diretamente no hardware de um equipamento eletrônico

Hash ..... É uma função resumo que transforma qualquer quantidade de bytes em um tamanho definido

LCD..... Liquid Crystal Display (monitor de cristal líquido)

PC ..... Personal Computer (Computador Pessoal)

PDA ..... Personal digital assistants (Handhelds ou Assistente Pessoal Digital)

RS-232..... Padrão para troca serial de dados binários entre um DTE (terminal de dados, de Data Terminal equipment) e um DCE (comunicador de dados, de Data Communication equipment).

UART..... Universal Asynchronous Receiver-Transmitter

1 INTRODUÇÃO

Devido à forte concorrência e comportamento dinâmico do mercado atual, apenas os pequeno/médio lotes de fabricação conseguem atender o exigente mercado. Nestas condições, as linhas de produção robotizada têm mostrado um dos melhores desempenhos "custo unitário" quando comparado com o trabalho manual e semi-automatizado (Rosheim, 1994). Existe um número crescente de pequenas empresas orientadas para clientes que apresentam fabricação de pequenos lotes ou produtos únicos concebidos para atendimento individual. Esses usuários são exigentes na qualidade dos produtos, e para atender esta demanda é necessário que o processo de soldagem seja automatizado de forma a atender às necessidades do cliente de tempo e alta qualidade. Estas empresas aplicam os conceitos de Produção Ágil (Kusiak, 2000) (Kusiak, 1986), que está baseada em configurações de fabricação flexíveis. Tudo leva a crer que, em futuro próximo exigirá máquinas mais potentes e flexíveis, a fim de atender aos pedidos de pequenas empresas, que precisam de mais interfaces remotas, linguagens de programação poderosas, controle de força, APIs poderosas para um alto nível de programação, etc.

O que torna a robótica importante é o fato de ser uma ciência de dispositivos engenhosos, construídos com precisão, alimentado por uma fonte de energia confiável, permanente e com programação flexível. Estes dispositivos robóticos não precisam necessariamente ter código aberto, mas sim a disponibilidade de APIs robustas, e com padrões tanto para *hardware* quanto para *software*, de forma que permita o acesso ao sistema com pouca limitação. A necessidade de APIs robustas ocorre principalmente em ambientes de pesquisa, onde um bom acesso aos recursos do dispositivo controlado é necessário, de modo que testes e novas idéias possam ser implementadas (Pires, 2006). Se este estiver disponível, então o integrador do sistema (ou mesmo pesquisador) não precisará de *software* de código aberto, pelo menos para os campos tradicionais da robótica (manipuladores robóticos industriais, robôs móveis e fontes de soldagem).

Infelizmente nem sempre existem as APIs ou quando há encontram-se em uma forma ineficiente ou restritiva. Este problema acaba restringindo ou até mesmo inviabilizando o desenvolvimento de novas tecnologias. Uma forma eficaz de resolver o problema seria criar uma API a partir da engenharia reversa do protocolo de comunicação do dispositivo ou da criação de uma nova interface de acesso ao dispositivo a ser controlado. Para isso

deve-se conhecer muito bem o processo e o funcionamento do equipamento ao qual se precisa criar uma API.

Este trabalho focalizou no desenvolvimento e implementação de uma API para uma fonte de soldagem e uma mesa posicionadora visando a integração destes com um computador de mão.

1.1 Objetivos

1.1.1 Geral

O presente trabalho tem como objetivo desenvolver e implementar uma interface de controle móvel para utilização em uma célula de soldagem, isto é, controlar uma mesa posicionadora e uma fonte de soldagem por meio de um PDA.

1.1.2 Específico

Construir uma interface física para troca de dados com a fonte de soldagem:

- Desenvolver um protocolo de comunicação para esta interface.
- Desenvolver o protocolo de comunicação da mesa de soldagem.
- Desenvolver um *software* para Windows Mobile que utilize estas interfaces.

2 REVISÃO BIBLIOGRÁFICA

2.1 Soldagem

Os processos de manufatura de união e corte são muito importantes para a atividade industrial (K.L. Moore D.S. Naidu, 2003). Destes, o corte é um processo relativamente simples e bem conhecido, o qual pode ser feito mecanicamente ou termicamente, utilizando fontes de calor como oxiacetileno ou plasma (Connor, 1987). Por outro lado, os processos básicos de união, tais como os de junção mecânica, colagem, brazagem e *soldering* de soldagem são mais sofisticados (Norrish, 1992).

De acordo com a Sociedade Americana de Soldagem (Connor, 1987), a soldagem é definida como a coalescência localizada de metais ou não-metais, produzida ou pelo aquecimento do material até a temperatura de soldagem, com ou sem aplicação de pressão, ou pela aplicação de pressão apenas, com ou sem adição de metal. Embora existam 40 ou mais processos de soldagem, apenas poucos processos são importantes para indústria. Na soldagem, o eletrodo revestido (SMAW) e a soldagem MIG/MAG (GMAW) são os processos mais utilizados na indústria.

2.1.1 Soldagem a arco

O termo "soldagem a arco" refere-se a um grupo vasto de processos de soldagem que empregam um arco elétrico como fonte de aquecimento para fundir e unir os metais. Acredita-se que a soldagem a arco seja a terceira maior categoria de processo, atrás de montagem e usinagem, em todas as indústrias de fabricação de metal (Holmes, 1979).

A física da soldagem é tida como um complexo fenômeno físico associado com a soldagem, incluindo eletricidade, calor, magnetismo, luz e som. Em particular, a maioria dos processos requer a aplicação de calor ou pressão, ou ambos, para produzir uma boa união entre as partes a serem soldadas. Uma forma comum de aquecimento para soldagem é utilizando um fluxo de corrente elétrica através da resistência do contato entre as superfícies das duas peças de trabalho. Processos de soldagem que obtêm calor por meio de fontes externas são geralmente classificados de acordo com o tipo da fonte de soldagem utilizado. Segundo Amos (1991), os processos nesta categoria são:

a) *Shielded Metal Arc Welding* (SMAW) é um processo de soldagem a arco conhecido também como soldagem com eletrodo revestido, onde um arco elétrico é mantido

entre a ponta de um eletrodo revestido e a superfície do metal de base, produzindo o calor necessário para união (Eagar, 1992).

b) *Gas Tungsten Arc Welding* (GTAW) ou TIG é um processo de soldagem a arco que utiliza um arco entre um eletrodo não consumível de tungstênio e a poça de soldagem com um gás de proteção.

c) *Gas Metal Arc Welding* (GMAW) ou MIG/MAG é um processo de soldagem que utiliza um arco entre um eletrodo consumível e a poça de soldagem com um gás de proteção fornecido externamente, sem aplicação de pressão. Na Europa, GMAW é também chamado de Metal Inert Gas (MIG) ou Metal Active Gas (MAG).

d) *Flux-Cored Arc Welding* (FCAW) ou soldagem com eletrodo tubular é um processo de soldagem a arco que utiliza um arco elétrico entre um eletrodo consumível e a poça de soldagem, com uma proteção vinda do fluxo contido dentro do eletrodo tubular, com ou sem proteção de uma fonte de gás externa.

e) *Submerged Arc Welding* (SAW) ou soldagem com arco submerso é um processo de soldagem a arco que produz a união dos metais através do aquecimento e depois com um arco elétrico entre o eletrodo metálico e a peça de trabalho. O arco e o metal fundido estão submersos em um fluxo acima da peça de trabalho.

f) *Electro-gas Welding* (EGW) ou soldagem eletro-gás é um processo de soldagem a arco que utiliza um arco entre um eletrodo consumível e a poça de soldagem, utilizando uma proteção de gás em volta do eletrodo com fluxo.

g) *Plasma Arc Welding* (PAW) ou soldagem a plasma é um processo de soldagem que produz a coalescência de metais por meio do aquecimento por um arco elétrico entre o eletrodo e a peça de trabalho, com a proteção de um gás ionizado.

2.1.2 Processo de soldagem GMAW (MIG/MAG)

GMAW (*Gas-metal arc welding*) ou soldagem a arco elétrico com gás de proteção é um processo que funde e une metais aquecendo-os com um arco estabelecido entre um arame de adição continuamente alimentado e o metal de base, como na Figura 1 (Kou, 2003).

A proteção do arco e da peça de soldagem fundida é normalmente obtida utilizando-se um gás inerte com o argônio e hélio, e é por isso que GMAW é também chamado de *Metal-Inert Gas* (MIG). Considerando-se que gases não inertes, particularmente o CO_2, é também utilizado, o nome GMAW parece ser mais apropriado.

2.1.2.1 Gases de proteção

Argônio e hélio, e suas misturas, são utilizados em metais não ferrosos, assim como em ligas de aço e aço inoxidável. A energia do arco é menos dispersa em um arco de argônio do que em um arco de hélio por causa da baixa condutividade térmica do argônio. Conseqüentemente, o arco de argônio possui um núcleo com alta energia e uma manta externa com menos energia térmica. Isto ajuda a produzir uma transferência axial estável de gotículas de metal pelo plasma do arco de argônio.

2.1.2.2 Modos de transferência metálica

O metal fundido na ponta do eletrodo pode ser transferido para a poça de fusão por três tipos básicos de transferência: globular, spray e curto-circuito.

A. Transferência Globular: Gotas de metal maiores ou próximas ao diâmetro do eletrodo viajam através do arco sob a influência da gravidade. A transferência globular geralmente não é suave e produz respingos.
B. Transferência em *spray*: Acima de uma corrente crítica, possui pequenas gotas discretas de metal que viajam através do arco sob a influência de uma força eletromagnética em uma freqüência e velocidade muito superior ao modo transferência globular. A transferência metálica é muito mais suave e com pouco respingo. O nível de corrente crítica depende do material, do tamanho do eletrodo e da composição do gás de proteção.
C. Transferência em curto-circuito: O metal fundido na ponta do eletrodo é transferido do eletrodo para a poça de fusão quando ele toca a superfície da poça, isto é, quando o curto-circuito ocorre. A transferência por curto-circuito ocorre com as correntes mais baixas de soldagem e diâmetros de eletrodo. Ele produz uma pequena poça e de rápida solidificação desejada na soldagem de seções finas e soldagem sobre a cabeça.

2.1.2.3 Vantagens e desvantagens

Como o GTAW, o GMAW pode ser bem suave quando é empregado o gás inerte. A principal vantagem do GMAW sobre o GTAW é a maior taxa de deposição, o qual possibilita a pequenas peças serem soldadas em velocidades maiores. Os processos de dupla tocha e duplo arame aumentam ainda mais a taxa de deposição do GMAW. Não é necessária muita habilidade para manter um pequeno arco estável como no GTAW.

Entretanto, as tochas GMAW podem ser muito volumosas e difíceis de acessar pequenas áreas e cantos.

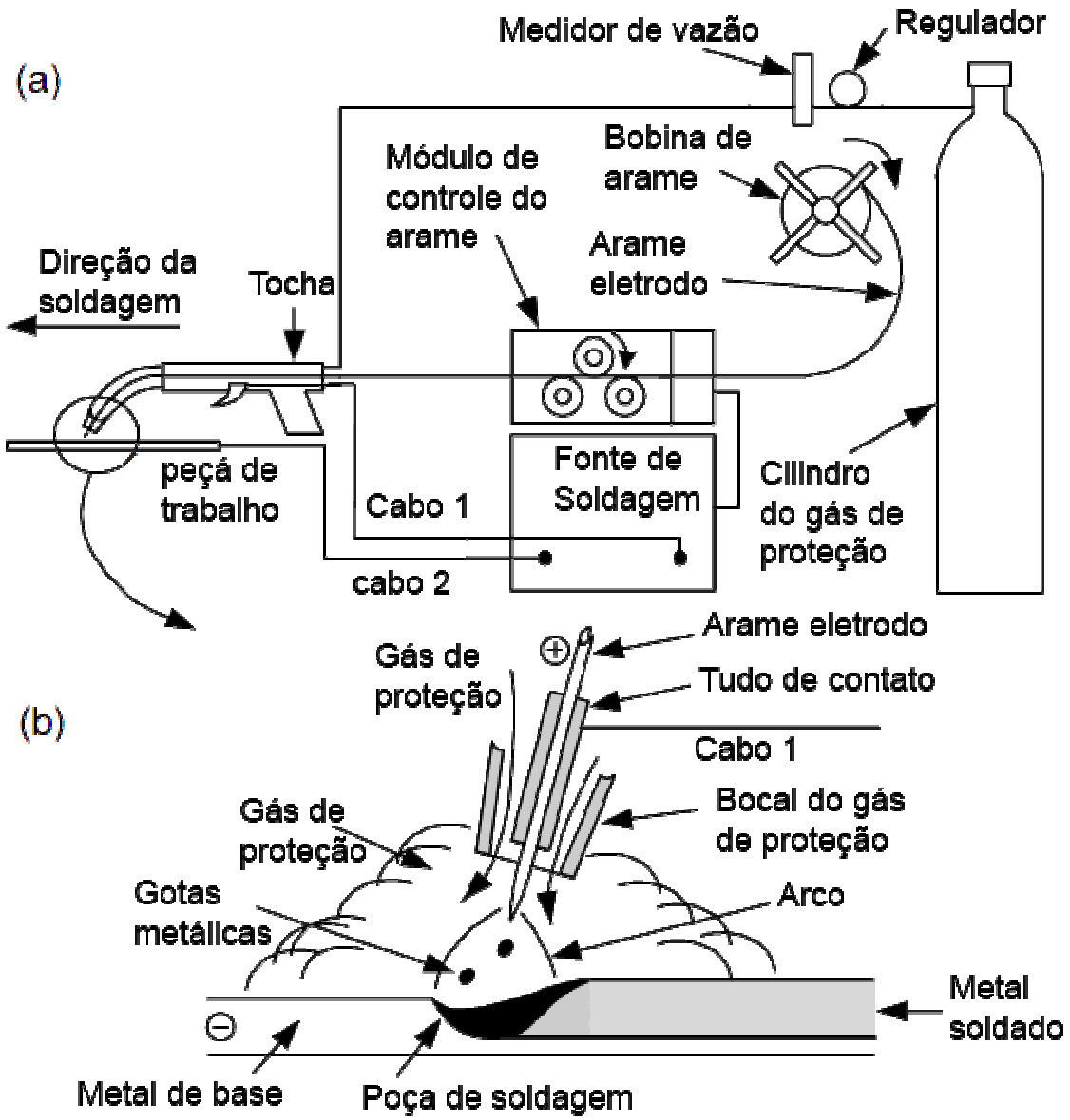


Figura 1 - GMAW (a) esquema do processo MIG/MAG ; (b) Região da poça de fusão. Imagem traduzida (Kou, 2003).

2.2 FONTES DE SOLDAGEM

Todo processo de soldagem requer alguma forma de energia para fundir e unir. Em geral as fontes de energia de soldagem são agrupadas dentro das cinco categorias (K.L. Moore D.S. Naidu, 2003):

1. Fontes elétricas são usadas para soldagem a arco, soldagem por resistência e soldagem por eletroescória;
2. Fontes químicas usadas em soldagem oxiacetileno e soldagem térmite.
3. Fontes por foco são usadas em lasers, feixe de laser pulsado, feixe de laser contínuo e feixe de elétrons;
4. Fontes destinadas à soldagem por fricção, soldagem por ultrassom e soldagem por explosão;
5. Outras fontes, destinadas para soldagem por difusão;

A tensão fornecida pelas companhias elétricas para a maioria das operações industriais é: 120V, 240V ou 480V com baixas correntes. As correntes usadas no processo de soldagem são muito altas, variando entre 30 a 1500 A, porém, as tensões são baixas, na faixa de 20 a 80 V. Por isso é necessário um equipamento de conversão para converter a alta tensão em baixa. A corrente necessária para soldar pode ser contínua (CC), alternada (CA) ou ambas. O equipamento de conversão geralmente utilizado é um ou uma combinação de:

1. Transformador - converte CA de baixa corrente e alta tensão para CA com alta corrente e baixa tensão;
2. Um motor-gerador destinado para converter CA ou CC para CA ou CC;
3. Um conversor de estado sólido, tais como um retificador controlado de silício ou um tiristor para converter CA para CC.
4. Um inversor de estado sólido para converter CC para CA.

Um simplificado sistema de classificação de várias fontes de soldagem pode ser visto na Figura 2 (Cary, 1989).

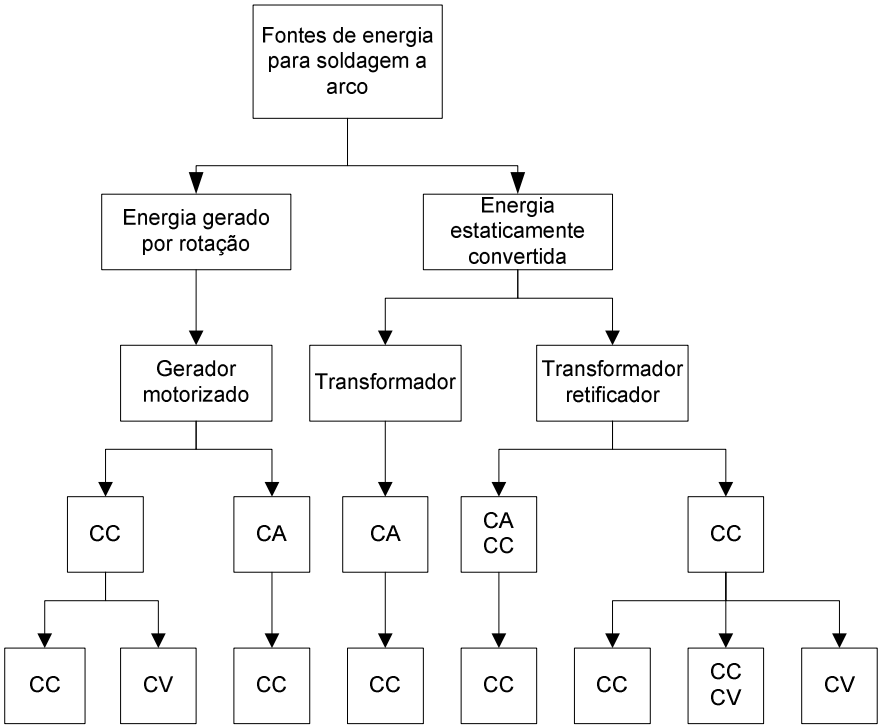


Figura 2 - Classificação das fontes de energia para soldagem. Imagem traduzida (K.L. Moore D.S. Naidu, 2003).

Todas as fontes de soldagem são descritas por dois tipos de características de operação: estática e dinâmica. A curva estática, relacionada à saída de tensão com a saída de corrente, é obtida sob condições estáticas, utilizando carregamentos resistivos. A curva dinâmica é determinada por meio da medição do transiente das saídas de tensão e de corrente. Uma importante característica das fontes de soldagem é o ciclo de trabalho, que é definido como a relação entre o tempo de arco (tempo de carga) com o tempo total do teste especificado. Então 60% de ciclo de trabalho significam que em 6 minutos a cada 10 a fonte irá fornecer a corrente de trabalho para o processo. Uma fonte de trabalho de 100% de ciclo de trabalho é projetada para fornecer a corrente de trabalho continuamente, sem exceder o máximo de temperatura aceitada. Para processos automáticos ou semi-automáticos, a fonte de soldagem deverá ter 100% do ciclo de trabalho.

2.2.1 Fonte de corrente constante (CC)

Uma fonte de soldagem de corrente constante apresenta uma característica volt-ampere estática que tende a produzir um corrente quase constante. Então, se o comprimento do arco variar por causa de alguma condição externa forçando o arco a variar a tensão, a corrente permanecerá substancialmente constante (Amos, 1991). Na vizinhança de qualquer ponto de operação a mudança na corrente é bem menor que a mudança na tensão.

A tensão em aberto ou sem carga é maior do que a tensão com carga, dependendo da resistência equivalente de todo o sistema.

Uma curva típica volt-ampere para fontes de soldagem convencionais é mostrada na Figura 3(a). Note a característica de queda da curva com declive negativo. Do ponto de operação P, um aumento relativamente grande na tensão resultará em um decréscimo relativamente baixo na corrente. Então, quanto maior o declive da curva, melhor ela funciona como uma fonte de corrente constante. Portanto em um processo com eletrodo consumível, a taxa de fusão do eletrodo será praticamente constante, mesmo com uma pequena mudança no comprimento do arco.

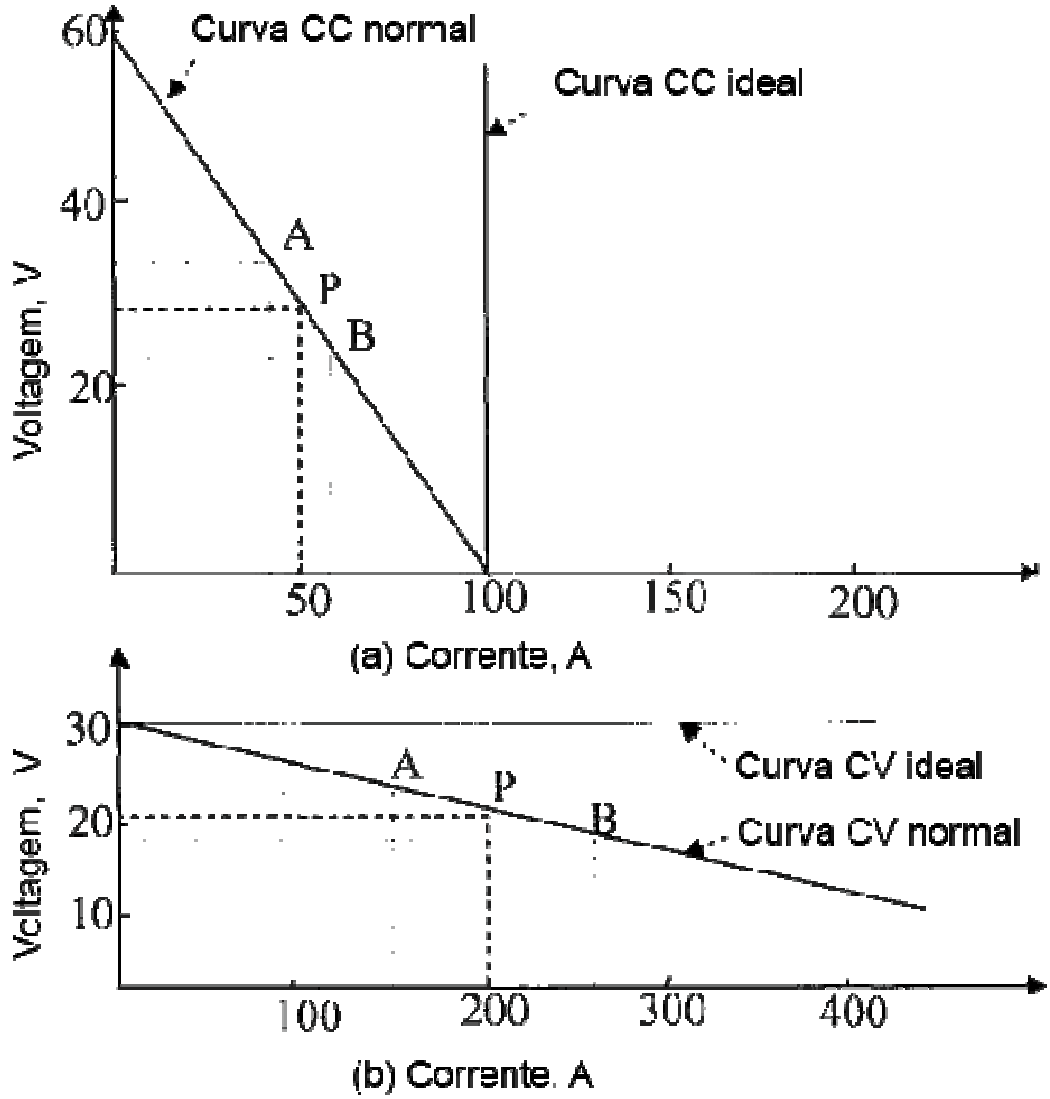


Figura 3 - Curva volt-ampere típica para: (a) fontes de soldagem de corrente constante e (b) fontes de soldagem de tensão constante. Imagem traduzida (K.L. Moore D.S. Naidu, 2003).

2.2.2 Fontes de tensão constante

Em uma fonte de soldagem com tensão constante a característica estática volt-ampere é tal que a fonte de soldagem fornece uma tensão de carregamento praticamente constante. Uma fonte de tensão constante é normalmente utilizada com um eletrodo consumível continuamente alimentado (Amos, 1991).

Uma curva volt-ampere para uma fonte de tensão constante pode ser vista na Figura 3(b). Onde, do ponto de operação P, uma grande mudança na corrente pode ser tolerado, com uma pequena mudança relativamente na tensão. Esta característica da fonte de soldagem com tensão constante é adequada para processos de soldagem com alimentação constante do eletrodo, tais como GMAW. Uma pequena mudança no comprimento do arco (e conseqüentemente da tensão) resultará em uma mudança relativamente alta na corrente, o qual irá conduzir automaticamente a um aumento ou uma diminuição da taxa de derretimento do eletrodo para restabelecer o comprimento do arco desejado. Este fenômeno é chamado de auto-regulação.

2.2.3 Fontes de energia de corrente constante (CC) e tensão constante (TC) combinada

A combinação das características da CC e TC pode ser obtida por uma única fonte de energia utilizando-se de uma variedade de circuitos de realimentação. Uma curva característica pode ser mostrada na Figura 4.

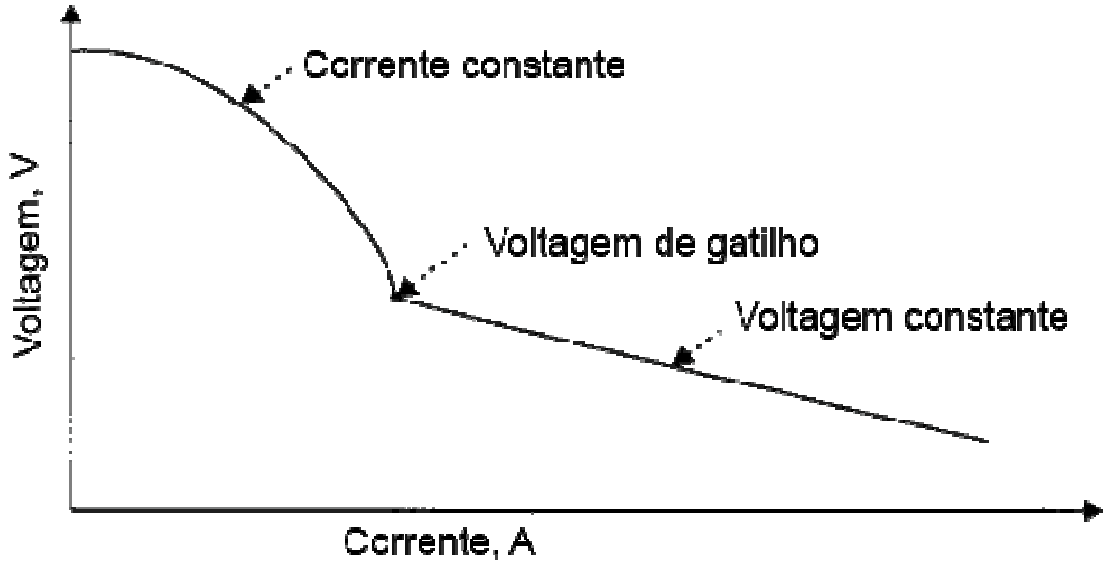


Figura 4 - Curva volt-ampere típica para fontes de energia CC e VC conbinada. Imagem traduzida (K.L. Moore D.S. Naidu, 2003).

2.2.4 Corrente pulsada

As fontes de energia mais utilizadas para o processo GMAW e GTAW são fontes de corrente pulsada. Fontes de corrente pulsada para GMAW são utilizadas para reduzir a energia do arco e a taxa de deposição enquanto preserva o modo spray desejado.

2.2.5 Inversoras

Com a introdução dos microprocessadores, as primeiras fontes de energia se tornaram inversores utilizando-se de retificadores controlado de silício. Uma fonte de energia classe H que era mais eficiente que a fonte de energia convencional classe A foi construída utilizando-se transistores chaveados e diodos (K. Andersen, 1989). Esta fonte de soldagem fornece até 45 V e 500 A com uma freqüência de resposta de várias dezenas de kHz.

2.2.6 Soldagem Automatizada

A soldagem industrial robotizada evoluiu ligeiramente, mas está longe de ser um processo tecnológico consolidado, pelo menos, de um modo geral (Weman, 2003). O processo de soldagem é complexo, difícil de parametrizar, monitorizar e controlar eficazmente (Rosheim, 1994) (Kusiak, 1986). Na verdade, a maioria das técnicas de soldagem não são totalmente compreendidas, ou seja, os efeitos sobre as juntas de soldagem. Por isso estas técnicas são realizadas utilizando como base modelos empíricos obtidos através da experiência em condições específicas. Os efeitos do processo de soldagem em superfícies soldadas não estão totalmente compreendidos. Soldagem pode, na maioria dos casos, impor temperaturas extremamente altas concentradas em pequenas zonas. Fisicamente, faz com que o material sofra expansões térmicas e ciclos de contração extremamente elevados, que introduzem alterações nos materiais que podem afetar o seu comportamento mecânico, juntamente com a deformação plástica (Bolmsjo, 1997) (Loureiro, 1998). Essas mudanças devem ser bem conhecidas, a fim de minimizar os efeitos.

Considerando o processo de soldagem MIG/MAG, a estabilidade do processo de soldagem é altamente sensível aos principais parâmetros de soldagem, especialmente corrente, tensão, velocidade do arame, *stick-out* (comprimento do arame a partir do contato elétrico), gás de proteção e comprimento do arco. Uma pequena mudança na distância entre a tocha de soldagem e o componente sendo soldado pode produzir uma considerável variação na corrente e na tensão. Corrente, tensão e gás de proteção influenciam o modo de transferência do metal de adição ao componente sendo soldado.

2.3 TELEOPERAÇÃO / TELEROBÓTICA / TELEMANIPULAÇÃO

O termo Tele é derivado do Grego e significa distância, generalizado para caracterizar a existência de uma barreira entre o usuário e o ambiente de trabalho. Esta barreira poderá ser transposta por meio de um controle remoto do robô no ambiente. Além da distância, outros tipos de barreiras podem ser impostas, por exemplo, perigo, grandes tarefas e pequenas tarefas de trabalho. Tais barreiras têm em comum que o usuário não pode ter acesso ao ambiente de trabalho.

Telerobótica é considerada como a robótica à distância, a qual é operada e controlada por pessoas e é uma das subáreas mais antiga da robótica. Qualquer decisão de alto nível, planejamento ou cognitiva são feitas por um usuário humano; já a execução mecânica é feita pelo robô (Siciliano & Oussama, 2008).

Mesmo que a separação física possa ser bem pequena entre o operador humano e o robô, podendo até compartilhar uma mesma sala, os sistemas telerobóticos são geralmente divididos em pelo menos duas partes: a primeira parte poderá ser constituída com o operador humano bem como todos os elementos necessários para possibilitar a conexão com o usuário, tais como joysticks, monitores, teclados e outros dispositivos de entrada/saída e, do outro lado, a parte remota que contém o robô, seus sensores e os elementos de controle. Uma recente inovação na telerobótica foi o uso de redes de computadores para transmitir informação entre as duas partes.

Muitos sinônimos são utilizados na telerobótica, tais como teleoperação e telemanipulação. O termo Telerobótica é o mais difundido e enfatiza o controle remoto de um robô por um humano. Teleoperação enfatiza a operação no nível das tarefas, enquanto a telemanipulação destaca a manipulação no nível do objeto.

Dentro da telerobótica, muitos tipos de arquitetura de controles têm sido empregadas, sendo as mais utilizadas:

- **Controle direto ou controle manual**, indica que o usuário controla diretamente o movimento do robô sem qualquer assistência automática.
- **Controle supervisionado** – representa que os comandos do usuário e retorno das informações ocorrem em um nível mais elevado e para tanto o robô requer certa autonomia e "inteligência".

- **Controle compartilhado**, este tipo de controle permite um grau mínimo de autonomia ou ajuda automática para dar assistência ao usuário.

Na grande maioria dos sistemas envolve parte de controle direto com o uso de um joystick ou um dispositivo similar na interface com o usuário para que possa aceitar os comandos do usuário. O joystick sendo um instrumento mecânico composto por sensores, este pode ser visto como um robô. O robô local é chamado de mestre e o robô remoto é definido como escravo, enquanto o sistema é referenciado com um sistema mestre-escravo. Para permitir o controle direto, o robô escravo é programado para seguir os movimentos pelo robô mestre, o qual está posicionado ao lado do usuário. Muitas vezes o robô mestre é uma réplica cinemática do robô escravo, provendo uma interface intuitiva.

Finalmente, a telepresença é freqüentemente definida como o objetivo final dos sistemas mestre-escravo e telerobótica em geral. Permite ao usuário não só a habilidade de manipular o ambiente remotamente, mas também identificar o ambiente de trabalho. O operador humano recebe informações de forma que se sinta presente no local remoto. O sistema mestre-escravo torna-se o meio com o qual o usuário interage com o ambiente remoto de tal forma que, idealmente, imagina-se em contato direto com o objeto, transformando o meio de controle propriamente dito, imperceptível. Se isso for atingido, é dito que o sistema mestre-escravo é transparente. (Manuel Ferre, 2007)

2.3.1 Interfaces

Uma interface, em ciência da computação, é a fronteira que define a forma de comunicação entre duas entidades (Christa Sommerer, 2008). Ela pode ser entendida como uma abstração que estabelece a forma de interação da entidade com o mundo exterior, através do encapsulamento dos detalhes internos da operação, permitindo que esta entidade seja modificada sem afetar as entidades externas que interagem com a mesma. Uma interface também pode promover um serviço de tradução para entidades que não falam a mesma linguagem, como no caso de humanos e computadores.

O conceito de interface é utilizado em diferentes áreas da ciência da computação e é importante no estudo da interação homem-máquina, no projeto de dispositivos de *hardware*, na especificação de linguagens de programação e também em projetos de desenvolvimento de *software*. A interface existente entre um computador e um humano é conhecida como interface do usuário, e as interfaces utilizadas para conectar componentes de *hardware* são chamadas de interfaces físicas.

2.3.1.1 Interface em programação e APIs

Em programação, a utilização de interfaces permite a composição de componentes de um *software* sem que a sua codificação seja conhecida. Um exemplo clássico de utilização de interfaces é o do sistema operacional que, por meio de uma interface de programação de aplicativos, permite que os programas utilizem os recursos do sistema (memória, CPU e etc) sem que os seus detalhes de codificação sejam conhecidos pelo programador. Este esquema isola e protege o sistema operacional de eventuais erros cometidos por uma aplicação.

Uma "*Application Programming Interface* (API)" (ou Interface de Programação de Aplicativos) é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades, por programas aplicativos, isto é, programas que não devem ser envolvidos em detalhes da codificação do *software*, mas apenas utilizar os serviços.

De modo geral, uma API é composta por uma série de funções acessíveis somente por programação, as quais permitem utilizar características do *software* menos evidentes para um usuário tradicional.

Por exemplo, um sistema operacional possui uma grande quantidade de funções na API, que permitem ao programador criar janelas, acessar arquivos, criptografar dados, etc. Mas a API dos sistemas operacionais costuma ser dissociada de tarefas mais essenciais, como manipulação de blocos de memória e acesso a dispositivos. Estas tarefas são atributos do *kernel* ou núcleo do sistema, e raramente são programáveis.

2.3.1.2 Interface física

Uma interface física, ou conector, é um dispositivo que efetua a ligação entre uma porta de saída de um determinado equipamento e a porta de entrada de outro (por exemplo, entre um computador e um periférico).

2.3.1.3 Interface (do usuário) de tela pequena

As telas pequenas são conhecidas como uma ferramenta para exibir informações dinâmicas e têm permitido uma aplicação significativa na indústria moderna (Kortum, 2008). Estas variam de telas muito simples, como pode ser visto em relógios, microondas, sistemas de alarme, e assim por diante, até telas gráficas altamente eficazes, como pode ser visto em telefones móveis, dispositivos médicos, minigames, e Assistentes Pessoais Digitais

(Personal Digital Assistant - PDAs). Tais produtos têm permitido alcançar uma fatia significativa do mercado, alcançando recordes de venda. As vendas de telefones celulares, que é apenas uma categoria de produtos de pequena tela, foram estimadas em 825 milhões de unidades em 2005 (Gohring, 2006) e têm mais de 2,6 bilhões de assinantes no mundo inteiro (Nystedt, 2006). Em comparação, as vendas de computador pessoal (PC) foram estimadas em 208,6 milhões de unidades em 2005 (Williams, 2006).

2.3.1.3.1 Tecnologia das telas

Até 1970, a, principal tecnologia de telas disponível aos projetistas eletrônicos era o tubo de raios catódicos (CRT). Embora o CRT seja ainda utilizado, ao lado de novas tecnologias em algumas áreas (em especial, na área de manufatura), este apresente dois grandes problemas: tamanho e consumo de energia. Os tubos de raios catódicos requerem grande quantidade de energia para disparar elétrons em uma tela a uma distancia relativamente grande. Apesar das melhorias contínuas no processo de engenharia do CRT, estas duas desvantagens ainda não foram superadas.

Em 1970, dois pesquisadores suíços, Schadt e Helfrich, construíram a primeira tela de cristal líquido (LCD) (Kortum, 2008). Enquanto os projetos de LCDs originais apresentaram suas próprias falhas, ou seja: alto custo, longo tempo de resposta e baixo contraste, a tecnologia superou as limitações do CRT, demandando menor consumo de energia e reduzida profundidade da tela. O LCD tem sofrido várias melhorias técnicas, o que levou ao aparecimento de diversos produtos com pequenas telas, tais como relógios digitais até poderosos laptops.

2.3.1.3.2 Tamanho, resolução e densidade de pontos da Tela

As telas são definidas pelo tamanho e pela resolução (número de pontos luminosos que ela pode mostrar). O tamanho físico da tela é medido pelo comprimento da diagonal em polegadas. As telas normalmente utilizam uma relação horizontal/vertical de 4:3. As novas telas chamadas de "wide-screen" apresentam relação de 16:9 e outras relações menos tradicionais (ex.: 3:2, 5:4 ou 16:10) em aplicações específicas.

A resolução da tela é fundamental para definir a quantidade de informação que pode ser apresentada na tela. Os pontos por polegadas (DPI) é a medida do número de pontos luminosos dentro de uma polegada quadrada da tela. Quanto maior o valor do DPI, maior é a densidade de pontos na tela e, consequentemente, melhor será a qualidade da mesma. Um

valor em DPI muito utilizado para telas é 72 pontos por polegada quadrada, entretanto, novas telas estão sendo comercializadas com resoluções acima de 200 DPIs.

Outra medida comum de resolução é o número de pontos horizontais e verticais que a tela é capaz de mostrar. A resolução de dispositivos de pequenas telas pode variar de 16x16, presente em telas de relógios, até telas que apresentam 800x600 pontos, que são encontradas em dispositivos com tecnologia de ponta. Novos dispositivos estão continuamente pressionando a indústria para disponibilizar telas com resoluções mais altas. Vários padrões de resolução podem ser vistos na Tabela 1.

Tabela 1 - Padrões de resolução de pequenas telas (Kortum, 2008).

Nome	Razão de aspecto	Resolução
QQVGA	4:3	160 x 120
QCIF	1.22:1	176 x 144
QCIFþ	4:5	176 x 220
QVGA	4:3	320 x 240
CGA	16:5	640 x 200
EGA	64:35	640 x 350
VGA	4:3	640 x 480
SVGA	4:3	800 x 600
XGA	4:3	1024 x 768

2.3.1.3.3 Smart Phones e PDAs

Os usuários não só querem dispositivos menores como também querem dispositivos que tenham tecnologia e capacidade de rede avançadas (Sarker, 2003). A combinação de baterias melhoradas, miniaturização de componentes e uma aparentemente interminável série de novas funcionalidades de *hardware* e *software* estão transformando dispositivos de tela pequena em verdadeiros computares portáteis.

A desvantagem do aumento da complexidade tecnológica, em geral, é uma redução na usabilidade. Este é o caso de muitos dispositivos de tela pequena de ponta. No entanto, uma melhor integração dos recursos está tornando mais fácil para os usuários a compreensão e o gerenciamento da complexidade de dados multimídia. PDAs com foco em usabilidade podem fornecer uma experiência computacional integrada fácil, apesar da complexidade da tecnologia.

2.3.2 Comunicação

Um canal de comunicação é um caminho sobre o qual a informação pode trafegar. Ela pode ser definida por uma linha física (fio) que conecta dispositivos de comunicação, ou por radiofreqüência, laser, ou outra fonte de energia radiante.

2.3.2.1 Comunicação Digital (Porta Serial)

Em comunicação digital, a informação é representada por bits de dados individuais, que podem ser encapsulados em mensagens de vários bits. Um byte (conjunto de 8 bits) é um exemplo de uma unidade de mensagem que pode trafegar através de um canal digital de comunicações. Um conjunto de *bytes* pode ser agrupado em um "frame" ou outra unidade de mensagem de maior nível. Esses múltiplos níveis de encapsulamento facilitam o reconhecimento de mensagens e interconexões de dados complexos.

Um canal no qual a direção de transmissão é inalterada é denominado como canal *simplex*. Por exemplo, uma estação de rádio é um canal simplex porque ela sempre transmite o sinal para os ouvintes e nunca é permitida a transmissão inversa.

Um canal *half-duplex* é um canal físico simples no qual a direção pode ser revertida. As mensagens podem fluir nas duas direções, mas nunca ao mesmo tempo. Em uma chamada de "walkie-talkie", uma parte fala enquanto a outra escuta. Depois de uma pausa, a outra parte fala e a primeira escuta. Falar simultaneamente resulta em sons que não podem ser compreendidos.

Um canal *full-duplex* permite que mensagens sejam trocadas simultaneamente em ambas as direções. Ele pode ser visto como dois canais *simplex*, um canal direto e um canal reverso, conectados nos mesmos pontos.

2.3.2.2 Porta serial

A maioria das mensagens digitais é mais longa que alguns poucos bits. Por não ser prático nem econômico transferir todos os bits de uma mensagem simultaneamente, a mensagem é quebrada em partes menores e transmitida seqüencialmente. A transmissão bit-serial converte a mensagem em um bit por vez através de um canal. Cada bit representa uma parte da mensagem. Os bits individuais são então rearranjados no destino para compor a mensagem original. Em geral, um canal irá passar apenas um bit por vez. A transmissão bit-serial é normalmente chamada de transmissão serial, e é o método de comunicação escolhido por diversos periféricos de computadores (Axelson, 2007).

A transmissão byte-serial converte 8 bits por vez através de 8 canais paralelos. Embora a taxa de transferência seja 8 vezes mais rápida que na transmissão bit-serial, são necessários 8 canais, e o custo poderá ser maior do que 8 vezes para transmitir a mensagem. Quando as distâncias são curtas, é factível e econômico usar canais paralelos como justificativa para as altas taxas de transmissão.

2.3.2.3 Protocolos e formatos

Uma porta serial de saída que funciona com um transmissor, ou dispositivo, envia bits um de cada vez para uma entrada de porta serial que funciona com um receptor, geralmente em outro computador. O cabo entre os dois computadores tem normalmente um caminho dedicado para cada direção. Algumas interfaces seriais apresentam um único caminho compartilhado com os transmissores revezando.

2.3.2.3.1 Comunicação assíncrona e síncrona

Em um protocolo assíncrono, a interface não inclui a linha de *clock*. Ao invés disso, cada computador fornece o seu *clock* para ser usado como referência de tempo. Computadores devem sincronizar a freqüência do *clock*, e as freqüências entre os computadores não podem ter grandes discrepâncias. Um *start bit* transmitido sincroniza os *clocks* do transmissor e do receptor. Em contraste, num protocolo síncrono, a interface inclui uma linha de *clock* normalmente controlada por outros computadores, e todos os bits transmitidos são sincronizados por este *clock*. Cada bit transmitido é válido após um tempo, definido para borda de subida ou descida do *clock*, dependendo do protocolo.

2.3.2.3.2 Tamanho da palavra

O UART (circuito integrado responsável pelas comunicações através de uma porta serial) transmite os dados em partes, denominados palavras. Cada palavra contém um *Start* bit, bits de dados, um bit opcional de paridade, e um ou mais *Stop* bits. A maioria dos UARTs suporta vários formatos de palavra. Um formato comum é 8-N-1, onde o transmissor envia cada palavra como um *start* bit, seguido por oito bits de dados e um *Stop* bit. Os bits de dados são transmitidos começando com o bit 0 (o bit menos significativo, ou LSB) (Figura 5). O "N" em 8-N-1 indica que as palavras não contêm um bit de paridade. Formatos que utilizam a bit de paridade podem utilizar um tipo de paridade *even, odd, mark* ou *space*. Um exemplo de formato usando paridade é 7-E-1 em que o transmissor envia um *Start* bit, sete bits de dados, um bit de paridade *even*, e um Stop bit.

O bit de paridade pode prover uma forma básica de detecção de erro. Se a paridade for *even*, os bits de dados somados aos bits da palavra deve resultar em quantidades pares do algarismo 1. Se a paridade for *Odd*, a somatória dos bits deve resultar em uma quantidade ímpar do algarismo 1.

Paridades *Mark* e *Space* são formas de paridade fixa. Com a paridade *Mark*, o bit de paridade é sempre um, e com a paridade s*pace* o bit de paridade é sempre zero.

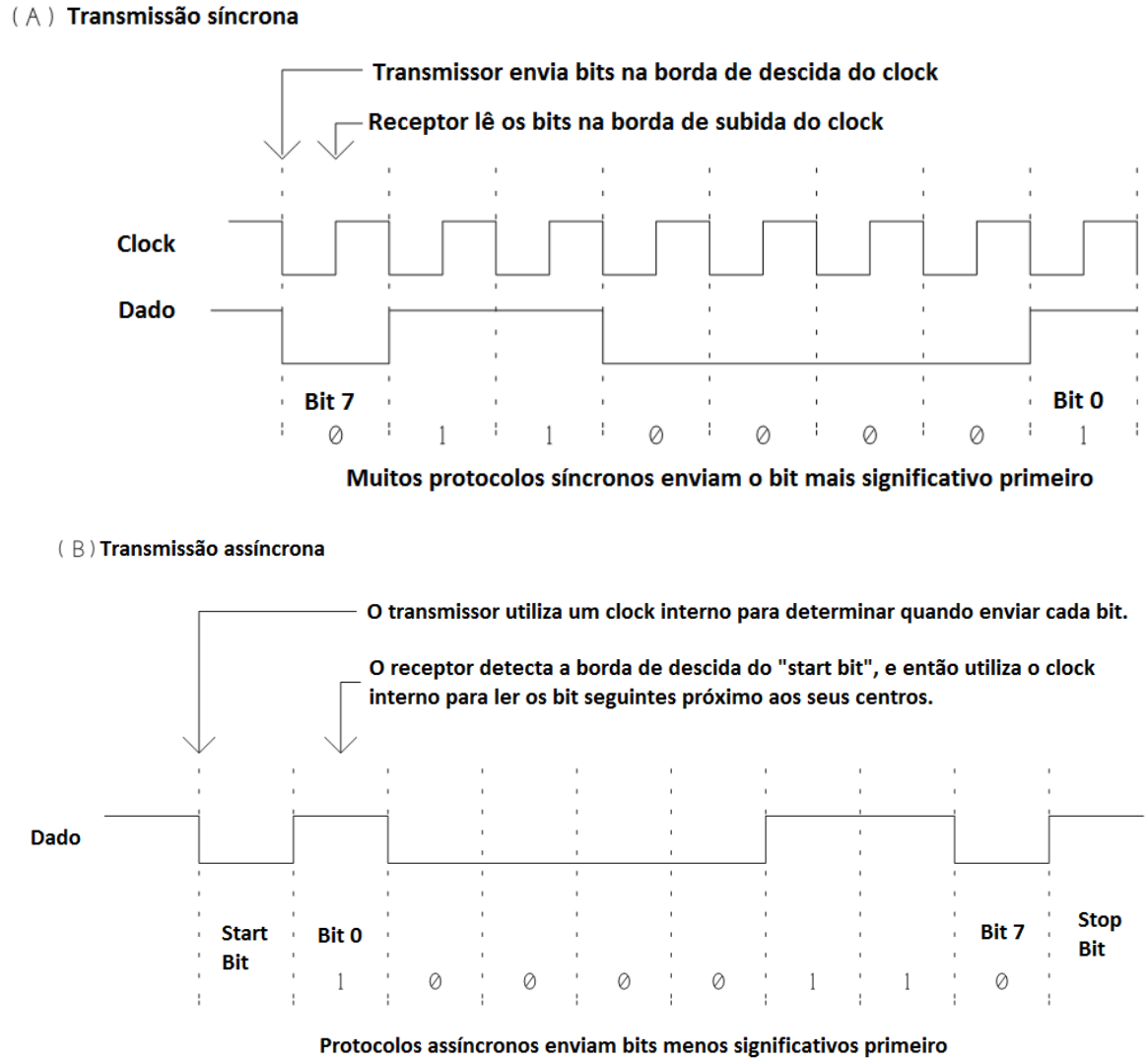


Figura 5 - Transmissões síncronas incluem a linha de clock, enquanto a transmissões assíncronas requerem que cada computador tenha o seu clock (Axelson, 2007).

Para receptores que utilizam tempo extra para aceitar o dado recebido, alguns UARTs possibilitam ao transmissor prorrogar o tempo do stop bit para o comprimento de 1,5 a 2 bits. O propósito original do stop bit mais longo era proporcionar às antigas máquinas de teledigitação maior tempo para voltar a um estado inativo.

2.3.2.3.3 Bit Rate e Baud Rate

Bit rate é o número de bits transmitidos ou recebidos por unidade de tempo, geralmente expresso em bits por segundo (bps). *Baud rate* é número de possíveis eventos, ou transições de dados por segundo. Em transmissões digitais por cabo básicas cada período de transição de dado representa um bit de dado, e o bit rate e o baud rate são iguais. Em linhas telefônicas, *modems* de alta velocidade usam chaveamento de fase e outras técnicas para codificar múltiplos bits em cada período de transição do dado, resultando em um baud rate que é menor que o bit rate. No uso popular, entretanto, o termo baud rate geralmente refere-se ao bit rate.

O número de caracteres transmitidos por segundo é igual ao bit rate dividido pelo número de bits da palavra. Com o formato 8-N-1, a transferência de *bytes* é 1/10 do bit rate porque cada palavra contém 10 bits: 1 start bit, 8 bits de dados e 1 stop bit. Então, um link de 9600 bps usando o formato 8-N-1 pode transmitir 960 *bytes* por segundo.

2.3.2.3.4 Controle de Fluxo

O controle de fluxo proporciona ao computador transmissor indicar a presença de dados a ser enviados, e o computador receptor pode informar sua disponibilidade de receber os dados. Os computadores devem usar o controle de fluxo numa conexão serial a menos que os *buffers* de recebimento sejam grandes o suficiente para armazenar todos os dados transmitidos até que o computador receptor possa ler. Em uma forma comum de controle de fluxo de *hardware*, o receptor ajusta uma linha dedicada para um estado definido quando estiver pronto para receber dados. O computador transmissor checa o estado da linha antes de enviar os dados. Se a linha não estiver no estado adequado, o computador transmissor aguarda para enviar futuramente. Controle de fluxo em duas direções requer uma linha para cada direção. Controle de fluxo é algumas vezes chamado de *handshaking*. Entretanto, um *handshake* completo requer uma comunicação de duas vias: o computador transmissor indica que existem dados a serem enviados, e o receptor indica disponibilidade de recebimento destes.

A especificação RS-232 atribui nomes a sinais de fluxo de controle. Em um PC, o sinal de entrada é *Clear To Send (CTS)* e o sinal de saida é *Request to Send (RTS).* O cabo que conecta dois PCs deve conectar cada saída RTS à entrada CTS do outro computador. Uma tensão positiva no RS-232 significa pronto para receber e uma tensão negativa significa que não está pronto.

Microcontroladores geralmente não possuem linhas CTS e RTS dedicadas. O *firmware* do dispositivo pode usar qualquer pino sobressalente para controle de fluxo. Códigos fonte podem usar nomes RTS e CTS ou nomes, tais como *flow control_in* e *flow_control_output* para evitar equívocos na identificação do sinal de entrada e de saída.

Dois sinais adicionais de fluxo de controle no RS-232 são o *Data Terminal Ready* (DTR) e *Data Set Ready* (DSR). Essas linhas foram definidas para fornecer informação sobre o estado da linha de telefone ou outro canal de comunicação em um modem que se conecta via RS-232 com outro computador ou terminal. Em PCs, o DTR é uma saída e o DSR é uma entrada. Os microcontroladores normalmente não possuem linhas DTR e DSR dedicadas. Portas sobressalentes com suporte do *firmware* podem prover estes sinais quando necessários.

Algumas ligações podem usar controle de fluxo por *software*, onde o computador receptor envia um código *Xon* para indicar que está pronto para receber e envia um código *Xoff* que informa ao transmissor para parar de enviar. Este método só funciona quando estiver mandando dados como texto puro ou outra codificação que não utiliza os códigos *Xon* e *Xoff*. O código *Xon* é normalmente 11, e o *Xoff* é 13. Alguns *software*s habilitam a opção de selecionar outros códigos.

Uma ligação pode usar métodos de controle de fluxo de *hardware* e *software* ao mesmo tempo. O computador transmissor envia dados somente quando a linha CTS do computador remoto estiver alta e o computador transmissor não recebeu um *Xoff*.

2.3.2.3.5 Buffers

Buffers de *hardware* e *software* podem ajudar a prevenir dados perdidos e possibilitar a transferência o mais rápido possível. Os *buffers* podem armazenar dados recebidos e dados esperando ser enviados. Em uma porta de controle de fluxo, um *buffer* de recebimento pode prevenir a perda dos dados através do armazenamento, até que o programa possa recuperar os dados.

Em uma porta com controle de fluxo e *buffer* de recebimento, o computador emissor envia grandes quantidades de dados mesmo se o receptor não conseguir processar esses dados imediatamente. *Buffers* de transmissão podem possibilitar ao *software* armazenar dados a serem enviados e avançar para outras tarefas.

Os *buffers* podem ser de *hardware*, de *software* ou ambos. Portas seriais em PCs normalmente têm *buffers* de *hardware* de 16 *bytes* incorporados no UARTs. Na direção de recebimento, o UART pode armazenar um total de 16 *bytes* até que o *software* possa lê-los. Na direção da transmissão, o UART pode armazenar até 16 *bytes* antes de transmiti-los, usando o protocolo selecionado. Alguns UART, incluído aqueles contidos em muitos dispositivos de portas COM virtuais, apresentam um maior *buffer*.

Drivers de porta COM em PCs mantêm um *buffer* de *software* que é programável e pode ser tão grande quanto a memória do sistema permitir. O *driver* transfere dados entre o *buffer* de *software* e *hardware* quando necessário.

Buffers em microcontroladores tendem a ser pequenos. Alguns UARTs não apresentam nenhum *buffer*. Um computador contendo um pequeno *buffer,* ou mesmo sem nenhum, poderá utilizar outros métodos para prevenir a perda de dados.

2.3.2.3.6 Checando erros

Um receptor pode usar checagem de erro para verificar se todos os dados chegaram corretamente. Formas de checar por erros em mensagens incluem bits de paridade, *checksum* e dados duplicados.

Ao se utilizar bibliotecas de porta serial em aplicações para PC, a aplicação só precisará selecionar o tipo da paridade. O *software* calcula automaticamente e posiciona o bit de paridade correto em cada palavra transmitida e pode acusar um erro ao receber dados com paridade incorreta. O *hardware* do microcontrolador e o *software* podem precisar que o *firmware* calcule e gere ou cheque o bit de paridade para cada palavra transmitida ou recebida.

O *checksum* caracteriza-se por um valor de checagem de erro obtido por meio de uma operação lógica ou matemática no conteúdo de um bloco de dados. A aplicação pode escolher entre uma variedade de métodos de calcular o *checksum*.

Um cálculo de *checksum* básico adiciona valores dos *bytes* contidos no bloco e usa o byte menos significativo do resultado como *checksum*. Um *checksum* para dados ASCII podem adicionar valores representados por cada par de caracteres.

O método de checagem de redundância cíclica (CRC) utiliza cálculos mais complexos para obter o valor de *checksum*.

Valores *Hash* representam *checksum*s muito seguros produzidos por funções *Hash* de códigos de detecção de mensagem. Para utilizar valores *Hash*, o emissor e receptor devem compartilhar a mesma chave, o qual é usado para criar o valor do *Hash* e verificar os dados recebidos.

O computador que receber os dados com *checksum* pode repetir o cálculo e obter novamente o *checksum*. Se o valor do *checksum* obtido pelo computador receptor não coincidir com o valor enviado originalmente, o sistema assume que não recebeu o mesmo dado transmitido pelo computador emissor. O computador receptor que detectou um erro pode notificar o computador emissor para que este possa tentar enviar os dados novamente ou realizar outra ação. Após um número de tentativas mal sucedidas, o computador transmissor pode: desistir, mostrar uma mensagem de erro ou um alarme sonoro quando necessário. Uma das desvantagens do *checksum* é que este gera uma sobrecarga em grandes blocos de dados.

O computador receptor deverá saber o que fazer quando a mensagem for menor do que o esperado ou o fim da mensagem não chegar. Ao invés de esperar para sempre, o *software* deve eventualmente interromper o recebimento e tentar notificar o computador emissor, se necessário.

Em outra forma de checagem de erro, o transmissor pode enviar cada mensagem duas vezes e o receptor verifica se as mensagens são iguais. É claro que isto significa que cada mensagem irá demorar o dobro do tempo para ser transmitida. Enviar dados duplicados pode ser útil quando os dados forem envidados ocasionalmente ou em séries consecutivas de dados em um ambiente propenso a erros. Várias conexões infravermelho utilizam este método.

2.3.2.4 Portas COM

Portas COM em PCs podem incluir portas e placas-mãe, cartões de expansão, conversores USB e servidores seriais. Outro nome para portas COM são porta de comunicação e *Comm Port*. Para cada porta COM, um *driver* do sistema operacional atribui um nome simbólico como COM1, COM2, e assim por diante, os quais são utilizados nas aplicações para detectar e acessar a porta. Versões recentes do Windows não limitam o número de portas COM. Claro que todo sistema possui uma quantidade limitada de recursos que limitará quantas portas COM podem ser usadas ao mesmo tempo.

2.3.2.4.1 Recursos da porta

Uma típica porta COM em uma placa mãe ou cartão de expansão contém uma UART que faz interface com o barramento do sistema, normalmente um barramento PCI. Cada UART utiliza uma série de oito endereços de porta. O primeiro endereço na série é o endereço base da porta. Muitas destas portas também apresentam uma linha IRQ associada para carregar as requisições de entrada.

Uma porta pode usar qualquer endereço e linha IRQ suportada pelo sistema.

A Tabela 2 mostra os endereços e linhas IRQs atribuídas a portas COM dos PCs no início.

Tabela 2 - Principais portas COM e seus respectivos endereços de memória (Axelson, 2007).

Porta	Endereço	IRQ
COM1	3F8h	4
COM2	2F8h	3
COM3	3E8h	4 or 11
COM4	2E8h	3 or 10

Portas antigas possuem frequentemente *jumpers*, chaves ou utilidades de configurações que possibilitam selecionar um endereço base e linha IRQ. Telas de configuração que permitem o acesso no *boot* podem permitir configurar portas na placa-mãe. Com alguns *hardware*s múltiplas portas podem partilhar uma mesma linha IRQ.

Uma porta de um conversor USB/serial não possui seu próprio endereço e linha IRQ. Em vez disso, a porta utiliza recursos compartilhados da USB.

2.3.2.4.2 Servidores de porta serial

Servidores seriais são dispositivos que possibilitam acessar portas seriais através da rede. O servidor serial contém um microcontrolador, um controlador *ethernet*, e um ou mais UARTs que são responsáveis pela interface com portas RS-232 ou RS-485. O servidor gerencia comunicações entre rede *ethernet* e portas seriais. Um servidor serial pode também fazer interface com uma rede sem fio (Wi-Fi). O mercado disponibiliza módulos de servidores seriais. Servidores de porta serial podem utilizar protocolos de internet definidos para comunicação de rede. A maioria dos servidores seriais se comunica via TCP. A especificação STD0007 define um padrão determinado de comunicação com um dispositivo e a troca dados com reconhecimento do número de seqüência, e outros recursos

que ajudam a garantir uma transferência confiável. Cada porta serial usa uma conexão TCP separada. O protocolo UDP (STD0006) é uma alternativa para aplicações que não necessitam da confiabilidade oferecida pelo protocolo TCP. Alguns servidores seriais usam conexões Telnet. A especificação Telnet (STD0008) define um protocolo para transmitir caracteres e controlar dados via conexão TCP.

Informações específicas para portas COM e *modems* podem utilizar protocolos definidos na opção de controle da porta de comunicação Telnet (RFC 2217). Este documento define comandos para configurar os parâmetros da porta COM e métodos de controle de fluxo, ler os sinais de *status* RS-232, escrever sinais de controle para RS-232, e ler informações de erro e outros dados de estado.

Software redirecionadores de porta serial, ou portas seriais virtuais, podem transformar os servidores de portas seriais em portas COM locais, dentro do sistema operacional. Aplicações de *software* podem acessar as portas remotas como se elas estivessem locais.

2.3.3 Aquisição de dados e controle (DAQ)

Aquisição de dados é o processo pelo qual um fenômeno físico do mundo real é transformado em um sinal elétrico que é medido e convertido para um formato digital para que possa ser processado e armazenado em um computador (John Park, 2003).

Na grande maioria das aplicações, o sistema de aquisição de dados (DAQ) é projetado não só para adquirir dados, mas também para agir sobre eles. Na definição de sistemas DAQ é, portanto, útil estender esta definição para incluir os aspectos de controle do sistema como um todo. O controle é o processo pelo qual sinais digitais de controle oriundos do *hardware* do sistema são convertidos para um formato de sinal utilizado por dispositivos de controle, tais como atuadores ou relês. Estes dispositivos, por sua vez, controlam um sistema ou processo. Onde um sistema for referido como sendo um sistema de aquisição de dados (DAQ), é possível que este apresente também funções de controle (Austerlitz, 2003).

2.3.3.1 Fundamentos da aquisição de dados

Um sistema de aquisição e controle de dados fornece qualidade e flexibilidade ao PC. Este sistema pode ser constituído de uma grande variedade de blocos distintos de *hardware* oriundos de equipamento de diferentes fabricantes. Uma das tarefas do integrador do sistema é juntar estes componentes individuais em um sistema funcional completo (Kirianaki, 2002).

Os elementos básicos do sistema de aquisição de dados são: sensores e transdutores; fios de campo; condicionamento de sinal; h*ardware* de aquisição de dados; PC (Sistema operacional) e *Software* de aquisição de dados, conforme diagrama apresentado na Figura 6.

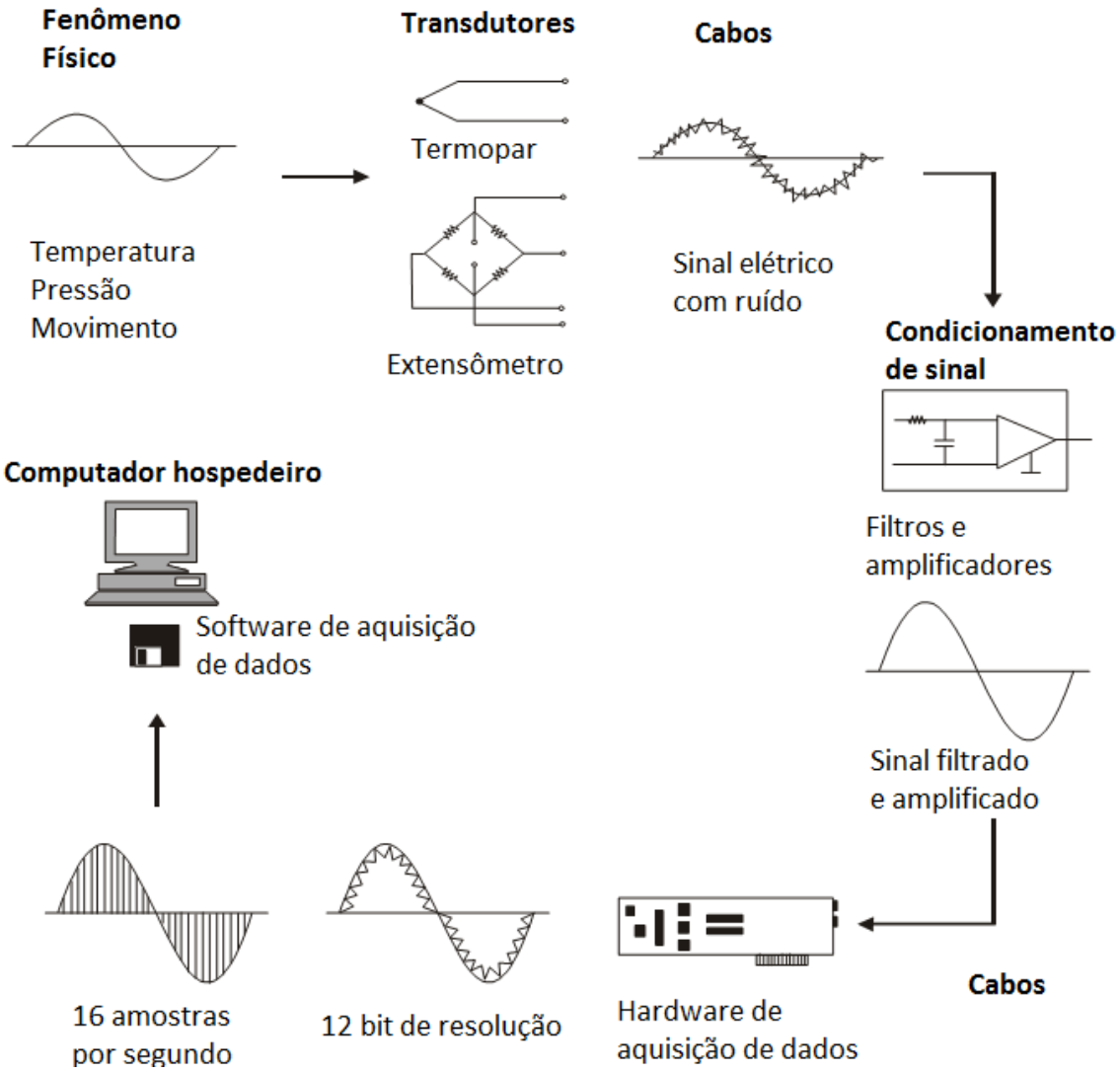


Figura 6 - Diagrama funcional de um sistema de aquisição de dados baseado em PC (Kirianaki, 2002).

Cada elemento do sistema de aquisição de dados em um PC é importante para ter uma precisão na medição e na coleta de dados a partir do processo ou fenômeno físico monitorado.

2.3.3.2 Transdutores e sensores

Transdutores de sensores apresentam sua própria interface entre o mundo real e o sistema de aquisição de dados, convertendo o fenômeno físico em sinais elétricos, os quais o *hardware* de condicionamento e/ou aquisição pode aceitar.

Transdutores permitem efetuar medição física e prover uma saída elétrica correspondente. Por exemplo, termopar, detector resistivo de temperatura (RTD), termistores e sensores IC que convertem temperatura em um sinal analógico, enquanto medidores de vazão produzem pulsos digitais cuja freqüência depende da intensidade da vazão.

Extensômetros e transdutores de pressão medem deslocamento e pressão, respectivamente, enquanto outros tipos de transdutores estão disponíveis para medir deslocamento linear e angular, velocidade e aceleração, luz, propriedades químicas (i.e. concentração de CO, pH), tensões, correntes, resistências ou pulsos, tais como sinais eletroencefalográficos (Ferreira, et al., 2006). Em cada caso, o sinal elétrico produzido é proporcional à quantidade física do experimento, sendo a medida de proporcionalidade estabelecida de acordo com alguma relação físico-elétrica entre o comportamento físico e os sinais elétricos.

2.3.3.3 Fios de campo e cabeamento de comunicação

Fios de campo representam a conexão física entre transdutores e sensores com o *hardware* de condicionamento e/ou aquisição de dados. Quando o fenômeno está localizado remotamente ao PC, então os fios de campo provêem o elo físico entre estes elementos de *hardware* e o computador hospedeiro do sistema DAQ. Se o elo físico for uma interface RS-232 ou RS-485, então este equipamento de fiação de campo é geralmente referenciado com cabo de comunicação.

Como os fios e cabos de conexão normalmente representam fisicamente o maior componente do sistema, estão normalmente susceptíveis a efeitos de ruídos externos, especialmente em ambientes industriais. O correto aterramento e proteção dos fios e cabos de conexão são importantes para reduzir os efeitos dos ruídos. Este componente passivo dos sistemas de aquisição e controle de dados é normalmente negligenciado, tornando-se um sistema impreciso e não confiável por causa das técnicas incorretas de fiação.

2.3.3.4 Condicionamento de sinal

Sinais elétricos gerados por transdutores normalmente precisam ser convertidos para uma forma aceitável para que o *hardware* de aquisição, particularmente o conversor analógico/digital, possa converter o sinal do dado recebido para o formato digital

estabelecido pelo operador. Além disso, a maioria dos transdutores exige alguma forma de excitação ou ponte para a realização correta e precisa da operação.

As principais tarefas realizadas pelo condicionamento de sinal são:

- Filtragem
- Amplificação
- Linearização
- Isolamento
- Excitação

Filtragem

Em ambientes ruidosos, pequenos sinais, da magnitude de mV, recebidos por sensores como termopar e extensômetros, dificilmente são captados sem o comprometimento dos dados. Muitas vezes o sinal do ruído é igual ou de maior magnitude do que o sinal enviado, neste caso, o ruído necessita ser filtrado (Ferreira, et al, 2006). Equipamentos condicionadores de sinal normalmente apresentam filtros passa-baixas projetados para eliminar ruídos de alta freqüência que podem conduzir a dados imprecisos.

Amplificação

Tendo filtrado o sinal, ele deve ser amplificado para aumentar a resolução. A máxima resolução é obtida amplificando-se o sinal de entrada, de forma que a variação máxima do sinal seja igual à faixa de entrada do conversor analógico/digital contido dentro do *hardware* de aquisição de dados. É recomendável colocar o amplificador o mais próximo possível fisicamente do evento, pois possibilita a redução dos efeitos do ruído nas linhas de sinais entre o transdutor e o *hardware* de aquisição de dados.

Linearização

Muitos transdutores, como termopares, mostram um relacionamento não linear com a quantidade física que estão medindo. Os métodos de linearização destes sinais de entrada variam de acordo com o equipamento de condicionamento de sinal. Por exemplo, no caso de termopares, alguns equipamentos possuem *hardware* de condicionamento de sinal adequado ao tipo do termopar, fornecendo uma amplificação e linearização do sinal ao mesmo tempo. Um método mais barato, fácil e flexível para condicionar um sinal é provido por produtos que realizam a linearização do sinal de entrada utilizando *software*.

Isolamento

Um equipamento de condicionamento de sinal pode também ser utilizado para isolar os sinais vindos dos transdutores para o computador onde podem ocorrer transientes de alta tensão dentro do sistema a ser monitorado, devido a uma descarga elétrica ou por falha elétrica. O isolamento protege contra danos, tanto aos equipamentos computadorizados quanto aos seus operadores.

Excitação

Produtos de condicionamento de sinal também fornecem sinais de excitação para alguns transdutores. Por exemplo: extensômetros, termistores e RTDs, os quais necessitam de sinais externos de excitação de tensão ou corrente.

3 METODOLOGIA

O trabalho foi subdividido em duas partes: 1) desenvolvimento de uma arquitetura de comunicação para viabilizar o controle de uma fonte de soldagem Fronius TransPuls Synergic utilizando o PDA Dell axim x51v, e 2) o desenvolvimento de *softwares* de controle.

3.1 Comunicação

Para estabelecer a comunicação entre o PDA e a fonte de soldagem foram desenvolvidas duas formas de se comunicar com a fonte de soldagem. A primeira consistiu em construir uma biblioteca (API) para controlar a fonte de soldagem utilizando uma interface serial RS232, por meio do mapeamento do protocolo LocalNet presente na fonte de soldagem. Na segunda forma de comunicação foi utilizada uma interface para robôs (ROB 5000), presente na fonte de soldagem, através de uma placa de aquisição de dados (National Instruments NI USB-6009).

3.1.1 Criação de uma API para o controle via Porta Serial

Para criação de uma API foi necessário realizar a engenharia reversa do protocolo de comunicação da fonte de soldagem Fronius TPS 5000.

3.1.1.1 Engenharia reversa do protocolo

Para que esta engenharia reversa fosse estabelecida foi necessário pesquisar o funcionamento do dispositivo através do entendimento da interação das partes (*Software*s de controles proprietários da Fronius e da fonte de soldagem) que compõem o sistema a ser analisado. No caso específico esta interação se caracterizou pelo tráfego de *bytes* dentro de uma comunicação serial. Neste trabalho as formas de observar os *bytes* transmitidos dentro de uma comunicação serial foram:

- O uso de *software*s espiões de porta serial;
- Bifurcação dos sinais para outras portas de comunicação;
- Utilização de dispositivo ou *software* intermediador;
- Osciloscópio

As duas primeiras formas apresentavam a mesma característica de bifurcar o sinal, entretanto, na primeira forma a bifurcação era realizada via um software, enquanto que na segunda era física, realizada por meio de um cabo serial especialmente projetado. Essa segunda forma foi a mais utilizada, pois possibilitava uma rápida visualização dos dados. Na terceira forma, foi empregado um *software* intermediador, cuja função era receber os dados transmitidos pelos dois dispositivos de comunicação (máquina de soldagem e o PC com software proprietário), além de armazenar e retransmitir os dados de um dispositivo para outro. O osciloscópio foi utilizado somente no começo do processo de monitoramento da comunicação para encontrar principalmente informações de baixo nível como baud-rate, bit de paridade, bits de parada.

Neste trabalho os processos de engenharia reversa do protocolo de comunicação escolhidos foram a segunda e a terceira formas de observação dos dados transmitidos, já que estes possibilitavam a criação da biblioteca (API) na medida em que os dados eram observados. Ou seja, o *software* desenvolvido para monitorar os *bytes* trafegados foi sendo modificado para decodificar os *bytes* na medida em que as partes do protocolo eram compreendidas.

3.1.1.2 Descobrindo o *baud rate*

O *baud rate* pode ser verificado de várias formas, como descrita abaixo, onde cada uma apresenta suas respectivas vantagens e desvantagens. Dentre elas as principais são:

1. Quando se tem uma conexão serial e não se sabe a tecnologia utilizada, como por exemplo, TTL, CMOS, ou outras tecnologias, recomenda-se utilizar um osciloscópio para medir a intensidade dos pulsos e a freqüência dos mesmos. Desta forma é possível identificar o tipo de tecnologia e o *baud rate* utlilizados.
2. Quando se conhece a tecnologia, mas os dispositivos seriais que se comunicam não são baseados em sistemas operacionais comercias como Microsoft Windows, Linux, e outros, pode ser criado um programa que possa ser conectado na porta serial utilizando todas as combinações possíveis de *baud rate*, bit de parada, bits de dados, etc. de forma que, ao conseguir receber *bytes* "randômicos" ou *bytes* diferentes de erro a configuração correta da porta serial seja encontrada.
3. Quando já se conhece a tecnologia embarcada da porta serial e este dispositivo se encontra dentro de um sistema operacional comercial, podem-se utilizar *software*s que espionam a porta serial para identificar o *baud rate* (Figura 7). Estes *software*s

além de mostrar as configurações utilizadas na conexão serial podem mostrar também os *bytes* trafegados na comunicação.

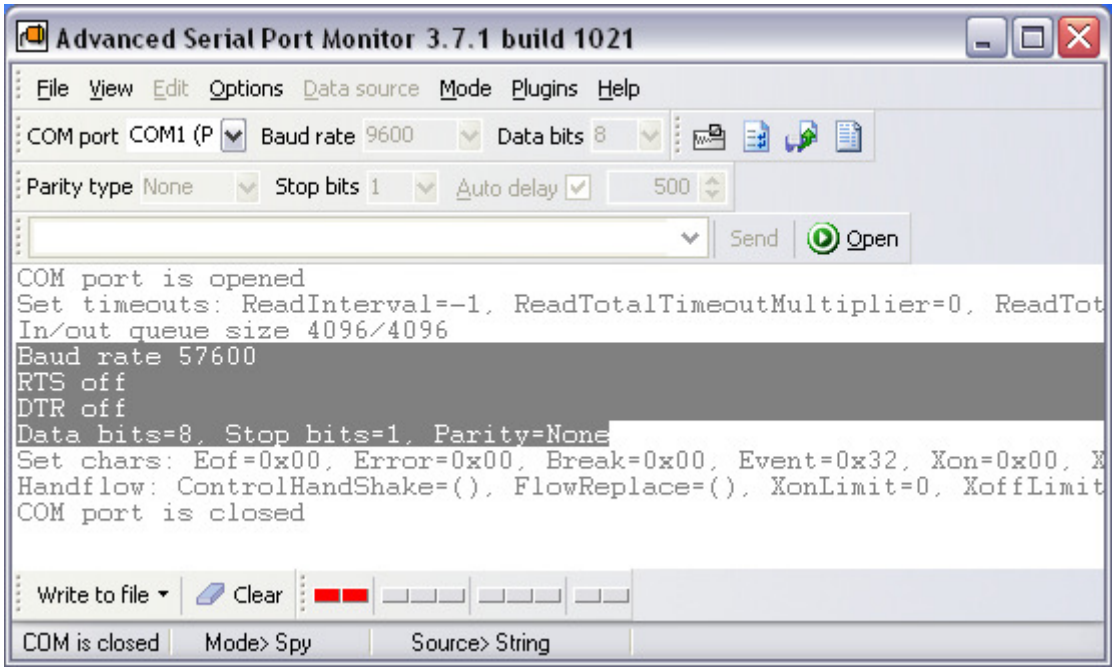


Figura 7 - Software espião de porta serial.

3.1.1.3 Descobrindo a porta COM

PCs podem ter múltiplas portas COM. Para decidir qual porta a ser usada, a aplicação fornece uma *combobox* que possibilita o usuário selecionar a COM. A aplicação pode salvar a última porta utilizada ou utilizar a porta padrão quando disponível. Ao utilizar conversores USB-serial, a interface física pode receber diferentes números de COM para cada vez que é conectada ou a cada mudança de porta USB.

3.1.1.4 Identificação da estrutura das mensagens

Para a identificação da estrutura das mensagens, foi montado um cabo serial que bifurcasse o sinal de transmissão (TX) de cada ponta da comunicação serial (Figura 8 e Figura 9). Em seguida foi utilizado um *software* (*DockLight*) que permitiu ler as duas portas seriais simultaneamente (Figura 10).

O último passo seguido dentro deste trabalho para a identificação da estrutura das mensagens foi o que demandou mais tempo, pois foi necessário procurar padrões dentro dos *bytes* trafegados, ou seja, toda comunicação serial foi realizada por meio de um protocolo de comunicação, e esse protocolo ou regras pode ser visualizado como estruturas padrões dentro da comunicação. A função do mapeamento do protocolo está em

correlacionar as ações ou reações apresentadas pelos dispositivos com os seus respectivos *bytes* transferidos.

Outra forma utilizada para definir a estrutura das mensagens foi por meio da busca por valores ou grandezas conhecidas ou esperadas dentro dos *bytes*.

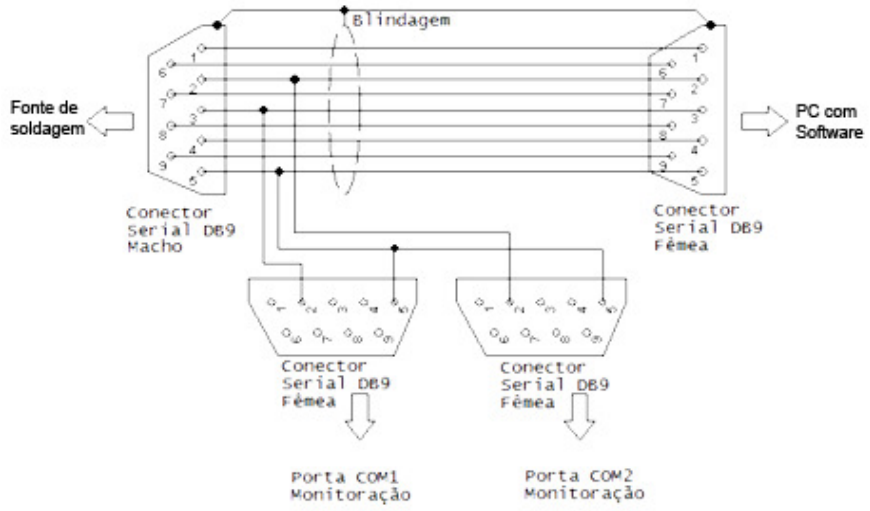


Figura 8 - Cabo serial com sinal do TX bifurcado.

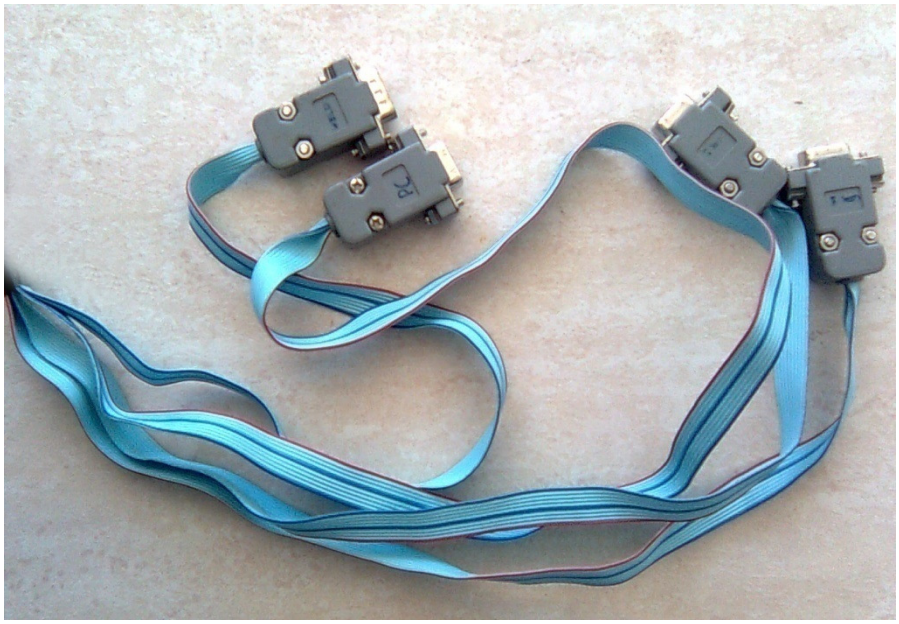


Figura 9 - Foto do cabo serial com sinal do TX bifurcado.

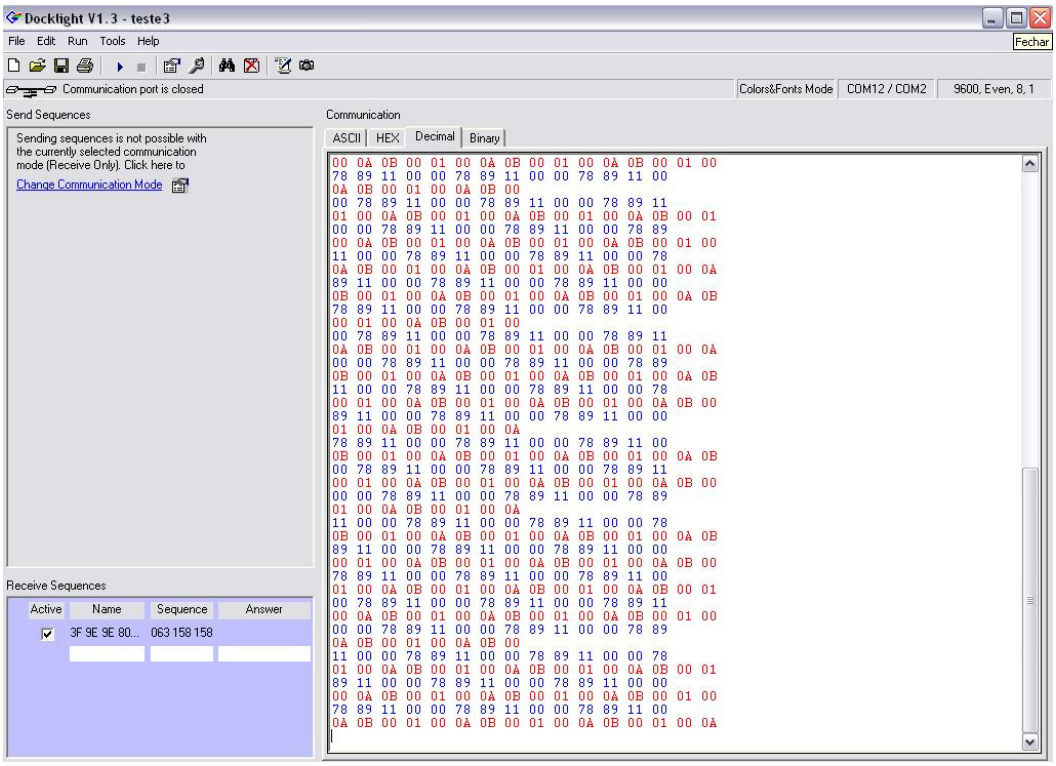


Figura 10 - Monitoramento dos *bytes* pelo programa *Docklight*

Durante o mapeamento do protocolo foi identificado que a fonte de soldagem que estava sendo mapeada não fornecia as funcionalidades desejadas (abertura/fechamento de arco, controle da corrente e aquisição de corrente, tensão e velocidade do arame) por meio da interface serial. Este fato inviabilizou a arquitetura originalmente projetada. Desta forma, para viabilizar o controle e a aquisição de dados da fonte de soldagem, uma nova arquitetura foi projetada, utilizando a interface robótica ROB 5000, também fornecida pela Fronius.

3.1.2 Controle da fonte Fronius TPS 5000 utilizando a Interface ROB 5000 e a placa de aquisição de dados NI USB-6009

O primeiro ponto observado ao utilizar a interface ROB 5000 foi que as tensões utilizadas tanto nos canais analógicos quanto nos digitais eram diferentes das tensões utilizadas na placa de aquisição de dados. Para conectar esses dispositivos com diferentes tensões, foi necessária a construção de uma placa de tratamento de sinais, que permitiu a interligação sem demais problemas de interoperabilidade.

3.1.2.1 Regulação da tensão

Para cada tipo de sinal (entrada analógica, saída analógica, entrada digital e saída digital) foi utilizado um tipo correspondente de correção da tensão, conforme Tabela 3:

Tabela 3 - Conversão de tensões para os tipos de sinais.

Sentido	Tipo	Tensão original (V)		Tensão corrigida (V)		Componente utilizado
		Mínima	Máxima	Mínima	Máxima	
Entrada	Analógica	0	5	0	10	Amplificador operacional
Saída	Analógica	0	10	0	5	Amplificador operacional
Entrada	Digital	0	5	0	24	Relê
Saída	Digital	0	24	0	5	Regulador de Tensão

3.1.2.2 Entrada analógica:

Como pode ser visto na Tabela 3 foi necessário projetar e construir um multiplicador analógico (BOURGERON, 1996) capaz de amplificar o sinal de entrada duas vezes. Isso foi feito com a configuração do amplificador operacional, apresentado no esquema mostrado na Figura 11 (Mancini, 2002):

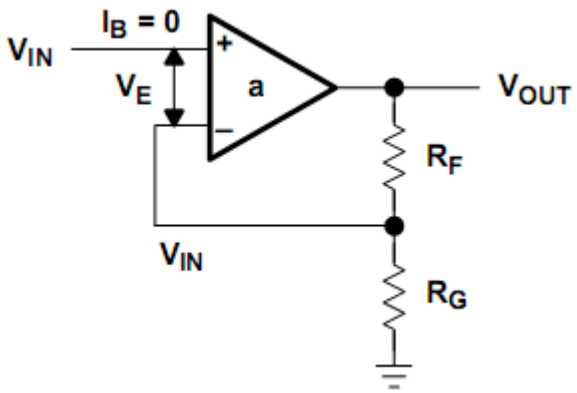


Figura 11 - Conversão da tensão para a entrada analógica.

Dado que:

$$V_{IN} = V_{OUT} \frac{R_G}{R_G + R_F}$$

Se for atribuído o valor 10kΩ para R_G e R_F, obtém-se:

$$V_{OUT} = V_{IN}\frac{(10+10)}{10} = 2V_{IN}$$

3.1.2.3 Saída analógica

Para que fosse possível realizar uma divisão por dois do sinal analógico proveniente da saída analógica foi necessário utilizar uma configuração diferente para o amplificador operacional. A divisão foi feita da seguinte forma:

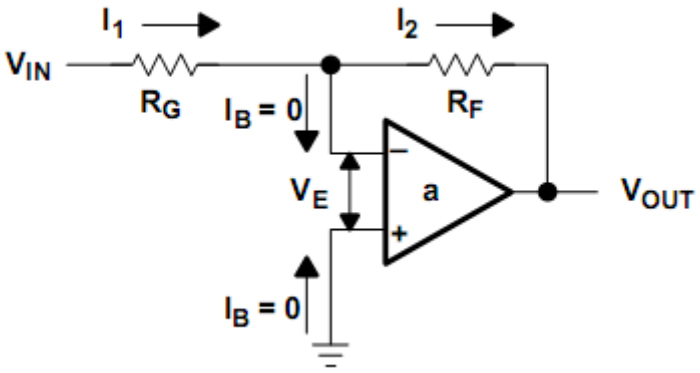


Figura 12 - Conversão da tensão para a saída analógica.

Dado que:

$$\frac{V_{OUT}}{V_{IN}} = -\frac{R_F}{R_G}$$

Se forem atribuídos os valores 5kΩ e 10 kΩ para R_F e R_G respectivamente tem-se:

$$V_{OUT} = -V_{IN}\frac{5}{10} = -\frac{1}{2}V_{IN}$$

Como o sinal invertido não era desejado, utilizou-se outro amplificador operacional para inverter novamente atribuindo o valor 10 kΩ para R_F e R_G.

$$V_{OUT} = -V_{IN}\frac{10}{10} = -V_{IN}$$

O que resultou na operação analógica

$$V_{OUT} = -\left(-\frac{1}{2}V_{IN}\right) = \frac{1}{2}V_{IN}$$

3.1.2.4 Entrada Digital

Para entrada digital foi utilizado o relê HHC66A-1C e o circuito integrado ULN2003 na configuração mostrada na Figura 13.

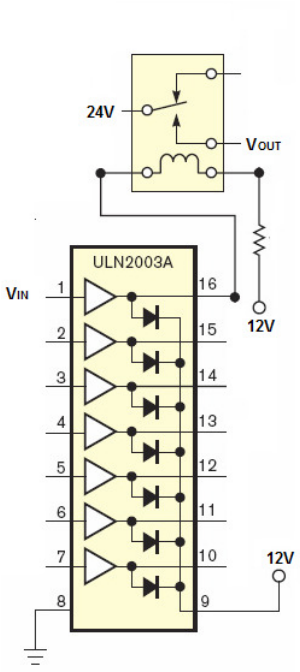


Figura 13 - Conversão da tensão para a entrada digital.

3.1.2.5 Saída Digital

Para obter o sinal de saída digital foi utilizado o regulador de tensão LM7805. Este componente regula qualquer tensão entre 6V e 35V para 5V.

3.1.2.6 Confecção da placa

Para obter a conversão de cada sinal, foi necessário projetar uma placa de circuito impresso por meio do *software* Cadsoft Eagle 5.1.0. Para a simplificação da placa, esta foi dividida em duas partes (digital e analógica), como pode ser visto na Figura 14 e na Figura 15, respectivamente.

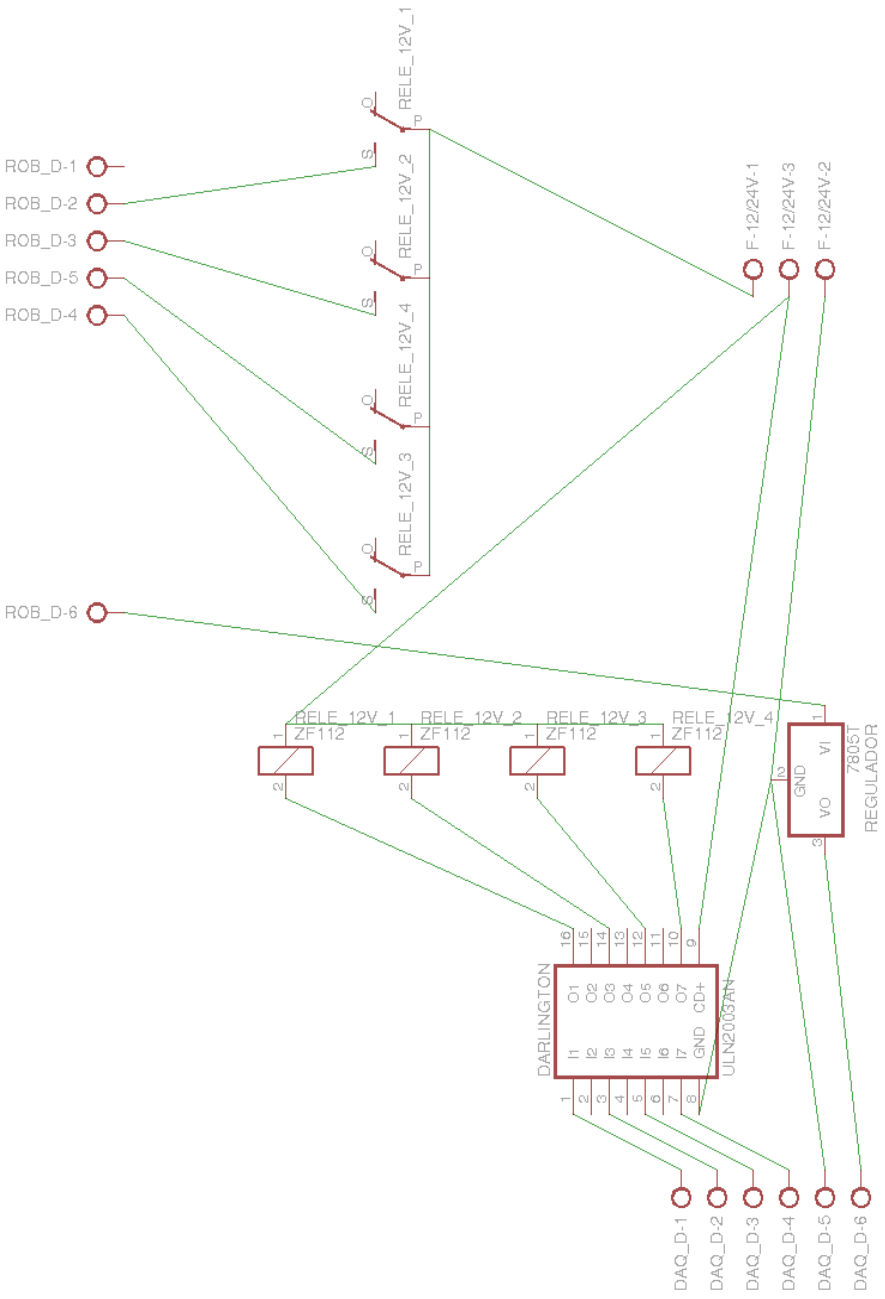


Figura 14 - Placa reguladora de tensão (parte digital).

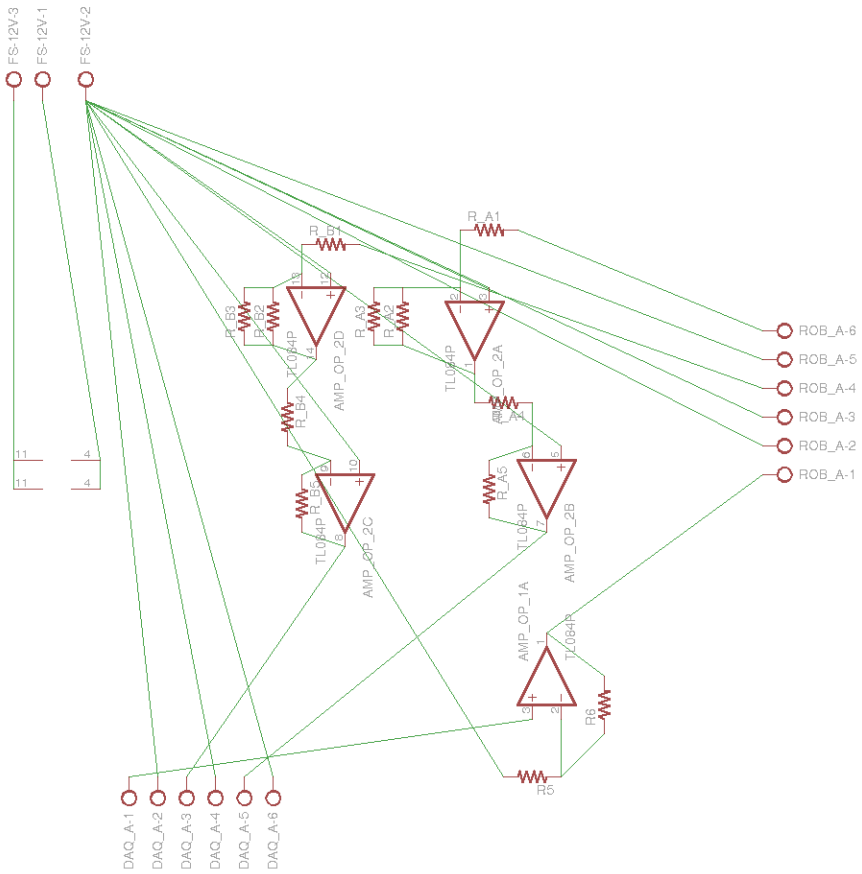
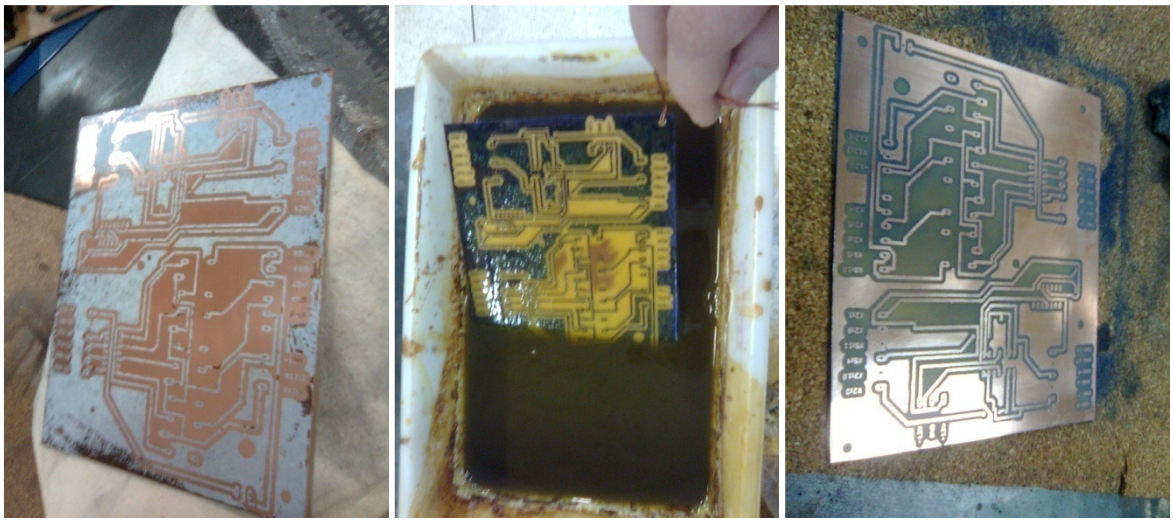


Figura 15- Placa reguladora de tensão (parte analógica).

Após o término da fase de diagramação da placa, foram posicionados os componentes nos locais desejados e em seguida roteadas as trilhas. Terminado este processo, foi obtida então uma imagem (APÊNDICE A – PLACA REGULADORA DE TENSÃO), a qual foi impressa em uma placa de cobre (Figura 16-a).

A confecção da placa foi realizada utilizando o processo de corrosão do cobre com uma solução de percloreto de ferro. A imagem das trilhas foi impressa utilizando uma impressora a laser sobre um papel Glossy A4 e transferida para placa através de um ferro de passar roupa. As imperfeições na transferência da tinta do papel para a placa foram corrigidas com caneta de transparência. Após 20 minutos de corrosão, conforme o esperado, foi possível visualizar que toda a parte da placa que não estava protegida pela imagem já estava corroída (Figura 16-b). Terminado o processo de corrosão, a placa foi enxaguada e lixada com palha de aço até que a imagem transferida fosse completamente removida da superfície (Figura 16-c).



(a) (b) (c)

Figura 16 - Corrosão da placa reguladora de sinais.

3.2 Software de controle

Com a construção da placa reguladora de sinais foi possível resolver o problema do meio físico de comunicação com a fonte de soldagem Fronius. Em seguida, foi desenvolvida a segunda etapa, ou seja, a de desenvolvimento do *software*. Devido ao fato do PDA não apresentar interface RS-232 ou *USB-host* necessária ao controle do dispositivo de aquisição de dados e da mesa posicionadora, foi necessário utilizar um servidor. Este funcionou como uma ponte de informações entre a mesa, a fonte de soldagem e o PDA.

3.2.1 Servidor

Foi desenvolvido o *software* do servidor, que funcionou em um PC convencional com porta serial e porta USB. Antes de iniciar o servidor, a porta TCP/IP e a porta COM, conectada à mesa posicionadora da tocha de soldagem, devem ser configuradas. O botão "Ativar", do *software* do servidor, disponibiliza o sistema para um cliente se conectar. Quando um cliente se conecta, este passa a ter o controle de funcionamento da fonte de soldagem e da mesa posicionadora. Todos os comandos enviados pelo cliente podem ser visualizados em um campo de texto na parte inferior do *software*, como pode ser visto na Figura 17.

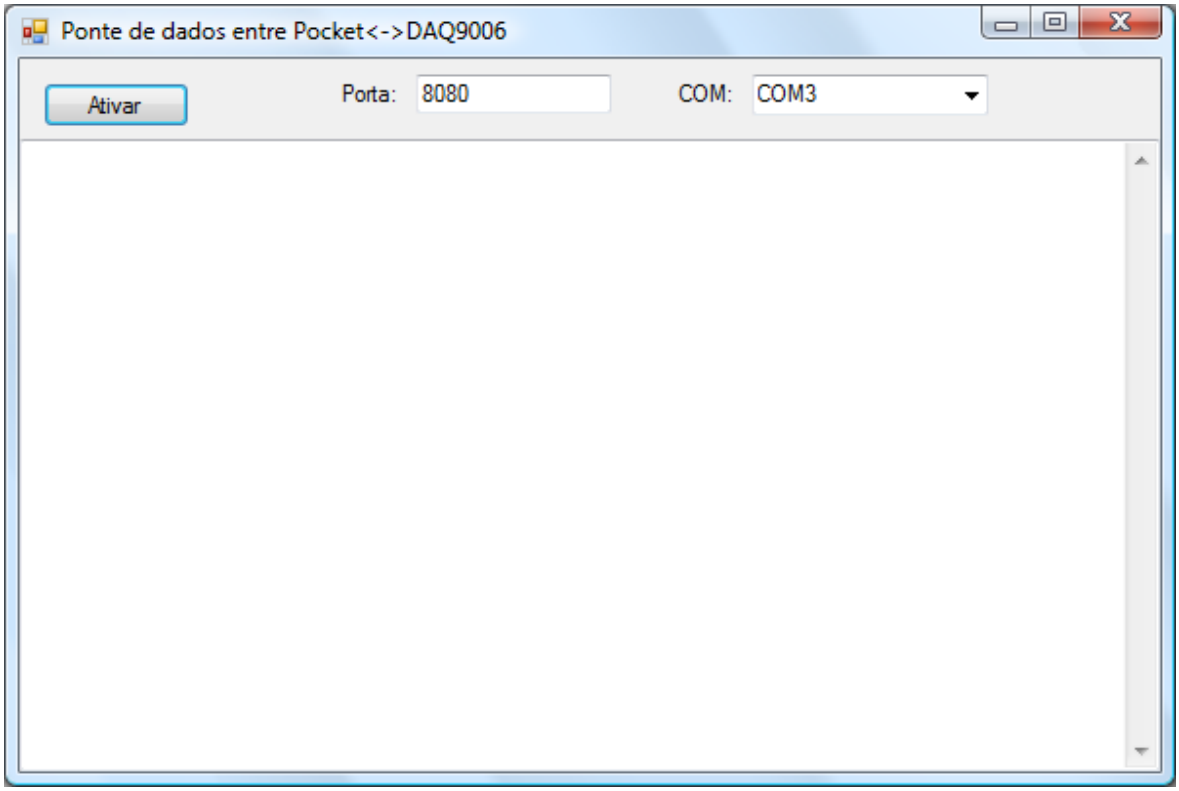


Figura 17 - Tela do software de controle do servidor.

3.2.2 Cliente

Foi criada uma tela principal para o *software* de controle, capaz de controlar a mesa de soldagem em conjunto com a fonte e também visualizar os parâmetros tensão, corrente e velocidade do arame vindos da fonte. A Figura 18 apresenta o leiaute da tela contendo 4 botões, 2 campos de texto, uma barra de rolagem e um gráfico.

O primeiro botão "Limpar Erros" solicita que a máquina limpe o registro de erros e conseqüentemente volte a funcionar. O botão rotulado com "Modo Normal" informa à fonte qual tipo de processo será utilizado (Normal ou Pulsado). O terceiro botão indica para a mesa qual o sentido do movimento em relação ao motor da mesma. Já o botão "Executar Tarefa" envia os comandos de movimentar a mesa e abrir/ fechar o arco, de forma que a tarefa de soldagem seja realizada no tempo e na velocidade correta.

O campo de texto "Velocidade do Processo:" informa qual a velocidade que a mesa deverá se deslocar. Já o campo "Duração do Processo:" informa o período durante o qual a mesa e o arco estarão ativos.

A barra de rolagem denominada como "Potência (%)" controla a potência de operação da fonte de soldagem.

O item de menu “configuração” abre a tela de configuração (Figura 19) responsável por gerenciar o IP e a porta na qual o servidor se encontra. O segundo item de menu, “Conectar”, conecta e desconecta ao servidor.

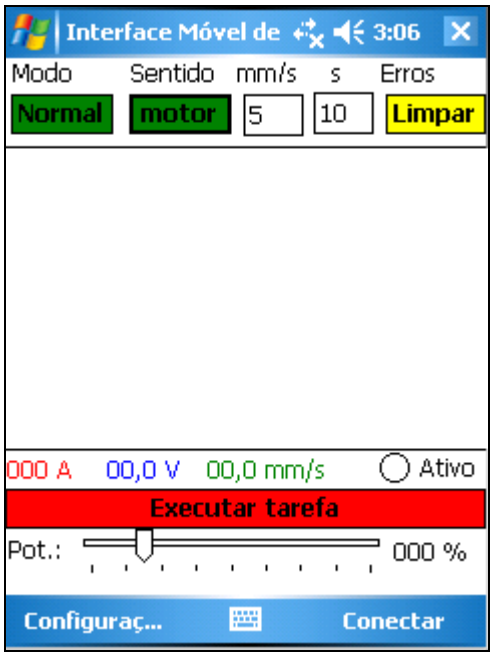


Figura 18 - Tela inicial do software de controle do Cliente.

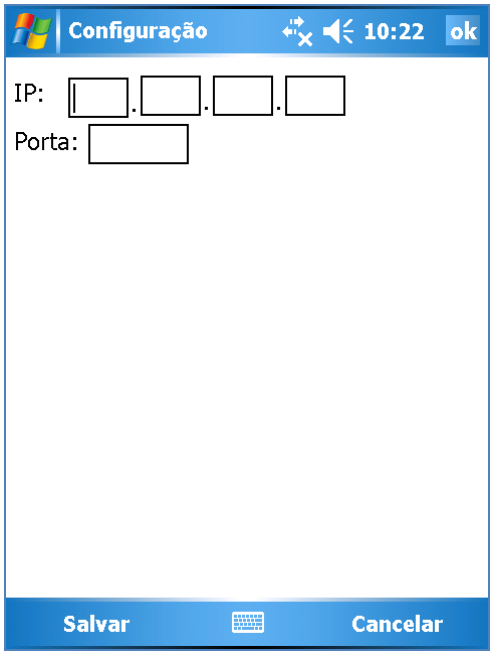


Figura 19 - Tela de configuração do software de controle do Cliente.

4 RESULTADOS E DISCUSSÃO

Em estudo anteriormente desenvolvidos no Laboratório do Grupo de Automação e Controle (Graco), da Universidade de Brasília foi utilizado a porta serial como forma de interface de comunicação entre o PDA e a fonte de soldagem Migatronic BDH 320 (Siqueira, 2006). No presente trabalho foi verificado que o protocolo de mapeamento da fonte de soldagem (Fronius TPS 5000) não fornecia suporte para a abertura e fechamento de arco, para o controle da potência e para a aquisição dos valores reais, o que inviabilizou a utilização da porta serial como forma de interface. Para solucionar este problema da interface física de comunicação com a fonte, foi necessário utilizar a interface ROB 5000 da Fonte de soldagem Fronius. Entretanto, a interface ROB 5000 apresentou incompatibilidade de tensão com o dispositivo de aquisição de dados, o que tornou necessário o desenvolvimento de uma placa reguladora de tensão capaz de transmitir informações por meio deste dispositivo de aquisição de dados, tendo sido utilizado o dispositivo NI USB-6009. Apesar do reduzido número de portas presente na placa NI USB-6009, por meio desta foi possível interfacear os principais parâmetros de soldagem, tais como potência, corrente, tensão e velocidade do arame.

4.1 Placa reguladora de tensão

Com a construção da placa reguladora de tensão (Figura 20) foi possível realizar testes de controle dos parâmetros de soldagem, constatando o adequado funcionamento das funcionalidades desenvolvidas (Abertura e fechamento de arco, controle de potência e aquisição de corrente, tensão e velocidade do arame).

A introdução desta placa reguladora de tensão entre o dispositivo de aquisição de dados e a interface robótica poderia interferir na exatidão dos valores convertidos analogicamente. Para verificar esta interferência, foi realizada a calibração dos valores analógicos dos parâmetros de entrada e de saída da interface robótica da fonte de soldagem.

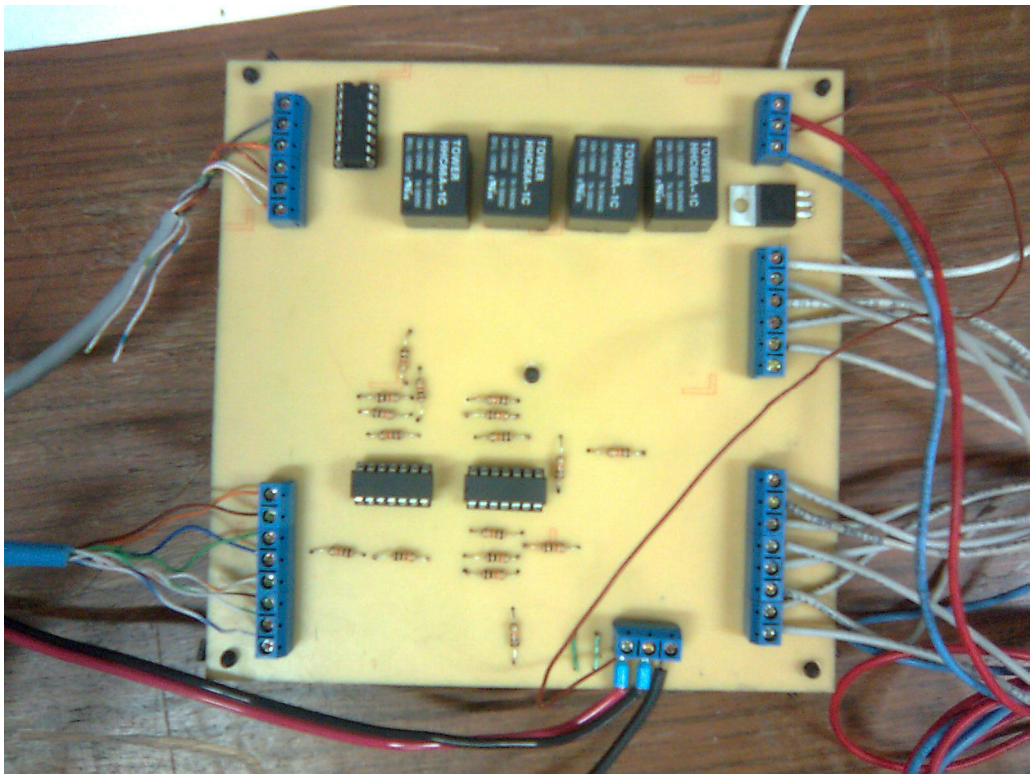


Figura 20 - Placa reguladora de tensão.

Essa calibração foi realizada por meio de *software*, na própria classe DAQ (APÊNDICE B), que faz o controle do dispositivo de aquisição de dados.

Para cada um dos paramentros analógicos: potência (Figura 21), corrente (Figura 22), tensão (Figura 23), e velocidade do arame (Figura 24) foi criada uma curva de calibração. Estas curvas foram construídas utilizando os respectivos valores dos parâmetros recebidos/enviados via software (dispositivo de aquisição de dados) e os valores mostrados no *display* da fonte de soldagem. Para cada um dos parâmentros de soldagem foi obtida uma equação para a linha de tendência dos pontos XY supracitados. Para os quatro parâmetros foram obtidas correlações altamente lineares, com coeficiente de correlação R^2 superiores a 0,9 e significativas (correlação de Pearson, P = 0,000). Estas curvas foram então utilizadas para calibração dos dados no servidor.

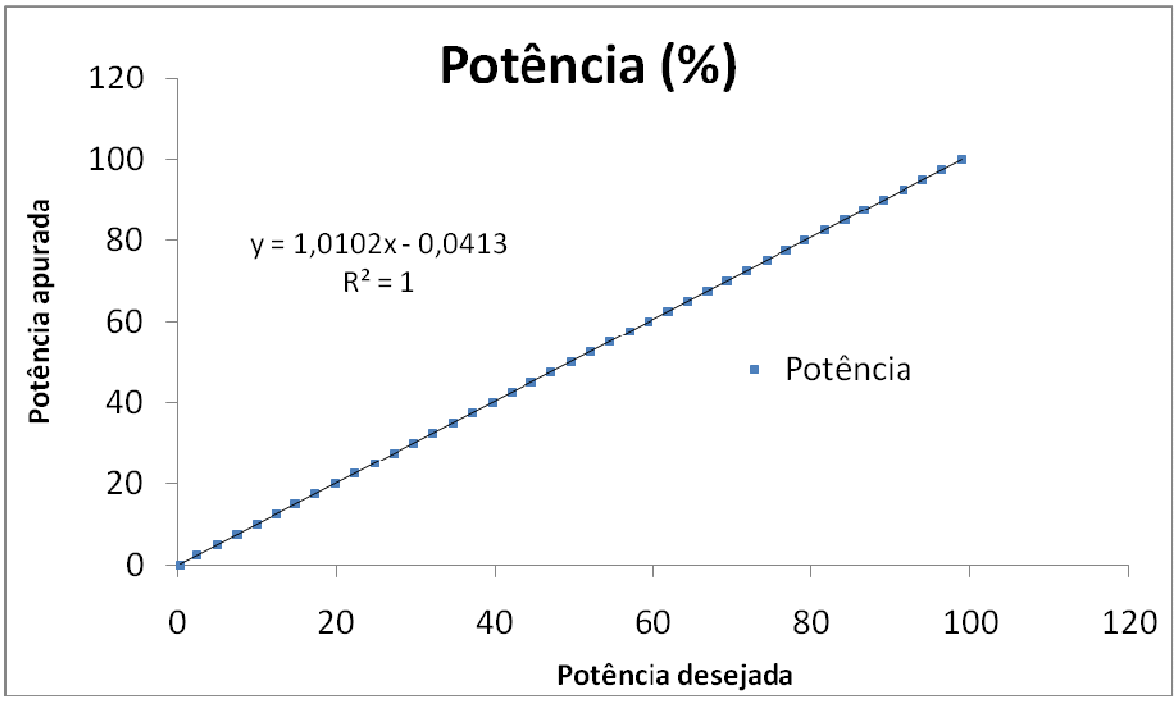


Figura 21 - Curva de calibração do parâmetro potência. Valores enviados ao dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 41).

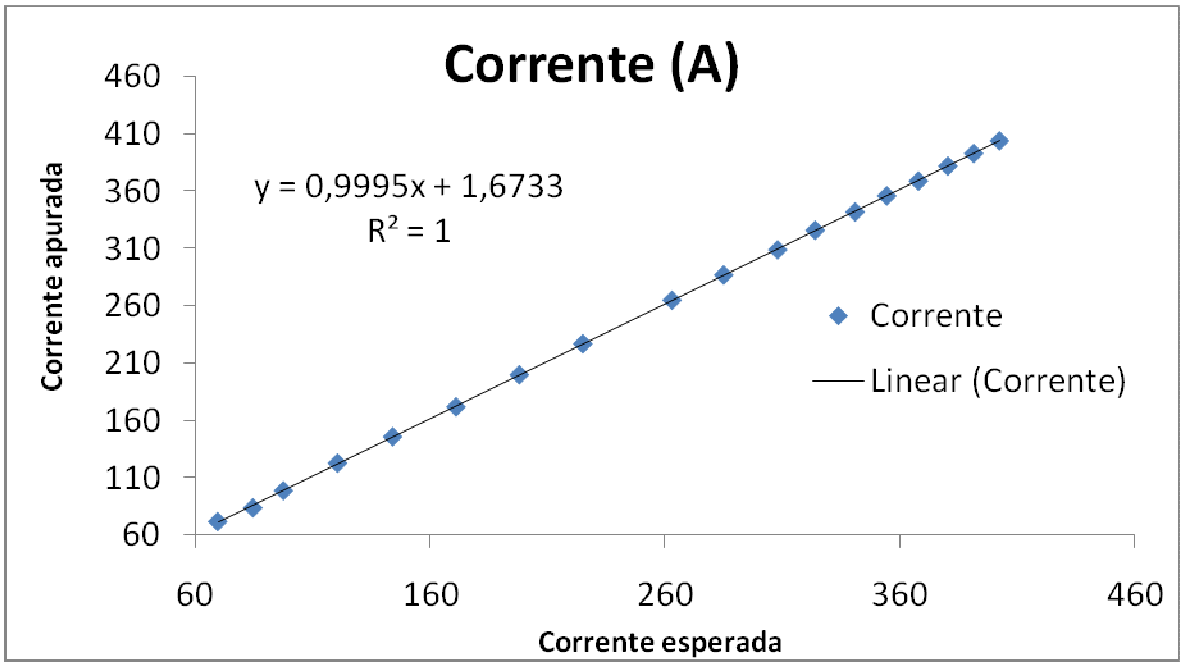


Figura 22 - Curva de calibração do parâmetro corrente. Valores recebidos do dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 18).

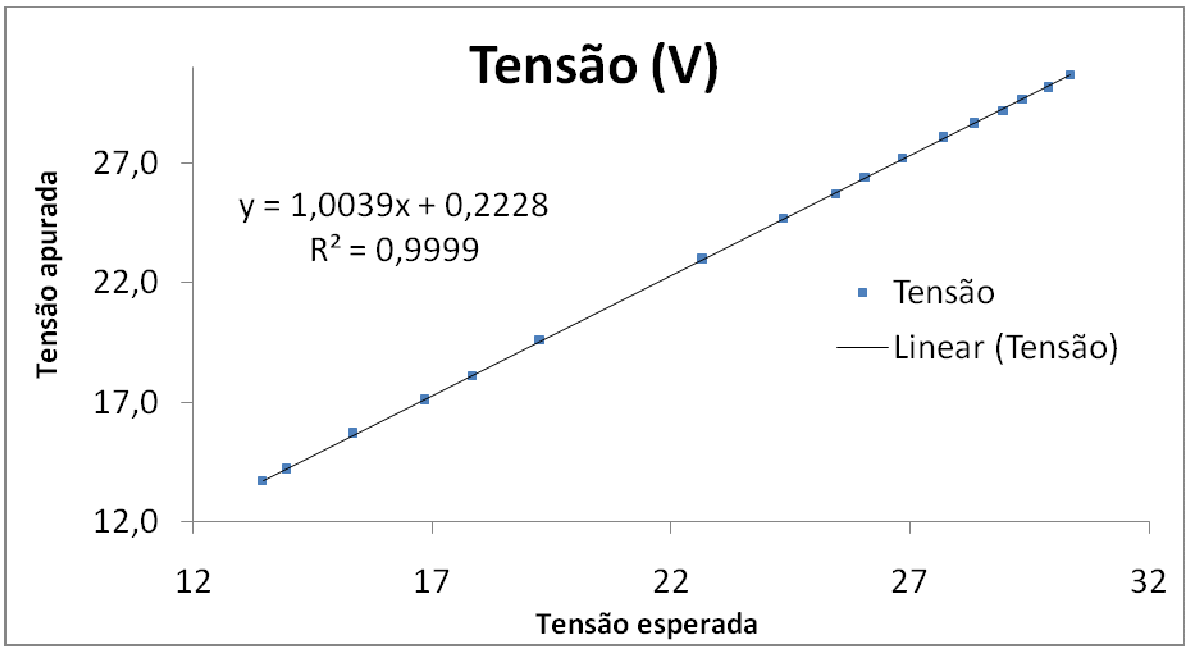


Figura 23 - Curva de calibração do parâmetro tensão. Valores recebidos do dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 17).

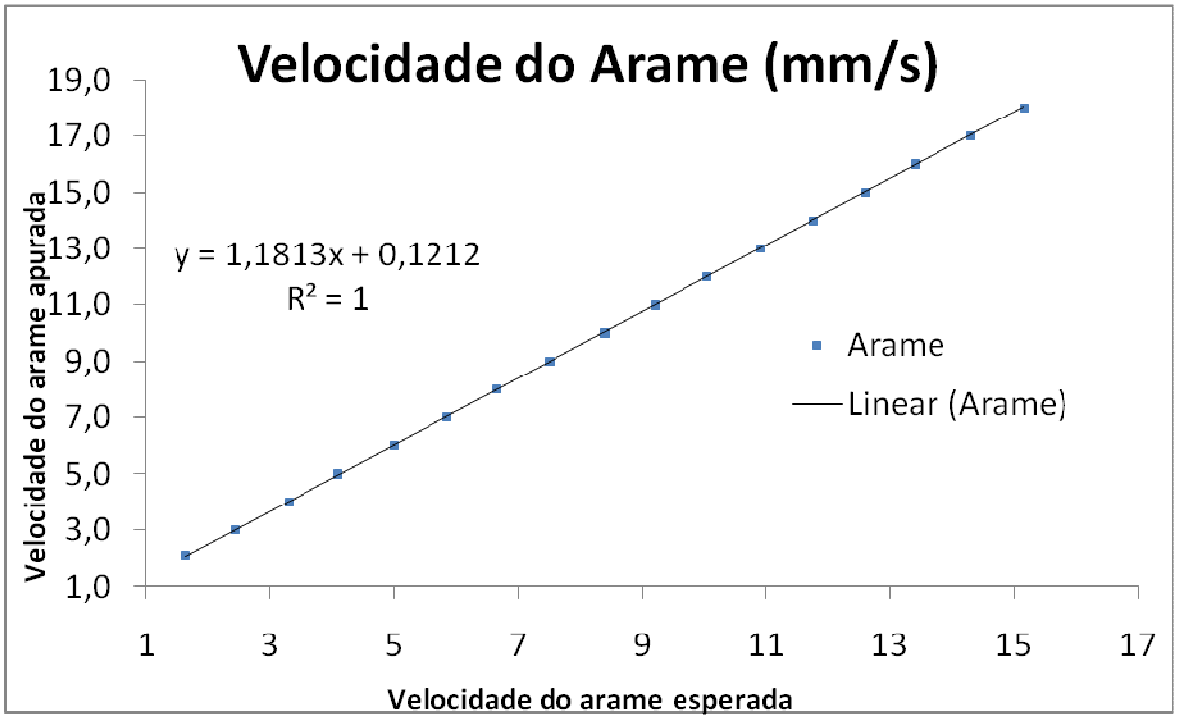


Figura 24 - Curva de calibração do parâmetro velocidade do arame. Valores recebidos do dispositivo de aquisição de dados versus valores mostrados no display da fonte de soldagem (n = 17).

4.2 Comunicação

Diferente do que ocorreu no projeto de desenvolvimento da interface para a fonte de soldagem MIGATRONIC (Siqueira, 2006), devido ao fato da interface de controle com a fonte de soldagem Fronius conter um conjunto de entradas e saídas analógicas e digitais, não foi possível utilizar um servidor de porta serial (Figura 25), tendo sido necessário, portanto, o desenvolvimento de outros equipamentos para intermediar a comunicação, como já foi mencionado anteriormente.

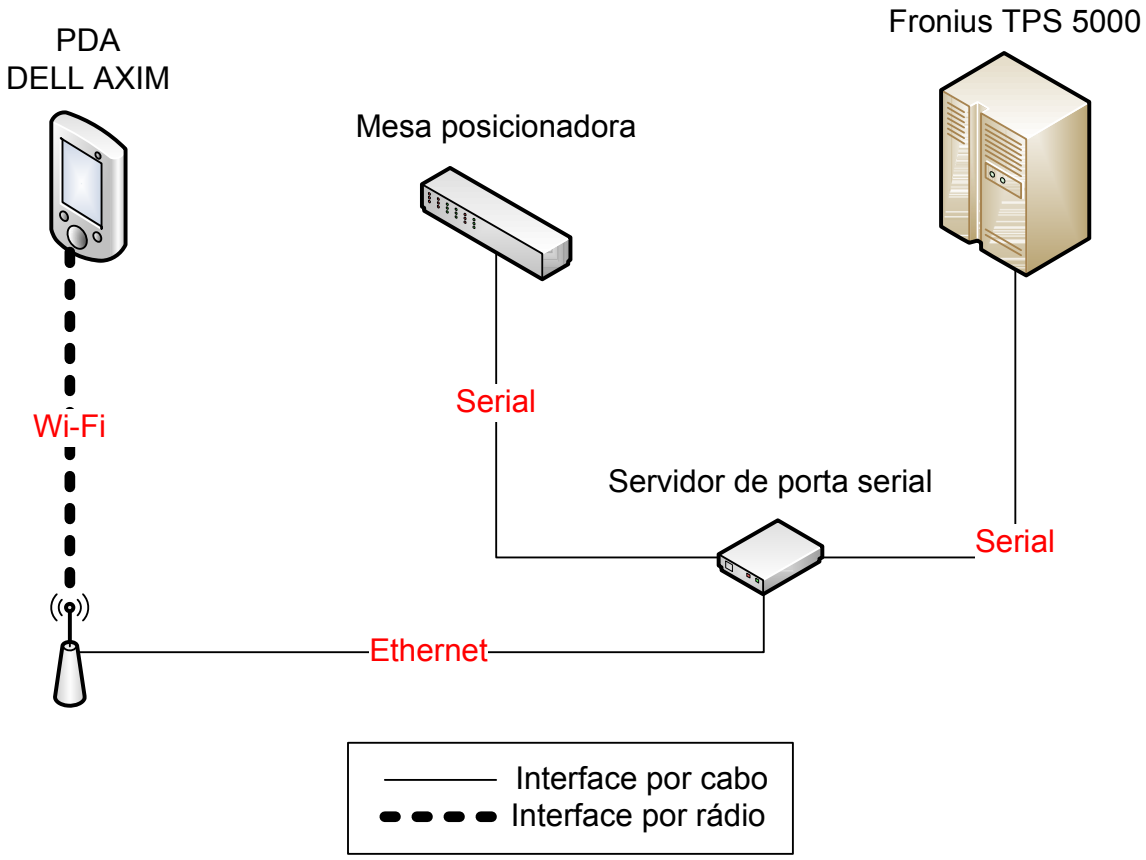


Figura 25 - Arquitetura da interface de controle móvel proposta no trabalho (Siqueira, 2006).

Visando a comunicação de dados entre a fonte de soldagem e o PDA, foi necessário centralizar a comunicação em um servidor. O servidor utilizou duas interfaces para se comunicar com a fonte de soldagem e a mesa posicionadora. Para o controle da mesa foi utilizado a porta serial RS-232 convencional. Para o controle da fonte de soldagem foi utilizado um dispositivo de aquisição de dados acoplado a uma placa reguladora de tensão que, por sua vez, estava conectada à interface robótica (ROB 5000) da fonte de soldagem.

O PDA fez parte da arquitetura como um cliente que acessa o servidor para controlar os dispositivos da célula de soldagem, utilizando alguns comandos predeterminados. O diagrama da arquitetura para a comunicação de dados proposto por este trabalho é mostrado na Figura 26.

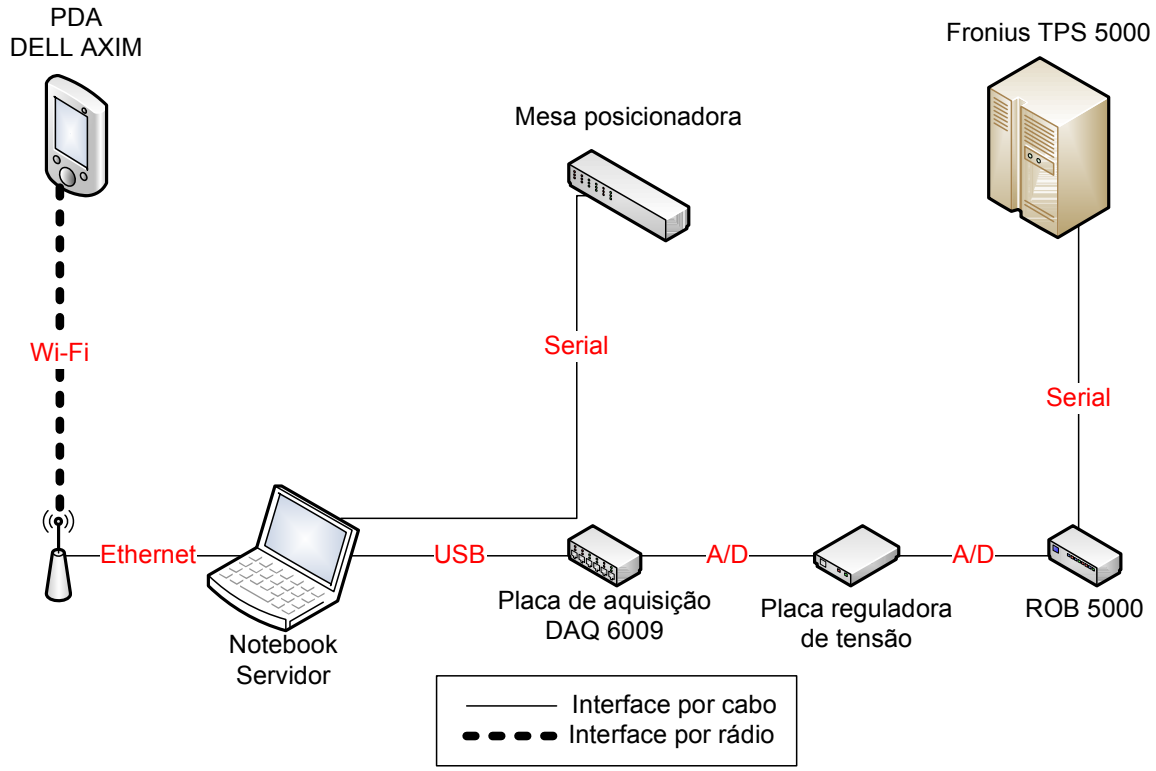


Figura 26 - Arquitetura da interface de controle móvel proposta para a célula de soldagem.

4.2.1 Controle da mesa posicionadora

Devido às limitações da controladora da mesa posicionadora (Figura 27), foi possível desenvolver apenas a movimentação da mesa no que se refere à velocidade, duração e deslocamento do movimento. Outras funcionalidades tais como parada de emergência, controle de velocidade em tempo de execução e indicação da posição absoluta não foram contempladas neste trabalho.

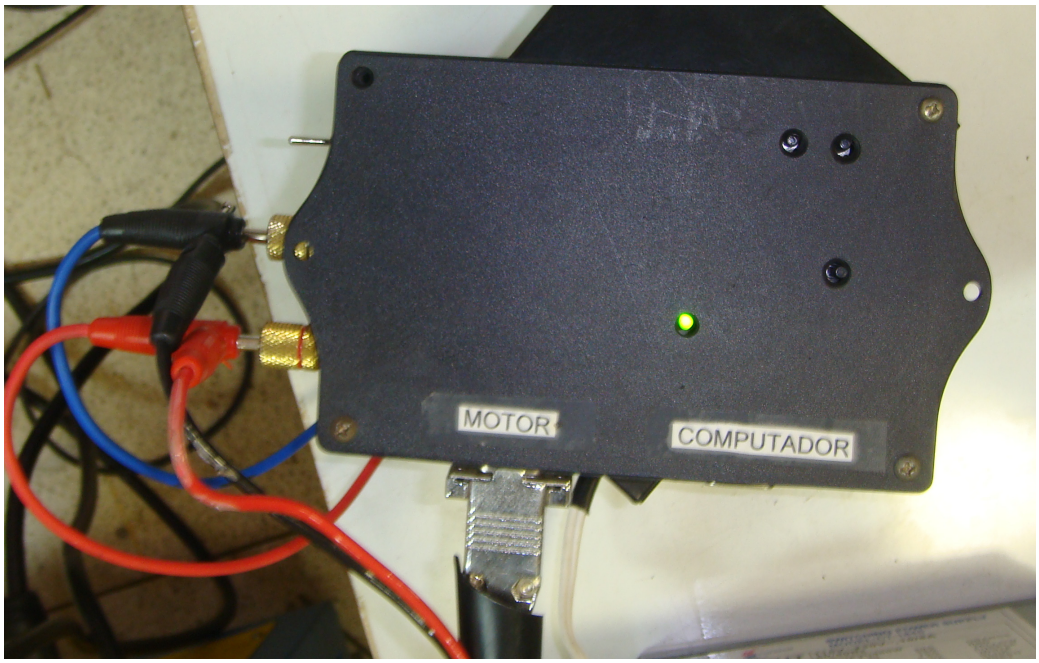


Figura 27 - Controladora da mesa posicionadora.

4.2.2 Interferência eletromagnética do arco na solda

Semelhante ao observado no projeto de automação da fonte de soldagem MIGATRONIC, no qual foi utilizado o processo TIG (Siqueira, 2006), no presente estudo não foi detectada na comunicação Wi-Fi interferência eletromagnética do arco formado pelo processo MIG/MAG. A comunicação manteve-se estável durante todo o processo, uma vez que não foi detectada qualquer queda na comunicação ou atraso no envio/recebimento dos dados. Todos os testes foram realizados a uma distância superior a meio metro, distância considerada livre de interferências conforme MISTODIE & RUSU (2007).

4.3 Softwares desenvolvidos

Visando a adequada comunicação de dados por meio da arquitetura proposta, foram desenvolvido dois *software*s. O primeiro software teve como objetivo funcionar como um servidor de *Socket* TCP/IP (compilado e executado em um PC convencional), que ficava esperando um cliente (o segundo *software* localizado dentro de um Pocket PC com Windows Mobile 5.0) conectar e enviar os comandos de controle. Tanto o *software* servidor quanto o cliente foram programados utilizando a linguagem C# e o compilador Visual Studio 2008 *Service Pack* 1.

4.3.1 Servidor

Com a função de centralizar todos os dispositivos a serem comunicados, o *software* servidor reuniu as funcionalidades de controle do dispositivo de aquisição de dados (classe DAQ), o controle de funcionamento da mesa (classe Protocolo) e a comunicação com o cliente TCP/IP (classe "WeldingSocket"), conforme diagrama de classe apresentado na Figura 31. A partir do diagrama da Figura 28 é possível visualizar a funcionalidade de cada classe dentro deste sistema de comunicação.

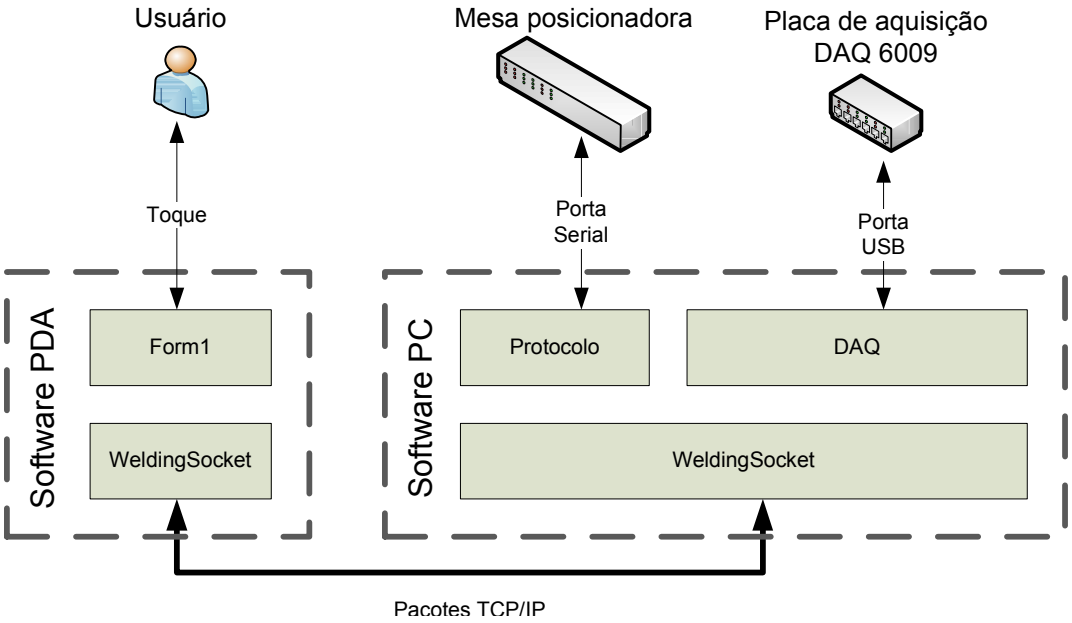


Figura 28 - Principais classes e suas distribuições dentro da comunicação.

4.3.2 Cliente

O *software* cliente, além de possuir a mesma classe ("WeldingSocket") de comunicação com o servidor (Figura 32), contém também uma classe de persistência para o salvamento dos dados recebidos do servidor durante o processo e ainda uma classe adicional para gerar o gráfico dos parâmetros de soldagem (corrente, tensão e velocidade do arame) em uma janela de tempo de dois segundos de duração. Os detalhamentos destas classes podem ser vistos no diagrama da Figura 32.

4.3.3 Emcapsulamento dos comandos e dados dentro da comunicação

Os comandos gerados pelo cliente, localizado no PDA, são encapsulados pelo objeto "strComandos" e, em seguida, serializada pela classe estática "Serializador", permitindo o envio dos comandos pela conexão *Socket* (gerenciada pelo objeto "WeldingSocket"). O

servidor, após receber o objeto serializado do cliente, deserializa-o para o objeto do tipo "strComandos", utilizando a mesma classe estática "Serializador". Com o objeto "strComandos", deserializado, o servidor envia os comandos correspondentes para a porta serial e/ou para o dispositivo de aquisição de dados por meio das classes Protocolo e DAQ, respectivamente, tal como mostrado no diagrama da Figura 29.

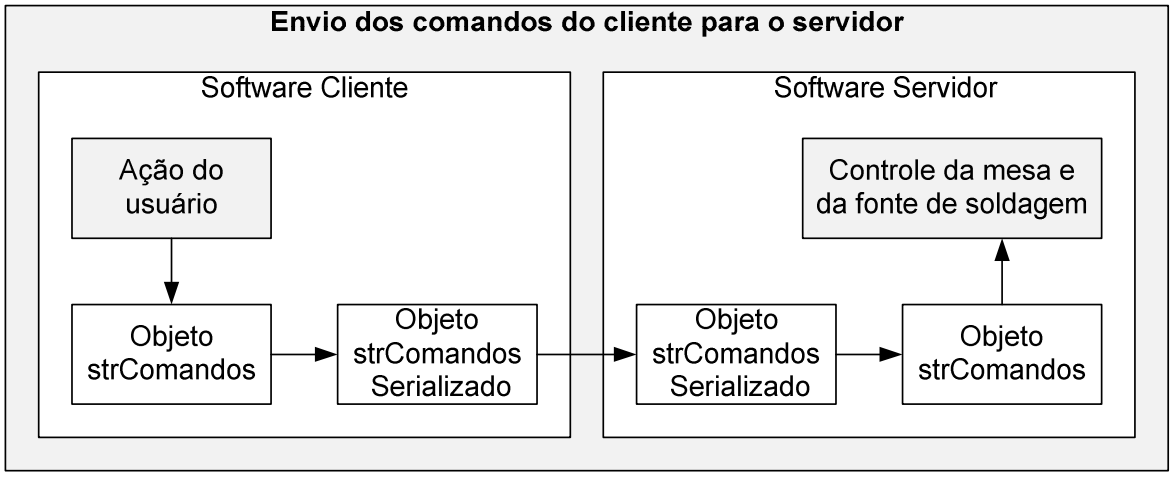


Figura 29 - Encapsulamento e serialização dos comandos de controle.

Os parâmetros de soldagens lidos por meio do dispositivo de aquisição de dados são retornados pela classe DAQ, encapsulados pelo objeto "strDados". Este objeto é serializado pela classe "Serializador" e enviado ao cliente pela classe "WeldingSocket". O *software* cliente do PDA, ao receber os dados pela classe "WeldingSocket", deserializa o objeto "strDados" com a classe "Serializador" e atualiza os controles de interface (campos de texto, gráfico e botões). Como pode ser percebido pelo diagrama da Figura 30.

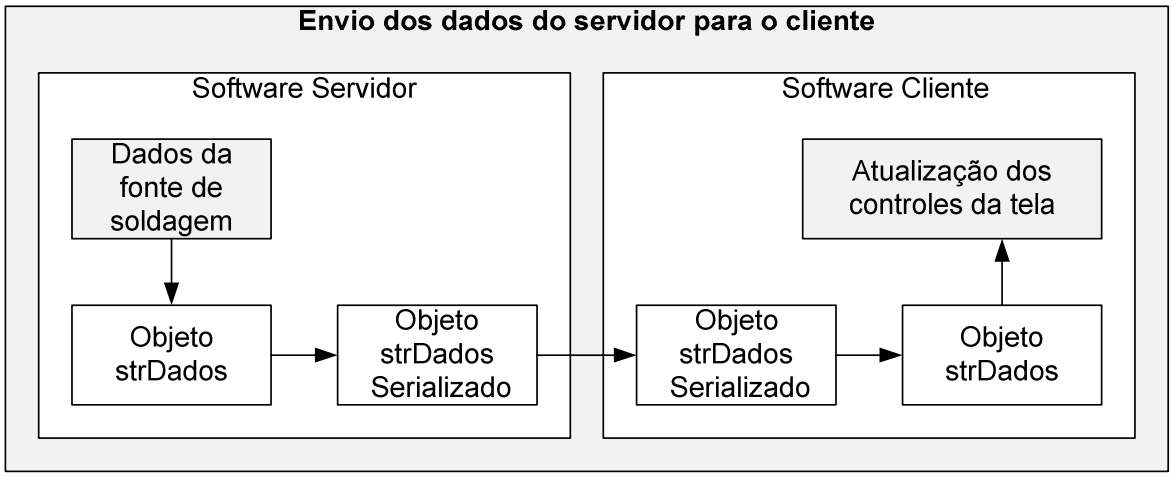


Figura 30 - Encapsulamento e serialização dos dados da fonte de soldagem.

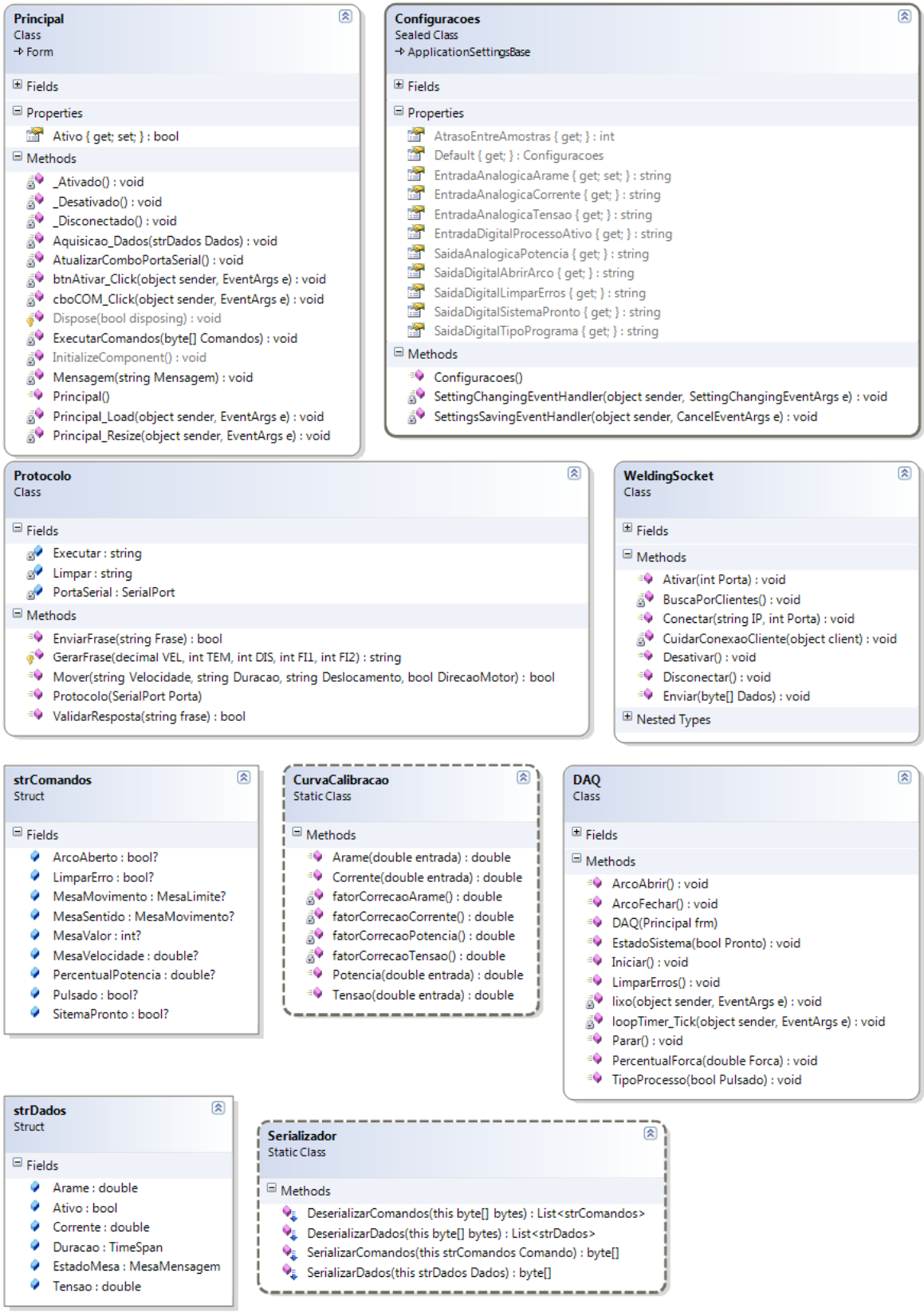


Figura 31 - Diagrama de Classe do software servidor

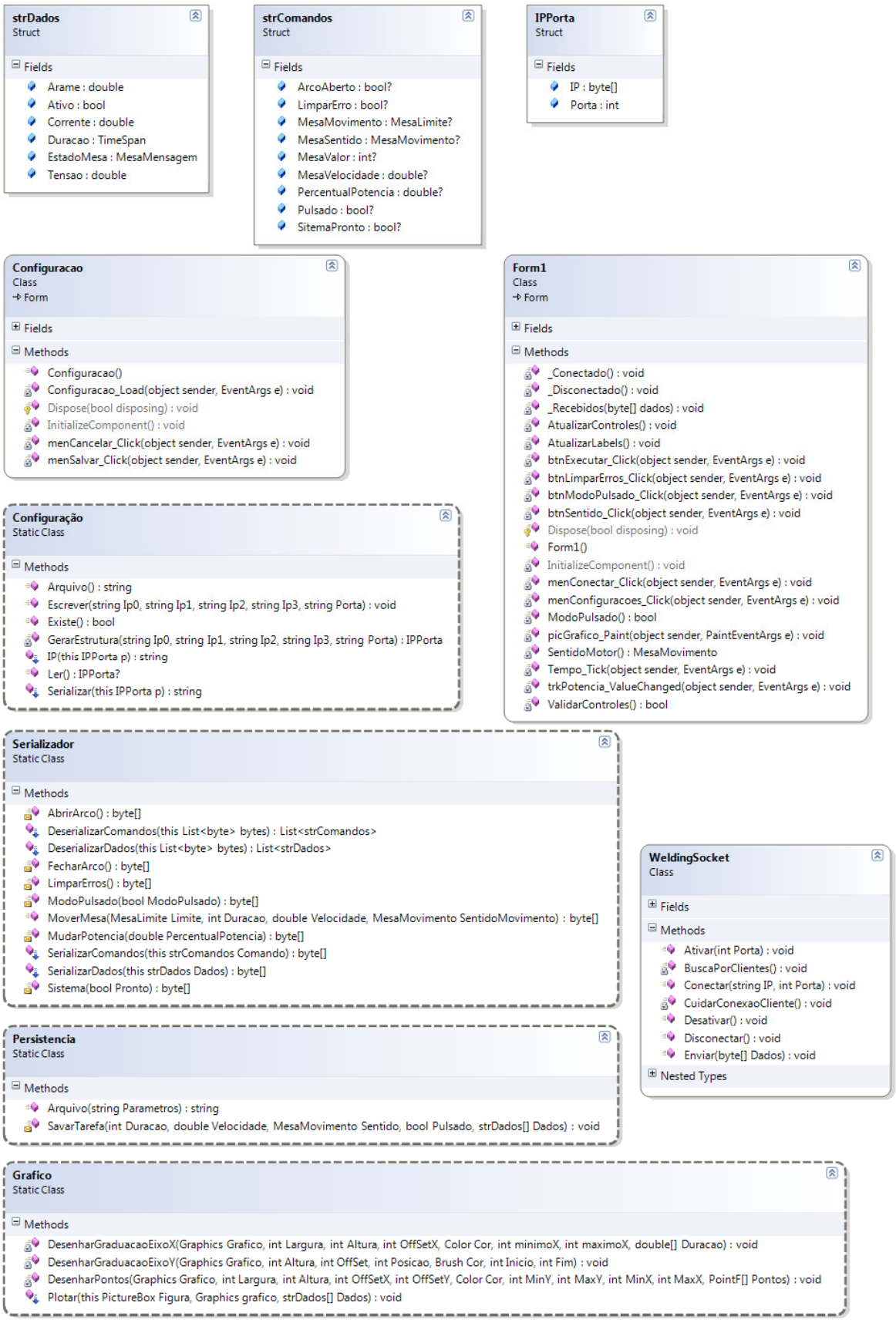


Figura 32 - Diagrama de Classe do *software* cliente

5 CONCLUSÃO

Esta dissertação de mestrado teve como objetivo geral desenvolver uma interface de controle móvel para utilização em uma célula de soldagem. Após o desenvolvimento do trabalho foi possível levantar algumas conclusões importantes:

- Devido às limitações presentes na interface serial da fonte de soldagem Fronius TPS 5000, não foi possível a comunicação com o PDA, por meio de porta serial, posto que os protocolos de comunicação presentes nos *software*s proprietários da fonte de soldagem não contemplam este nível de controle.
- Para viabilizar a interface de controle móvel, foi desenvolvido um servidor que associado a uma placa de aquisição de dados, permitiu o encapsulamento das funcionalidades básicas da fonte Fronius, proporcionando seu comando por meio de um dispositivo móvel.
- A velocidade na qual o software desempenhou as funções durante a realização dos do *software* cliente sugere que o PDA possa embarcar um número maior de componentes na interface gráfica, como por exemplo, o *streamming* de vídeo ou ambientes 3D para melhorar o nível da telepresença da arquitetura de controle.
- Efeitos da interferência eletromagnética (do arco elétrico) na comunicação WiFi não foram percebidos, em testes de comunicação realizados com a interface implementada.
- A partir da arquitetura desenvolvida foi possível controlar a velocidade e o deslocamento da mesa posicionadora, assim como controlar a potência de operação da fonte de soldagem, bem como visualizar os valores reais de corrente, tensão e velocidade do arame da fonte de soldagem.

5.1 Projetos futuros

Com a conclusão deste trabalho foi possível observar que algumas melhorias podem ser realizadas, tais como:

- Substituir o conjunto computador-placa de aquisição-placa reguladora por um sistema embarcado.
- Mapear o protocolo de comunicação da interface ROB 5000 com a fonte de soldagem Fronius TPS 5000.
- Adicionar uma câmera para visualização do processo remotamente.
- Substituir a mesa posicionadora por um manipulador.

6 REFERÊNCIAS BIBLIOGRÁFICAS

Amos, D. (1991). *Shielded Metal Arc Welding* (8 ed.). (O`Brien, Ed.) Miami, FL: American Welding Society.

Austerlitz, H. (2003). *Data acquisition techniques using PCs.* Academic Press.

Axelson, J. (2007). *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems.* Lakeview Research.

Bolmsjo, G. (1997). *Sensor System in Arc Welding.* Lud Institute of Technoogy, Production and Materials Engineering Department.

BOURGERON, R. (1996). *1300 Esquemas E Circuitos Eletronicos.* Hemus.

Cary, H. B. (1989). *Modern Welding Technology* (2 ed.). Englewood Cliffs, NJ: Printice Hall.

Christa Sommerer, L. C. (2008). *The Art and Science of Interface and Interaction Design.* Springer.

Connor, L. P. (1987). *Welding Handbook.* Miami, FL: American Welding Society.

Eagar, T. W. (1992). *Resistance welding: a fast, inexpensive and deceptively simple process.* Gatlinburg, TN: Conference on Trends in Welding Research.

Ferreira, A., Bastos, T. F., Sarcinelli, M., Cheein, F., Postigo, J., & Carelli, R. (2006, July 9-12). Teleoperation of an Industrial Manipulator Through a TCP/IP Channel Using EEG signals. *Industrial Electronics, 2006 IEEE International Symposium* , pp. 3066 -3071.

Ferreira, A., Celeste, W. C., Bastos, T. F., & Sarcinelli, M. (2007). Development of Interfaces for Impaired People Based on EMG and EEG.

Gohring, N. (2006). *Mobile phone sales topped 800 million in 2005, says IDC*. Fonte: InfoWorld: http://www.infoworld.com/article/06/01/27/74861_HNmobilephonesales_1.html

Holmes, J. G. (1979). *A flexible robot arc welding system.* Proceedings of the American Society of Mechanical Engineers.

Interface (ciência da computação). (s.d.). Fonte: Wikipédia: http://pt.wikipedia.org/wiki/Interface_(ciência_da_computação)

John Park, S. M. (2003). *Practical data acquisition for instrumentation and control systems.* Newnes.

K. Andersen, G. E. (1989). *A class-H amplifier power source used as a high-performance welding research tool.* Gatlinburg, TN: Proceedings of the 2nd International Conference on Trends in Welding Research.

K.L. Moore D.S. Naidu, S. O. (2003). *Modeling, Sensing and Control of Gas Metal Arc Welding*. Elsevier.

Kirianaki, N. V. (2002). *Data acquisition and signal processing for smart sensors.* John Wiley and Sons.

Kortum, P. (2008). *HCI beyond the GUI: design for haptic, speech, olfactory and other nontraditional interfaces.* Elsevier/Morgan Kaufmann.

Kou, S. (2003). *Welding metallurgy* (2 ed.). John Wiley and Sons.

Kusiak, A. (2000). *Computational Intelligence.* John Wiley & Sons.

Kusiak, A. (1986). *Modeling and Design of Flexible Manufacturing System.* Elsevier Science Publishers.

Loureiro, A. V. (1998). *The influence od Heat Input and the Torch Weaving Moviment on Robotized MIG Weld Shape.* Internation Journal for the Joining of Material.

Mancini, R. (2002). Op Amps for everyone. Texas Instruments Inc.

Manuel Ferre, M. B. (2007). *Advances in telerobotics.* Berlin: Springer.

Mistodie, L. R., & RUSU, C. C. (2007). Arc Welding Workspace Risk Factors Evaluation and Monitoring. *TECHNOLOGIES IN MECHANICAL ENGINEERING* , 85-92.

Norrish, J. (1992). *Advanced welding processes.* Bristol, UK: Intitute of Physics Publishing.

Nystedt, D. (2006). *Mobile subscribers to reach 2.6B this year*. Fonte: InfoWorld: http://www.infoworld.com/article/06/11/10/HNmobilesubscribers_1.html

Pires, J. N. (2006). *Welding robots : technology, systems issues and applications.* London: Springer.

Rosheim, M. E. (1994). *Robot Evolution: The Development of Anthrobots.* New York: Willey & Sons.

Sarker, S. &. (2003). Understanding mobile handheld device use and adoption. *Communications of the ACM* , 46(12):35–40.

Siciliano, B., & Oussama, K. (2008). *Springer Handbook of Robotics.* Springer.

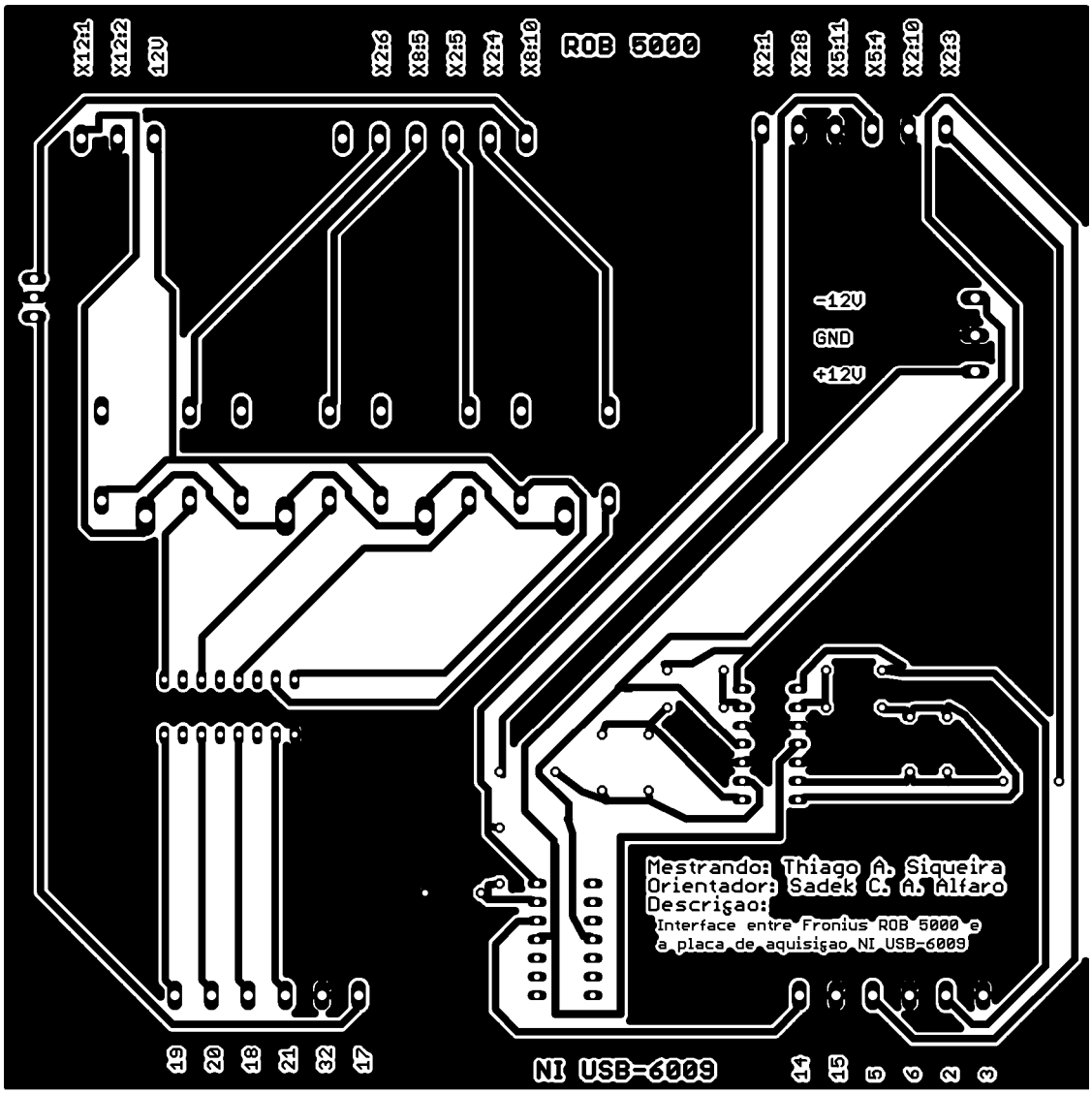
Siqueira, T. A. (2006). Controle de uma célula de soldagem através de um PDA. *Trabalho final* . Brasília, Brasil: Universidade de Brasília, Departamento de Engenharia Mecânica.

Weman, K. (2003). *Welding processes handbook.* Woodhead Pub.

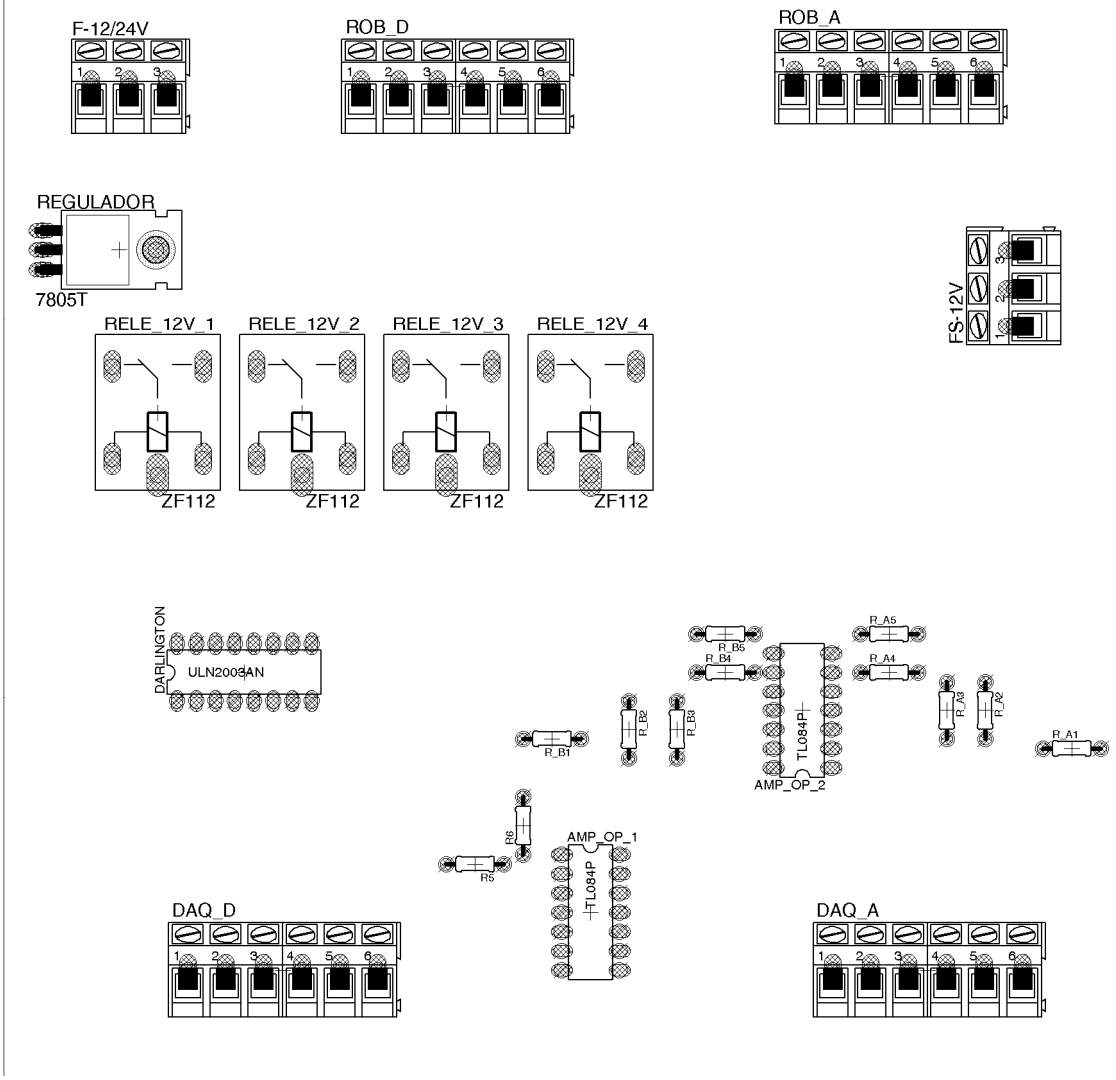
Williams, M. C. (2006). *Update: PCmarket achieved double-digit growth in 2005*. Fonte: Infoworld: http://www.infoworld.com/article/06/01/19/74317_HNgrowthin2005_1.html

APÊNDICE A – PLACA REGULADORA DE TENSÃO

Trilhas



Componentes



APÊNDICE B - CÓDIGO FONTE DO SERVIDOR

CurvaCalibracao.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PonteDados
{
    /// <summary>
    /// Classe estática responsável por corrigir os valores analógicos tranferidos ou
recebidos da placa de aquisição de dados
    /// </summary>
    public static class CurvaCalibracao
    {

        private static double fatorCorrecaoPotencia() { return 0.05; }
        private static double fatorCorrecaoCorrente() { return 200; }
        private static double fatorCorrecaoTensao() { return 20; }
        private static double fatorCorrecaoArame() { return 5; }
        // private static double fatorAdCorrecaoArame() { return 0.2; }

        /// <summary>
        /// Corrige o valor da potência para que possa ser enviada corretamente à placa de
aquisição
        /// </summary>
        /// <param name="entrada">Valor de potência desejada de 0% a 100%</param>
        /// <returns>Valor de potência corrigido</returns>
        public static double Potencia(double entrada)
        {
            double temp = fatorCorrecaoPotencia() * (Math.Pow(0.00002 * entrada, 2) +
1.0086 * entrada - 0.0155);
            return (temp > 5) ? 5 : ((temp < 0) ? 0 : temp);
        }
        /// <summary>
        /// Corrige o valor de corrente recebida da placa de aquisição
        /// </summary>
        /// <param name="entrada">Valor de corrente recebido da placa</param>
        /// <returns>Valor de corrente corrigido</returns>
        public static double Corrente(double entrada)
        {
            return 0.9995 * entrada * fatorCorrecaoCorrente() + 1.6733;
        }
        /// <summary>
        /// Corrige o valor de tensão recebida da placa de aquisição
        /// </summary>
        /// <param name="entrada">Valor de tensão recebido da placa</param>
        /// <returns>Valor de tenão corrigido</returns>
        public static double Tensao(double entrada)
        {
            return 1.0039 * entrada * fatorCorrecaoTensao() + 0.2228;
        }
        /// <summary>
        /// Corrige o valor de velocidade do arame recebida da placa de aquisição
        /// </summary>
        /// <param name="entrada">Valor de velocidade do arame recebido da placa</param>
        /// <returns>Valor de velocidade do arame corrigido</returns>
        public static double Arame(double entrada)
        {
            return 1.1813 * entrada * fatorCorrecaoArame() + 0.1212;
        }
    }
}
```

DAQ.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NationalInstruments.DAQmx;
using System.Net.Sockets;
using System.Threading;
using NationalInstruments;

namespace PonteDados
{
    /// <summary>
    /// Classe responsável por controlar a placa de aquisição de dados
    /// </summary>
    public class DAQ
    {
        // public event EventHandler Dados;
        Principal Formulario;
        DateTime Agora;
        public bool Ativo = false;
        //
        private Task TaskAqCorrente;
        private Task TaskAqTensao;
        private Task TaskAqArame;
        private Task TaskCoPotencia;
        private Task TaskCoProcessoAtivo;
        private Task TaskCoTipoProcesso;
        private Task TaskCoAbrirArco;
        private Task TaskCoSistemaPronto;
        private Task TaskCoLimparErros;
        private AnalogMultiChannelReader ReaderAqCorrente;
        private AnalogMultiChannelReader ReaderAqTensao;
        private AnalogMultiChannelReader ReaderAqArame;
        private DigitalSingleChannelReader ReaderCoProcessoAtivo;
        private AnalogSingleChannelWriter WriterCoPotencia;
        private DigitalSingleChannelWriter WriterCoTipoProcesso;
        private DigitalSingleChannelWriter WriterCoAbrirArco;
        private DigitalSingleChannelWriter WriterCoSistemaPronto;
        private DigitalSingleChannelWriter WriterCoLimparErros;
        private System.Windows.Forms.Timer loopTimer;
        /// <summary>
        /// Contrutor da classe
        /// </summary>
        /// <param name="frm">Recebe uma instância do formulário Principal</param>
        public DAQ(Principal frm)
        {
            Formulario = frm;
            Agora = DateTime.Now;
            loopTimer = new System.Windows.Forms.Timer();
            loopTimer.Tick += new EventHandler(loopTimer_Tick);
        }
        private void lixo(object sender, EventArgs e)
        {
        }
        /// <summary>
        /// Evento de "Tick" do clock
        /// </summary>
        /// <param name="sender">loopTimer</param>
        /// <param name="e"></param>
        private void loopTimer_Tick(object sender, EventArgs e)
        {
            try
            {
                strDados saida = new strDados();
                saida.Corrente =
Math.Round(CurvaCalibracao.Corrente(ReaderAqCorrente.ReadSingleSample()[0]), 0);
                saida.Tensao =
Math.Round(CurvaCalibracao.Tensao(ReaderAqTensao.ReadSingleSample()[0]), 0);
                saida.Arame =
Math.Round(CurvaCalibracao.Arame(ReaderAqArame.ReadSingleSample()[0]), 1);
                saida.Ativo = ReaderCoProcessoAtivo.ReadSingleSampleMultiLine()[0];

                saida.EstadoMesa = MesaMensagem.Espera;
```

```
                saida.Duracao = DateTime.Now - Agora;
                //TODO responder esta estrutura
                // Dados.Invoke(saida, e);
                Formulario.Invoke(Formulario.m_DadosAtualizados, new object[] { saida });
            }

            catch (Exception exception)
            {
                loopTimer.Enabled = false;
                TaskAqCorrente.Dispose();
                TaskAqTensao.Dispose();
                throw exception;
            }
        }
        /// <summary>
        /// Para a aquisição dos dados
        /// </summary>
        public void Parar()
        {
            loopTimer.Enabled = false;
            TaskAqCorrente.Dispose();
            Ativo = false;
        }
        /// <summary>
        /// Inicia a aquisição dos dados
        /// </summary>
        public void Iniciar()
        {
            try
            {
                //TaskAqCorrente
                TaskAqCorrente = new Task();

TaskAqCorrente.AIChannels.CreateVoltageChannel(Configuracoes.Default.EntradaAnalogicaCorren
te, "", (AITerminalConfiguration)(-1), 0, 5, AIVoltageUnits.Volts);
                TaskAqCorrente.Control(TaskAction.Verify);
                ReaderAqCorrente = new AnalogMultiChannelReader(TaskAqCorrente.Stream);
                //TaskAqTensao
                TaskAqTensao = new Task();

TaskAqTensao.AIChannels.CreateVoltageChannel(Configuracoes.Default.EntradaAnalogicaTensao,
"", (AITerminalConfiguration)(-1), 0, 5, AIVoltageUnits.Volts);
                TaskAqTensao.Control(TaskAction.Verify);
                ReaderAqTensao = new AnalogMultiChannelReader(TaskAqTensao.Stream);
                //TaskAqArame
                TaskAqArame = new Task();

TaskAqArame.AIChannels.CreateVoltageChannel(Configuracoes.Default.EntradaAnalogicaArame,
"", (AITerminalConfiguration)(-1), 0, 5, AIVoltageUnits.Volts);
                TaskAqArame.Control(TaskAction.Verify);
                ReaderAqArame = new AnalogMultiChannelReader(TaskAqArame.Stream);
                //TaskCoPotencia
                TaskCoPotencia = new Task();

TaskCoPotencia.AOChannels.CreateVoltageChannel(Configuracoes.Default.SaidaAnalogicaPotencia
, "aoChannel", 0, 5, AOVoltageUnits.Volts);
                WriterCoPotencia = new AnalogSingleChannelWriter(TaskCoPotencia.Stream);
                //
                TaskCoProcessoAtivo = new Task();

TaskCoProcessoAtivo.DIChannels.CreateChannel(Configuracoes.Default.EntradaDigitalProcessoAt
ivo, "myChannel", ChannelLineGrouping.OneChannelForEachLine);
                ReaderCoProcessoAtivo = new
DigitalSingleChannelReader(TaskCoProcessoAtivo.Stream);
                //
                TaskCoTipoProcesso = new Task();

TaskCoTipoProcesso.DOChannels.CreateChannel(Configuracoes.Default.SaidaDigitalTipoPrograma,
"", ChannelLineGrouping.OneChannelForEachLine);
                WriterCoTipoProcesso = new
DigitalSingleChannelWriter(TaskCoTipoProcesso.Stream);
                //
                TaskCoAbrirArco = new Task();

TaskCoAbrirArco.DOChannels.CreateChannel(Configuracoes.Default.SaidaDigitalAbrirArco, "",
ChannelLineGrouping.OneChannelForEachLine);
                WriterCoAbrirArco = new DigitalSingleChannelWriter(TaskCoAbrirArco.Stream);
```

```
                //
                TaskCoSistemaPronto = new Task();

TaskCoSistemaPronto.DOChannels.CreateChannel(Configuracoes.Default.SaidaDigitalSistemaPront
o, "", ChannelLineGrouping.OneChannelForEachLine);
                WriterCoSistemaPronto = new
DigitalSingleChannelWriter(TaskCoSistemaPronto.Stream);
                //
                TaskCoLimparErros = new Task();

TaskCoLimparErros.DOChannels.CreateChannel(Configuracoes.Default.SaidaDigitalLimparErros,
"", ChannelLineGrouping.OneChannelForEachLine);
                WriterCoLimparErros = new
DigitalSingleChannelWriter(TaskCoLimparErros.Stream);
                //
                loopTimer.Interval = Configuracoes.Default.AtrasoEntreAmostras;
                loopTimer.Enabled = true;
                //
                // Inicializar saidas
                WriterCoAbrirArco.WriteSingleSampleSingleLine(true, false);
                WriterCoLimparErros.WriteSingleSampleSingleLine(true, false);
                WriterCoSistemaPronto.WriteSingleSampleSingleLine(true, false);
                WriterCoTipoProcesso.WriteSingleSampleSingleLine(true, false);
                Ativo = true;
            }
            catch (DaqException exception)
            {
                //disable the timer and dispose of the task
                loopTimer.Enabled = false;
                TaskAqCorrente.Dispose();
                Ativo = false;
            }
            catch (Exception exception)
            {
                throw exception;
            }
        }
        /// <summary>
        /// Envia o valor da potência para a placa de aquisição
        /// </summary>
        /// <param name="Forca">Valor percentual (Corrigido) da potência</param>
        public void PercentualForca(double Forca)
        {
            if (Ativo)
            {
                WriterCoPotencia.WriteSingleSample(true, CurvaCalibracao.Potencia(Forca));
            }
        }
        /// <summary>
        /// Envia o tipo do processo a ser utilizado (Pulsado ou padrão)
        /// </summary>
        /// <param name="Pulsado">É pulsado?</param>
        public void TipoProcesso(bool Pulsado)
        {
            if (Ativo)
            {
                WriterCoTipoProcesso.WriteSingleSampleSingleLine(true, Pulsado);
            }

        }
        /// <summary>
        /// Envia o Estado do sistema
        /// </summary>
        /// <param name="Pronto">Pronto?</param>
        public void EstadoSistema(bool Pronto)
        {
            if (Ativo)
            {
                WriterCoSistemaPronto.WriteSingleSampleSingleLine(true, Pronto);
            }
        }
        /// <summary>
        /// Envia o sinal de limpar erros
        /// </summary>
        public void LimparErros()
        {
            if (Ativo)
```

```
            {
                WriterCoLimparErros.WriteSingleSampleSingleLine(true, true);
                Thread.Sleep(10);
                WriterCoLimparErros.WriteSingleSampleSingleLine(true, false);
            }
        }
        /// <summary>
        /// Envia o comando para abrir o arco
        /// </summary>
        public void ArcoAbrir()
        {
            if (Ativo)
            {
                WriterCoAbrirArco.WriteSingleSampleSingleLine(true, true);
            }
        }
        /// <summary>
        /// Envia o comando para fechar o arco
        /// </summary>
        public void ArcoFechar()
        {
            if (Ativo)
            {
                WriterCoAbrirArco.WriteSingleSampleSingleLine(true, false);
            }
        }
    }
}
```

Mesa.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;

namespace PonteDados
{
    /// <summary>
    /// Classe responsável por controlar a mesa posicionadora
    /// </summary>
    public class Protocolo
    {
        SerialPort PortaSerial;
        string Executar = ((char)13).ToString() + "EXE" + ((char)13).ToString();
        string Limpar = ((char)13).ToString() + "CLR" + ((char)13).ToString();
        /// <summary>
        /// Contrutor
        /// </summary>
        /// <param name="Porta">Instância da porta serial a ser utilizada</param>
        public Protocolo(System.IO.Ports.SerialPort Porta)
        {
            PortaSerial = Porta;
        }
        /// <summary>
        /// Envia a frase utilizando a instância da porta serial passada pelo contrutor
        /// </summary>
        /// <param name="Frase">Dados a serem enviados pela porta serial</param>
        /// <returns></returns>
        public bool EnviarFrase(string Frase)
        {
            try
            {
                if (!PortaSerial.IsOpen)
                {
                    PortaSerial.Open();
                }
                PortaSerial.Write(Frase);
                return true;
            }
            catch
            {
                return false;
            }
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="Velocidade"></param>
        /// <param name="Duracao"></param>
        /// <param name="Deslocamento"></param>
        /// <param name="DirecaoMotor"></param>
        /// <returns></returns>
        public bool Mover(string Velocidade, string Duracao, string Deslocamento, bool
DirecaoMotor)
        {
            if (!PortaSerial.IsOpen)
            {
                PortaSerial.Open();
            }
            decimal VEL;
            int TEM;
            int DIS;
            int FI1;
            int FI2;
            try { VEL = decimal.Parse(Velocidade); }
            catch { throw new Exception("O valor da velocidade deve ser um decimal!"); }
            try { TEM = int.Parse(Duracao); }
            catch { throw new Exception("O valor da duração deve ser um inteiro!"); }
            try { DIS = int.Parse(Deslocamento); }
            catch { throw new Exception("O valor do deslocamento deve ser um inteiro!"); }
            if (VEL < 1 || VEL > 10)
            {
```

```
            throw new Exception("Velocidade inválida!");
        }
        if (TEM < 0 || TEM > 120)
        {
            throw new Exception("Duração inválida!");
        }
        if (DIS < 0 || DIS > 100)
        {
            throw new Exception("Deslocamento inválido!");
        }
        FI1 = DirecaoMotor ? 1 : 0;
        FI2 = DirecaoMotor ? 0 : 1;
        string frase = GerarFrase(VEL, TEM, DIS, FI1, FI2);
        PortaSerial.DiscardInBuffer();
        PortaSerial.DiscardOutBuffer();


        EnviarFrase(frase);
        System.Threading.Thread.Sleep(500);
        int contador = 0;
        bool resp = ValidarResposta(frase);
        while (!resp && contador < 5)
        {
            PortaSerial.DiscardInBuffer();
            PortaSerial.DiscardOutBuffer();
            EnviarFrase(Limpar);
            System.Threading.Thread.Sleep(500);
            EnviarFrase(frase);
            contador++;
            resp = ValidarResposta(frase);
        }
        if (resp)
        {
            EnviarFrase(Executar);
        }
        else
        {
            return false;
        }
        return true;
    }
    /// <summary>
    ///
    /// </summary>
    /// <param name="frase"></param>
    /// <returns></returns>
    public bool ValidarResposta(string frase)
    {
        int bytesParaLer = PortaSerial.BytesToRead;
        char[] saida = new char[bytesParaLer];
        PortaSerial.Read(saida, 0, bytesParaLer);
        string resposta = new string(saida);
        if (resposta == "")
        {
            return false;
        }
        else
        {
            return frase.Replace('.', ',') == resposta.Replace('.', ',');
        }
    }
    /// <summary>
    /// Gerador da frase a partir dos parâmetros
    /// </summary>
    /// <param name="VEL">Velocidade</param>
    /// <param name="TEM">Tempo</param>
    /// <param name="DIS">Deslocamento</param>
    /// <param name="FI1">Flag1</param>
    /// <param name="FI2">Flag2</param>
    /// <returns></returns>
    protected string GerarFrase(decimal VEL, int TEM, int DIS, int FI1, int FI2)
    {
        string CR = ((char)13).ToString();
        string saida = "";
        saida += CR + "VEL:" + VEL.ToString("##0.0") + CR;
        saida += CR + "TEM:" + TEM.ToString("##0") + CR;
        saida += CR + "DIS:" + DIS.ToString("##0") + CR;
```

```
            saida += CR + "FI1:" + FI1.ToString("##0") + CR;
            saida += CR + "FI2:" + FI2.ToString("##0") + CR;
            return saida;
        }

    }
}
```

Welding*Socket*.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.Net;
using System.Threading;

namespace MeuSocket
{
    public delegate void Conectado();
    public delegate void Ativado();
    public delegate void Disconectado();
    public delegate void Desativado();
    public delegate void Recebido(byte[] Dados);
    /// <summary>
    /// Classe responsável por gerenciar a conexão socket TCP
    /// </summary>
    public class WeldingSocket
    {
        enum TipoSocket
        {
            NaoDefinido, Cliente, Servidor
        }
        private TipoSocket Tipo = TipoSocket.NaoDefinido;
        public TcpClient Cliente;
        private TcpListener Servidor;
        private Thread ServidorThread;
        private Thread ClienteThread;
        private NetworkStream WeldingStream;
        public bool Conectado = false;
        public bool? Ativo = null;
        //Delegates
        public Conectado DelegateConectado;
        public Ativado DelegateAtivado;
        public Disconectado DelegateDisconectado;
        public Desativado DelegateDesativado;
        public Recebido DelegateRecebido;
        /// <summary>
        /// Conecta o Cliente TCP
        /// </summary>
        /// <param name="IP">Numero do IP do servidor, ex.: 192.168.0.100</param>
        /// <param name="Porta">Numero da porta do servidor, ex.: 8080</param>

        public void Conectar(string IP, int Porta)
        {
            Tipo = TipoSocket.Cliente;
            Cliente = new TcpClient();
            Servidor = null;
            IPEndPoint pontoFinal = new IPEndPoint(IPAddress.Parse(IP), Porta);
            Cliente.Connect(pontoFinal);
            WeldingStream = Cliente.GetStream();
            Conectado = true;
        }
        /// <summary>
        /// Ativa o Servidor TCP
        /// </summary>
        /// <param name="Porta">Número da porta TCP o qual o servidor ficará
escutando</param>
        public void Ativar(int Porta)
        {
            Tipo = TipoSocket.Servidor;
            Ativo = true;
            this.Servidor = new TcpListener(IPAddress.Any, Porta);
            this.ServidorThread = new Thread(new ThreadStart(BuscaPorClientes));
            this.ServidorThread.Start();
        }
        /// <summary>
        /// Busca por novos clientes
        /// </summary>
        private void BuscaPorClientes()
        {
            this.Servidor.Start();
```

```
            while (Ativo == true)
            {
                Cliente = this.Servidor.AcceptTcpClient();
                ClienteThread = new Thread(new
ParameterizedThreadStart(CuidarConexaoCliente));
                ClienteThread.Start(Cliente);
                DelegateAtivado();
            }
        }
        /// <summary>
        /// Cuida da conexão com o cliente
        /// </summary>
        /// <param name="client">Instância da conexão Socket com o cliente</param>
        private void CuidarConexaoCliente(object client)
        {
            Cliente = (TcpClient)client;
            Cliente.ReceiveTimeout = 50;
            WeldingStream = Cliente.GetStream();
            while (Ativo == true)
            {
                if (WeldingStream.DataAvailable)
                {
                    try
                    {
                        byte[] dados = new byte[4096];
                        int DadosLidos = WeldingStream.Read(dados, 0, 4096);
                        DelegateRecebido(dados.Take(DadosLidos).ToArray());
                    }
                    catch
                    {
                        if (ClienteThread.IsAlive)
                        {
                            ClienteThread.Abort();
                        }
                        DelegateDesativado();
                    }
                }
                else
                {
                    if (!Cliente.Client.Connected)
                    {
                        DelegateDisconectado();
                    }
                }
            }
            Cliente.Close();
        }
        /// <summary>
        /// Envia dados pela conexão TCP
        /// </summary>
        /// <param name="Dados">Dados a serem enviados</param>
        public void Enviar(byte[] Dados)
        {
            switch (Tipo)
            {
                case TipoSocket.NaoDefinido:
                    break;
                case TipoSocket.Cliente:
                    if (Conectado)
                    {
                        if (Cliente.Connected)
                        {
                            Cliente.Client.Send(Dados);
                        }
                    }
                    break;
                case TipoSocket.Servidor:
                    if (Ativo == true)
                    {
                        if (Cliente != null && Cliente.Connected)
                        {
                            if (ClienteThread.IsAlive)
                            {
                                Cliente.Client.Send(Dados);
                            }
                        }
                        else
```

```
                    {
                        DelegateDisconectado();
                    }
                }
                break;
            default:
                break;
        }
    }
    /// <summary>
    /// Desconecta do Servidor
    /// </summary>
    public void Disconectar()
    {
        Conectado = false;
        if (Tipo == TipoSocket.Cliente)
        {
            Cliente.Close();
            WeldingStream = null;
            DelegateDisconectado();
        }
    }
    /// <summary>
    /// Desativa o servidor
    /// </summary>
    public void Desativar()
    {
        Ativo = false;
        if (Tipo == TipoSocket.Servidor)
        {
            if (ClienteThread != null)
            {
                ClienteThread.Abort();
            }
            ServidorThread.Abort();
            Servidor.Stop();
            if (Servidor.Server.Connected)
            {
                Servidor.Server.Close();
                DelegateDisconectado();
            }
            DelegateDesativado();
        }
    }
  }
}
```

Serializador.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PonteDados
{
    /// <summary>
    /// Classe estática que serializa e deserializa os objetos ( Objetos <-> byte[] )
    /// </summary>
    public static class Serializador
    {
        /// <summary>
        /// Serializa o objeto de dados strDados em um array de bytes
        /// </summary>
        /// <param name="Dados">Instância do objeto strDados</param>
        /// <returns>Array de bytes</returns>
        public static byte[] SerializarDados(this strDados Dados)
        {
            string temp = "E=" + (Dados.Ativo ? "A" : "N") + ";C=" +
Dados.Corrente.ToString("000") + ";T=" + Dados.Tensao.ToString("00.0") + ";V=" +
Dados.Arame.ToString("00.0") + ";M=" + ((int)Dados.EstadoMesa).ToString("0") + ";D=" +
((int)Dados.Duracao.TotalMilliseconds).ToString() + "\n";
            return temp.ToCharArray().Select(n => (byte)n).ToArray();
        }
        /// <summary>
        /// Deserializa o objeto de dados a partir de um array de bytes
        /// </summary>
        /// <param name="bytes">Array de bytes a ser deserializados</param>
        /// <returns>Lista de dados</returns>
        public static List<strDados> DeserializarDados(this byte[] bytes)
        {
            string temp = new string(bytes.Select(n => (char)n).ToArray());
            string[] Estruturas = temp.Split('\n');
            List<strDados> EstruturasTipadas = new List<strDados>();
            foreach (string est in Estruturas)
            {
                string[] SubStr = est.Split(';');
                if (SubStr.Length == 3)
                {
                    strDados s = new strDados();
                    s.Ativo = SubStr[0] == "E=A";
                    s.Corrente = double.Parse(SubStr[1].Replace("C=", "").Replace(".",
","));
                    s.Tensao = double.Parse(SubStr[2].Replace("T=", "").Replace(".", ","));
                    s.Arame = double.Parse(SubStr[3].Replace("V=", "").Replace(".", ","));
                    s.EstadoMesa = (MesaMensagem)int.Parse(SubStr[4].Replace("M=", ""));
                    s.Duracao = new TimeSpan(0, 0, 0, 0, int.Parse(SubStr[5].Replace("D=",
"")));

                    EstruturasTipadas.Add(s);
                }
            }
            return EstruturasTipadas;
        }
        /// <summary>
        /// Serializa o objeto de comandos num array de bytes
        /// </summary>
        /// <param name="Comando">Comando a ser serializado</param>
        /// <returns>Array de bytes</returns>
        public static byte[] SerializarComandos(this strComandos Comando)
        {
            string temp = "";
            temp += ((Comando.ArcoAberto == null) ? "" :
((bool)Comando.ArcoAberto).ToString()).PadLeft(5, ' ') + ";";
            temp += ((Comando.LimparErro == null) ? "" :
((bool)Comando.LimparErro).ToString()).PadLeft(5, ' ') + ";";
            temp += ((Comando.Pulsado == null) ? "" :
((bool)Comando.Pulsado).ToString()).PadLeft(5, ' ') + ";";
            temp += ((Comando.SitemaPronto == null) ? "" :
((bool)Comando.SitemaPronto).ToString()).PadLeft(5, ' ') + ";";
            temp += ((Comando.PercentualPotencia == null) ? "" :
((double)Comando.PercentualPotencia).ToString("000.0")) + ";";
            //Mesa
```

```
            temp += ((Comando.MesaVelocidade == null) ? "" :
((double)Comando.MesaVelocidade).ToString("000.0")) + ";";
            temp += ((Comando.MesaMovimento == null) ? "" :
((int)Comando.MesaMovimento).ToString("0")) + ";";
            temp += ((Comando.MesaValor == null) ? "" :
((int)Comando.MesaValor).ToString("000")) + ";";
            temp += ((Comando.MesaSentido == null) ? "" :
((int)Comando.MesaSentido).ToString("0")) + ";" + "\n";
            //
            return temp.ToCharArray().Select(n => (byte)n).ToArray();
        }
        /// <summary>
        /// Deserializa o objeto de comandos a partir de um array de bytes
        /// </summary>
        /// <param name="bytes">Array de bytes a serem deserializados</param>
        /// <returns>Lista de comandos</returns>
        public static List<strComandos> DeserializarComandos(this byte[] bytes)
        {
            string Mensagem = new string(bytes.Select(n => (char)n).ToArray());
            List<strComandos> Estruturas = new List<strComandos>();

            foreach (string Linha in Mensagem.Split('\n'))
            {
                try
                {
                    strComandos Estrutura = new strComandos();
                    string[] est = Linha.Split(';').Select(n => n.Trim()).ToArray();
                    Estrutura.ArcoAberto = string.IsNullOrEmpty(est[0]) ? (bool?)null :
bool.Parse(est[0]);
                    Estrutura.LimparErro = string.IsNullOrEmpty(est[1]) ? (bool?)null :
bool.Parse(est[1]);
                    Estrutura.Pulsado = string.IsNullOrEmpty(est[2]) ? (bool?)null :
bool.Parse(est[2]);
                    Estrutura.SitemaPronto = string.IsNullOrEmpty(est[3]) ? (bool?)null :
bool.Parse(est[3]);
                    Estrutura.PercentualPotencia = string.IsNullOrEmpty(est[4]) ?
(double?)null : double.Parse(est[4].Replace(".", ","));
                    //Mesa
                    Estrutura.MesaVelocidade = string.IsNullOrEmpty(est[5]) ? (double?)null
: double.Parse(est[5].Replace(".",","));
                    Estrutura.MesaMovimento = string.IsNullOrEmpty(est[6]) ?
(MesaLimite?)null : (MesaLimite)int.Parse(est[6]);
                    Estrutura.MesaValor = string.IsNullOrEmpty(est[7]) ? (int?)null :
int.Parse(est[7]);
                    Estrutura.MesaSentido = string.IsNullOrEmpty(est[8]) ?
(MesaMovimento?)null : (MesaMovimento)int.Parse(est[8]);
                    //
                    Estruturas.Add(Estrutura);
                }
                catch
                {

                }
            }

            return Estruturas;
        }
    }
    /// <summary>
    /// Estrutura contendo todos os possíveis dados
    /// </summary>
    public struct strDados
    {
        public double Corrente;
        public double Tensao;
        public double Arame;
        public bool Ativo;
        public MesaMensagem EstadoMesa;
        public TimeSpan Duracao;
    }
    /// <summary>
    /// Estrutura contendo todos os possíveis comando
    /// </summary>
    public struct strComandos
    {
        public bool? SitemaPronto;
        public bool? Pulsado;
```

```
        public bool? ArcoAberto;
        public bool? LimparErro;
        public double? PercentualPotencia;
        public double? MesaVelocidade;
        public MesaLimite? MesaMovimento;
        public int? MesaValor;
        public MesaMovimento? MesaSentido;
    }
    /// <summary>
    /// Enumerável dos estados da mesa
    /// </summary>
    public enum MesaMensagem
    {
        Espera, Enviado, NaoEnviado
    }
    /// <summary>
    /// Enumerável do tipo de parâmentro que irá definir o limite do movimento
    /// </summary>
    public enum MesaLimite
    {
        Deslocamento, Duracao
    }
    /// <summary>
    /// Enumerável do sentido de movimentação da mesa
    /// </summary>
    public enum MesaMovimento
    {
        EmDirecaoAoMotor, ContrarioAoMotor
    }
}
```

APÊNDICE C - CÓDIGO FONTE DO CLIENTE

Persistencia.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;

namespace MobileWelding
{
    /// <summary>
    /// Classe estática para persistência dos dados
    /// </summary>
    public static class Persistencia
    {
        /// <summary>
        /// Gera o nome do arquivo a partir dos parâmetros
        /// </summary>
        /// <param name="Parametros">Parâmentros de soldagem utilizados no processo</param>
        /// <returns>Nome do arquivo</returns>
        public static string Arquivo(string Parametros)
        {
            string Aplicacao =
System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName()
.CodeBase);
            string Cartao = "SD Card";
            return Cartao + "\\" + DateTime.Now.ToString("yyyy-MM-dd HH-mm-ss") +
Parametros + ".txt";
        }
        /// <summary>
        /// Salva o processo em um arquivo
        /// </summary>
        /// <param name="Duracao">Duração</param>
        /// <param name="Velocidade">Velocidade</param>
        /// <param name="Sentido">Sentido do movimento</param>
        /// <param name="Pulsado">Tipo do processo</param>
        /// <param name="Dados">Dados coletados durante o processo</param>
        internal static void SavarTarefa(int Duracao, double Velocidade,
PonteDados.MesaMovimento Sentido, bool Pulsado, PonteDados.strDados[] Dados)
        {
            string Linha = "";
            string Parametros = " D-" + Duracao.ToString() + " V-" +
Velocidade.ToString("#0") + " S-" + (Sentido == PonteDados.MesaMovimento.ContrarioAoMotor ?
"CM" : "M") + " P-" + (Pulsado ? "S" : "N");
            foreach (PonteDados.strDados d in Dados)
            {
                Linha += d.Duracao.ToString() + ";";
                Linha += d.Corrente.ToString("##0") + ";";
                Linha += d.Tensao.ToString("#0.0") + ";";
                Linha += d.Arame.ToString("#0.0") + ";";
                Linha += Environment.NewLine;
            }
            var temp = new System.IO.StreamWriter(Arquivo(Parametros), false);
            temp.WriteLine(Linha);
            temp.Close();
            temp.Dispose();
            temp = null;
        }
    }
}
```

Grafico.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using PonteDados;

namespace GraficoXY
{
    /// <summary>
    /// Classe estática para gerar o gráfico
    /// </summary>
    public static class Grafico
    {
        /// <summary>
        /// Plota o gráfico em um PictureBox
        /// </summary>
        /// <param name="Figura">Controle PictureBox onde será desenhado o gráfico</param>
        /// <param name="grafico">Objeto Grafico do controle PictureBox</param>
        /// <param name="Dados">Dados a serem plotados</param>
        public static void Plotar(this System.Windows.Forms.PictureBox Figura,
System.Drawing.Graphics grafico, strDados[] Dados)
        {
            Graphics Grafico = Graphics.FromImage(Figura.Image);

            int Altura = Figura.Size.Height;
            int Largura = Figura.Size.Width;
            int OffSet = 5;

            Grafico.Clear(Color.White);
            int minimoX = (int)Dados.Min(n => n.Duracao.TotalMilliseconds);
            int maximoX = (int)Dados.Max(n => n.Duracao.TotalMilliseconds);

            DesenharGraduacaoEixoY(Grafico, Altura, OffSet, 2, new SolidBrush(Color.Red),
70, 470);
            DesenharGraduacaoEixoY(Grafico, Altura, OffSet, 35, new SolidBrush(Color.Blue),
13, 32);
            DesenharGraduacaoEixoY(Grafico, Altura, OffSet, 60, new
SolidBrush(Color.Green), 2, 22);

            DesenharPontos(Grafico, Largura, Altura, 80, OffSet, Color.Red, 70, 470,
minimoX, maximoX, Dados.Select(n => new PointF() { X = n.Duracao.TotalMilliseconds, Y =
n.Corrente }).ToArray());
            DesenharPontos(Grafico, Largura, Altura, 80, OffSet, Color.Blue, 13, 32,
minimoX, maximoX, Dados.Select(n => new PointF() { X = n.Duracao.TotalMilliseconds, Y =
n.Tensao }).ToArray());
            DesenharPontos(Grafico, Largura, Altura, 80, OffSet, Color.Green, 2, 22,
minimoX, maximoX, Dados.Select(n => new PointF() { X = n.Duracao.TotalMilliseconds, Y =
n.Arame }).ToArray());

            DesenharGraduacaoEixoX(Grafico, Largura, Altura, 80, Color.Yellow, minimoX,
maximoX, Dados.Select(n => n.Duracao.TotalMilliseconds).ToArray());
            grafico = Grafico;
        }
        /// <summary>
        /// Desenha o eixo X de coordenadas
        /// </summary>
        /// <param name="Grafico">Objeto gráfico onde será desenhado o eixo</param>
        /// <param name="Largura">Largura do gráfico</param>
        /// <param name="Altura">Altura do gráfico</param>
        /// <param name="OffSetX">Distância horizantal das bordas</param>
        /// <param name="Cor">Cor do eixo</param>
        /// <param name="minimoX">Valor mínimo da escala X</param>
        /// <param name="maximoX">Valor máximo da escala Y</param>
        /// <param name="Duracao">Largura de tempo da janela</param>
        private static void DesenharGraduacaoEixoX(Graphics Grafico, int Largura, int
Altura, int OffSetX, Color Cor, int minimoX, int maximoX, double[] Duracao)
        {
            double[] PontosFiltrados = Duracao.Where(m => m >= minimoX && m <=
maximoX).Select(n => n - n % 1000).Distinct().ToArray();
            int[] PontoCorrigidos = new int[PontosFiltrados.Length];
            for (int i = 0; i < PontosFiltrados.Length; i++)
            {
                if (PontosFiltrados[i] < 0)
```

```
                {
                    PontoCorrigidos[i] = 0;
                }
                else
                {
                    PontoCorrigidos[i] = (int)((Largura - OffSetX) * (PontosFiltrados[i] -
minimoX) / (maximoX - minimoX) + OffSetX);
                }
            }
            for (int i = 0; i < PontoCorrigidos.Length; i++)
            {
                if (PontoCorrigidos[i] >= OffSetX)
                {
                    Grafico.DrawLine(new Pen(Cor, 2), (int)PontoCorrigidos[i], Altura,
(int)PontoCorrigidos[i], Altura - 10);
                }
            }
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="Grafico">Objeto gráfico onde será desenhado o eixo</param>
        /// <param name="Largura">Largura do gráfico</param>
        /// <param name="Altura">Altura do gráfico</param>
        /// <param name="OffSetX">Distância horizantal das bordas</param>
        /// <param name="OffSetY">Distância vertical das bordas</param>
        /// <param name="Cor">Cor do eixo</param>
        /// <param name="MinY">Valor mínimo da escala Y</param>
        /// <param name="MaxY">Valor máximo da escala Y</param>
        /// <param name="MinX">Valor mínimo da escala X</param>
        /// <param name="MaxX">Valor máximo da escala X</param>
        /// <param name="Pontos"></param>
        private static void DesenharPontos(Graphics Grafico, int Largura, int Altura, int
OffSetX, int OffSetY, Color Cor, int MinY, int MaxY, int MinX, int MaxX, PointF[] Pontos)
        {
            PointF[] PontosFiltrados = Pontos.Where(m => m.X >= MinX && m.X <=
MaxX).ToArray();
            Point[] PontoCorrigidos = new Point[PontosFiltrados.Length];
            double l = (double)Largura;
            double a = (double)Altura;
            double oX = (double)OffSetX;
            double oY = (double)OffSetY;
            double miX = (double)MinX;
            double maX = (double)MaxX;
            double miY = (double)MinY;
            double maY = (double)MaxY;
            for (int i = 0; i < PontosFiltrados.Length; i++)
            {
                double x = PontosFiltrados[i].X;
                double y = PontosFiltrados[i].Y;
                Point temp = new Point();
                if (x < 0)
                {
                    temp.X = 0;
                }
                else
                {
                    //Interpolacao de X
                    temp.X = (int)((l - oX) * (x - miX) / (maX - miX) + oX);
                }
                if (y < 0)
                {
                    temp.Y = 0;
                }
                else
                {
                    //Interpolacao de Y
                    // temp.Y = (int)((a - 2 * oY) * (y - miY) / (maY - miY) + oY);
                    temp.Y = (int)((2 * oY - Altura) * (y - miY) / (maY - miY) + (a - oY));
                }
                PontoCorrigidos[i] = temp;

            }
            for (int i = 1; i < PontoCorrigidos.Length; i++)
            {
                Grafico.DrawLine(new Pen(Cor), PontoCorrigidos[i - 1].X, PontoCorrigidos[i
- 1].Y, PontoCorrigidos[i].X, PontoCorrigidos[i].Y);
```

```
            }
        }
        /// <summary>
        /// Desenha o eixo Y de coordenadas
        /// </summary>
        /// <param name="Grafico">Objeto gráfico onde será desenhado o eixo</param>
        /// <param name="Altura">Altura do gráfico</param>
        /// <param name="OffSet">Distância das bordas</param>
        /// <param name="Posicao"></param>
        /// <param name="Cor">Cor dos pontos</param>
        /// <param name="Inicio"></param>
        /// <param name="Fim"></param>
        private static void DesenharGraduacaoEixoY(System.Drawing.Graphics Grafico, int
Altura, int OffSet, int Posicao, System.Drawing.Brush Cor, int Inicio, int Fim)
        {
            int tamanhoFonte = 25;//pixels
            int Segmentos = (int)Math.Floor(((float)(Altura - 2 * OffSet)) / tamanhoFonte);
            float Escala = ((float)Fim - (float)Inicio) / (float)Segmentos;

            for (int i = 0; i <= Segmentos; i++)
            {
                float PosicaoX = Posicao;
                float PosicaoY = Altura - OffSet - (i + (float)0.5) * tamanhoFonte;
                float Valor = i * Escala + Inicio;
                Grafico.DrawString(Valor.ToString("##0"), new
Font(FontFamily.GenericMonospace, 6, FontStyle.Regular), Cor, PosicaoX, PosicaoY);
            }
        }
    }
    /// <summary>
    /// Classe contendo o ponto com precisão de ponto Flutuante
    /// </summary>
    public class PointF
    {
        public double X;
        public double Y;
    }
}
```

Configuracao.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace MobileWelding
{
    /// <summary>
    /// Classe estática utilizada para ler e salvar os dados de configuração
    /// </summary>
    public static class Configuração
    {
        /// <summary>
        /// Retorna o nome completo do arquivo que conterá as configurações
        /// </summary>
        /// <returns>Nome do arquivo</returns>
        public static string Arquivo()
        {
            string Aplicacao =
System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().GetName()
.CodeBase);
            //string Cartao = "SD Card";
            //return Cartao + "\\Ethernet.txt";
            return Aplicacao + "\\Ethernet.txt";
        }
        /// <summary>
        /// Checa se o arquivo de configuração existe
        /// </summary>
        /// <returns>Verdadeiro se o arquivo existir</returns>
        public static bool Existe()
        {
            return Ler() != null;
        }
        /// <summary>
        /// Retorna o objeto de configuração a partir do arquivo
        /// </summary>
        /// <returns>nulo se o arquivo não existir ou estiver danificado</returns>
        public static IPPorta? Ler()
        {
            try
            {

                if (System.IO.File.Exists(Arquivo()))
                {
                    //var arquivo = System.IO.File.Open("Ethernet.txt",
System.IO.FileMode.OpenOrCreate);
                    //byte[] Bytes = new byte[arquivo.Length];
                    //arquivo.Read(Bytes, 0, Bytes.Length);
                    //arquivo.Close();
                    //string ConfArq = new string(Bytes.Select(n => (char)n).ToArray());

                    var teste = new System.IO.StreamReader(Arquivo());
                    string ConfArq = teste.ReadLine();
                    teste.Close();
                    if (ConfArq != null)
                    {
                        IPPorta confIP = new IPPorta();
                        string IP = ConfArq.Split(',')[0];
                        string Porta = ConfArq.Split(',')[1];
                        confIP.Porta = int.Parse(Porta);
                        confIP.IP = IP.Split('.').Select(n => byte.Parse(n)).ToArray();
                        return confIP;
                    }
                    return null;
                }
                else
                {
                    return null;
                }
            }
            catch (Exception ex)
            {
```

```
                return null;
            }
        }
        /// <summary>
        /// Salva as configurações no arquivo texto
        /// </summary>
        /// <param name="Ip0">Primeiro byte do IP</param>
        /// <param name="Ip1">Segundo byte do IP</param>
        /// <param name="Ip2">Terceiro byte do IP</param>
        /// <param name="Ip3">Quarto byte do IP</param>
        /// <param name="Porta">Porta TCP</param>
        public static void Escrever(string Ip0, string Ip1, string Ip2, string Ip3, string
Porta)
        {
            string Linha = GerarEstrutura(Ip0, Ip1, Ip2, Ip3, Porta).Serializar();
            var temp = new System.IO.StreamWriter(Arquivo(), false);
            temp.WriteLine(Linha);
            temp.Close();
        }
        /// <summary>
        /// Gera objeto IPPorta a partir dos parâmetros
        /// </summary>
        /// <param name="Ip0">Primeiro byte do IP</param>
        /// <param name="Ip1">Segundo byte do IP</param>
        /// <param name="Ip2">Terceiro byte do IP</param>
        /// <param name="Ip3">Quarto byte do IP</param>
        /// <param name="Porta">Porta TCP</param>
        /// <returns>Objeto com IP e porta</returns>
        private static IPPorta GerarEstrutura(string Ip0, string Ip1, string Ip2, string
Ip3, string Porta)
        {
            IPPorta p = new IPPorta();
            byte[] bytes = new byte[4];
            bytes[0] = byte.Parse(Ip0);
            bytes[1] = byte.Parse(Ip1);
            bytes[2] = byte.Parse(Ip2);
            bytes[3] = byte.Parse(Ip3);
            p.IP = bytes;
            p.Porta = int.Parse(Porta);
            return p;
        }
        /// <summary>
        /// Tranforma o objeto IPPorta em um String
        /// </summary>
        /// <param name="p">Objeto IPPorta a ser transformado</param>
        /// <returns></returns>
        public static string Serializar(this IPPorta p)
        {
            return new StringBuilder().AppendFormat("{0}.{1}.{2}.{3},{4}", p.IP[0],
p.IP[1], p.IP[2], p.IP[3], p.Porta).ToString();
        }
        /// <summary>
        /// Tranforma o objeto IPPorta em um String sem a porta
        /// </summary>
        /// <param name="p">Objeto IPPorta a ser transformado</param>
        /// <returns></returns>
        public static string IP(this IPPorta p)
        {
            return new StringBuilder().AppendFormat("{0}.{1}.{2}.{3}", p.IP[0], p.IP[1],
p.IP[2], p.IP[3]).ToString();
        }
    }
    /// <summary>
    /// Estrutura contendo o IP e a porta
    /// </summary>
    public struct IPPorta
    {
        public byte[] IP;
        public int Porta;
    }
}
```

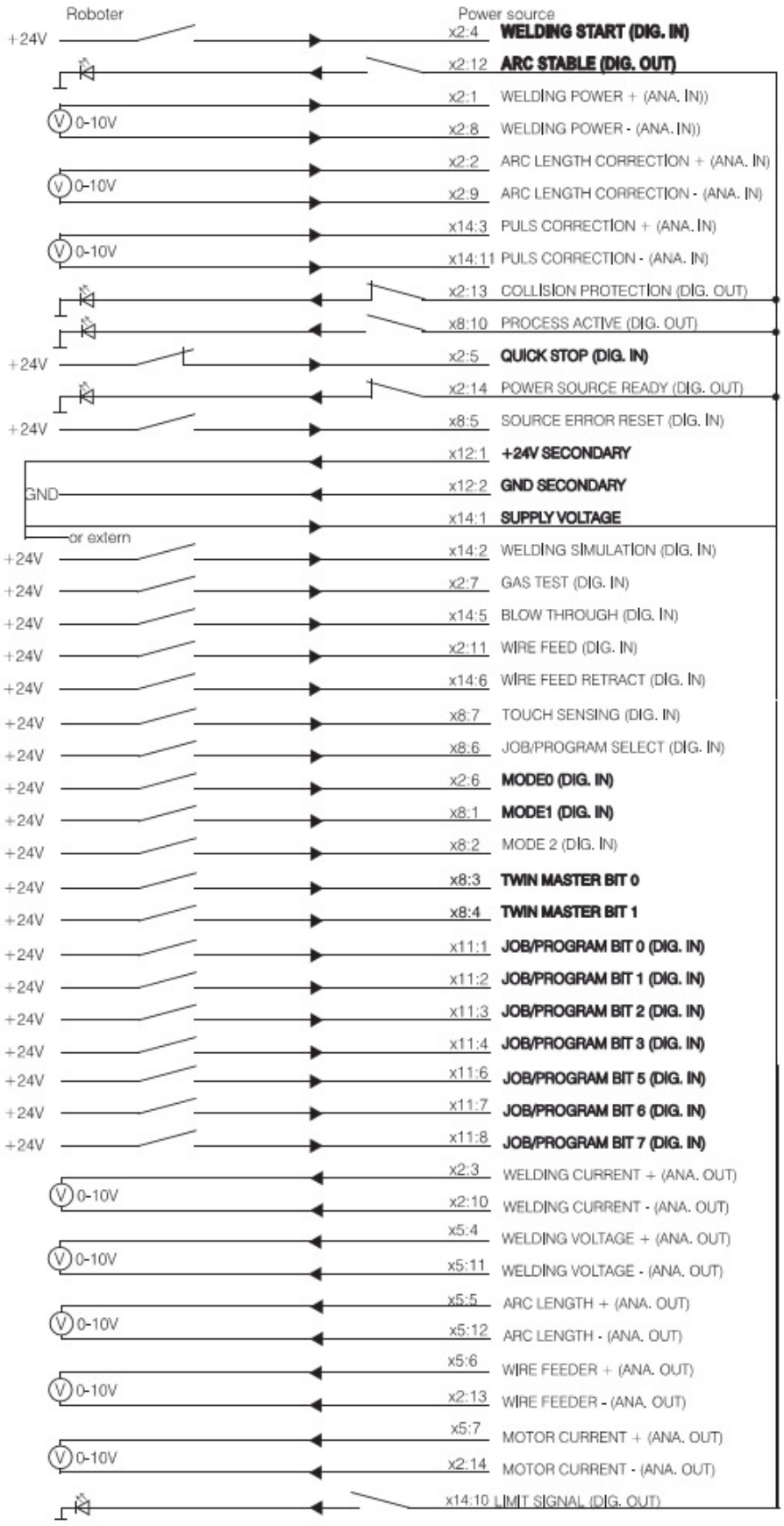
Welding*Socket*.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.Net;
using System.Threading;
using System.IO;

namespace Meu*Socket*
{
    public delegate void Conectado();
    public delegate void Ativado();
    public delegate void Disconectado();
    public delegate void Desativado();
    public delegate void Recebido(byte[] Dados);
    public class Welding*Socket*
    {
        enum Tipo*Socket*
        {
            NaoDefinido, Cliente, Servidor
        }
        private Tipo*Socket* Tipo = Tipo*Socket*.NaoDefinido;
        private TcpClient Cliente;
        private TcpListener Servidor;
        private Thread ServidorThread;
        private Thread ClienteThread;
        private NetworkStream WeldingStream;
        public bool Conectado = false;
        public bool? Ativo = null;
        //Delegates
        public Conectado DelegateConectado;
        public Ativado DelegateAtivado;
        public Disconectado DelegateDisconectado;
        public Desativado DelegateDesativado;
        public Recebido DelegateRecebido;
        /// <summary>
        /// Conecta o Cliente TCP
        /// </summary>
        /// <param name="IP">Numero do IP do servidor, ex.: 192.168.0.100</param>
        /// <param name="Porta">Numero da porta do servidor, ex.: 8080</param>
        public void Conectar(string IP, int Porta)
        {
            Tipo = Tipo*Socket*.Cliente;
            Cliente = new TcpClient();
            Servidor = null;
            IPEndPoint pontoFinal = new IPEndPoint(IPAddress.Parse(IP), Porta);
            Cliente.Connect(pontoFinal);
            WeldingStream = Cliente.GetStream();

            Conectado = true;
            ClienteThread = new Thread(new ThreadStart(CuidarConexaoCliente));
            ClienteThread.Start();
        }
        public void Ativar(int Porta)
        {
            Ativo = true;
            this.Servidor = new TcpListener(IPAddress.Any, Porta);
            this.ServidorThread = new Thread(new ThreadStart(BuscaPorClientes));
            this.ServidorThread.Start();
        }
        private void BuscaPorClientes()
        {
            //this.Servidor.Start();
            //while (Ativo == true)
            //{
            //    Cliente = this.Servidor.AcceptTcpClient();
            //    ClienteThread = new Thread(new
ParameterizedThreadStart(CuidarConexaoCliente));
            //    ClienteThread.Start(Cliente);
            //}
        }
        private void CuidarConexaoCliente()
```

```
{
    //WeldingStream = Cliente.GetStream();
    try
    {
        while (Conectado == true)
        {
            int tamanho = Cliente.Client.Available;
            if (tamanho > 0)
            {
                byte[] dados = new byte[tamanho];
                int BytesLidos = Cliente.Client.Receive(dados);
                DelegateRecebido(dados.Take(BytesLidos).ToArray());
            }
        }
    }
    catch (SocketException ex)
    {
        Conectado = false;
        DelegateDisconectado();
    }
    try
    {
        Cliente.Close();
        ClienteThread.Abort();
    }
    catch { }
}
public void Enviar(byte[] Dados)
{
    try
    {
        switch (Tipo)
        {
            case TipoSocket.NaoDefinido:
                break;
            case TipoSocket.Cliente:
                if (Conectado)
                {
                    if (Cliente.Client.Connected)
                    {
                        Cliente.Client.Send(Dados);
                    }
                }
                break;
            case TipoSocket.Servidor:
                //if (Ativo == true)
                //{
                //    if (Servidor.Server.Connected)
                //    {
                //        if (ClienteThread.IsAlive)
                //        {
                //            if (Cliente.Connected)
                //            {
                //                Cliente.Client.Send(Dados);
                //            }
                //        }
                //    }
                //}
                break;
            default:
                break;
        }
    }
    catch (SocketException ex)
    {
        Conectado = false;
        DelegateDisconectado();
    }
}
public void Disconectar()
{
    Conectado = false;
    if (Tipo == TipoSocket.Cliente)
    {
        Cliente.Close();
        WeldingStream = null;
        DelegateDisconectado();
```

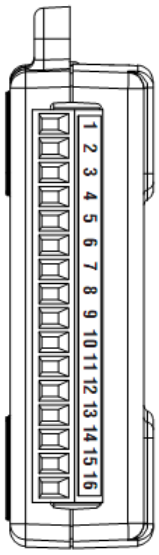
```
            }
        }
        public void Desativar()
        {
            //Ativo = false;
            //if (Tipo == TipoSocket.Servidor)
            //{
            //    if (ClienteThread.IsAlive)
            //    {
            //        ClienteThread.Abort();
            //    }

            //    Servidor.Stop();
            //    if (Servidor.Server.Connected)
            //    {
            //        Servidor.Server.Close();
            //        DelegateDisconectado();
            //    }
            //    DelegateDesativado();
            //}
        }
    }
}
```

ANEXO A – PINAGEM DA INTERFACE ROB 5000

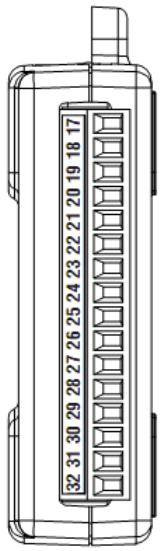


ANEXO B – NI DAQ 6009

Terminais analógicos

Module	Terminal	Signal, Single-Ended Mode	Signal, Differential Mode
	1	GND	GND
	2	AI 0	AI 0+
	3	AI 4	AI 0–
	4	GND	GND
	5	AI 1	AI 1+
	6	AI 5	AI 1–
	7	GND	GND
	8	AI 2	AI 2+
	9	AI 6	AI 2–
	10	GND	GND
	11	AI 3	AI 3+
	12	AI 7	AI 3–
	13	GND	GND
	14	AO 0	AO 0
	15	AO 1	AO 1
	16	GND	GND

Terminais digitais

Module	Terminal	Signal
	17	P0.0
	18	P0.1
	19	P0.2
	20	P0.3
	21	P0.4
	22	P0.5
	23	P0 6
	24	P0.7
	25	P1.0
	26	P1.1
	27	P1.2
	28	P1.3
	29	PFI 0
	30	+2.5 V
	31	+5 V
	32	GND