Universidade Federal
de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

RELATÓRIO DE ESTÁGIO INTEGRADO

RELATÓRIO DE ATIVIDADES DESENVOLVIDAS NO LABORATÓRIO DE AUTOMAÇÃO DO SENAI STENIO LOPES

Campina Grande, Paraíba
Janeiro de 2016

AUGUSTO JOSÉ SILVA FIRMO

RELATÓRIO DE ATIVIDADES DESENVOLVIDAS NO LABORATÓRIO DE AUTOMAÇÃO DO SENAI STENIO LOPES

Relatório de estágio integrado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Controle e Automação, Eletrônica de Potência

Orientador:
Professor Saulo Oliveira Dornellas Luiz, D. Sc.

Campina Grande, Paraíba
Janeiro de 2016

AUGUSTO JOSÉ SILVA FIRMO

RELATÓRIO DE ATIVIDADES DESENVOLVIDAS NO LABORATÓRIO DE AUTOMAÇÃO DO SENAI STENIO LOPES

Relatório de estágio integrado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Controle e Automação, Eletrônica de Potência

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Saulo Oliveira Dornellas Luiz, **D. Sc.**
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho a todos os amigos que me rodeiam, como também a todos meus familiares sempre presentes em minha vida, seja nas horas boas ou ruins.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, por todo esforço feito para que tivesse uma melhor educação.

A esta instituição por me acolher e proporcionar meu crescimento intelectual e civil.

Aos amigos de curso, pelas incontáveis noites de estudos, extremamente necessárias para que conseguisse chegar aonde estou, e pelas risadas sempre necessárias para o abrandamento do corpo e da mente.

“Little darling, the smiles returning to the faces
Little darling, it seems like years since it's been here
Here comes the sun, here comes the sun
And I say it's all right”
The Beatles

“Remember when you were young?
You shone like the sun
Shine on, you crazy Diamond
Now there's a look in your eyes
Like black holes in the sky
Shine on, you crazy Diamond
You were caught in the crossfire
Of childhood and stardom,
Blown on the steel breeze
Come on you target
For faraway laughter;
Come on you stranger, you legend,
You martyr, and shine"
Pink Floyd

“Ph’ngluimglw’ nafh Cthulhu R’ylehwgah’naglfhtagn.”
H. P. Lovecraft

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Processos de automação, sejam em determinadas máquinas ou em processos industriais completos, possuem o objetivo de aumentar sua eficiência, produção e confiabilidade, reduzindo o consumo de energia, matérias primas e o esforço humano.

Nestes processos, são aplicados computadores ou dispositivos capazes de realizar operações lógicas, como controladores lógicos programáveis (CLP), microcontroladores, sistemas digitais de controle distribuído (SDCD) ou controle numérico computadorizado (CNC). Além também do uso de sistemas de supervisão, permitindo assim o compartilhamento de dados importantes dos processos, trazendo maior agilidade e confiabilidade em decisões a serem tomadas sobre os mesmos.

O estágio foi realizado no Senai Stenio Lopes, localizado em Campina Grande. Tendo início em 02 de fevereiro de 2015 e término em 01 de outubro de 2015. A unidade além de conter diversos cursos técnicos, também presta serviços a empresas externas, sendo o laboratório de automação um dos prestadores. O estágio foi realizado particularmente neste laboratório.

No estágio foram desenvolvidos projetos na área de automação e controle de processos, usando tecnologias como microcontroladores e *softwares* de supervisório para criação de IHM's (*Interface* Homem Máquina).

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados conceitos e definições abrangentes aos temas principais deste trabalho, tais como descrições de *softwares* e *hardwares* usados e desenvolvidos e tipos de comunicações entre estes.

2.1 FUNDAMENTAÇÃO PARA A PRIMEIRA ATIVIDADE:

Aqui serão apresentados conceitos e definições necessárias para o desenvolvimento da primeira atividade.

2.1.1 *SOFTWARE* SUPERVISÓRIO ELIPSE E3:

A Elipse *Software* é uma empresa brasileira com foco em produção de *software* de supervisão para a automação industrial, criada em Porto Alegre no início dos anos 90. Em 2000 começa o desenvolvimento do Elipse E3, sendo este a terceira geração de *software* de supervisório da empresa. Deste seu lançamento em 2001, este supervisório vem sendo usado em diversos tipos de sistemas, como Centros de Operações de empresas elétricas, plantas industriais de diversas finalidades, sistemas de telemedição e controle de energia, automação e controle predial, mineração, entre outros (ELIPSE SOFTWARE, 2014).

O Elipse E3 é composto por quatro módulos principais. O E3 *Server*, um servidor de aplicações, onde os principais processos são executados, incluindo a comunicação em tempo real com os equipamentos de controle. Também responsável por enviar dados e telas aos clientes conectados em rede. O E3 *Studio*, uma ferramenta de configuração das aplicações do E3, agindo como a plataforma de desenvolvimento do supervisório. Inclui editores gráficos e de *scripts* (VBScript), permite também que um projeto seja editado por várias pessoas ao mesmo tempo, ou que vários E3 *Studios* estejam conectados ao mesmo servidor remoto, com múltiplas configurações. O E3 *Viewer*, este permite operar as aplicações residentes no servidor em qualquer computador com o programa executável *Viewer*. Não é necessário instalar a aplicação na máquina cliente, pois todos os componentes (telas, bibliotecas, bancos de dados, etc.)

serão baixados e registrados automaticamente. E o E3 *Admin,* este é o módulo responsável pela interface do E3 *Server* e de outros módulos do E3 com o usuário. Através dele o mesmo pode enviar comandos ao E3 *Server*, utilizando o ícone na área de notificação do *Windows* e controlar o domínio pela linha de comando (ELIPSE SOFTWARE, 2014).

2.1.2 PROTOCOLO DE COMUNICAÇÃO MODBUS:

O protocolo Modbus é uma estrutura de mensagem aberta desenvolvida pela Modicon em 1979 e é utilizada para a comunicação em dispositivos do tipo mestre/escravo ou cliente/servidor. É um dos protocolos mais usados em equipamentos industriais e devido às suas características, este protocolo também está presente em aplicações como instrumentos e equipamentos de laboratório, automação residencial, automação automobilísticas, etc. (SOUZA, 2009).

Em suas aplicações, tal protocolo pode ser implementado nos padrões RS-232, RS-485 ou Ethernet TCP/IP. A velocidade de comunicação é variada de acordo com cada um desses padrões, como também o comprimento máximo dos cabos da rede e o número de dispositivos conectados.

O padrão RS-232 é usado apenas para a comunicação do tipo ponto a ponto, permitindo assim somente dois dispositivos na rede, sendo obrigatoriamente um mestre e um escravo. A velocidade máxima deste padrão é de aproximadamente 115 kbps e a distância máxima entre os dispositivos da rede é em torno de 30 m.

O padrão RS-485 vem como uma clara evolução do padrão RS-232, permitindo a comunicação entre até 32 dispositivos, com velocidades podendo chegar em torno de 12 Mbps e com distância entre dispositivos de até 1200 m, sendo sem dúvida um dos padrões mais utilizados na indústria.

O padrão Ethernet TCP/IP industrial é bem semelhante ao normal, porém sendo implementada de maneira mais robusta, para suportar o ambiente de fábricas. Podem usar *hubs, switches* ou roteadores para compor sua rede. Pode chegar a até 100 Mbps com distâncias de 100 m ou 200 m.

Existem dois modos de transmissão do protocolo Modbus, ASCII (American Standard Code for Information Interchange) ou RTU (Remote Terminal Unit). Estes definem a forma como os *bytes* da mensagem serão transmitidos e como será o empacotamento da mensagem, como também sua descompactação. Não é possível usar

dois modos de transmissão em uma mesma rede Modbus. Geralmente o modo de transmissão é selecionado na etapa de configuração dos equipamentos, porém é muito comum que dispositivos só utilizem um tipo de modo de transmissão, como alguns CLP's ou inversores de frequência que usam somente o modo RTU (FREITAS, 2014).

O protocolo Modbus possui ao todo 256 endereços. Onde 0 é o endereço de *broadcast*, ou seja, quando o mestre envia uma mensagem para o endereço 0, todos os escravos a recebem. Os endereços de 1 a 247 são destinados a escravos e dos endereços 248 a 255 são endereços reservas para aplicações implementadas particularmente para a rede em questão. O mestre não possui endereço, somente os escravos.

O mestre especifica o tipo de função solicitada ao escravo por meio de um código de função, usada para acessar um tipo especifico de dado. Na tabela 1 podem ser vistos os códigos existentes para o protocolo Modbus (FREITAS, 2014).

Tabela 1: Funções do Modbus.

Código da função	Descrição
1	Leitura de bloco de bits do tipo *coil* (saída discreta).
2	Leitura de bloco de bits do tipo entradas discretas.
3	Leitura de bloco de registradores do tipo *holding*.
4	Leitura de bloco de registradores do tipo *input.*
5	Escrita em um único bit do tipo *coil* (saída discreta).
6	Escrita em um único registrador do tipo *holding*.
7	Ler o conteúdo de 8 estados de exceção.
8	Prover uma série de testes para verificação da comunicação e erros internos.
11	Obter o contador de eventos.
12	Obter um relatório de eventos.
15	Escrita em bloco de bits do tipo *coil* (saída discreta).
16	Escrita em bloco de registradores do tipo *holding*.
17	Ler algumas informações do dispositivo.
20	Ler informações de um arquivo.
21	Escrever informações em um arquivo.
22	Modificar o conteúdo de registradores de espera por

	meio de operações lógicas.
23	Combina ler e escrever em registradores numa única transação.
24	Ler o conteúdo da fila FIFO de registradores.
43	Identificação do modelo do dispositivo.

No tipo de comunicação Modbus usando ASCII, cada byte de mensagem é enviado como dois caracteres ASCII. Os dispositivos escravos monitoram a rede constantemente para o início da mensagem, quando esta é enviada pelo mestre, todos os dispositivos da rede decodificam o campo de endereço para determinar qual escravo deve receber a mensagem. O início da mensagem é reconhecido pelo caractere (:). O formato para cada mensagem no modo ASCII é:

- 1 bit de início;
- 7 bits de dados, começando pelo menos significativo;
- 1 bit de paridade par/ímpar, ou sem bit de paridade;
- 1 bit de parada ou 2 bits caso não se use paridade.

Intervalos de até um segundo podem decorrer entre os caracteres dentro da mensagem. Caso ocorra um intervalo maior, o dispositivo assume a ocorrência de erro. O campo de checagem de erros é baseado no método LRC (Longitudinal Redundancy Check). Este método possui 16 bits para a checagem de erro, sendo dividido como:

- 1 bit de *start*;
- 2 bits de endereço;
- 2 bits de função;
- 7 bits de dados, começando pelo menos significativo;
- 2 bits para checagem de erro LRC;
- 2 bits para fim de mensagem.

No modo de transmissão RTU não existe um caractere específico para a indicação do fim ou começo de uma mensagem. Esta identificação é feita pela ausência de transmissão de dados na rede, por um tempo mínimo de 3,5 vezes o tempo de transmissão de um byte de dados. Sendo assim, caso uma mensagem seja iniciada após a decorrência deste tempo mínimo, os elementos da rede irão assumir que o próximo caractere recebido é de uma nova mensagem. Da mesma maneira, os elementos da rede irão assumir o fim de uma mensagem caso esse tempo decorra após o recebimento de dados (FREITAS, 2014)

O campo de checagem de erros do modo RTU é baseado no método CRC (Cyclical Redundancy Checking). Este é dividido da seguinte maneira:

- 8 bits de endereço;
- 8 bits para a determinação da função;
- 8 bits de dados;
- 16 bits para a checagem de erro CRC.

O modo de transmissão TCP/IP utiliza a pilha TCP/IP para a comunicação e adiciona ao quadro Modbus um cabeçalho chamado MBAP (Modbus Application Protocol). Tendo seu modelo de transmissão dividido da seguinte maneira:

- 7 bytes de cabeçalho MBAP, sendo:
- 2 bytes para identificação da resposta para a transação;
- 2 bytes para a identificação do protocolo usado;
- 2 bytes para a contagem de todos os bytes;
- 1 byte para identificação de escravos.
- 8 bits para determinação da função;
- 8 bits de dados.

2.1.3 PLACA ARDUINO

Arduino é uma plataforma para prototipagem eletrônica que utiliza o conceito de hardware livre. Neste conceito deve-se prover a disponibilização irrestrita de informações sobre o projeto de hardware, tais como digramas eletrônicos, estrutura de produtos, layout de placas de circuito impresso, rotinas de baixo nível e qualquer outra informação necessária para uma construção from-scratch do projeto.

Os modelos mais utilizados da placa Arduino (Duemilanove e Uno) foram concebidas com base em um microcontrolador Atmel AVR montado em placa única. Eles possuem suporte de entrada/saída embutidos. A linguagem de programação padrão é baseada em C/C++.

A plataforma surgiu para que amadores, entusiastas e outras pessoas que não teriam acesso a controladores e ferramentas mais sofisticadas, pudessem criar seus projetos, tornando-os acessíveis, flexíveis e com baixo custo. Uma placa Arduino, em geral, possui os seguintes elementos em sua construção:

- Um microcontrolador
- Placa base com reguladores de tensão adequados ao microcontrolador.
- Linhas de Entrada/Saída digitais e analógicas.
- Linhas de comunicação serial.
- Interface USB para programação e interação com um computador hospedeiro.

Seguindo estas características comuns, ao decorrer da existência do projeto foram desenvolvidas diversas placas microcontroladoras, cada uma carrega um codinome e especificações de hardware próprias.

Placa Arduino Mega:

A placa Arduino Mega[1] é baseada no microcontrolador ATmega2560, possuindo 54 pinos de entrada/saída digitais (dos quais 15 podem ser utilizados como saída PWM), 16 entradas analógicas, cristal de cerâmica com 16 MHz de clock que define a frequência de operação do microcontrolador, conexão USB, pino de alimentação

[1] Disponível em: https://www.arduino.cc/en/Main/ArduinoBoardMega2560

externa padrão barrel e botão de reset. A placa possui todos os circuitos auxiliares para o funcionamento do microcontrolador, bastando ao usuário conectá-la a um computador via cabo USB para começar a utilizá-la. Além disso o processo de gravação de rotinas é facilitado por haver um bootloader que automatiza este passo.

Uma vista frontal da placa Arduino Mega é apresentada na Fig.1.

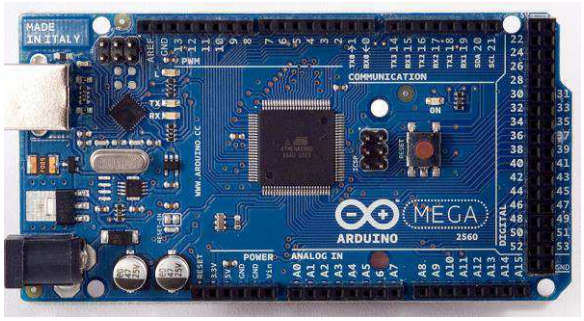

Figura 1:Vista frontal da placa Arduino Mega.

Programação:

A programação do microcontrolador ATmega2560 do Arduino Mega pode ser feita por meio da IDE Arduino, apresentada na Fig. 2. A linguagem de desenvolvimento é C/C++com recursos exclusivos desenvolvidos com base na linguagem Wiring. O microcontrolador vem com um bootloader pré-gravado, o que permite o upload de novos códigos com o pressionar de um botão, sem uso de hardware para programação externa, como no caso de gravadores de flash.

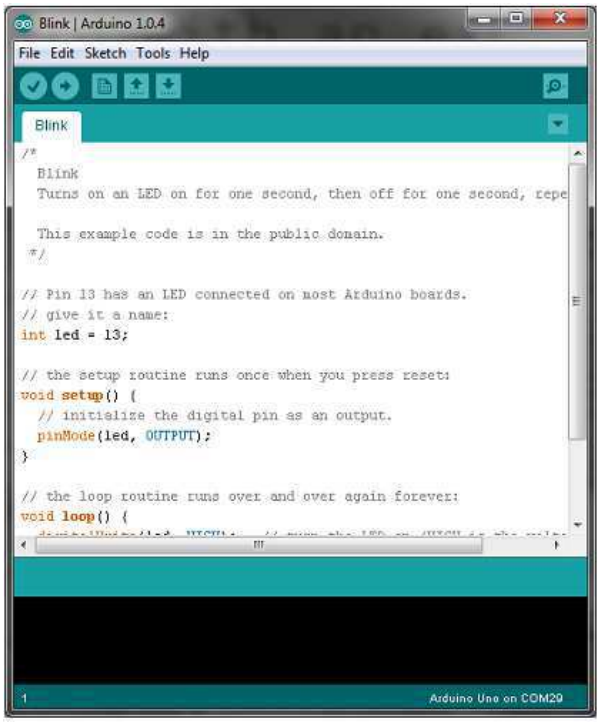

Figura 2: Ambiente de programação do Arduino

Sempre que é feita uma gravação de rotina no controlador, ocorrerá uma reinicialização do circuito, forçando a sobrescrita de dados na memória RAM. A reinicialização também pode ser feita por meio do botão reset ou mesmo de um circuito interligado ao pino RESET controlado via software.

2.2 FUNDAMENTAÇÃO PARA A SEGUNDA ATIVIDADE:

Aqui serão apresentados conceitos e definições necessárias para o desenvolvimento da segunda atividade.

2.2.1 *SOFTWARE* SUPERVISÓRIO ELIPSE MOBILE:

O Elipse Mobile é uma plataforma móvel para a integração com sistemas de automação, possibilitando monitorar indicadores e comandar equipamentos do seu processo. Possuindo interface clara, simples e objetiva, é possível agrupar seus dados em páginas com informações sobre cada uma das partes do projeto desenvolvido. Podendo ser acessados via Internet ou *smartphone*.

2.3 FUNDAMENTAÇÃO PARA A TERCEIRA ATIVIDADE:

Aqui serão apresentados conceitos e definições necessárias para o desenvolvimento da terceira atividade.

2.3.1 SISTEMAS DE CONTROLE DE MALHA FECHADA:

Um sistema que estabeleça uma relação de comparação entre a saída e a entrada de referência, utilizando a diferença como meio de controle, é denominado sistema de controle de malha fechada (OGATA, 2010).

Nestes sistemas, o sinal de erro atuante, é a diferença entre o sinal de entrada (ou referência) e o sinal de realimentação (que pode ser o próprio sinal de saída ou uma função do sinal de saída). Esse sinal de erro, realimenta o controlador, de modo a

minimizar o erro e mudar a saída do sistema para o valor desejado. Na Fig. 3 é apresentado o diagrama de blocos para um sistema de controle em malha fechada.

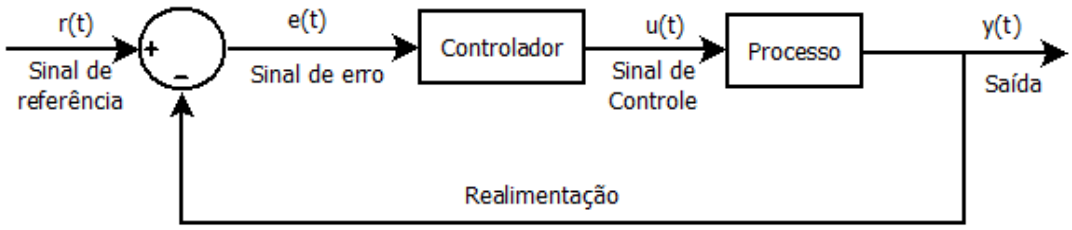

Figura 3: Diagrama de blocos de um sistema de controle em malha fechada.

Onde: r(t) é o sinal de referência;
e(t) é o sinal de erro;
u(t) é o sinal de controle;
y(t) é o sinal de saída.

2.3.2 SISTEMAS DE CONTROLE DIGITAIS:

Praticamente todos os sistemas de controle que são implementados hoje são baseados em sistemas computacionais. O diagrama de blocos correspondente a esse sistema é apresentado na Fig. 4.

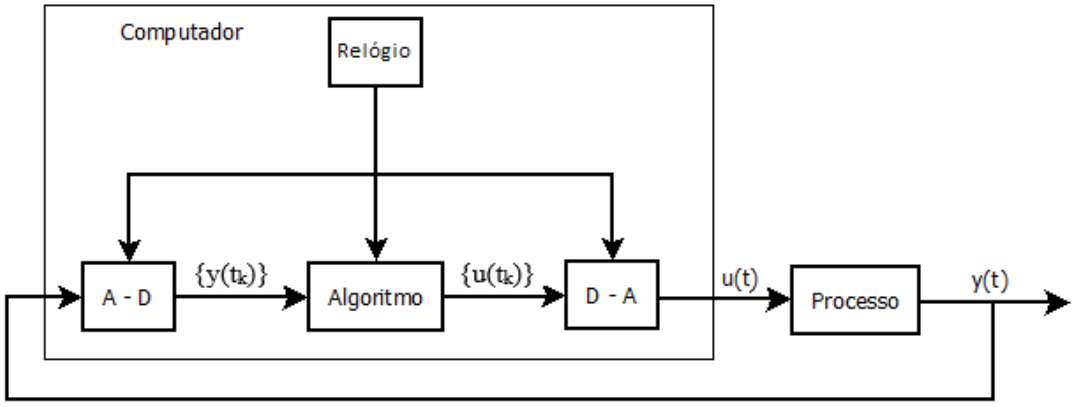

Figura 4: Diagrama de blocos de um sistema de controle computacional.

A saída do processo y(t) é um sinal contínuo no tempo. Esta saída é convertida para a forma digital pelo conversor analógico – digital (A - D). A conversão é feita no tempo de amostragem, t_k. O sinal convertido $y(t_k)$ é então processado pelo algoritmo feito no computador, gerando um sinal de controle discreto no tempo, $u(t_k)$. Este sinal é convertido para um sinal analógico no tempo pelo conversor digital – analógico (D - A),

gerando o sinal u(t), responsável pelo controle do processo. Os eventos são sincronizados por um relógio em tempo real no computador.

Os sistemas de controle computacionais, contêm ambos tipos de sinais, contínuos e discretos no tempo, sendo muitas vezes chamados de sistemas de dados amostrados. (ASTROM, 1997)

2.3.3 CONTROLE PID:

A utilidade dos controladores PID está na sua aplicabilidade geral à maioria dos sistemas de controle. Em particular, quando o modelo matemático da planta não é conhecido e, portanto, métodos de projeto analítico não podem ser utilizados, controles PID se mostram os mais úteis. Na área dos sistemas de controle de processos, sabe-se que os esquemas básicos de controle PID e os PID modificados provaram sua utilidade, conferindo um controle satisfatório (OGATA, 2010).

Os controladores PID são compostos por três ações:

- Controle Proporcional;
- Controle Integral;
- Controle Derivativo.

O conjunto destes três tipos de controle forma o controlador PID. Porém formas de controle usando somente um ou dois tipos de controle podem ser adotadas, sendo estas apresentadas a seguir.

Controle proporcional (P):

A ação proporcional apresenta um sinal de controle proporcional ao sinal de erro. Sendo assim, esta ação pode ser definida como a amplitude de correção proporcional à amplitude do desvio, como apresentada em (1).

$$u(t) = K_p e(t) \tag{1}$$

Onde K_p é o ganho proporcional.

Este tipo de controlador se torna limitado por não permitir na maioria dos processos, um erro de regime estacionário nulo.

Controle proporcional - integral (PI):

Com o objetivo de garantir erro nulo em regime permanente, recorre-se à ação integral. Assim, é usada a ação proporcional em conjunto com a ação integral. Sendo a saída deste controlador apresentada em (2).

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau) d\tau \tag{2}$$

Onde K_i é o ganho integral.

Este tipo de controlador se torna limitado quando se deseja um regime transitório rápido.

Controle proporcional -integral – derivativo (PID):

Para se evitar o regime transitório lento quando se aplica o controlador PI, torna-se necessário o uso da ação derivativa, fazendo com que o controle aja antecipando o processo e assim aplicando sua resposta mais rapidamente. O uso isolado desta ação de controle pode tornar o sistema instável. A lei de controle do controlador PID é apresentada em (3):

$$u(t) = K_p e(t) + K_i \int_{t_0}^{t} e(\tau) d\tau + K_D \frac{de(t)}{dt} \tag{3}$$

Onde K_D é o ganho derivativo.

Para a aplicação em sistemas digitais, torna-se necessário o desenvolvimento de controladores PID digitais. Para isso, é necessário o desenvolvimento de certos algoritmos para a substituição da integral e derivada do erro, para processos numéricos que possam ser resolvidos em sistemas digitais.

Controle PID digital:

A equação geral do PID digital pode ser obtida por meio da discretização da função de transferência do PID analógico apresentada em (3). Usando o tempo de amostragem como mostrado em (4).

$$t = nT \tag{4}$$

Onde T é o período de amostragem e n é um inteiro correspondendo ao número da amostra.

Sendo assim, obtém-se a equação discretizada, como apresentada em (5):

$$u(nT) = K_P e(nT) + K_i T \sum_{j=0}^{n} e(jT) + K_D \frac{e[nT] - e([n-1]T)}{T} \tag{5}$$

Normalizando o termo do período de amostragem T, obtemos a equação discretizada final, apresentada em (6).

$$u[n] = K_P e[n] + K_i T \sum_{j=0}^{n} e[j] + K_D \frac{e[n] - e[n-1]}{T} \tag{6}$$

2.3.4 RETIFICADORES DE ONDA:

Um retificador é um circuito que converte um sinal AC em um sinal unidirecional. Os diodos são extensivamente utilizados em retificadores. O retificador monofásico de meia-onda é o tipo mais simples e não usado em aplicações industriais, entretanto, seu estudo é útil na compreensão do princípio de operação do retificador. Este tipo de retificador e suas formas de onda são apresentadas na Fig. 5:

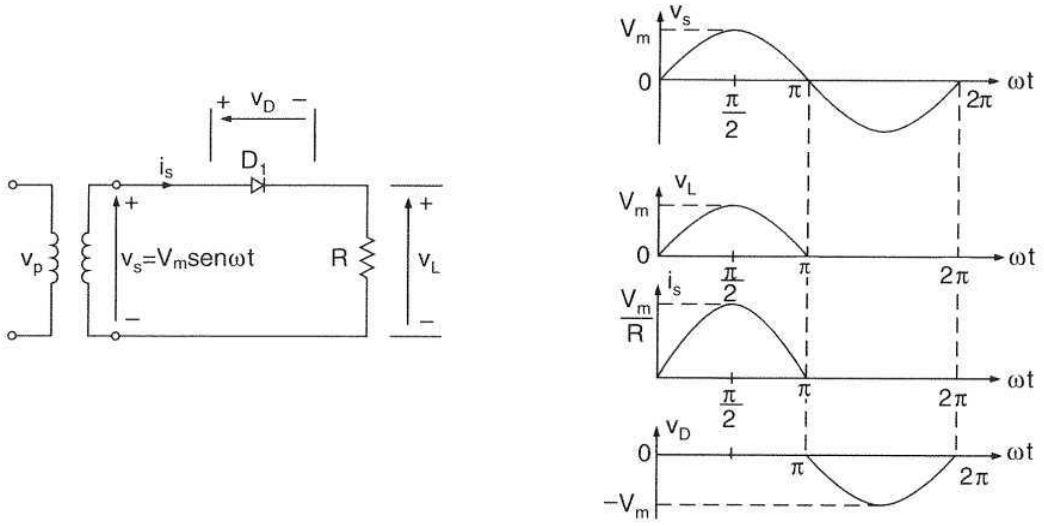

Figura 5:Diagrama do retificador de meia-onda e suas formas de onda.
Fonte: Eletrônica de potência: Circuitos, dispositivos e aplicações, Muhammad, Rashid,

Durante o semiciclo positivo da tensão de entrada, o diodo D_1 conduz e a tensão de entrada aparece sobre a carga. Durante o semiciclo negativo da tensão de entrada, o diodo está em corte e a tensão de saída é zero.

Já os retificadores de onda completa podem ser obtidos com o uso de quatro diodos, como apresentado na Fig. 6:

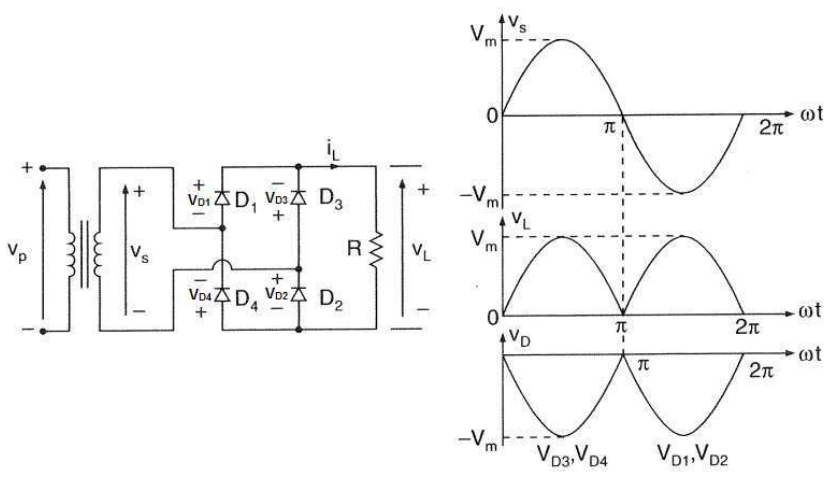

Figura 6: Diagrama do retificador de onda completo e suas formas de onda
Fonte: Eletrônica de potência: Circuitos, dispositivos e aplicações, Muhammad, Rashid,

Durante o semiciclo positivo da tensão de entrada, a potência é fornecida à carga através dos diodos D_1 e D_2. Durante o semiciclo negativo da tensão de entrada, a potência é entregue pelos diodos D_3 e D_4. Este tipo de circuito é conhecido como retificador em ponte e comumente utilizado em aplicações industriais.

2.3.5 CIRCUITO DETECTOR DE ZERO:

Para o uso de microcontroladores em conjunto com a rede elétrica, é necessário fazer uma sincronização entre ambos, de modo que os comandos gerados por um microcontrolador sejam feitos no momento desejado. Para isso, é possível utilizar o circuito apresentado na Fig. 7, fazendo com que o mesmo reconheça quando ocorre a passagem pelo zero na rede elétrica.

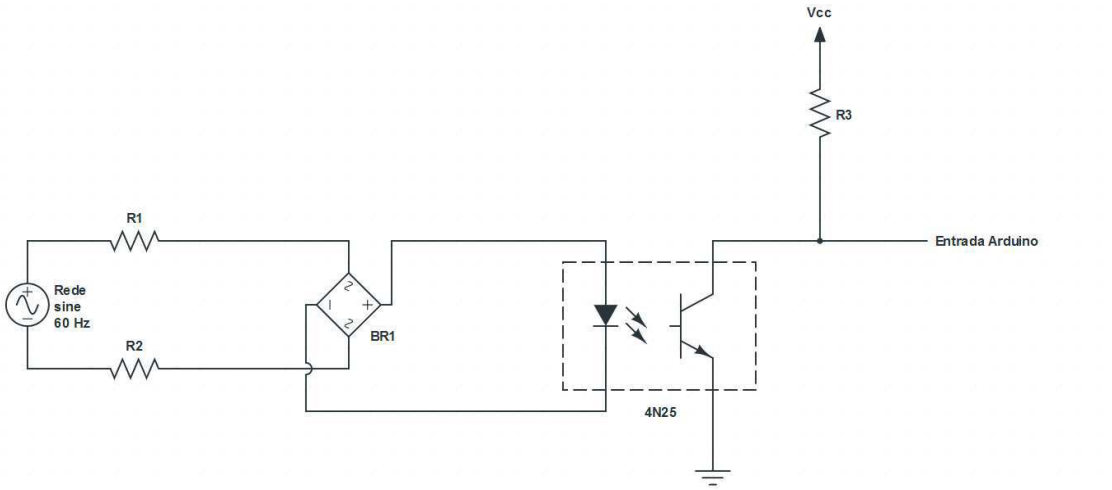

Figura 7: Diagrama do circuito detector de zero

No circuito acima, o componente BR1 é um retificador de onda completa em ponte, como mostrado anteriormente. O componente 4N25 é um optoacoplador de sentido único. O principal motivo do uso de optoacopladores aqui, é a isolação que este fornece entre o microcontrolador e a rede elétrica, prevenindo possíveis danos que possam ser causados. Os resistores R_1 e R_2 são para a limitação de tensão e corrente circulando pela parte esquerda do circuito. O resistor R_3 tem a função de *Pull-Up* evitando oscilações de tensão na entrada do microcontrolador, e assim prevenindo erros.

A forma de onda entregue ao led do optoacoplador é semelhante à forma de onda da carga na Fig.6, porém com um valor de pico bem menor, já que boa parte da tensão da rede encontra-se nos resistores R1 e R2. Enquanto o valor da tensão sobre o led é superior a zero, o mesmo está aceso, fazendo com que o transistor esteja em condução, ligando o Vcc ao terra através do resistor de *Pull-Up*, e levando o nível lógico zero ao microcontrolador. Quando ocorre a passagem pelo zero, o led apaga, fazendo com que o transistor pare de conduzir, e o Vcc então é ligado à entrada do microcontrolador através do resistor de *Pull-Up*, levando o nível lógico um ao microcontrolador. Como essa passagem ocorre em um tempo muito curto, o nível alto

se manifesta como um pulso no microcontrolador, devendo este ter uma interrupção no seu código para o reconhecimento do mesmo, e assim, efetuando a sincronização entre a tensão da rede e a lógica do microcontrolador.

2.3.6 CONTROLE DE POTÊNCIA EM CORRENTE ALTERNADA:

O fluxo de potência sobre uma carga AC pode ser controlado através da variação do valor eficaz da tensão AC aplicada à carga. As aplicações mais comuns para este tipo de controle são aquecimento industrial, mudança de derivação de transformadores sob cargas, controle de iluminação, controle de velocidade de máquinas de indução polifásicas e controle de eletroímãs AC.

Existem dois tipos de controle em cargas AC. Controle liga-desliga, ou controle por ângulo de fase. Neste trabalho foi adotado o controle por ângulo de fase e este será explicado com mais detalhes.

No controle por ângulo de fase, a tensão AC entregue à carga é controlada através de disparos em chaves, sendo normalmente usadas para este tipo de ação tiristores. Este tipo de controlador é apresentado na Fig 8:

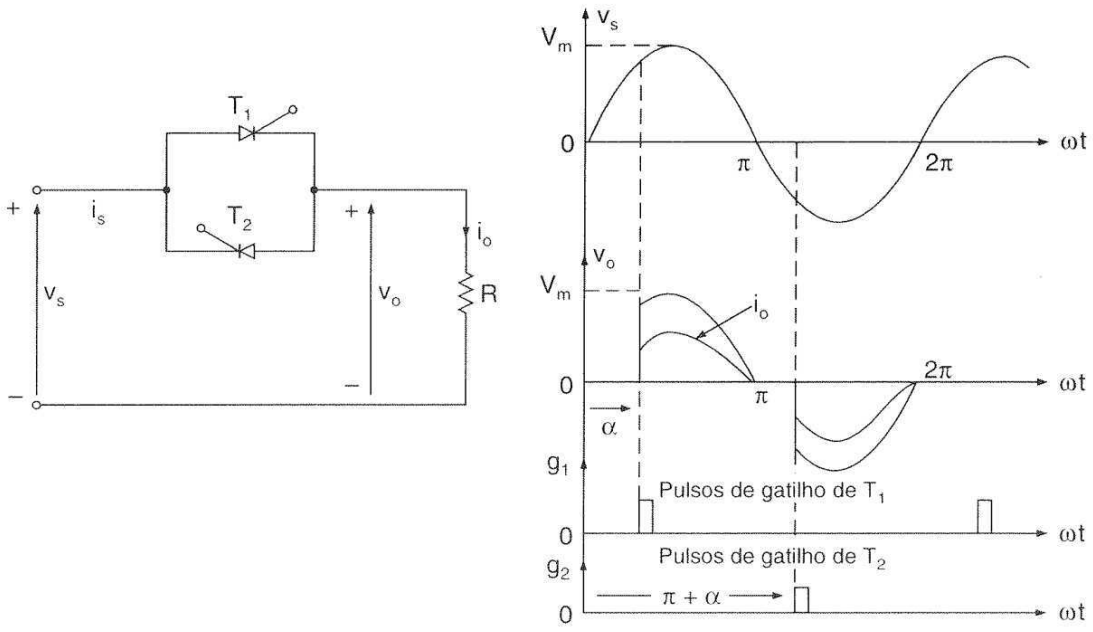

Figura 8: Diagrama do controle por ângulo de fase.
Fonte: Eletrônica de potência: Circuitos, dispositivos e aplicações, Muhammad, Rashid,

Durante o semiciclo positivo da tensão de entrada, o fluxo de potência é controlado através do ângulo de disparo do tiristor T_1, e no semiciclo negativo pelo tiristor T_2. Os pulsos de disparo dos tiristores são defasados de 180°.

A tensão eficaz de saída pode ser obtida da seguinte forma:

$$V_{RMS} = \left[\frac{1}{T}\int_0^T v_O(t)^2 d(t)\right]^{1/2} \tag{7}$$

Onde: V_{RMS} é a tensão eficaz de saída;

v_O é a tensão de saída;

T é o período da tensão de saída.

Sendo a tensão de entrada $v_s = \sqrt{2}V_s sen(\omega t)$ e $\alpha_1 = \alpha_2 = \alpha$, e aplicando em (7), temos:

$$V_O = \left[\left\{\frac{2}{2\pi}\int_\alpha^\pi 2\,{V_s}^2 sen^2(\omega t) d(\omega t)\right\}\right]^{1/2} \tag{8}$$

$$V_O = \left[\frac{4{V_s}^2}{4\pi}\int_\alpha^\pi (1-\cos(2\omega t))\, d(\omega t)\right]^{1/2} \tag{9}$$

$$V_O = V_s\left[\frac{4}{\pi}\left(\pi - \alpha + \frac{sen(2\alpha)}{2}\right)\right]^{1/2} \tag{10}$$

Então, variando-se α de 0 a π em (10), a tensão eficaz na saída poderá variar de V_S a 0.

Porém, para cargas indutivas a corrente do tiristor T_1 não cai a zero em $\omega t = \pi$, quando a tensão de entrada começa a ficar negativa. O tiristor continua a conduzir até que sua corrente caia a zero em $\omega t = \beta$. E assim o ângulo de condução do tiristor T_1 será $\delta = \beta - \alpha$ e depende do ângulo de disparo α e do ângulo do fator de potência da carga θ. As formas de onda para a corrente no tiristor e tensão de entrada são apresentados na Fig 9.

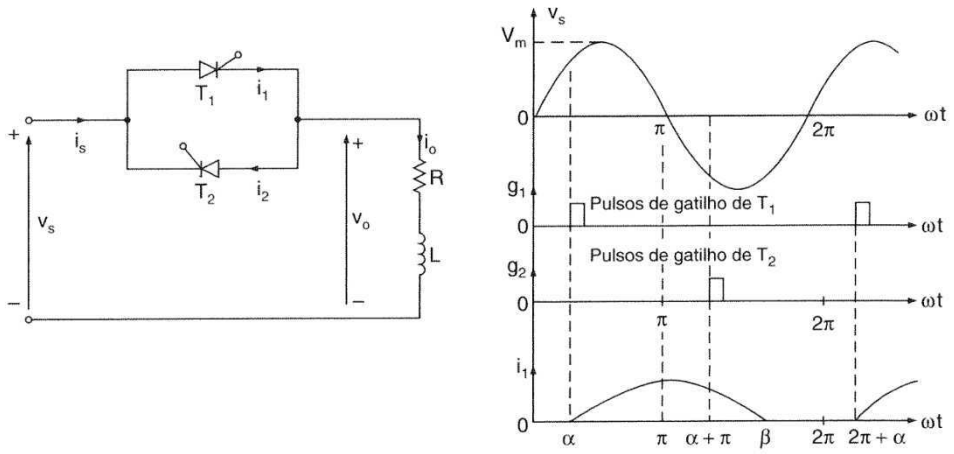

Figura 9: Diagrama do controle por ângulo de fase com carga indutiva.
Fonte: Eletrônica de potência: Circuitos, dispositivos e aplicações, Muhammad, Rashid,

Sendo: $v_s = \sqrt{2}V_s sen(\omega t)$, a corrente i_1 do tiristor T_1 pode ser encontrada a partir de:

$$L\frac{di_1}{dt} + Ri_1 = \sqrt{2}V_s sen(\omega t) \tag{11}$$

Onde: L é a indutância da carga;

R é a resistência da carga.

Sendo a solução de (11):

$$i_1 = \frac{\sqrt{2}V_s}{Z} sen(\omega t - \theta) + A_1 e^{-(R/L)t} \tag{12}$$

Onde a impedância da carga é $Z = [R^2 + (\omega L)^2]^{1/2}$ e o ângulo do fator de potência da carga é $\theta = tan^{-1}(\omega L/R)$.

A constante A_1 pode ser determinada usando como condições iniciais, $\omega t = \alpha$ e $i_1 = 0$. A partir (12), temos:

$$A_1 = -\frac{\sqrt{2}V_s}{Z} sen(\alpha - \theta) e^{(R/L)(\alpha/\omega)} \tag{13}$$

Substituindo o valor de A_1 de (13) em (12), teremos:

$$i_1 = \frac{\sqrt{2}V_s}{Z}\left[sen(\omega t - \theta) - sen(\alpha - \theta) e^{(R/L)(\frac{\alpha}{\omega} - t)}\right] \tag{14}$$

A equação apresentada em (14) se faz importante para a determinação do ângulo β, que é quando a corrente i_1 cai para zero e o tiristor T_1 é desligado. Usando a condição inicial $i_1(\omega t = \beta) = 0$, podemos fazer:

$$sen(\beta - \theta) = sen(\alpha - \theta) e^{(R/L)(\alpha - \beta)/\omega} \tag{15}$$

O ângulo β também é chamado de ângulo de extinção, e pode ser determinado a partir de (15) através de métodos numéricos. Uma vez que β tenha sido determinado, o ângulo de condução do tiristor T_1 pode ser determinado como:

$$\delta = \beta - \alpha \tag{16}$$

A partir de (7), pode-se determinar a tensão eficaz de saída como:

$$V_O = \left[\left\{\frac{2}{2\pi}\int_{\alpha}^{\beta} 2\,{V_s}^2 sen^2(\omega t)d(\omega t)\right\}\right]^{1/2} \tag{17}$$

$$V_O = \left[\frac{4{V_s}^2}{4\pi}\int_{\alpha}^{\beta}(1-\cos(2\omega t))\,d(\omega t)\right]^{1/2} \tag{18}$$

$$V_O = V_s\left[\frac{4}{\pi}\left(\beta - \alpha + \frac{sen(2\alpha)}{2} - \frac{sen(2\beta)}{2}\right)\right]^{1/2} \tag{19}$$

Na prática, como o ângulo de condução, δ, não pode exceder a π e a corrente de carga tem que passar por zero, o ângulo de disparo α não pode ser menor que θ, sendo assim, a faixa de controle do ângulo de disparo é:

$$\theta \leq \alpha \leq \pi \tag{20}$$

No controle de máquinas elétricas, os métodos apresentados aqui são recomendados para aplicações em baixa potência, como resfriadores por ventiladores, sopradores térmicos, aquecedores, bombas hidráulicas, etc. Isto se dá principalmente pelo conteúdo harmônico gerado ser alto.

2.3.7 CHAVES TIPO TRIAC:

Como mostrado, para o controle da potência proposto, é necessário o uso de dois tiristores ligados em antiparalelo, este tipo de configuração pode ser encontrado em componentes tipo TRIAC. Seu diagrama interno é apresentado na Fig.10.

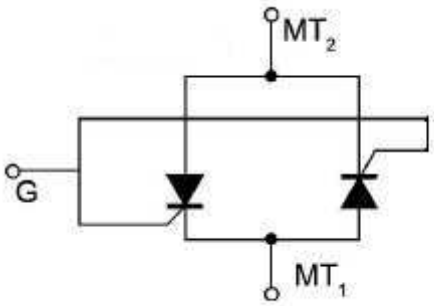

Figura 10: Diagrama interno do TRIAC.
Fonte: Datasheet do componente.

A principal vantagem do uso de TRIAC's no lugar de pares de tiristores está na lógica de disparo mais simples, pois ambos tiristores possuem o mesmo terminal *Gate*. Outras vantagens são a redução no preço, e redução de espaço em placas de circuito impresso.

Para a mudança entre o estado de bloqueio e o estado de condução do TRIAC pode se dar tanto com o terminal MT_1 positivo em relação ao terminal MT_2, como na situação inversa. Os pulsos de disparo no *Gate* podem ser positivos ou negativos. Para que o TRIAC continue em seu estado de condução, uma vez suprimido o sinal de disparo do mesmo, é necessário que a corrente que atravessa o mesmo seja superior à sua corrente de manutenção.

A curva característica de um TRIAC é apresentada na Fig 11. Supondo que não tenha sido aplicado nenhum pulso de tensão no *Gate*. Caso aumente-se a tensão entre MT_1 e MT_2, ocorrerá a existência de uma pequena corrente de fuga. Caso esta tensão aumente até ultrapassar o limite de V_{RO} o TRIAC entrará na região de ruptura, e assim para o estado de condução (baixa impedância interna), mantendo-se assim até que a corrente através do mesmo seja menor que a corrente de manutenção. Caso a polaridade da tensão nos terminais do TRIAC seja invertida, o mesmo irá funcionar no terceiro quadrante, com uma característica inteiramente simétrica. Com a aplicação de um pulso de tensão no *Gate* o TRIAC irá entrar na característica de condução até que a corrente que passa pelo mesmo seja inferior ao valor da corrente de manutenção.

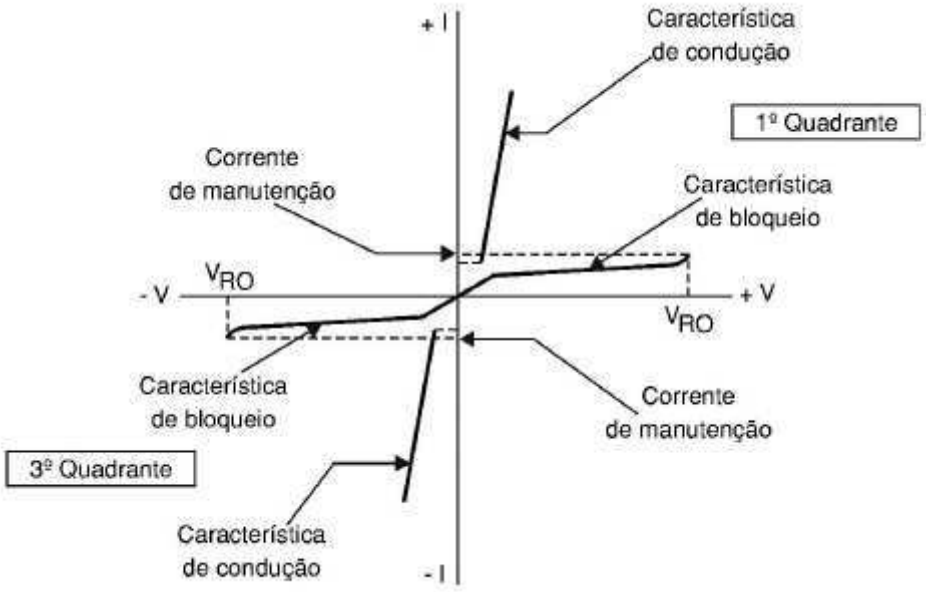

Figura 11: Curva característica do TRIAC.
Fonte: Datasheet do componente.

2.3.8 CIRCUITO DE PROTEÇÃO TIPO SNUBBER:

Snubbers são pequenos circuitos com o intuito de amortecer os transientes que ocorrem na comutação de cargas. Estes transientes podem além de forçar o dispositivo de comutação, causar sua queima, como também danos à carga. Em cargas indutivas, tais circuitos têm também a função de absorver a energia gerada na sua comutação.

Um dos tipos mais comuns destes circuitos é o *Snubber* RC, formado por um circuito RC série ligado em paralelo à chave comutadora. Este é apresentado na Fig. 12:

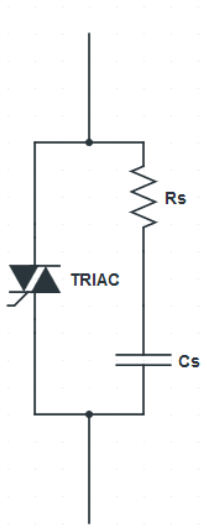

Figura 12: Diagrama do circuito Snubber RC.

Onde os valores de R_s e C_s podem ser obtidos através de (21) e (22):

$$R_s = \frac{V_{RMS}}{I_{máx}} \tag{21}$$

$$C_s = \frac{1}{{V_{RMS}}^2 . f} \tag{22}$$

Onde: V_{RMS} é a tensão sobre o comutador durante o período de bloqueio;

$I_{máx}$ é a máxima corrente suportada pelo comutador;

f é a frequência de comutação.

3 LISTA DE ATIVIDADES DESENVOLVIDAS

Neste capítulo serão apresentadas, de maneira sucinta, informações relativas ao trabalho de estágio desenvolvido pelo autor.

3.1 DESENVOLVIMENTO DE COMUNICAÇÃO SERIAL MODBUS ENTRE ELIPSE E3 E ARDUINO:

Para a configuração de Modbus escravo para o Arduino, é necessário o uso de bibliotecas adicionais que cubram essa função. Como tudo que envolve Arduino é de código aberto, existem bibliotecas para esse intuito criadas por terceiros. Muitas destas não contemplam tudo que deveria ser usado para a comunicação com o Elipse E3. Sendo assim, foram necessários testes de muitas bibliotecas ModbusSlave até encontrar a que oferecesse todos os requisitos necessários para a realização dos testes pedidos. Muitas delas não permitiam a escrita nos registradores pelo mestre, outras não possuíam formas de instanciação destes registradores em código para implementação de lógicas usando seus valores, ou os registradores com endereços acima de 40000 eram somente habilitados para leitura de portas digitais, deixando a escrita destas ou mesmo o uso das portas analógicas ou de PWM (pulse width modulation) impossíveis.

A biblioteca encontrada que atendia a todos os requisitos com poucas alterações foi a fornecida pelo jpmzometa[2]. Com ela foi possível o desenvolvimento de um código teste para a implementação das tarefas solicitadas e configuração de escravo para o Arduino.

O código da biblioteca encontra-se na primeira parte do Anexo A. Este foi dividido em blocos para o melhor entendimento:

- Bloco 1: Aqui é instanciado o pino Txenpin (seu valor é definido em código). Os endereços 0 ou 1 são definidos para redes RS–232. Os

[2] Disponível em: https://sites.google.com/site/jpmzometa/arduino-mbrt/arduino-modbus-slave

demais valores, são definidos para redes RS - 485. Também são instanciadas as funções de modbus, sendo estas, ler um bloco de registradores, escrever um único registrador e escrever em um bloco de registradores.

- Bloco 2: Aqui são definidas constantes usadas nas funções suportadas apresentadas no bloco 1.
- Bloco 3: Implementação do CRC (Cyclic Redundance Check), contendo as entradas *buf* (matriz contendo a mensagem a ser enviada para o controlador mestre), *start* (início do *loop* no crc do contador, normalmente 0), *cnt* (quantidade de *bytes* na mensagem a ser enviada para o controlador mestre). A saída *temp* (retorna o *byte* crc para a mensagem).
- Bloco 4: As funções implementadas nesse bloco constroem o pacote de consulta Modbus, sendo estes de leitura de bloco, escrita em bloco e escrita única. Também é construído um bloco de exceção, para a determinação de erros.
- Bloco 5: Implementação para adicionar uma soma de verificação para o fim de um pacote. Sendo assim tem-se a confirmação do envio do pacote de informações.
- Bloco 6: Implementa uma função que retorna o número total de caracteres enviados do escravo para o mestre. Este retorno é enviado para o mestre Modbus.
- Bloco 7: Função para monitorar um pedido do mestre Modbus. Retorna o número total de caracteres recebidos ou uma palavra negativa em caso de falha.
- Bloco 8: Esta função retorna uma palavra positiva quando o pedido está correto, caso ocorra erro, retorna uma palavra negativa.
- Bloco 9: Função para verificar se o pedido pode ser processado pelo escravo. Retorna 0, caso positivo ou uma palavra negativa em caso de erro.
- Bloco 10: Função para escrever nos registradores do escravo e retorna o número de registradores escritos.

- Bloco 11: Função para a escrita de dados na matriz de registradores do escravo.
- Bloco 12: Função para escrever um único valor inteiro em um único registrador do escravo.
- Bloco 13: Lê os registradores do escravo e envia para o mestre Modbus.

O código desenvolvido em Arduino para a execução das funções pedidas pode ser encontrado na segunda parte do Anexo A, que também foi dividido em blocos, sendo estes apresentados a seguir:

- Bloco 01: Aqui é incluída a biblioteca ModbusSlave.h, como também é declarado a função Modbus responsável por receber os parâmetros de configuração de comunicação, sendo neste exemplo chamada de elipse_E3.
- Bloco 02: Aqui são declarados o número de registradores que serão usados, como também é declarado o número total de registradores.
- Bloco 03: Declaração da variável para o *watchdog*, que irá implementar o tempo de espera sem informação para que ocorra a separação da mensagem.
- Bloco 04: Declaração das variáveis:
  SLAVE: para a definição do endereço de escravo.
  BAUD: para o baudrate em bps.
  PARITY: para o tipo de paridade, sendo definido “n” para sem paridade, “e” para paridade par e “o” para paridade ímpar.
  TXENPIN: para o tipo de topologia usada, 0 ou 1 desabilita essa função (uso de nenhum escravo), valores maior ou igual a 2 para topologia de ponto ou multiponto (um escravo ou vários escravos).
- Bloco 05: Configuração da função Modbus para o recebimento dos parâmetros de comunicação como também o recebimento dos registradores.
- Bloco 06: Declaração dos pinos do Arduino como entrada ou saída e também seu estado inicial.
- Bloco 07: Atualização dos parâmetros recebidos pela função Modbus e da implementação do *watchdog* para a separação das mensagens.

- Bloco 08: Implementação da lógica usada para implementação das funções desejadas.

Para a comunicação serial no Elipse E3, é utilizado o driver ModiconModbus Master, disponível para *download* no próprio site da Elipse. Além de portas seriais, este driver proporciona acesso padrão do nível físico e proporciona interfaces para ethernet (via TCP/IP ou UDP/IP), modem (através de TAPI) e RAS (Remote Access Server).

A configuração do driver utiliza recursos como independência de nível físico, geração de logs, configuração off-line e gerenciamento de conexões.

O primeiro passo no Elipse E3 para a configuração da comunicação Modbus é inserir o driver ModiconModbus Master no projeto em questão. Este procedimento é apresentado na Fig. 13.

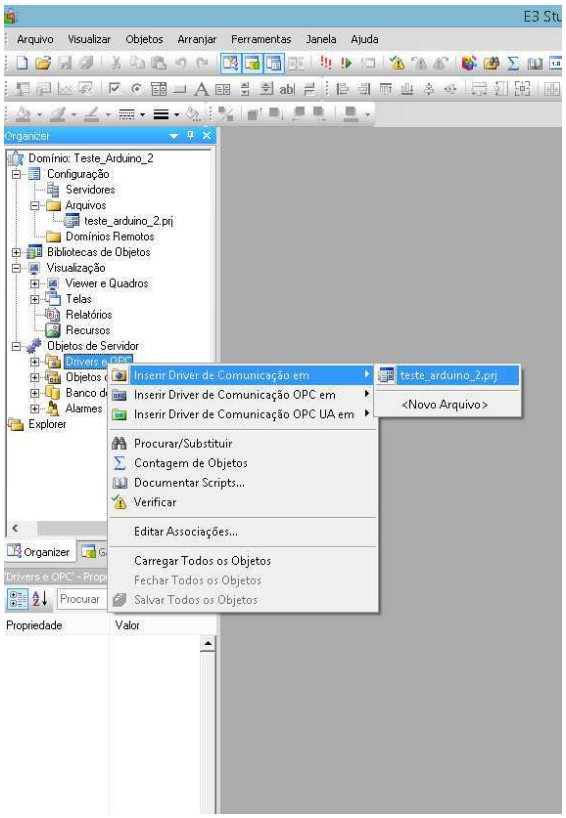

Figura 13: Adição do Driver ModconModbus Master.

Assim que adicionado, a janela ilustrada na Fig. 14 é aberta. Esta é a aba de operações disponíveis, sendo também possível criar outras operações ou modificar as existentes com os comandos na parte direita.

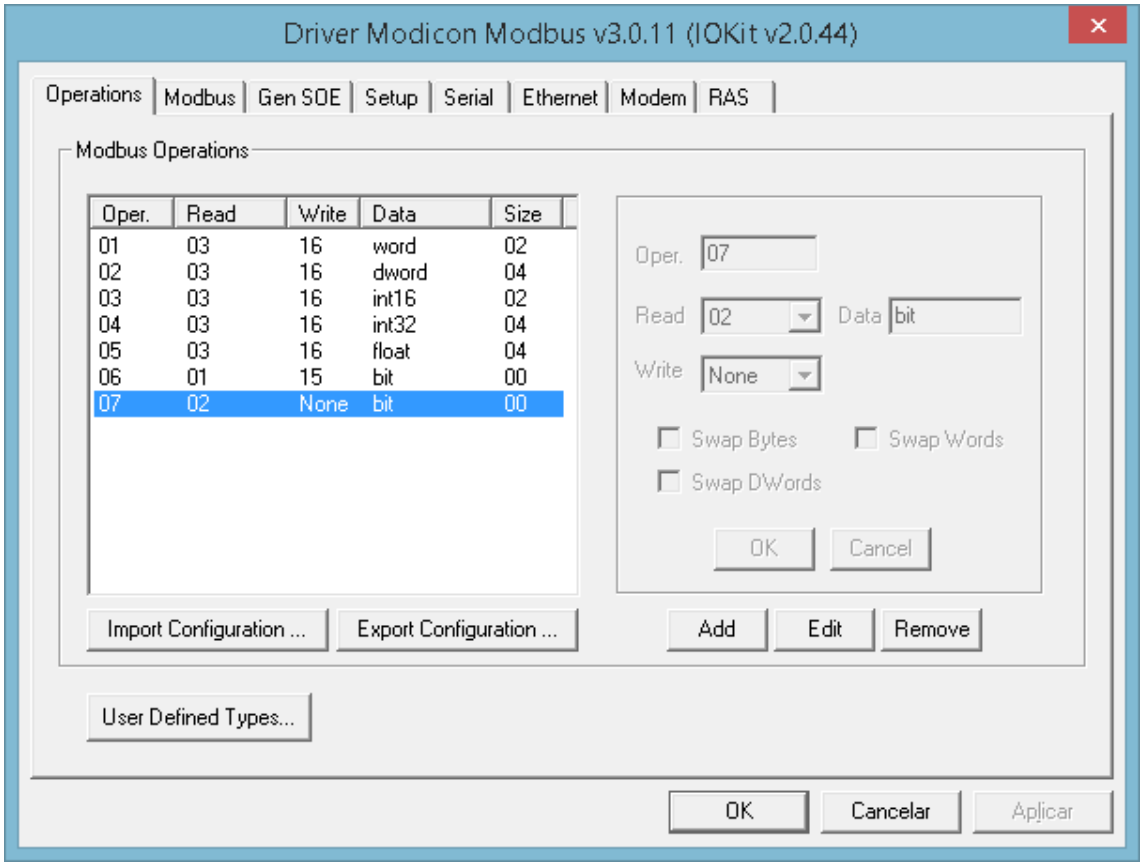

Figura 14: Operações de Modbus.

As operações pré-definidas pelo driver são as seguintes:

- 1: Leitura e escrita em um dado tipo *word* (palavra de 16 bits);
- 2: Leitura e escrita em um dado tipo *dword* (palavra de 32 bits);
- 3: Leitura e escrita em um dado tipo inteiro de 16 bits;
- 4: Leitura e escrita em um dado tipo inteiro de 32 bits;
- 5: Leitura e escrita em um dado tipo *float*;
- 6: Leitura e escrita em um dado tipo *bit;*
- 7: Leitura de um dado tipo *bit*.

Na aba Modbus apresentada na Fig. 15, são definidas as opções do protocolo, como o modo de transmissão (RTU ou ASCII), o modelo de endereçamento inicial, o endereço de escravo padrão e a forma de aviso de erros.

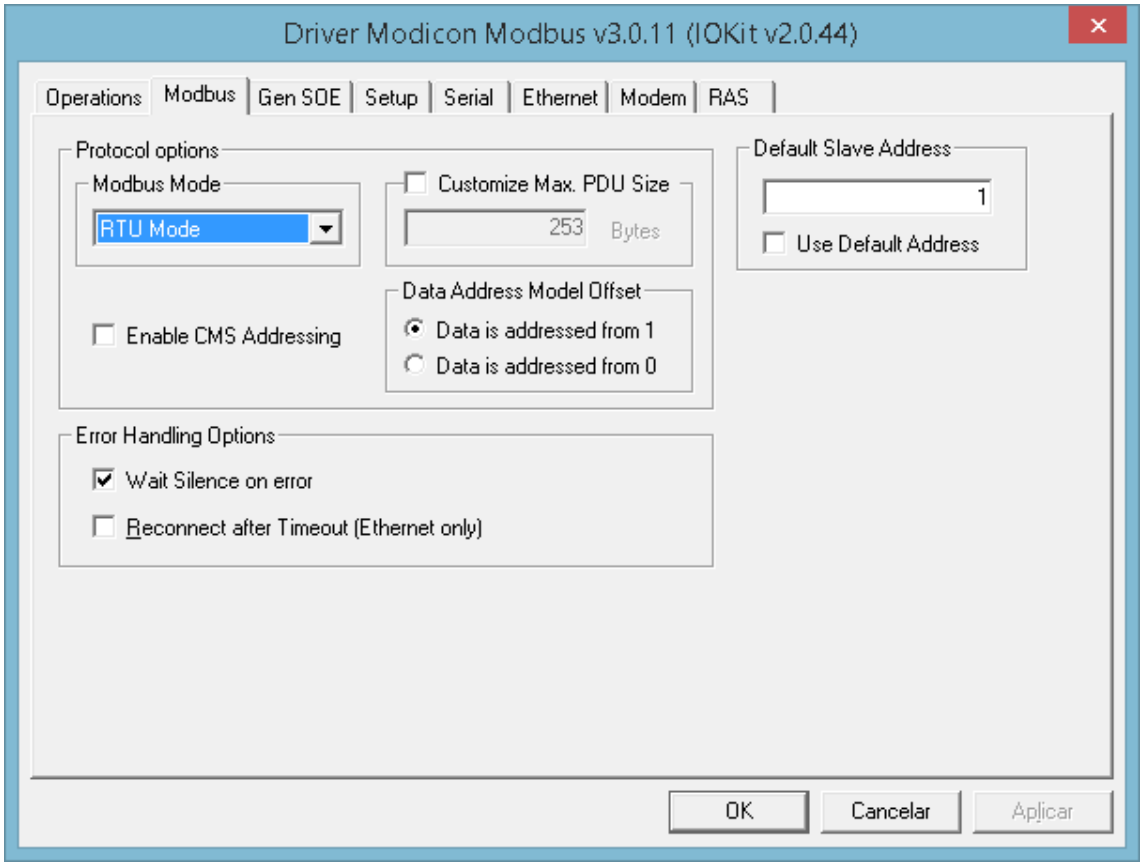

Figura 15: Aba Modbus.

Na aba Setup, apresentada na Fig. 16, é definido o tipo de conexão (Serial ou Ethernet) e o tempo de aguardo para resposta do equipamento.

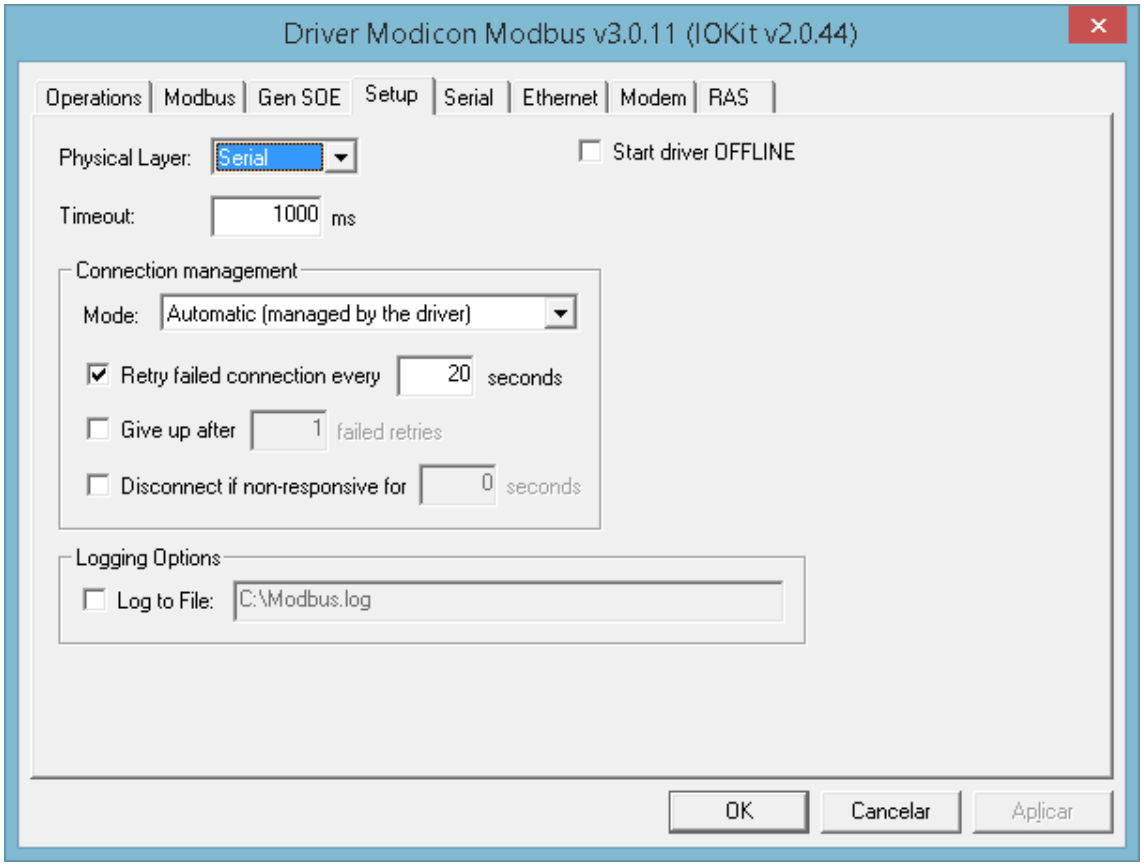

Figura 16: Aba Setup.

Com as definições na aba *Setup* concluídas, podem ser feitas as configurações na barra serial, como endereço serial de comunicação, taxa de transferência de dados, tipos de dados, paridade e *bit* de parada. A aba Serial é apresentada na Fig. 17.

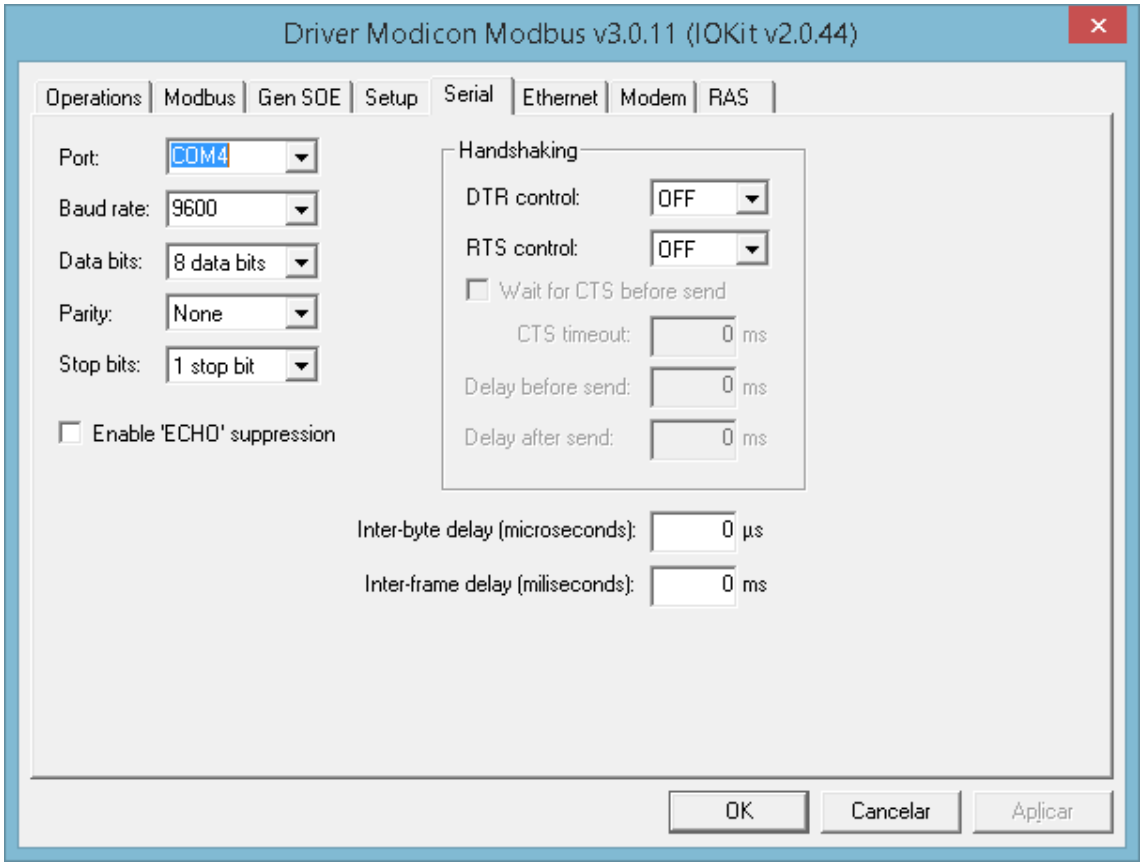

Figura 17: Aba Serial.

Após a configuração da aba Serial, a configuração para o modo de transmissão Modbus RTU está concluída e podem ser criadas *tags* de comunicação no driver configurado anteriormente para receber os dados enviados pelo Arduino no Elipse E3, a criação destas *tags* é apresentada nas Figuras 18 e 19.

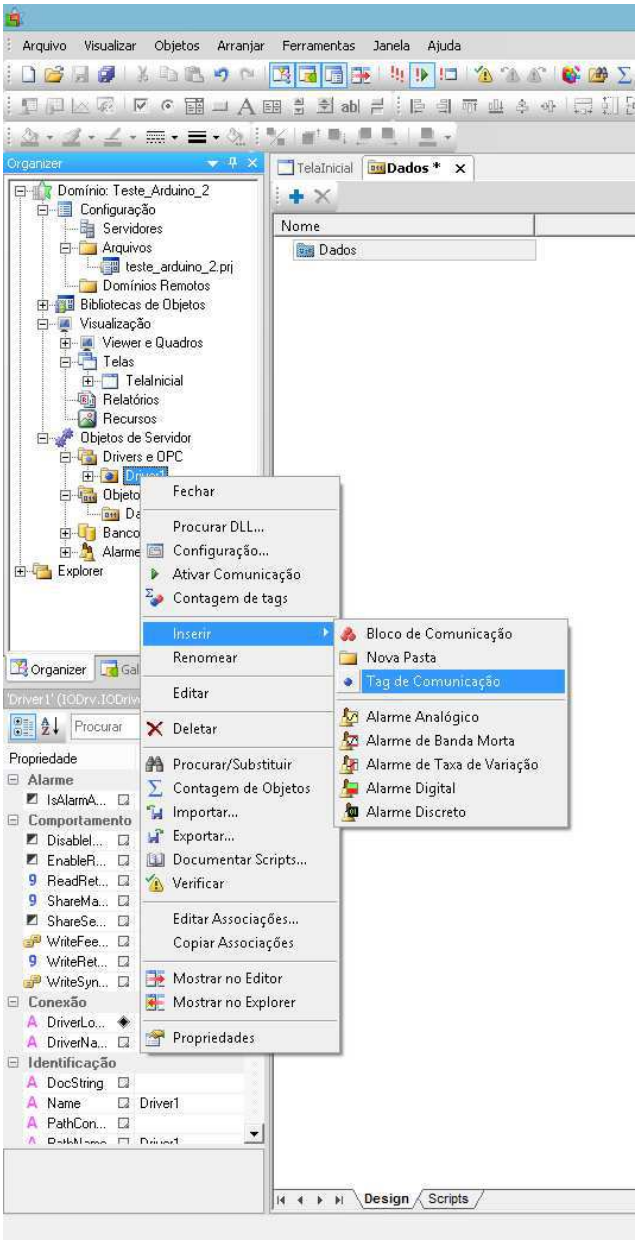

Figura 18: Inserção de *Tags* de comunicação.

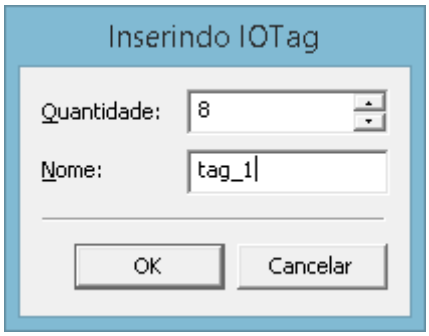

Figura 19: Janela IOTag

Após a realização destes passos, a aba driver deverá conter as *tags* criadas como é apresentado na Fig. 20, restando apenas registrar os endereços de comunicação, como

também as operações que deverão ser realizadas. Caso todos os passos estejam bem executados, ao iniciar a comunicação, as *tags* deverão ficar na cor azul, e a qualidade de sinal será de 192, mostrando que a comunicação está ocorrendo de maneira correta. Isso é apresentado na Fig. 21.

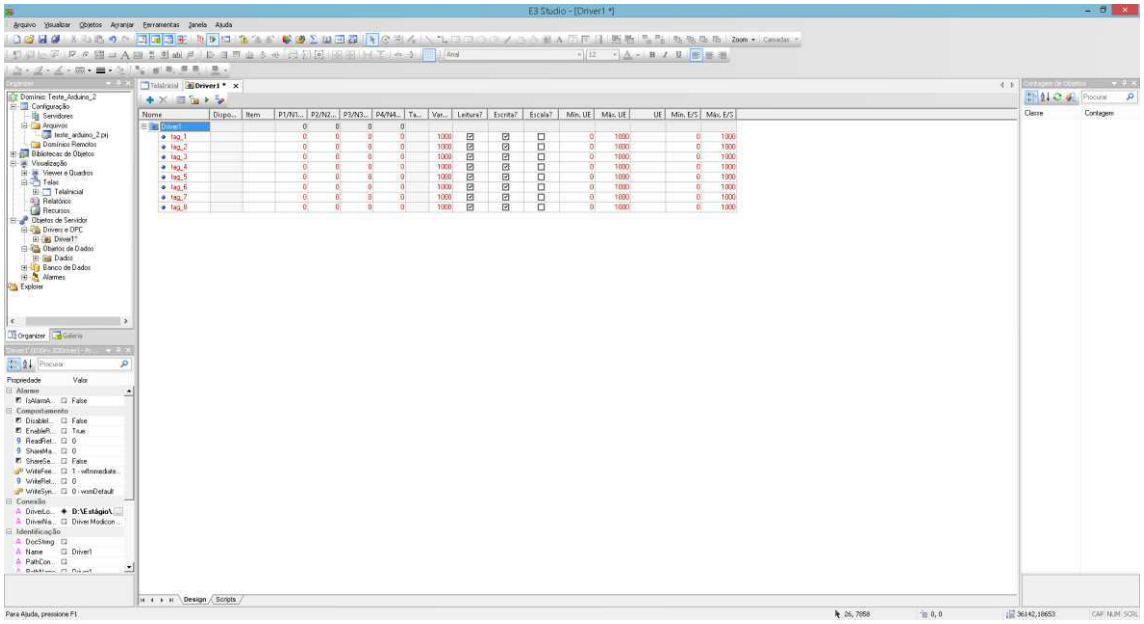

Figura 20: Criação de *tags*.

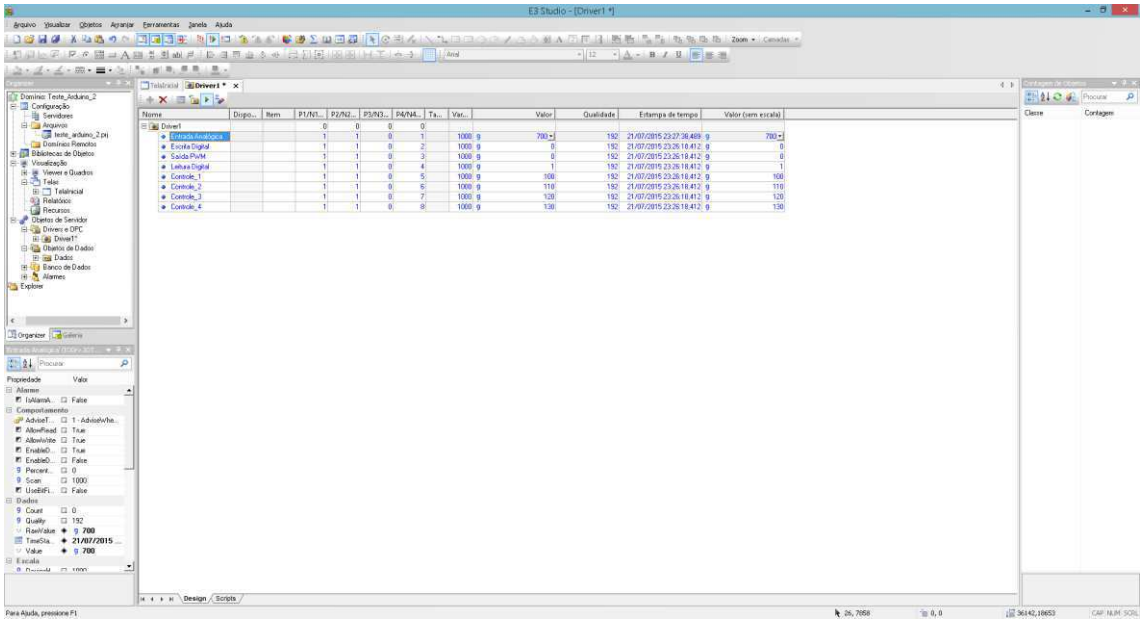

Figura 21: Comunicação de *tags*.

Para um teste final, foi criada uma interface gráfica do E3 *Viewer* como apresenta a Fig. 22. Mostrando que a comunicação ocorre em tempo real, e pode ser visualizada em um sistema supervisório.

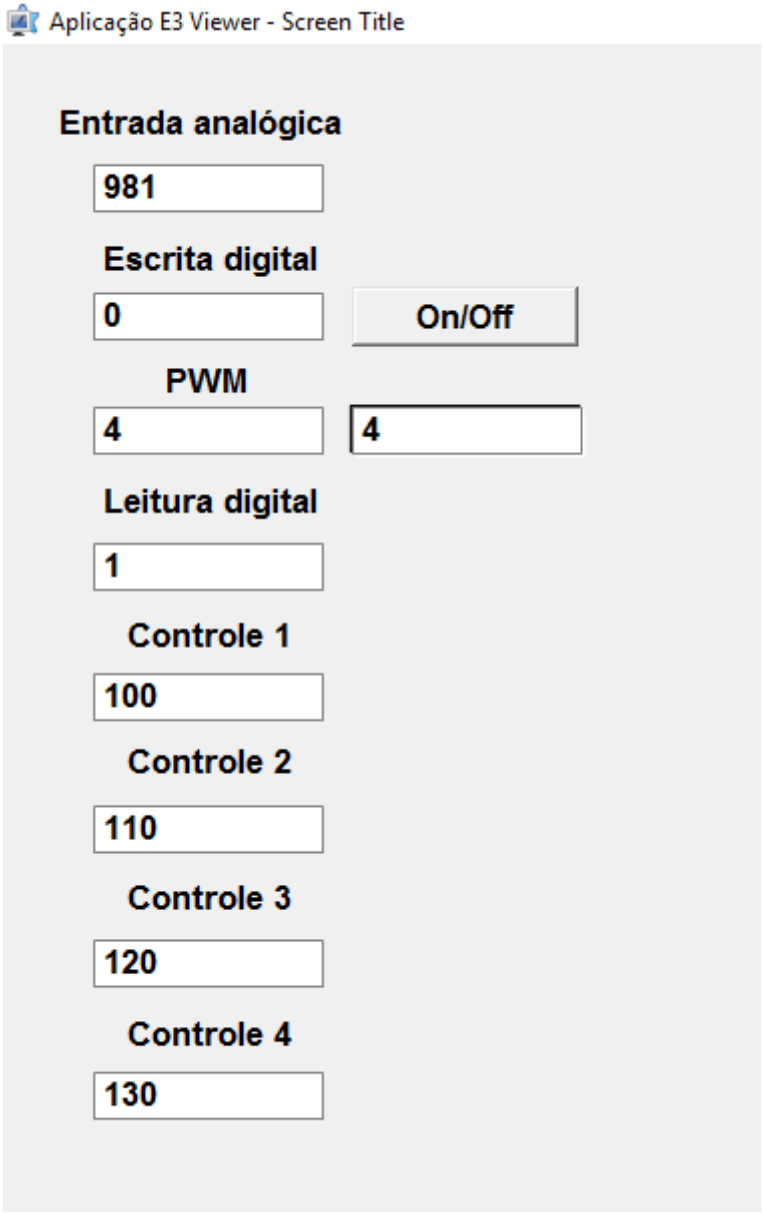

Figura 22: Tela de supervisório criada.

3.2 ESTABELECER A COMUNICAÇÃO SERIAL ENTRE O SOFTWARE ELIPSE MOBILE E A PLACA ARDUINO UNO.

A Elipse fornece uma biblioteca para a comunicação com Arduino, fornecendo assim protocolo próprio de comunicação, sendo este chamado de *protocolo Elipse Mobile/Arduino*. Para a realização da comunicação, a placa Arduino deve estar conectada via cabo serial no mesmo computador do servidor.

Importa-se a biblioteca fornecida pela Elipse Mobile para a placa Arduino e assim pode ser desenvolvido o código que atenda às necessidades pedidas. O código desenvolvido encontra-se na primeira parte do Anexo B.

Porém a biblioteca oferecida pela Elipse Mobile não contemplava a escrita na memória interna do Arduino. Para tanto, foi necessário o desenvolvimento de novos parâmetros que contemplassem essa função, para que fosse adicionado à biblioteca padrão da Elipse Mobile. Sendo assim, foi alterada a função AnalogReadComand() para que quando selecionada a entrada analógica A5, e leitura fosse realizada nas memórias internas do Arduino no lugar da entrada analógica A5. O aplicativo Elipse Mobile não permite a mudança do nome da função. As alterações feitas podem ser vistas na segunda parte do Anexo B.

Ao final do processo, todas as formas de conexão pedidas foram contempladas e a tela de supervisório feita para o teste pode ser visualizada na Fig. 23, e a visualização em dispositivo móvel é apresentada na Fig. 24.

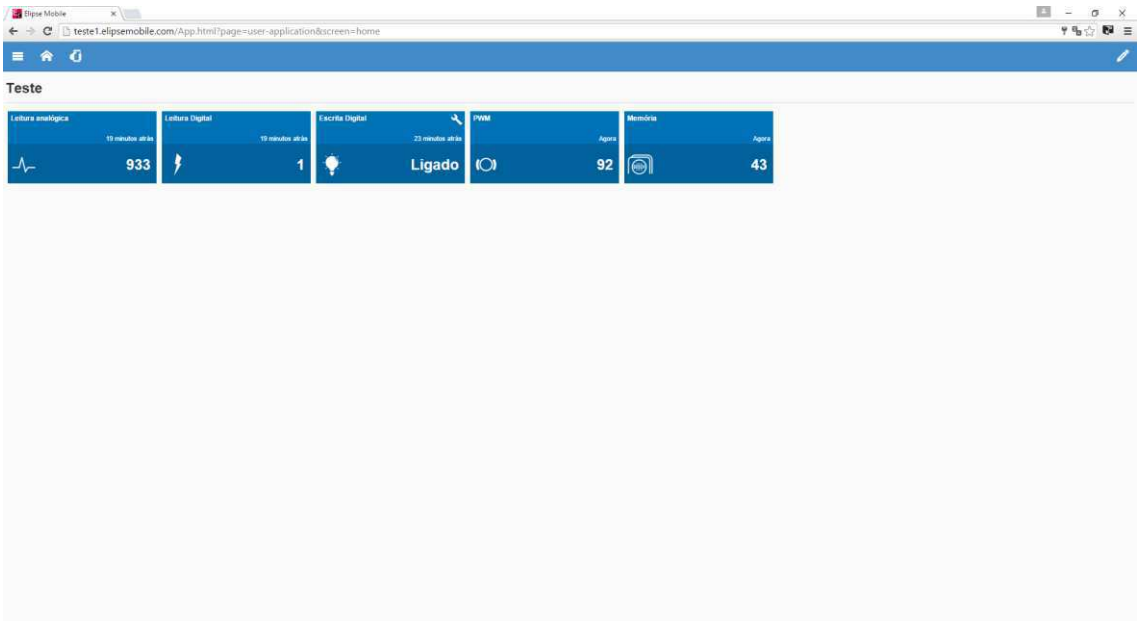

Figura 23: Supervisório criado para teste do Elipse Mobile.

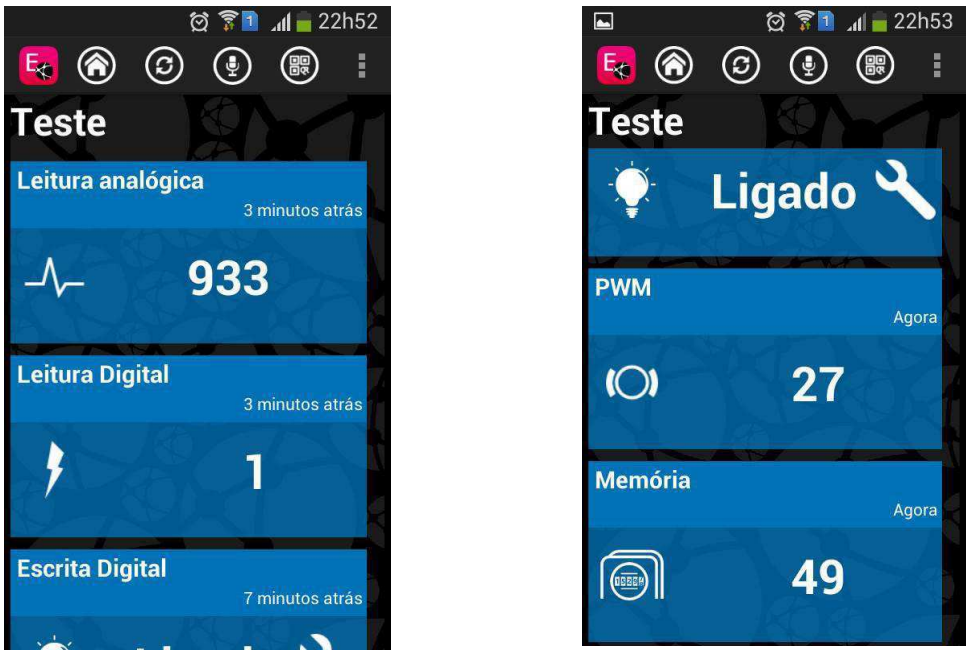

Figura 24: Supervisório visto através de dispositivo móvel.

3.3 DESENVOLVIMENTO DE CONTROLE DE POTÊNCIA AC POR MEIO DE ARDUINO:

3.3.1 PARTE ELETRÔNICA:

Tendo como base as informações fornecidas no embasamento teórico, a primeira etapa a ser realizada foi a definição dos valores dos componentes do circuito detector de zero.

Os valores dos resistores R_1 e R_2 devem ser calculados de modo a reduzir a corrente que passa pelo led do optoacoplador a um máximo de 60 mA, porém a corrente ideal de operação para este é de 10 mA (valores fornecidos pelo *datasheet* do componente). O circuito detector de zero é apresentado novamente na Fig 25. Usando-se a lei das malhas, é obtido:

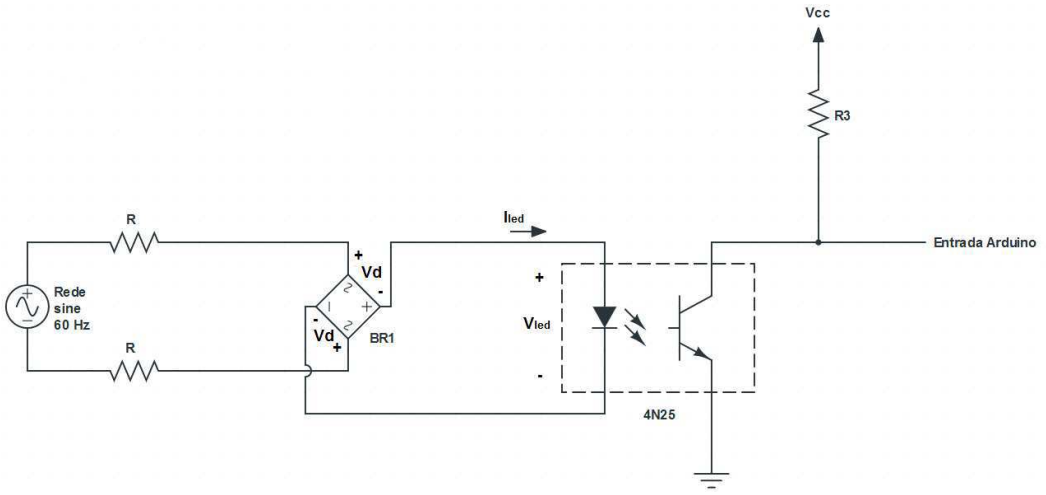

Figura 25: Diagrama do circuito detector de zeros

$$220 - RI_{led} - V_D - V_{led} - V_D - RI_{led} = 0 \quad (23)$$

Para $I_{led} = 10mA$, $V_D = 0{,}7V$ e $V_{led} = 1{,}8\,V$, obtêm-se o valor de 10,84 kΩ para R. Como este valor não é comercial, foram adotados resistores de 12 kΩ. Foi escolhido o valor de 12 kΩ, superior a 10,84 kΩ, para que a corrente não excedesse o valor de 10 mA, pois valores acima disto, diminuem a eficiência do componente, segundo o *datasheet* do mesmo.

Como estes resistores estão limitando a tensão vinda da rede elétrica, é importante saber qual é a dissipação de potência sobre estes, a fim de evitar a sua queima. Como a corrente dos mesmos é de aproximadamente 10 mA, temos:

$$P_R = RI_{led}{}^2 \quad (24)$$

Onde: P_R é a potência sobre cada resistor.

Obtendo-se uma potência dissipada de 1,2 W, sendo assim, para uma boa margem de segurança, são adotados resistores de 2 W.

Como as entradas do Arduino podem suportar no máximo uma corrente de 40 mA, o valor do resistor R_3 de *Pull-Up* deve garantir uma corrente menor que esta. Usando a Lei de Ohm, o valor mínimo necessário para R_3 é de 125Ω. Sendo assim, para uma boa margem de segurança, foi adotado um resistor de 1 kΩ.

Possuindo todos os valores dos componentes, o circuito detector de zero, foi simulado no *Proteus.* O circuito montado no mesmo como também as formas de onda da rede e da entrada do microcontrolador são apresentados respectivamente na Fig.26 e 27.

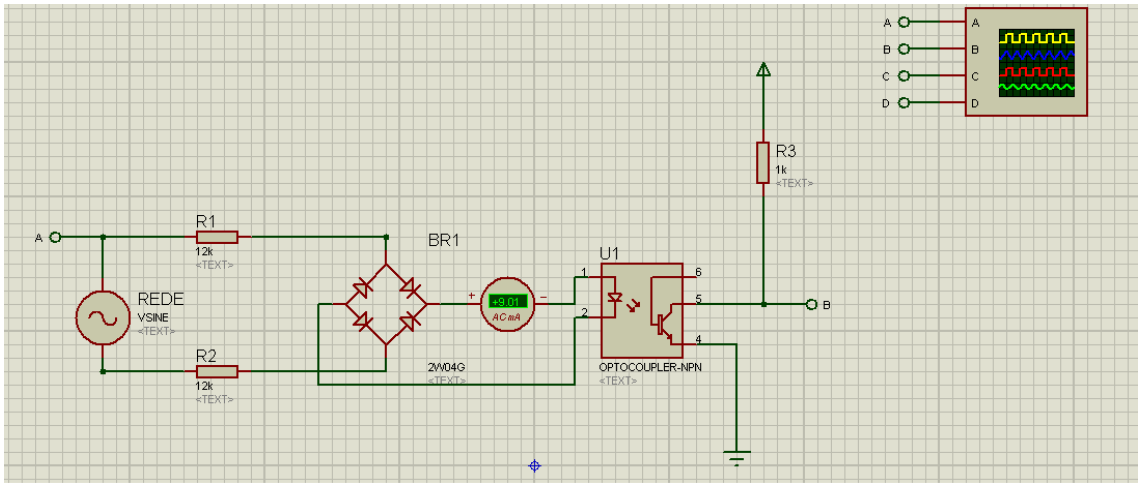

Figura 26: Simulação do Circuito detector de zero.

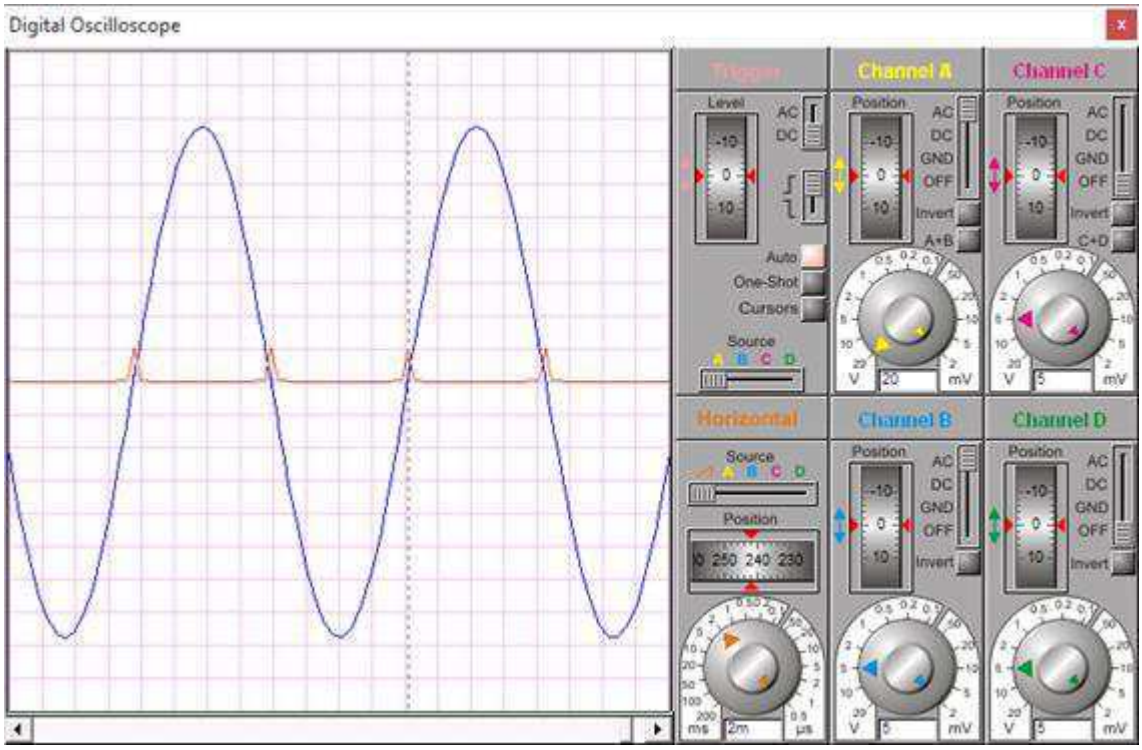

Figura 27: Formas de onda da rede (azul) e saída do detector de zero (vermelho).

Nas Figuras 26 e 27 pode-se observar que a corrente desejada é obtida, além do pulso de tensão nas passagens por zero da rede, levando nível lógico alto ao microcontrolador.

Feito isso, é necessário então o desenvolvimento do circuito de acionamento, responsável pelo acionamento das chaves utilizadas, no tempo de disparo correto. Para isso, foi desenvolvido o circuito que é apresentado na Fig. 28:

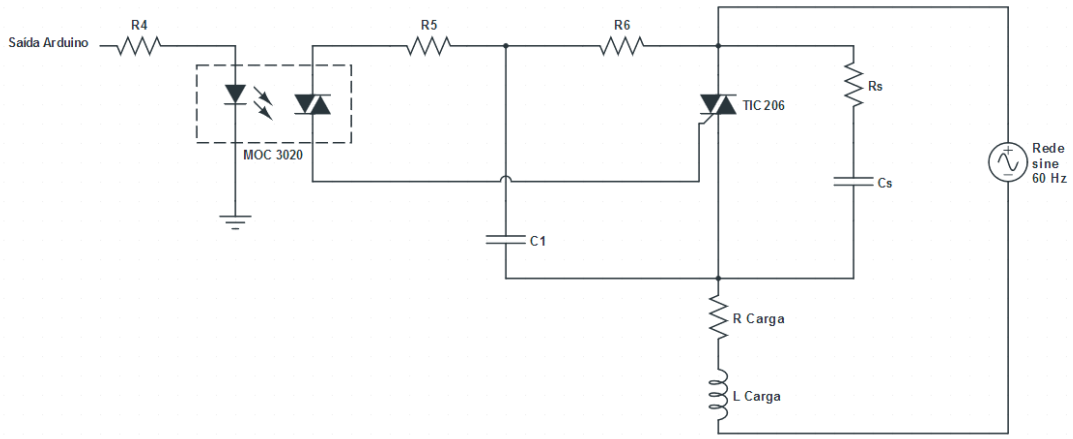

Figura 28: Diagrama do circuito Acionador.

O led do MOC 3020 também possui corrente ideal de operação de 10 mA (valor fornecido pelo *datasheet* do componente). Como a saída em nível alto do Arduino possui 5 V, usando a Lei de Ohm, chega-se ao valor de 500 Ω para o resistor R_4.

O resistor R_s e capacitor C_s constituem o circuito de proteção *Snubber*. Sabendo que a tensão sobre o TRIAC no seu bloqueio é 220 V_{RMS}, a corrente máxima suportada pelo TIC 206 é de 25 A (valor fornecido pelo *datasheet* do componente) e sua frequência de comutação é de 120 Hz, pois este comuta duas vezes a cada ciclo completo da tensão da rede. Usando (21) e (22), obtemos o valor de 8,8 Ω para R_s. Como este valor não é comercial, adotou-se então um resistor de 10 Ω. Para C_s, obtém-se o valor de 189,4 nF, novamente um valor não comercial, e desta forma, foram usados dois capacitores cerâmicos de 100 nF em paralelo.

Os valores para R_5, R_6, e C_1, foram obtidos por simulações. É desejável que seus valores sejam obtidos desta maneira, já que os mesmos irão variar de acordo com a carga. Devem ser tomados valores que resultem em atraso inicial curto, e que evitem flutuações no circuito, podendo causar acionamentos indesejáveis ao TRIAC.

O circuito de acionamento foi simulado no *Proteus,* tendo como carga lâmpadas em paralelo, simulando a iluminação de um ambiente. Este é apresentado na Fig. 29:

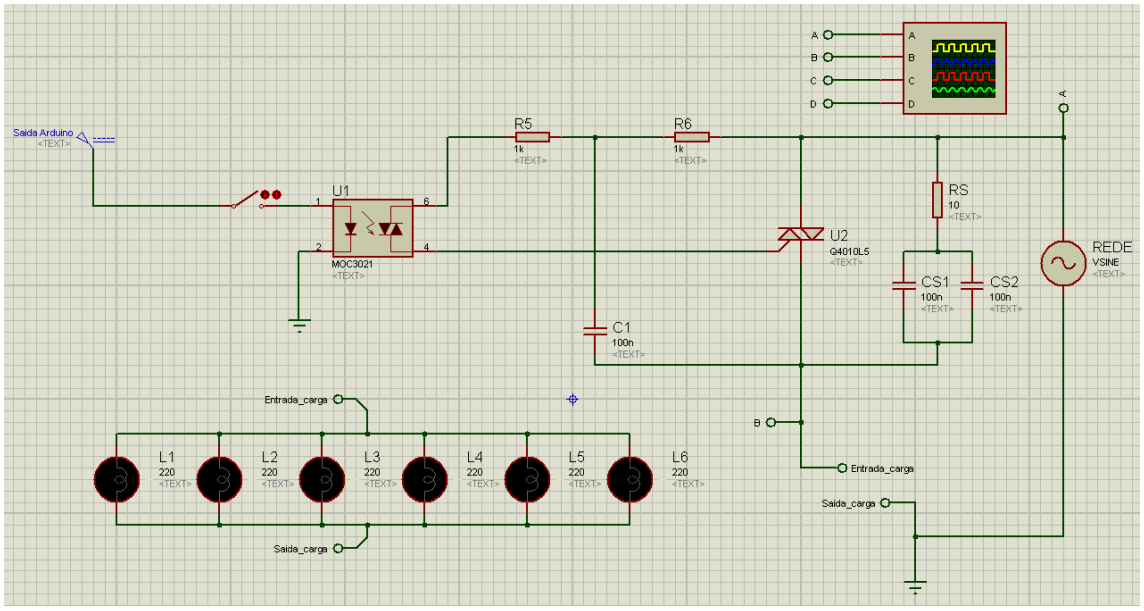

Figura 29: Simulação do circuito acionador.

Usando-se para C_1 o valor de 100 nF e valores iniciais de 1 kΩ para R_5 e R_6, obtém-se a forma de onda de tensão sobre as lâmpadas apresentada na Fig. 30 ao se ligar a chave:

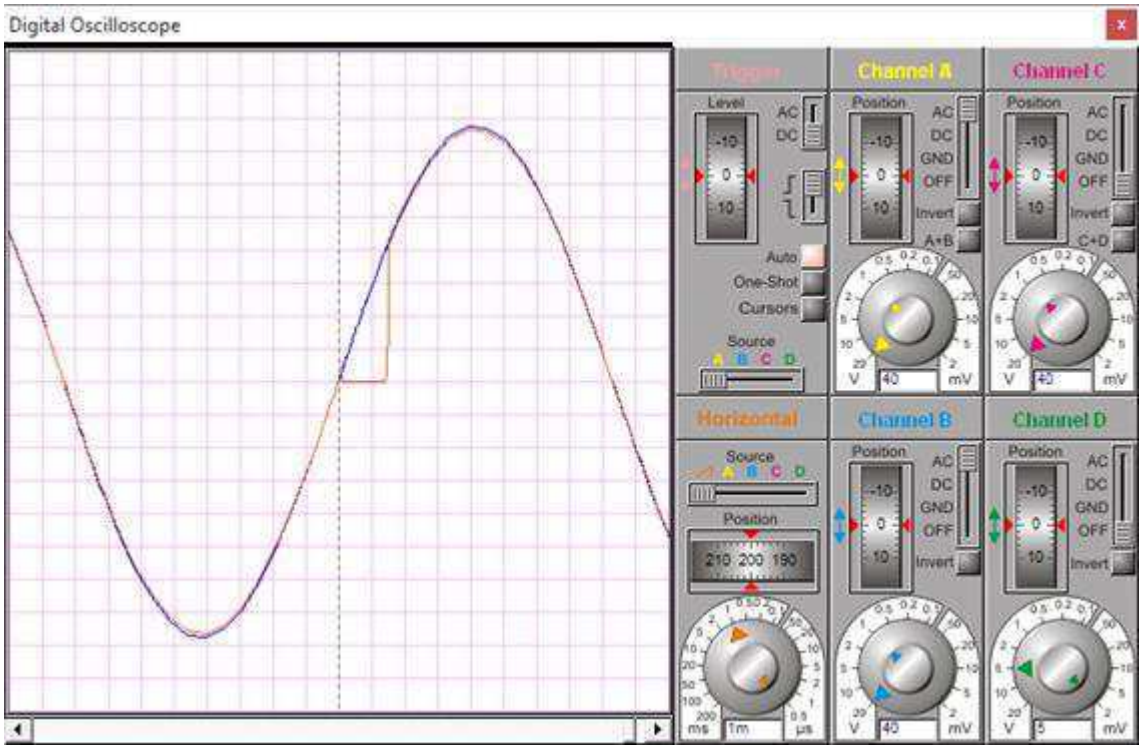

Figura 30: Forma de onda sobre as lâmpadas (vermelho) com R5 e R6 com 1 kΩ.

Nota-se um grande atraso no disparo do TRIAC nos semiciclos positivos da tensão da rede. Variando-se os valores de R_5 e R_6, para 180 Ω e 330 Ω respectivamente,

obtém-se a forma de onda apresentada na Fig.31. Tendo assim um atraso inicial admissível para aplicações.

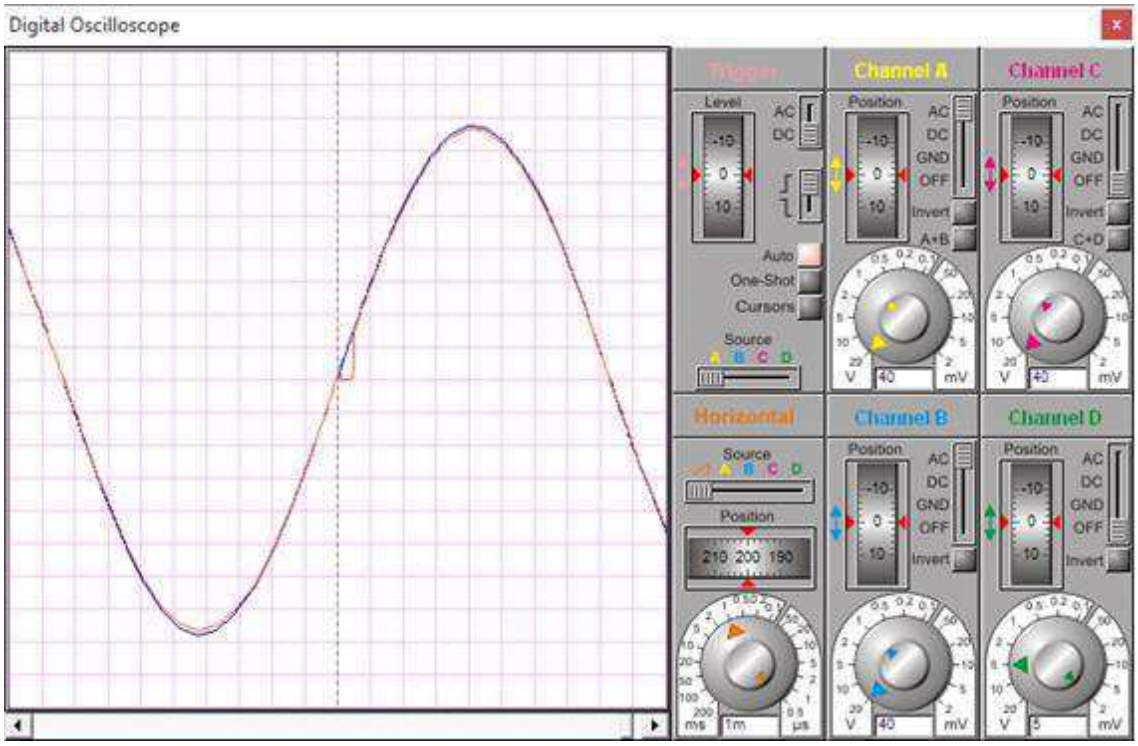

Figura 31: Forma de onda sobre as lâmpadas (vermelho) com R5 e R6 com 180 Ω e 330 Ω respectivamente.

Foram também verificadas as tensões sobre cada resistor. 2,07V sobre R_5, 3,84 sobre R_6, e 0,14V sobre R_s. Sendo assim, podem ser usados resistores de 1/2 W.

3.3.2 PARTE LÓGICA:

Com toda parte eletrônica desenvolvida, foi então desenvolvida a parte lógica. Primeiramente foi desenvolvido o controle por potenciômetro. O código desenvolvido encontra-se na primeira parte do Anexo C. Este foi dividido em blocos para o melhor entendimento:

- Bloco 01: Aqui são definidas as variáveis usadas no programa. A variável “nivel” é do tipo *volatile* por ter seu valor variante na chamada da rotina de interrupção. A variável “resolucao” não necessita ser do tipo *volatile* por ter seu valor fixo. As saídas PWM do Arduino possuem 8 bits, sendo assim, podem ser representadas por valores que variam de 0 a 255, possuindo então 256 níveis. O tempo de meio-ciclo da onda da tensão da rede possui 8333,333 µs. A resolução então é definida pela

divisão do tempo de meio-ciclo da onda da tensão da rede, pelo número de níveis da saída PWM.

- Bloco 02: Aqui é definida a função “void ativa_TRIAC()”. Esta é responsável por gerar os pulsos para o *Gate* do TRIAC nos ângulos de defasagens corretos. Primeiramente é esperado o tempo do ângulo de disparo definido pelo potenciômetro. Atingido este tempo, o pulso na saída é ativado por 83 µs (tempo de 1% do meio-ciclo).
- Bloco 03: Aqui são definidas as funções “void setup()” e “void loop()”. Na primeira são definidas as entradas e saídas utilizadas e a função “attachInterrupt” responsável pela interrupção que deve ser usada. Esta possui o formato “attachInterrupt(pino_da_interrupcao, função_chamada, modo)”. Na forma como está definida, usa o pino digital 2 como responsável pela verificação da interrupção, chama a função “void ativa_TRIAC()” e isso ocorre em quando varia do nível lógico 0 para o nível lógico 1. Na função “void loop()” é realizada a leitura do valor do potenciômetro pela porta analógica A0. Como esta possui 10 bits, as leituras podem ser representadas por valores que variam de 0 a 1023. Estes valores são linearmente convertidos para valores entre 5 e 245 pela função “map”. Não foram usados os valores 0 a 255 pois, nestes extremos, muitas vezes a sincronia entre Arduino e tensão da rede era perdida.

Para seu teste, foram então integrados os circuitos de detecção de zero e o circuito de acionamento, como também o chip ATmega 328p, para a simulação do Arduino. Os mesmos são apresentados nas Fig. 32 e 33 respectivamente.

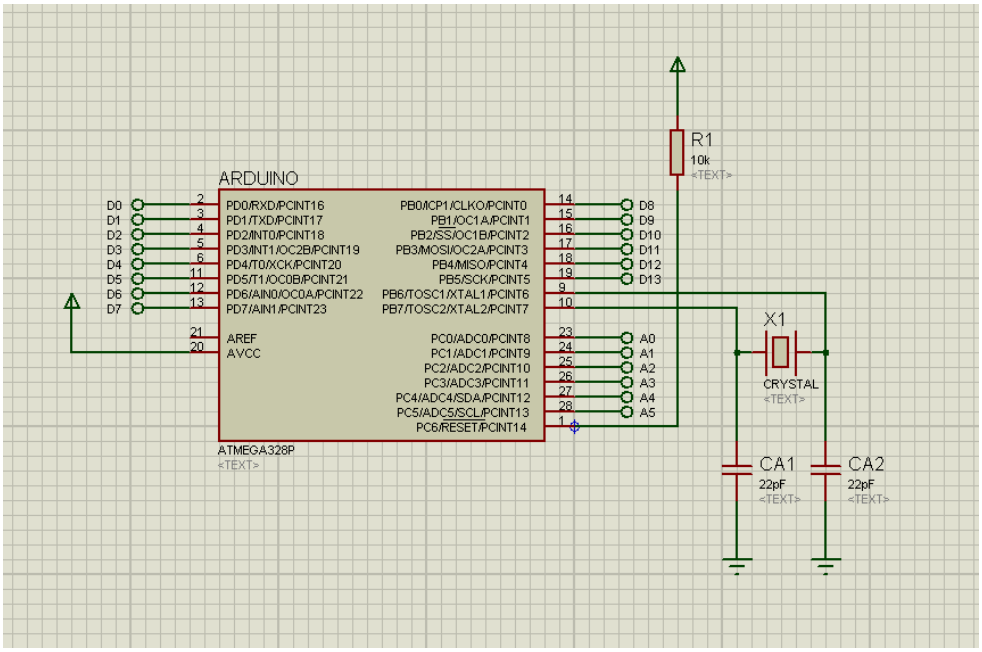

Figura 32: Simulação do Arduino.

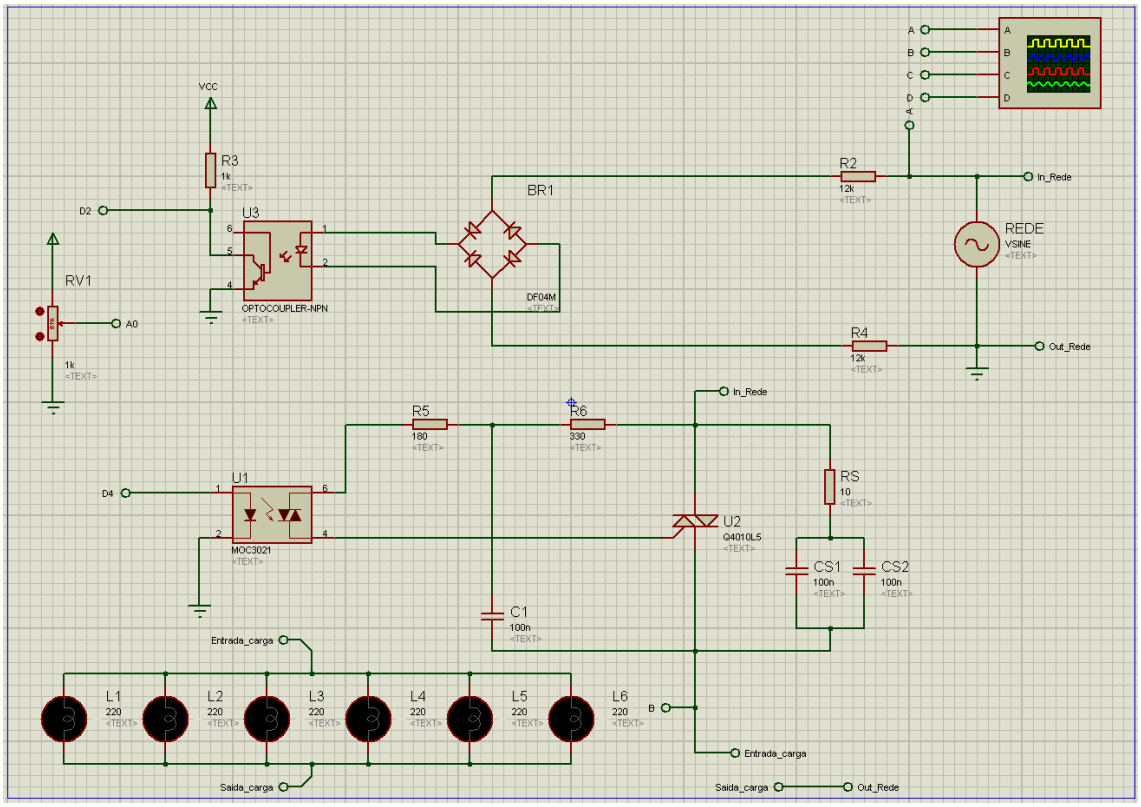

Figura 33: Simulação do sistema completo.

O código presente na primeira parte do Anexo C, foi carregado para o ATmega 328p simulado. Seu fluxograma é apresentado na Fig. 34. Obtendo-se então a forma de onda da tensão sobre a carga apresentada na Fig.35, tendo seu ângulo de disparo controlado pelo potenciômetro usado.

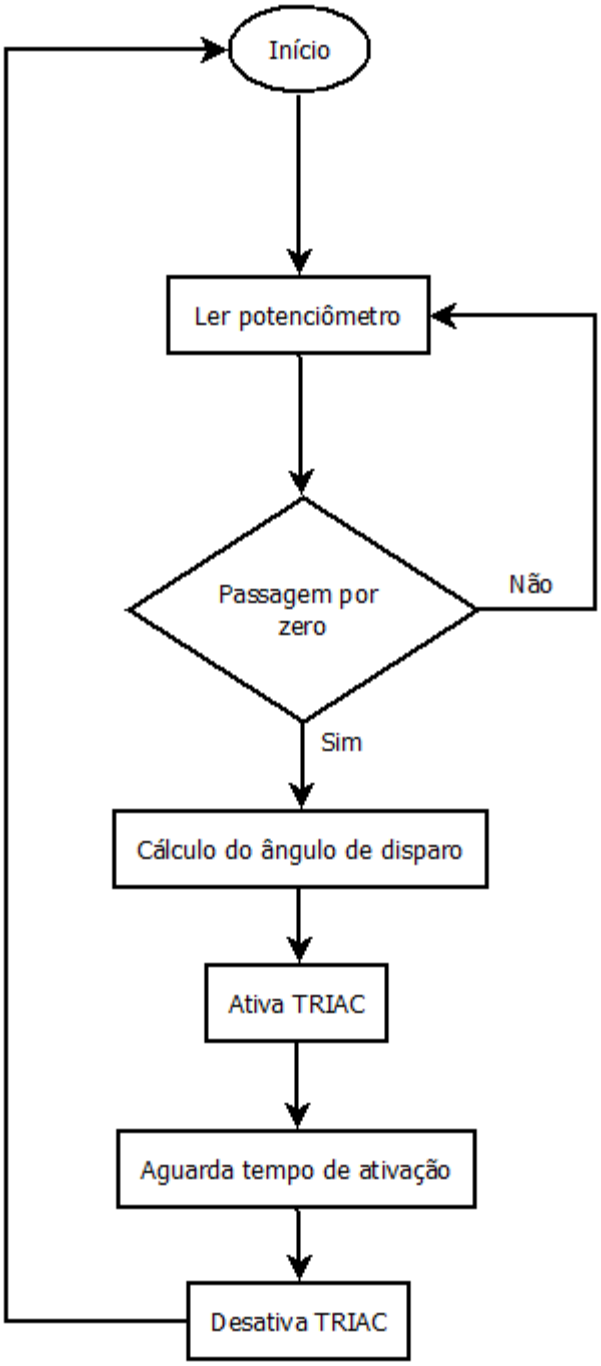

Figura 34: Fluxograma do controle por potenciômetro.

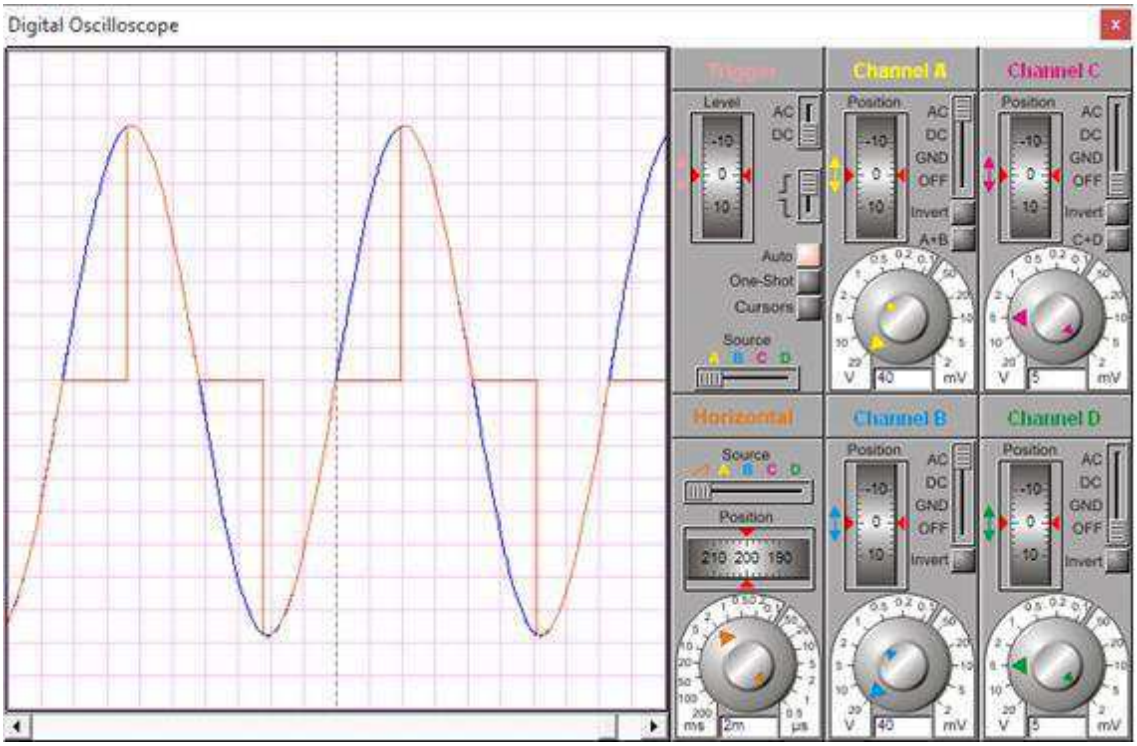

Figura 35: Forma de onda da tensão da carga (vermelho).

Com a conclusão desta etapa, começou então a construção do PID de fato. Tendo como base as informações fornecidas no embasamento teórico, foi desenvolvido um código para a implementação do PID em Arduino com comunicação serial com o Elipse E3. Este código encontra-se na segunda parte do Anexo C. Como vários conceitos usados neste código já foram abordados em tópicos anteriores, estes não serão explicados novamente. Sendo detalhada apenas a parte definida no bloco 01.

- Bloco 01: O tempo de amostragem usado pelo Arduino é calculado usando a função “millis()”, esta retorna o tempo decorrido em que o programa está em funcionamento em milissegundos, sendo assim a variável “ultimo_processo” recebe o tempo em que o programa está funcionando e o tempo de amostragem é calculado pela subtração do tempo atual pelo tempo decorrido no último processo. O erro é calculado pela subtração do “setpoint” pela leitura do sensor usado no processo. O “erro_anterior” recebe o valor atual do “erro”, para que assim na próxima iteração este seja o valor anterior. As ações proporcionais, integrais e derivativas são calculadas como mostradas em (6) e o sinal de controle do PID é aplicado no sinal PWM.

A planta escolhida para os testes foi de iluminação, sendo esta formada por uma lâmpada fluorescente compacta, funcionando como atuador e um LDR, sendo este o sensor de luz. Ambos em um ambiente fechado. É apresentado na Fig. 36 o circuito montado em *protoboard*.

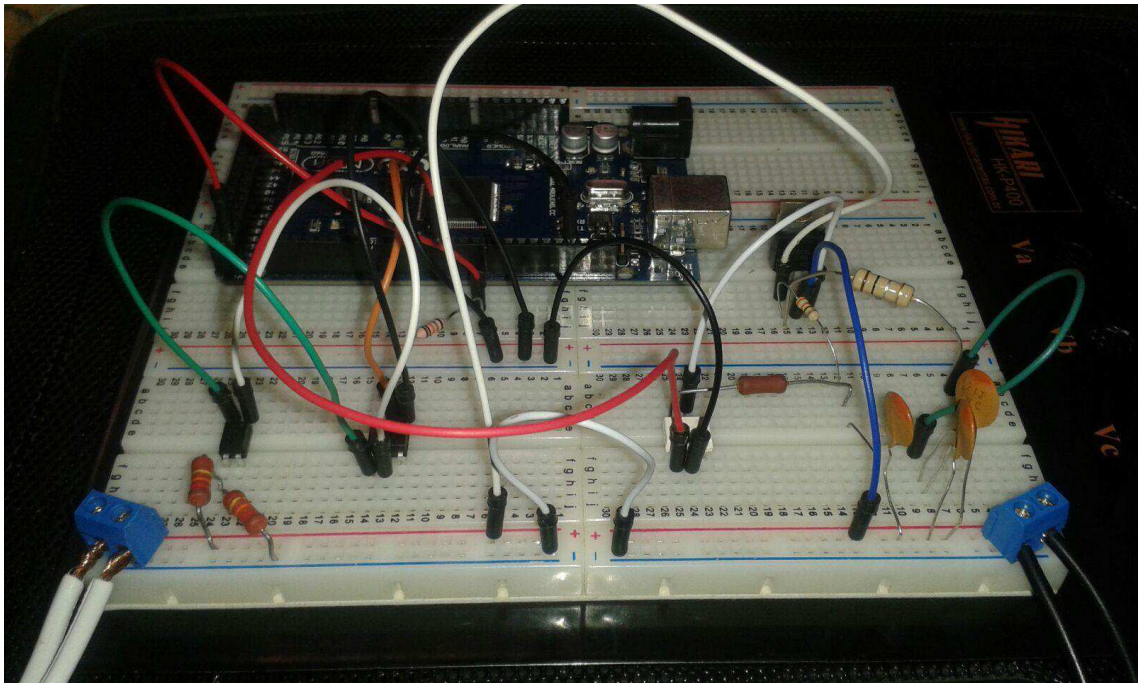

Figura 36: Fotografia do circuito montado em *protoboard*.

Usando os mesmos processos para a comunicação entre Arduino e Elipse E3 mostradas na primeira atividade, foi desenvolvido o supervisório de testes apresentado na Fig. 37.

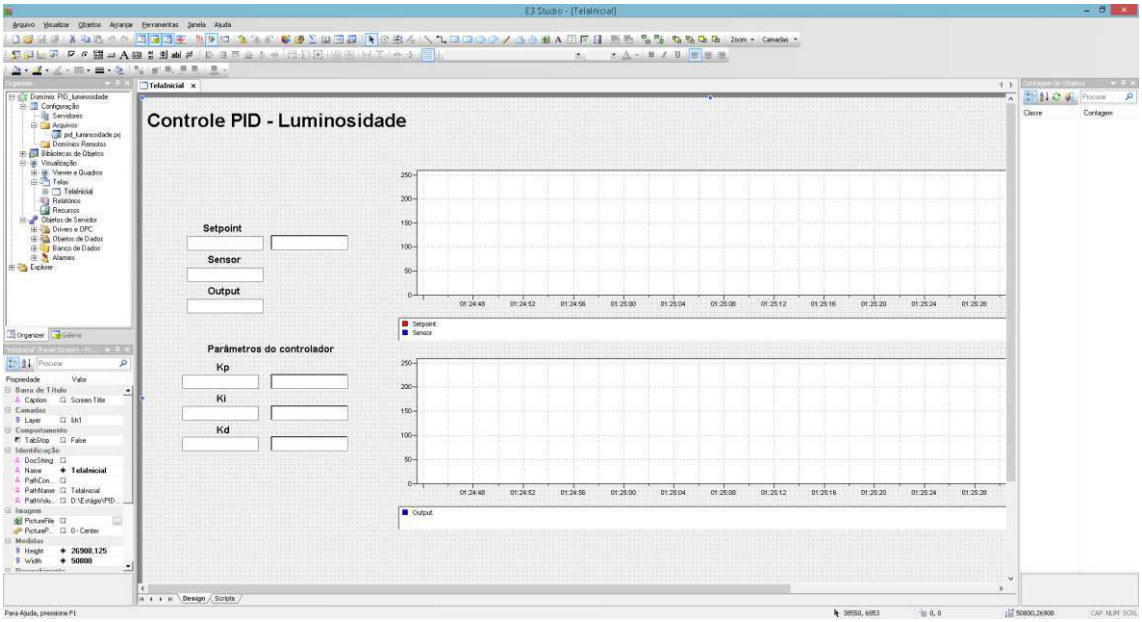

Figura 37: Supervisório desenvolvido para o teste do PID em Arduino.

Foi então feita uma bateria de testes para determinação correta do funcionamento do projeto, primeiramente foi testado a componente proporcional, em seguida a componente proporcional e integral e por fim todas as componentes, proporcional, integral e derivativa.

Na Fig.38 é apresentado o teste com o termo Kp possuindo o valor 5 e com variações no “Setpoint”.

Figura 38: Controle com Kp = 5, escolhido como exemplo.

É possível notar que o controle é feito, porém com erro de regime permanente alto, como dito antes. Para diminuir este tipo de erro, é necessário o uso da ação integral, sendo assim, foi feito um segundo teste com $k_p = 5$ e $k_i = 10$ e mudanças no “Setpoint”, apresentado na Fig.39.

Figura 39: Controle com Kp = 5 e Ki = 10, escolhidos como exemplo.

Nota-se então um controle bem mais preciso do que o conseguido a partir do uso somente da ação proporcional. Para que o controle se torne mais rápido, é necessário a adição da ação derivativa, sendo assim, foi realizado um terceiro teste, usando $K_p = 5$, $K_i = 10$ e $K_i = 5$, e mudanças no "Setpoint", apresentado na Fig. 40.

Figura 40: Controle com Kp = 5, Ki = 10 e Kd = 5., escolhidos como exemplo

Observa-se assim que as ações proporcional, integral e derivativa do controlador PID tiveram funcionamento como esperado. Os ganhos foram escolhidos como um exemplo, pois as especificações temporais para a malha fechada não foram informadas para a execução desse trabalho de estágio. O supervisório Elipse E3 pode também ser implantado em demais processos tais como resfriadores por ventiladores, sopradores térmicos, aquecedores, bombas hidráulicas, etc.

3.4 AUTOMAÇÃO DE ESTAÇÃO DE TRATAMENTO DE EFLUENTES:

Devido a questões contratuais, o autor deste documento não poderá apresentar nomes, detalhes importantes do processo produtivo ou demais informações da empresa atendida. Durante o tempo de estágio, o autor participou da primeira etapa do processo de automação da estação, o estudo do funcionamento da mesma e determinação dos equipamentos necessários.

A estação de tratamento em questão tem como finalidade básica enquadrar os despejos líquidos oriundos do setor produtivo, dentro dos padrões de emissão exigidos

pela legislação vigente. A mesma tem seu diagrama de funcionamento apresentado na Fig. 41. O processo de tratamento consta de quatro etapas distintas. Tratamento preliminar, tratamento físico-químico, tratamento biológico e sistema de desidratação.

No tratamento preliminar, o efluente é preparado para ser tratado, removendo-se sólidos grosseiros, sedimentáveis ou flutuantes, por meio das unidades de gradeamento, caixa de areia e peneira. No gradeamento são retirados todos os sólidos grosseiros, que devido ao seu tamanho ou natureza, possam causar danos como desgaste de bombas, obstrução em tubulações, baixa na eficiência de tratamento, entre outros. A caixa de areia tem por objetivo reter toda areia e outros detritos que tenham passado pela fase de gradeamento. Na fase de peneiramento, ocorre o processo de remoção dos sólidos que tenham passado pelas duas etapas anteriores.

O tratamento físico-químico tem por objetivo homogeneizar e manter toda a massa líquida em completa mistura, para posterior ajuste de pH. Este tratamento é realizado no tanque de equalização e no tanque de ajuste de pH. No primeiro, ocorre a uniformização das características do efluente, como valor de pH, condições aeróbias favoráveis para evitar odores e regular o fluxo para um regime constante. No tanque de acerto de pH, ocorre a operação de acerto de pH, preparando o efluente para as próximas etapas.

O tratamento biológico tem por objetivo reduzir o teor de matéria orgânica biodegradável remanescente, que não foi removida nos tratamentos anteriores. É constituído pelo tanque de aeração e pelo decantador secundário. No tanque de aeração ocorre o consumo da matéria orgânica por micro-organismos, formados por flocos biológicos, a massa total destes flocos é denominada lodo ativado. No decantador secundário ocorre o processo de sedimentação do lodo ativado, o retorno deste para o tanque de aeração, visando manter a massa de micro-organismos, e o descarte do excesso do mesmo para os leitos de secagem. Por fim, no sistema de desidratação o lodo ativado passa pelos leitos de secagem, onde ocorre o processo de evaporação da água e percolação.

A metodologia utilizada para a realização do diagnóstico foi baseada em visita técnica para a observação *in loco* e coleta de informações com colaboradores da empresa. A visita passou por um momento de entendimento do processo, diagnóstico de falhas no funcionamento e levantamento das necessidades apontadas pela empresa com o intuito de melhorar o processo e garantir o funcionamento automatizado na estação de tratamento de efluentes.

Os equipamentos necessários para o processo são apresentados na Tabela 2.

Tabela 2: Equipamentos necessários para a automação.

	Equipamentos
1	Inversor de frequência
2	Controlador Programável
3	Medidor de vazão ultrassônico
4	Transmissor de nível ultrassônico
5	Transmissor de pH sem isolamento de 2 fios, 0 a 14 pH
6	Eletrodo de pH para submersão em CPVC
7	Transmissor de OD
8	Sensor de OD para submersão
9	Transmissor de ORP sem isolamento de 2 fios
10	Eletrodo de ORP para submersão em CPVC
11	Válvula On/Off
12	Válvula solenóide
13	Painel de Comando
14	Turbidimetro
15	Fluido de calibração do sensor de pH
16	Fluido de calibração do sensor de OD
17	Software supervisório
18	Bomba de recalque helicoidal

Com a implantação deste sistema a empresa obterá ganhos em vários pontos, não somente no processo de tratamento que terá um maior controle, mas também a redução no consumo de energia elétrica uma vez que o acionamento de motores e bombas será automatizado. Além disso, haverá a flexibilidade na estação, sendo possível alterar o processo de tratamento de acordo com a demanda da fábrica uma vez que o nível de pH é fator determinante no tipo de tratamento que o efluente vai necessitar, garantindo

assim o controle e a ação no sistema em tempo hábil, evitando variações na eficiência da estação de tratamento de efluentes.

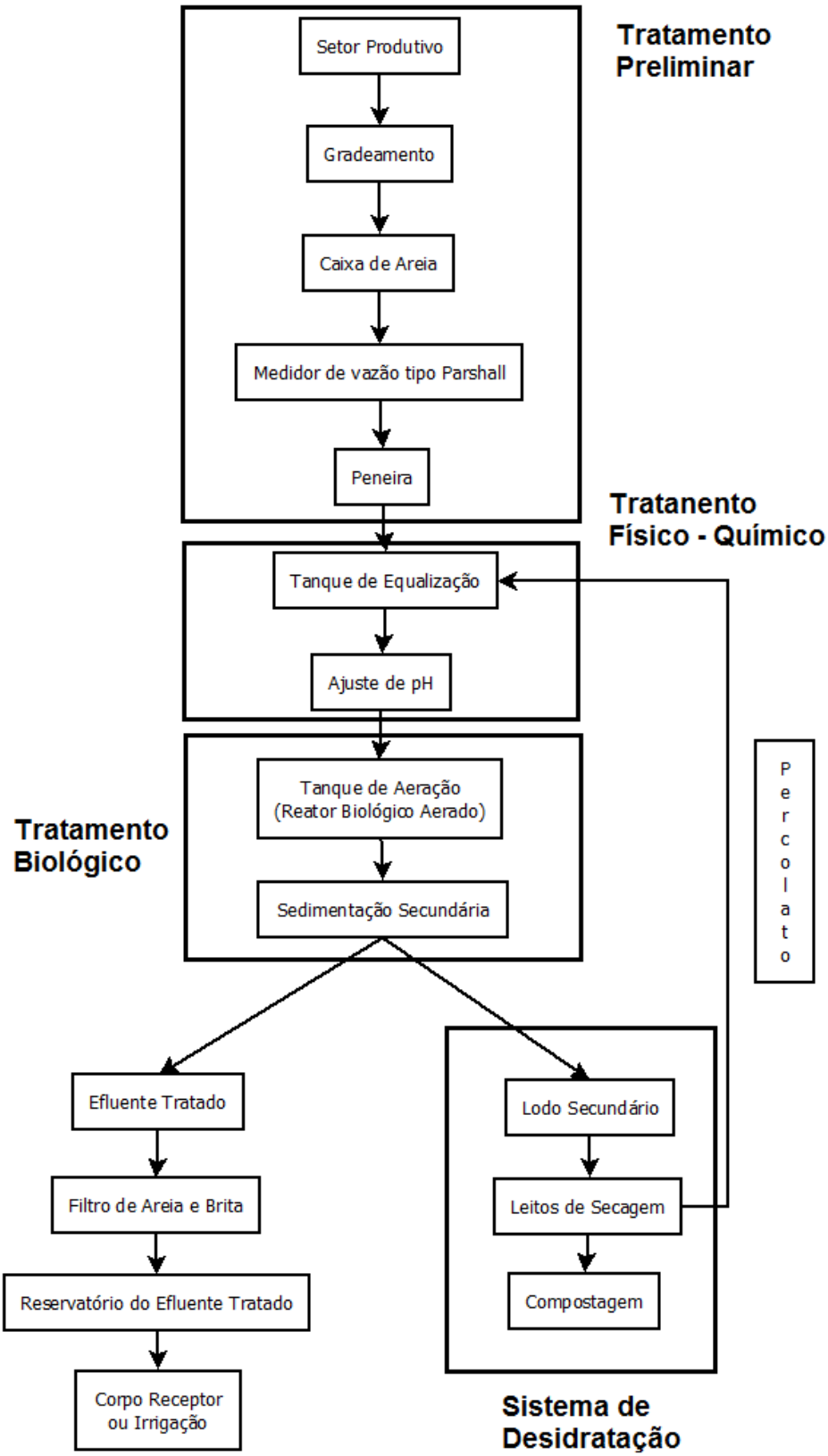

Figura 41: Fluxograma da estação de tratamento de efluentes.

3.5 COMPETIÇÃO "MEU ESTÁGIO, NOSSA IDEIA":

A competição "Meu estágio, nossa ideia", é realizada pelo IEL (Instituto Euvaldo Lodi), envolvendo todos os estagiários do SENAI e SESI da Paraíba. Os mesmos são convidados a apresentarem projetos desenvolvidos ao longo do tempo de estágio. O autor foi premiado com o projeto "Sistemas supervisórios e controle PID para a indústria". Na Fig.42 é apresentado o mesmo durante a premiação da cerimônia.

Figura 42: Cerimônia de premiação, "Meu estágio, nossa ideia".

3.6 INSCRIÇÃO DE PROJETO PARA O INOVA SENAI:

O autor, juntamente com outros desenvolvedores, inscreveu um projeto intitulado "Máquina de impressão tridimensional com o uso de polímeros" para o INOVA SENAI. A proposta deste projeto pode ser vista no anexo D.

4 CONCLUSÕES

O desenvolvimento das atividades relacionadas neste relatório, envolveram vários conhecimentos adquiridos ao longo da graduação do autor, principalmente das disciplinas Controle Analógico, Controle Digital, Arquitetura de Sistemas Digitais, Eletrônica e Eletrônica de Potência. Isso evidenciou pontos importantes das ementas das disciplinas de graduação, assim como a necessidade de atualização do profissional, pois a tecnológica está em constante progresso. Também mostrou a importância do trabalho em grupo em laboratórios de desenvolvimento tecnológico.

Durante a realização do estágio, o autor encontrou várias dificuldades, sendo as principais o uso de vários *softwares* e conhecimentos práticos não vistos durante a graduação do mesmo.

Por fim, o trabalho realizado durante o período de estágio ajudou o autor a se firmar na profissão de Engenheiro Eletricista, como algo agradável e digno de orgulho do mesmo.

BIBLIOGRAFIA

ASTROM, KARL J., “Computer-Controlled Systems”, Prentice Hall, 1997.

Elipse Software, “Manual do E3 para desenvolvedores”, 2014.

FREITAS, CARLOS M., Protocolo Modbus: Fundamentos e aplicações. Disponível em: http://www.embarcados.com.br/protocolo-modbus/. Acessado em 10/03/2015.

OGATA, KATSUHIKO,”Engenharia de Controle Moderno”, Pearson, 2010.

RASHID, MUHAMMAD H., “Eletrônica de potência: Circuitos, dispositivos e aplicações”, Makron Books, 1999.

SOUZA, VITOR A., O protocolo Modbus. Disponivel em: http://www.cerne-tec.com.br/Modbus.pdf. Acessado em 10/03/2015.

TEIXEIRA, P. R. F.; FARIA R.A., Instrumentista de Controle: Fundamentos de Controle. Universidade Tecnológica Federal do Paraná – UTFPR, Curitiba, 2006.

ANEXO A: CÓDIGOS DESENVOLVIDOS NA PRIMEIRA ATIVIDADE

Parte 1: Biblioteca ModbusSlave:

```
/*******************************Bloco 01*****************************/

Unsignedint Txenpin;


enum{
FC_READ_REGS=0x03,
FC_WRITE_REG=0x06,
FC_WRITE_REGS=0x10
};

/*******************************Bloco 02*****************************/

constunsigned char fsupported[]=
{FC_READ_REGS,FC_WRITE_REG,FC_WRITE_REGS};

enum{
MAX_READ_REGS=0x7D,
MAX_WRITE_REGS=0x7B,
MAX_MESSAGE_LENGTH=256
};


enum{
RESPONSE_SIZE=6,
EXCEPTION_SIZE=3,
CHECKSUM_SIZE=2
};

enum{
NO_REPLY=-1,
EXC_FUNC_CODE=1,
EXC_ADDR_RANGE=2,
EXC_REGS_QUANT=3,
EXC_EXECUTE=4
};

enum{
SLAVE=0,
FUNC,
START_H,
START_L,
REGS_H,
```

```
REGS_L,
BYTE_CNT
};

/*******************************Bloco 03*****************************/

Unsigned int crc(unsigned char *buf,unsigned char start,
unsignedcharcnt)
{
Unsigned char i,j;
Unsigned temp,temp2,flag;

temp=0xFFFF;

for(i=start;i<cnt;i++){
temp=temp^buf[i];

for(j=1;j<=8;j++){
flag=temp&0x0001;
temp=temp>>1;
if(flag)
temp=temp^0xA001;
}
}

temp2=temp>>8;
temp=(temp<<8)|temp2;
temp&=0xFFFF;

return (temp);
}


/*******************************Bloco 04*****************************/

Void build_read_packet (unsigned char slave, unsigned char function,
Unsigned char count,unsigned char *packet)
{
packet[SLAVE]=slave;
packet[FUNC]=function;
packet[2]=count*2;
}

Void build_write_packet (unsigned char slave, unsigned char function,
Unsigned int start_addr,
Unsigned char count,
Unsigned char *packet)
{
packet[SLAVE]=slave;
packet[FUNC]=function;
packet[START_H]=start_addr>>8;
packet[START_L]=start_addr&0x00ff;
packet[REGS_H]=0x00;
packet[REGS_L]=count;
}

Void build_write_single_packet (unsigned char slave,unsigned char function,
```

```
Unsigned int write_addr, unsigned int reg_val, unsigned char *packet)
{
packet[SLAVE]=slave;
packet[FUNC]=function;
packet[START_H]=write_addr>>8;
packet[START_L]=write_addr&0x00ff;
packet[REGS_H]=reg_val>>8;
packet[REGS_L]=reg_val&0x00ff;
}

Void build_error_packet (unsigned char slave, unsigned char function,
Unsigned char exception, unsigned char *packet)
{
packet[SLAVE]=slave;
packet[FUNC]=function+0x80;
packet[2]=exception;
}


/*******************************Bloco 05*****************************/

Void modbus_reply(unsigned char *packet, unsigned char string_length)
{
Int temp_crc;

temp_crc=crc(packet,0,string_length);
packet[string_length]=temp_crc>>8;
string_length++;
packet[string_length]=temp_crc&0x00FF;
}


/*******************************Bloco 06*****************************/

Int send_reply(unsigned char *query,unsigned char string_length)
{
Unsigned chari;

if(Txenpin>1){
UCSR0A=UCSR0A|(1<<TXC0);
digitalWrite(Txenpin,HIGH);
delayMicroseconds(3640);
}

modbus_reply(query,string_length);
string_length+=2;

for(i=0;i<string_length;i++){
Serial.print(query[i],BYTE);
}

if(Txenpin>1){
while(!(UCSR0A&(1<<TXC0)));
digitalWrite(Txenpin,LOW);
}

returni;
```

```
}

/*******************************Bloco 07*****************************/

int receive_request(unsignedchar*received_string)
{
int bytes_received=0;

while(Serial.available()){
received_string[bytes_received]=Serial.read();
bytes_received++;
if(bytes_received>=MAX_MESSAGE_LENGTH)
return NO_REPLY;
}

return (bytes_received);
}

/*******************************Bloco 08*****************************/

int modbus_request (unsigned char slave,unsigned char *data)
{
int response_length;
unsigned int crc_calc=0;
unsigned int crc_received=0;
unsigned char recv_crc_hi;
unsigned char recv_crc_lo;

response_length=receive_request(data);

if(response_length>0){
crc_calc=crc(data,0,response_length-2);
recv_crc_hi=(unsigned)data[response_length-2];
recv_crc_lo=(unsigned)data[response_length-1];
crc_received=data[response_length-2];
crc_received=(unsigned)crc_received<<8;
crc_received=
crc_received|(unsigned)data[response_length-1];
if(crc_calc!=crc_received){
return NO_REPLY;
}

if(slave!=data[SLAVE]){
return NO_REPLY;
}
}
return (response_length);
}

/*******************************Bloco 09*****************************/


int validate_request (unsigned char *data, unsigned char length,
unsigned int regs_size)
{
int i,fcnt=0;
unsigned int regs_num=0;
```

```
unsigned int start_addr=0;
unsigned char max_regs_num;

for(i=0;i<sizeof(fsupported);i++){
if(fsupported[i]==data[FUNC]){
fcnt=1;
break;
}
}
if(0==fcnt)
return EXC_FUNC_CODE;

if(FC_WRITE_REG==data[FUNC]){
regs_num=((int)data[START_H]<<8)+(int)data[START_L];
if(regs_num>=regs_size)
return EXC_ADDR_RANGE;
return 0;
}

regs_num=((int)data[REGS_H]<<8)+(int)data[REGS_L];

if(FC_READ_REGS==data[FUNC])
max_regs_num=MAX_READ_REGS;
elseif(FC_WRITE_REGS==data[FUNC])
max_regs_num=MAX_WRITE_REGS;

if((regs_num<1)||(regs_num>max_regs_num))
returnEXC_REGS_QUANT;

start_addr=((int)data[START_H]<<8)+(int)data[START_L];
if((start_addr+regs_num)>regs_size)
returnEXC_ADDR_RANGE;

return 0;
}


/*******************************Bloco 10*****************************/

int write_regs(unsigned int start_addr,unsigned char *query,int *regs)
{
int temp;
unsigned inti;

for(i=0;i<query[REGS_L];i++){
/* mudar reghi_byte para temp */
temp=(int)query[(BYTE_CNT+1)+i*2]<<8;
/* OR com lo_byte         */
temp=temp|(int)query[(BYTE_CNT+2)+i*2];

regs[start_addr+i]=temp;
}
return i;
}

/*******************************Bloco 11*****************************/
```

```
int preset_multiple_registers (unsigned char slave,
unsigned int start_addr,
unsigned char count,
unsigned char *query,
int *regs)
{
unsigned char function=FC_WRITE_REGS;
int status=0;
unsigned char packet[RESPONSE_SIZE+CHECKSUM_SIZE];

build_write_packet(slave,function,start_addr,count,packet);

if(write_regs(start_addr,query,regs)){
status=send_reply(packet,RESPONSE_SIZE);
}

return (status);
}

/*****************************Bloco 12*****************************/

int write_single_register (unsigned char slave,
unsigned intwrite_addr,unsigned char *query,int *regs)
{
unsigned char function=FC_WRITE_REG;
int status=0;
unsigned int reg_val;
unsigned char packet[RESPONSE_SIZE+CHECKSUM_SIZE];

reg_val=query[REGS_H]<<8|query[REGS_L];
build_write_single_packet(slave,function,write_addr,reg_val,packet);
regs[write_addr]=(int)reg_val;

status=send_reply(packet,RESPONSE_SIZE);

return (status);
}

/*****************************Bloco 13*****************************/


int read_holding_registers (unsigned char slave,unsigned int start_addr,

unsigned char reg_count, int *regs)
{
unsigned char function=0x03;
int packet_size=3;
int status;
unsigned int i;
unsigned char packet[MAX_MESSAGE_LENGTH];

build_read_packet(slave,function,reg_count,packet);

for(i=start_addr;i<(start_addr+(unsignedint)reg_count);
i++){
packet[packet_size]=regs[i]>>8;
packet_size++;
```

```
packet[packet_size]=regs[i]&0x00FF;
packet_size++;
}

status=send_reply(packet,packet_size);

return (status);
}


void configure_mb_slave (long baud,char parity,char txenpin)
{
Serial.begin(baud);

switch (parity){
case 'e':// 8E1
UCSR0C|=((1<<UPM01)|(1<<UCSZ01)|(1<<UCSZ00));
//    UCSR0C &= ~((1<<UPM00) | (1<<UCSZ02) | (1<<USBS0));
break;
case 'o':// 8O1
UCSR0C|=((1<<UPM01)|(1<<UPM00)|(1<<UCSZ01)|(1<<UCSZ00));
//    UCSR0C &= ~((1<<UCSZ02) | (1<<USBS0));
break;
case 'n':// 8N1
UCSR0C|=((1<<UCSZ01)|(1<<UCSZ00));
//    UCSR0C &= ~((1<<UPM01) | (1<<UPM00) | (1<<UCSZ02) | (1<<USBS0));
break;
default:
break;
}

if(txenpin>1){// pino 0 & pino 1 são reservados para RX/TX
Txenpin=txenpin;/* definir variável global */
pinMode(Txenpin,OUTPUT);
digitalWrite(Txenpin,LOW);
}

return;
}

unsigned long Nowdt=0;
unsigned int lastBytesReceived;
const unsigned longT35=5;

int update_mb_slave(unsigned char slave,int *regs,
unsigned int regs_size)
{
unsigned char query[MAX_MESSAGE_LENGTH];
unsigned char errpacket[EXCEPTION_SIZE+CHECKSUM_SIZE];
unsigned int start_addr;
int exception;
int length=Serial.available();
unsigned long now=millis();

if(length==0){
lastBytesReceived=0;
return 0;
```

```
}

if(lastBytesReceived!=length){
lastBytesReceived=length;
Nowdt=now+T35;
return 0;
}
if(now<Nowdt)
return 0;

lastBytesReceived=0;

length=modbus_request(slave,query);
if(length<1)
return length;


exception=validate_request(query,length,regs_size);
if(exception){
build_error_packet(slave,query[FUNC],exception,
errpacket);
send_reply(errpacket,EXCEPTION_SIZE);
return (exception);
}


start_addr=((int)query[START_H]<<8)+
(int)query[START_L];
switch (query[FUNC]){
case FC_READ_REGS:
return read_holding_registers(slave,
start_addr,
query[REGS_L],
regs);
break;
case FC_WRITE_REGS:
return preset_multiple_registers(slave,
start_addr,
query[REGS_L],
query,
regs);
break;
case FC_WRITE_REG:
write_single_register(slave,
start_addr,
query,
regs);
break;
}
}
```

Parte 2: Código desenvolvido para aplicação em Arduino:

```
/*******************************Bloco 01*****************************/
#include <ModbusSlave.h>
ModbusSlaveelipse_E3;
/*******************************Bloco 02*****************************/
enum{
MB_40001,
MB_40002,
MB_40003,
MB_40004,
MB_40005,
MB_40006,
MB_40007,
MB_40008,

MB_REGS=8
};

int regs[MB_REGS];
/*******************************Bloco 03*****************************/
unsigned long wdog=0;//watchdog
/*******************************Bloco 04*****************************/

void setup(){

const unsigned char SLAVE=1;
const long BAUD=9600;
const char PARITY='n';
const char TXENPIN=2;
/*******************************Bloco 05*****************************/

elipse_E3.configure(SLAVE,BAUD,PARITY,TXENPIN);
elipse_E3.setup_regs(regs,MB_REGS);

/*******************************Bloco 06*****************************/
pinMode(A0,INPUT);
pinMode(10,OUTPUT);
pinMode(11,OUTPUT);
pinMode(12,OUTPUT);
pinMode(13,OUTPUT);

digitalWrite(13,LOW);
}

/*******************************Bloco 07*****************************/

void loop(){

if(elipse_E3.update())
wdog=millis();
if((millis()-wdog)>3000);

/*******************************Bloco 08*****************************/

//Registrador 1 lendo a porta analógica A0
```

```
regs[MB_40001]=analogRead(A0);

//Registrador 2 escrevendo na porta 12 do arduino (pode ser feito com if/else)
switch(regs[MB_40002]){
case 1:
digitalWrite(12,HIGH);
break;
case 0:
digitalWrite(12,LOW);
break;
default:
digitalWrite(12,LOW);
}

//Registrador 3 como pwm da porta 11
if(regs[MB_40003]>0){
analogWrite(11,map(regs[MB_40003],0,100,0,255));//Para ocorrer a conversão de valor digitado no elipse e porcentagem de tensão na saída
}

//Registrador 4 lendo o valor digital da porta 13
regs[MB_40004]=digitalRead(13);

//Registrador 5
regs[MB_40005]=100;

//Registrador 6
regs[MB_40006]=110;

//Registrador 7
regs[MB_40007]=120;

//Registrador 8
regs[MB_40008]=130;
}
```

ANEXO B: CÓDIGOS DESENVOLVIDOS NA SEGUNDA ATIVIDADE

Parte 1: Código para Arduino:

```
#include <ElipseMobile.h>
#include <EEPROM.h>

ElipseMobile elipse;

void setup()
{
//Definição das portas a serem usadas
elipse.DigitalTags(14);// 14 digital tags
elipse.AnalogTags(6);//6 analogtags

// Velocidade de bps
Serial.begin(9600);

//Definição de entradas e saídas
pinMode(A0,INPUT);
pinMode(A1,INPUT);
pinMode(A2,INPUT);
pinMode(A3,INPUT);
pinMode(A4,INPUT);
pinMode(A5,INPUT);
pinMode(3,INPUT);
pinMode(4,INPUT);
pinMode(5,INPUT);
pinMode(6,INPUT);
pinMode(7,OUTPUT);
pinMode(8,OUTPUT);
pinMode(9,OUTPUT);
pinMode(10,OUTPUT);
pinMode(11,OUTPUT);
pinMode(12,OUTPUT);
pinMode(13,OUTPUT);
}

void loop()
{

EEPROM.write(1,10);
elipse.ProcessCommands();
}
```

Parte 2: Adição feita a biblioteca ElipseMobile:

```
void AnalogReadCommand()
{
//Receives 1 characters and sends 2 characters
while(Serial.available()<=0)
{
}
char port=Serial.read();//-1 if no data is available
uint16_treadResult=0;
if(port<=4)
{
readResult=analogRead(port);
}
else
{
switch(port > 4){
case 1:
readResult=EEPROM.read(1);
break;
case 2:
readResult=EEPROM.read(2);
break;
}
}
SendAnalogResponse(1,readResult);
}
```

ANEXO C: CÓDIGOS DESENVOLVIDOS NA TERCEIRA ATIVIDADE

Parte 01: Controle por potenciômetro:

```
/*********************************Bloco 01*****************************/

#define TRIAC 4

volatile int nivel;
float resolucao=8333.333/256;
/*********************************Bloco 02*****************************/
Void ativa_TRIAC(){
delayMicroseconds((resolucao*(256-nivel)));
digitalWrite(TRIAC,HIGH);
delayMicroseconds(83);
digitalWrite(TRIAC,LOW);

}//fim ativa_TRIAC
/*********************************Bloco 03*****************************/
void setup(){
pinMode(TRIAC,OUTPUT);
pinMode(A0,INPUT);
attachInterrupt(0,ativa_TRIAC,RISING);

}//fim setup

void loop(){
nivel=map(analogRead(A0),0,1023,5,245);

}//fim loop
```

Parte 02: Controle por PID:

```
//Declaração Modbus
#include <ModbusSlave.h>
//Definição da saida do gate
#define TRIAC 4
//Declaração do mbs
ModbusSlavePID_Luminosidade_E3;
//Declarando os registradores:
enum{
MB_40001,
MB_40002,
MB_40003,
MB_40004,
MB_40005,
MB_40006,

//Número total de registradores
MB_REGS=6
};

int regs[MB_REGS];

unsigned long wdog=0;//watchdog

//Definição das variáveis usadas no PID
double setpoint,sensor;
double kp,kd,ki;
double erro,erro_anterior;
double P,I,D;
double pid;
long ultimo_processo;
float tempo_amostragem;

//Definição das variáveis usadas na interrupção
volatile int nivel;
float resolucao=8333.333/256;

//Definição da função ativa triac
void ativa_TRIAC(){
delayMicroseconds((resolucao*(256-nivel)));
digitalWrite(TRIAC,HIGH);
delayMicroseconds(83);
digitalWrite(TRIAC,LOW);

}//fim ativa_TRIAC

void setup(){

const unsigned char SLAVE=1;
const long BAUD=9600;
const char PARITY='n';
const char TXENPIN=2;

PID_Luminosidade_E3.configure(SLAVE,BAUD,PARITY,TXENPIN);
PID_Luminosidade_E3.setup_regs(regs,MB_REGS);
```

```
pinMode(A0,INPUT);
pinMode(A1,INPUT);
pinMode(10,OUTPUT);
pinMode(TRIAC,OUTPUT);

}//fim setup

void loop(){

//Atualização do loop:
if(PID_Luminosidade_E3.update())
wdog=millis();
if((millis()-wdog)>3000);

//Definição da interrupção
attachInterrupt(0,ativa_TRIAC,RISING);

//Determinação do setpoint, Kp, Ki e Kd
setpoint=regs[MB_40001];
kp=regs[MB_40004];
ki=regs[MB_40005];
kd=regs[MB_40006];

//Definição das portas:
sensor=map(analogRead(A1),0,1023,0,255);

//==============================Bloco 01==========================

//Definição do tempo de amostragem T
tempo_amostragem=(millis()-ultimo_processo)/1000;
ultimo_processo=millis();



//Calculo do erro:
erro=setpoint-sensor;

//Ação proporcional
P=erro*kp;

//Ação integral
I=I+(erro*ki)*tempo_amostragem;

//Ação derivativa
D=((erro-erro_anterior)/tempo_amostragem)*kd;

//Implementação do erro anterior
erro_anterior=erro;


//Definição do pid
pid=P+I+D;

if (pid>245) analogWrite(3,245);

else if (pid<5) analogWrite(3,5);
```

```
else analogWrite(3,pid);

//================================================================


//Declaração dos registradores

regs[MB_40002]=sensor;
regs[MB_40003]=pid;
}//fim loop
```

ANEXO D: PROPOSTA DE PROJETO PARA O INOVA SENAI

MÁQUINA DE IMPRESSÃO TRIDIMENSIONAL COM O USO DE POLIMEROS

Augusto José Silva Firmo
SENAI-CEPSL, Campina Grande, Paraíba
ajsfirmo@gmail.com

Tiago Luis da Silva Oliveira
SENAI-CEPSL, Campina Grande, Paraíba
tiagoluisdasilvaoliveira@yahoo.com.br

Claudiano Sampaio
SENAI-CEPSL, Campina Grande, Paraíba
claudianosampaio@gmail.com

Vinicius de Moraes Nascimento
SENAI-CEPSL, Campina Grande, Paraíba
viniciusdemoraes@fiepb.org.br

Resumo

Apresentar um modelo de impressora 3D com o uso de novos materiais para concepção de protótipos com o uso de polímeros formados por cadeias de monômeros e materiais foto curáveis, unidos a técnicas de memória de forma, este método de fabricação visa aumentar as opções de novos materiais construtivos para a cadeia produtiva além de oferecer materiais com características mecânicas diferenciadas e com o resultado orientado à redução de tempo e custos na fabricação de produtos.

Palavras-chave: Impressão 3D, polímero, foto curável, memória de forma, nanômetro, nanotecnologia.

INTRODUÇÃO

As impressoras 3D tiveram seu surgimento em meados da década de 70 e ficou no anonimato durante muito tempo. Com o advento de novas tecnologias e o surgimento do movimento “faça você mesmo” estas incríveis máquinas tem ganhado cada vez mais mercado. A prototipagem rápida é uma técnica de fabricação que visa a produção de protótipos em pequena escala para facilitar a concepção e testes de novos produtos. Atualmente existem

diversas técnicas de impressão e este projeto propõe o uso de novos métodos e técnicas para melhoria nos processos de prototipação.

DESENVOLVIMENTO

1. A concepção de novas técnicas de produção é um caminho sem volta onde os desafios da indústria e sociedade exigem soluções sem precedentes para resolver problemas de igual magnitude. Este projeto visa a concepção de uma máquina impressão tridimensional com o uso de novos materiais. Para o desenvolvimento deste projeto é necessário o estudo de diversos materiais como polímeros formados por cadeias de monômeros e materiais foto curáveis, unidos a técnicas de memória de forma para que seja possível a criação de novos métodos de construção. Estas técnicas de fabricação são extremamente necessárias para resolver problemas específicos inerentes aos materiais, tempo de fabricação e acabamento.
    1.1. O tempo de impressão é um detalhe essencial neste contexto pois o uso de novas técnicas prometem a redução de tempo de impressão e consequentemente os custos.
    1.2. No quesito acabamento, o uso de resinas seria uma boa alternativa para dispensar as etapas de pôs processamento comuns em alguns processos atuais.
    1.3. O uso de novos materiais visa propor soluções no que tange a características fisicas e mecânicas inerente a resistência, permeabilidade, durabilidade e outros atributos.
2. Os monômeros são compostos formados apenas por uma molécula, a união destes compostos em cadeias moleculares é o que forma o polímero, podendo ser essa união feita de forma linear, ramificada, ou o que queremos, resultar em uma estrutura tridimensional. O processo de fotocura se dá na junção destes monômeros para a formação das principais cadeias que compõem a peça impressa. E o processo de memória de forma irá permitir que a peça seja programada para tomar o formato desejado.

Viabilidade técnico e econômica

Para pesquisa e desenvolvimento do projeto foram incorridos custos operacionais relativos a mão de obra direta, custos indiretos e materiais diretos conforme tabelas abaixo:

Tabela 1- Mão de obra

Item	Unidade	Quantidade	Carga horária (h)	Custo da hora (R$)	Total (R$)
Técnico	Pessoa	4	30	12,00	1.440,00
TOTAL	**R$1.440,00**				

Fonte: Autor

Tabela 2 - Materiais diretos

Insumos	Unidade	Quantidade	Preço unitário (R$)	Total (R$)
Polímero	Pç	2	80,00	160,00
Monômero	Pç	2	20,00	40,00
Laser UV	Pç	1	500,00	500,00
Resina foto curável	Pç	10	30,00	300,00
Insumos para construir impressora 3 eixos	Pç	1	3.980,00	3.980,00
Liga de níquel-titânio	m	10	200,00	200,00
Insumos para construção do canhão de laser	Pç	2	100,00	100,00
ABS	CC	0.2	1280,00	256,00
Placa de circuito impresso	Pç	1	200,00	200,00
Chapa de acrílico	Pç	1	150,00	
TOTAL	**R$ 5.886,00**			

Fonte: Autor

Impactos

O projeto tem como uma de suas principais vantagens o uso de materiais polímeros no processo de impressão, o que irá vir a resolver problemas encontrados atualmente na questão de resistência, permeabilidade, durabilidade e outras características mecânicas dos materiais usados em outras impressoras 3D atualmente.

A união entre o depósito de material polímero e seu processo de cura, faz desnecessário o uso de materiais de suporte, reduzindo seu tempo de impressão, tornando isso outro fator bastante positivo, já que esse tempo é um dos fatores que mais pesa no custo final de peças impressas.

O tamanho reduzido na impressão base torna esse sistema ideal para ambientes onde o espaço é reduzido e comprometido, como aplicações navais ou espaciais além da grande facilidade no transporte e armazenamento de peças.

A principal desvantagem encontrada é o custo do material usado na impressão, que seria superior aos materiais mais comum em impressoras mais simples.

Conclusão

Com advento da nanotecnologia é sabido que existirá cada vez mais possibilidades para construção de novos produtos a partir da disponibilidade de novos materiais, com esta afirmação podemos concluir que, existem uma serie de alternativas para trabalhar as técnicas de construção, entretanto o referido projetos irá tratar apenas da manipulação destes materiais a um nível macro fazendo o uso de tecnologias como catalisação de polímeros de cadeias curtas foto cura de resinas e o uso de materiais com propriedades de memorização.

Referências

SALMORIA, GEAN V. Prototipagem rápida por impressão 3d com resinas foto curáveis: uma análise sobre as tecnologias disponíveis no mercado nacional. Universidade Federal de Santa Catarina, 2007.

MANO, E. B.; MENDES, L. C. Introdução a polímeros. – 2. ed. rev. e ampl. –
São Paulo: Edgard Blucher, 2004.

CANEVAROLO Jr., Sebastião V. Ciência dos polímeros: um texto básico para tecnólogos e engenheiros /Sebastião V. Canevarolo Jr. - São Paulo: Artliber Editora, 2002.

FERNANDES, F. M. B. Ligas com memória de forma. Departamento de Ciência dos Materiais / CENIMAT, Universidade Nova de Lisboa, 2003.

MACHADO, L.G. ; SAVI, M.A. Aplicações odontológicas das ligas com memória de forma. Revista Brasileira de Odontologia. vol. 59, n. 5, pp. 302-306, 2002.

OTSUKA K.; WAYMAN C. M. Shapememorymaterials. Cambridge University Press, 1998.