UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
DEPARTAMENTO ACADÊMICO DE MECÂNICA
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

GUILHERME SCHUSTER MONTEIRO
TASSIO PETRY

# DESENVOLVIMENTO DE PROTÓTIPO PARA ENSAIOS DE TÉCNICAS DE CONTROLE PROGRAMADAS EM MICROCONTROLADOR

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2012

GUILHERME SCHUSTER MONTEIRO
TASSIO PETRY

# DESENVOLVIMENTO DE PROTÓTIPO PARA ENSAIOS DE TÉCNICAS DE CONTROLE PROGRAMADAS EM MICROCONTROLADOR

Trabalho de Conclusão de Curso de graduação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Mecatrônica Industrial dos Departamentos Acadêmicos de Eletrônica e Mecânica – DAELN e DAMEC - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. M.Sc. Guilherme Alceu Schneider

CURITIBA

2012

GUILHERME SCHUSTER MONTEIRO

TASSIO PETRY

# DESENVOLVIMENTO DE PROTÓTIPO PARA ENSAIOS DE TÉCNICAS DE CONTROLE PROGRAMADAS EM MICROCONTROLADOR

Este trabalho de conclusão de curso foi apresentado no dia 26 de novembro de 2012, como requisito parcial para obtenção do título de Tecnólogo em mecatrônica industrial, outorgado pela Universidade Tecnológica Federal do Paraná. Os alunos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado

_____

Prof. Milton Luiz Polli
Coordenador de Curso
Departamento Acadêmico de Mecânica

_____

Prof. Décio Estevão do Nascimento
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

**BANCA EXAMINADORA**

_____

Prof. Ph.D. Hugo Vieira Neto

_____

Prof. M.Sc. Guilherme Alceu Schneider
Orientador

_____

Prof. M.Sc. Valmir de Oliveira

(Obs: O Documento original contendo as assinaturas encontra-se em posse da coordenação).

# RESUMO

PETRY, Tassio Vinícius; MONTEIRO, Guilherme Schuster. Desenvolvimento de protótipo para ensaios de técnicas de controle programadas em microcontrolador. 2012. 81 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Mecatrônica Industrial) – Departamento Acadêmico de Eletrônica, Departamento Acadêmico de Mecânica, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

O presente trabalho tem por objetivo elaborar o Protótipo para Ensaio de Técnicas de Controle Programadas em Microcontrolador. Esse controle se dará por meio da parametrização de valores de entrada - comumente utilizados em controles Proporcional, Integral, Derivativo (PID), com a combinação de diferentes formas de discretização. O protótipo conta com uma interface no *software* Elipse E3 passível de comunicação com o *hardware* e sua programação se faz através da utilização do *software* de programação MikroC, implementado no microcontrolador (PIC) 18F4520 sendo utilizada a linguagem C para o seu desenvolvimento. O projeto utiliza o modbus *Remote Terminal Unit* (RTU) para realizar a comunicação e o *driver* modbus RTU para comunicar-se com o *software* supervisório Elipse.



**Palavras-chave**: Técnicas de controle programadas em microcontrolador. Controle discretizado por meio de microcontroladores.

## ABSTRACT

PETRY, Tassio Vinícius; MONTEIRO, Guilherme Schuster. Desenvolvimento de protótipo para ensaios de técnicas de controle programadas em microcontrolador. 2012. 81 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Mecatrônica Industrial) – Departamento Acadêmico de Eletrônica, Departamento Acadêmico de Mecânica, Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

The purpose of the present work is the design of a prototype for experiments using control techniques programmed in microcontrollers. This control will be done by parameterizing the entrance values commonly used in PID controllers with the combination of different ways of discretization. The prototype has an interface developed in Elipse E3 which enables communication with the hardware. The project's programming is done utilizing the software MikroC in a PIC 18F4520 with C as programming language. This Project utilizes modbus RTU to communicate and the modbus RTU *driver* to communicate with Elipse.



**Key words**: Control techniques programmed in microcontrollers. Discretized control in microcontrollers.

# LISTA DE FIGURAS

# LISTA DE TABELAS

**SUMÁRIO**

# 1 INTRODUÇÃO

Em meados dos anos 30 do século XX, matemáticos desenvolveram um algoritmo chamado controle PID (Proporcional, Integral, Derivativo). Contudo, no panorama internacional, os controladores PID industriais surgiram a partir dos anos 80, quando os microprocessadores conseguiram atingir uma capacidade de processamento e memória capazes de executar alguns algoritmos PID. No Brasil, apenas em 1993 uma empresa nacional lançou um equipamento semelhante, porém limitado se comparado aos produtos estrangeiros. Hoje, são possíveis de se encontrar no mercado, controladores com vários tipos de algoritmos de controle, inclusive com autoajuste de parâmetros por métodos de sintonia variados, ficando a critério do técnico a escolha dos mesmos (CAON 1999).

O PID é usado nas indústrias para o controle dos processos. Tais controles são feitos em malha fechada, pois a resposta do sistema retorna ao controle sob forma de erro, ou seja, a subtração entre valor desejado e valor medido. Para exemplificar apresenta-se um controle contínuo de um tanque de água, como demonstrado na figura 1.

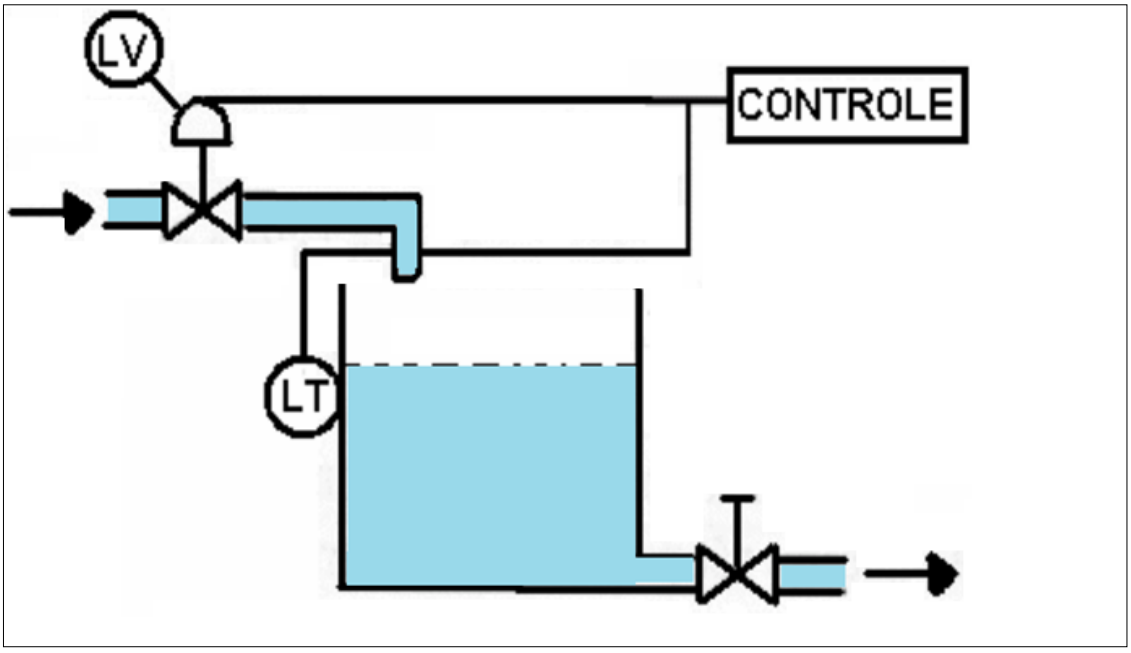


**Figura 1 - Tanque de nível, exemplo de controle contínuo**
**Fonte: SCHNEIDER (2011, p. 5).**

No tanque observa-se a válvula de nível LV para enchimento, outra válvula, para esvaziamento - a qual está fora da malha de controle e o sensor de nível LT. Desconsiderando a pressão da água, com as duas válvulas abertas no mesmo percentual e supondo-se válvulas iguais, tem-se a estabilidade do sistema, onde o montante de água que sai é o mesmo que entra. Porém, assume-se que o tanque esteja vazio e a válvula não controlada não tem sua abertura alterada, permanecendo constante em um percentual entre 0 e 100%. A partir do momento em que o controle percebe que o nível do tanque está vazio, ele atua na válvula LV, que é manipulada de forma a permitir a passagem total do líquido até que o sensor LT reconheça o tanque cheio. Quando LT detectar o tanque cheio, o controlador irá reduzir a abertura da válvula manipulada, fazendo com que o fluxo de entrada e saída de água se estabilize.

Entretanto, alguém pode alterar a abertura da válvula que não está presente no controle. Nesse momento, o sensor LT percebe que houve uma variação e envia a informação para o controle. O controle gera um sinal de correção para a válvula LV e essa dinâmica ocorre até a estabilização do sistema.

Diante disso, torna-se importante a descrição de duas topologias, em Malha Aberta e em Malha Fechada.

Sistema de controle em malha aberta é o nome dado aos sistemas nos quais o sinal de saída não causa nenhuma interferência na ação de controle a ser tomada, ou seja, não há a retroação e não há o sinal de erro, há somente o sinal de entrada. Ao receber o sinal de entrada a ação é executada, porém o resultado não repercute em uma nova ação, sendo assim o processo torna-se dependente de uma prévia calibração. Exemplos de controle em malha aberta também são muito simples. A máquina de lavar roupas não verifica o coeficiente de limpeza realizado, executando o ciclo programado independentemente da quantidade de sabão, por exemplo. A figura 2 mostra um exemplo de um sistema de controle à malha aberta (OGATA, 1998).

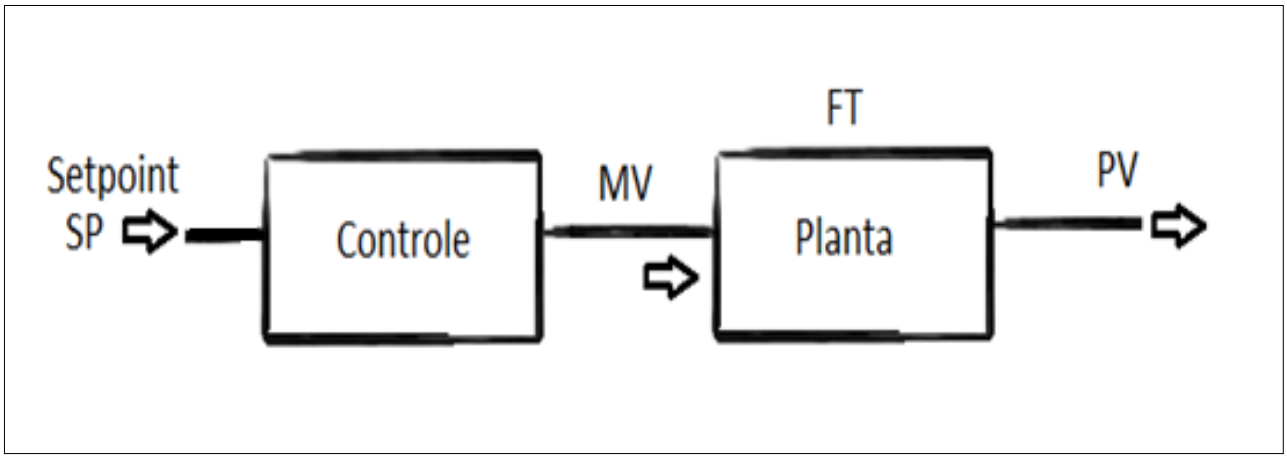


**Figura 2 - Controle de Malha Aberta**
**Fonte: Autoria Própria.**

Segundo Cruz (2004), nos sistemas de controle em malha fechada, a saída é utilizada para alterar o controle, por isso é sinônimo de sistemas com realimentação. O controlador é um dispositivo que usa o erro do comparador, entre o valor desejado (*setpoint*) e o valor real. Um exemplo disso é a geladeira doméstica. Por meio do botão de regulagem de temperatura, a geladeira mantém-se fria, a despeito das perturbações externas, tais como variações da temperatura ambiente.

Sistemas de controle a malha fechada são sistemas de controle com retroação, isto é, o sistema mantém uma relação entre a saída e a referência. O erro é obtido pela subtração entre o sinal de saída do processo e o sinal de entrada. A diferença entre o sinal de entrada e o sinal de retroação, excita o controlador de modo que este atue no processo com o objetivo de reduzir cada vez mais o próprio erro, trazendo o valor do sinal de saída para o valor desejado (OGATA, 1998).

Existem inúmeros exemplos de controles a malha fechada em diversas áreas (controle de temperatura, vazão, refrigeração, pressão, entre outros). O sistema de refrigeração de *notebooks* é um exemplo. Sabendo-se que existe uma faixa de temperatura segura para o funcionamento do computador e supondo que a faixa de 30 a 40ºC seja ideal para o funcionamento, o controle não acionará o ventilador caso a temperatura esteja nesta faixa, porém se ultrapassar os 40ºC a diferença entre a faixa e o sinal de saída explicitará um valor positivo e o ventilador será acionado para corrigir

essa desconformidade (OGATA, 1998). A figura 3 mostra um sistema de controle à malha fechada.

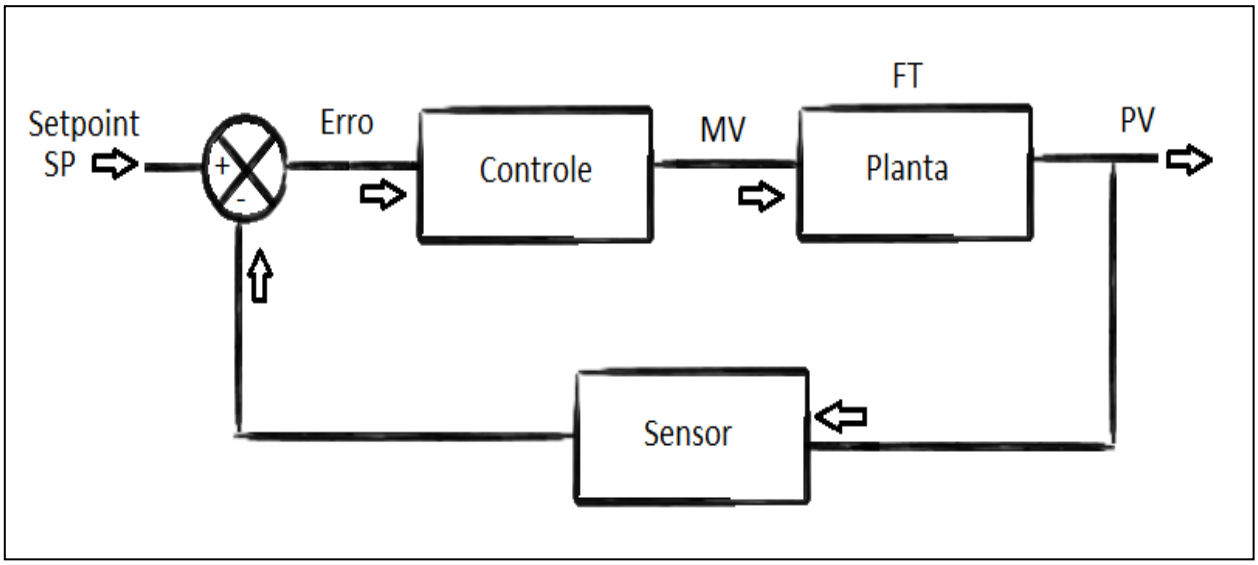


**Figura 3 - Controle de Malha Fechada**
**Fonte: Adaptado de OGATA (1998, p. 54).**

Visto isso, passa-se para a análise do distúrbio.

O distúrbio causa uma variação indesejada no processo. Em malha fechada, sendo um sistema de refrigeração, o distúrbio pode ser criado a partir e uma fonte de calor que elevará a temperatura indesejadamente.

## 1.1 Objetivo Geral

Desenvolver um sistema (*hardware*, *firmware* e a planta didática) para utilizar como ferramenta de ensaio de controles PID industriais.

## 1.2 Objetivos Específicos

- Pesquisar sobre arranjos PID;

- Desenvolver um *hardware*;

- Estudar a discretização de alguns arranjos PID;

- Desenvolver um *firmware* para os arranjos PID;

- Desenvolver a estrutura da planta;

- Desenvolver o modelo didático da planta;

- Desenvolver a interface de comunicação *hardware* – PC;

- Realizar ensaios para levantamento de dados quantitativos a respeito dos PID;

## 1.3 Problema

A motivação presente na elaboração da ferramenta acadêmica de ensaios de controles utilizando PID microcontrolados é a de demonstrar o funcionamento dos arranjos de PID na prática e aplicar os processos de discretização, buscando mostrar como pode ser feita a discretização do PID e sua respectiva aplicação em um processo real. Há também a demonstração didática dos arranjos de PID para aperfeiçoar os estudos dos acadêmicos e interessados na área, assim como há a demonstração da integração dos PID com o sistema supervisório que são corriqueiramente implementados nas indústrias. A escassez de plantas trabalhadas com PID para ensaio e a não prática do desenvolvimento de algoritmos de PID também motivaram a elaboração do trabalho.

## 1.4 Justificativa

O trabalho visa a difundir o conhecimento dos controladores PID, dos programas supervisórios e da integração entre as áreas para todos os interessados, bem como fornecer base para futuros estudos acerca dos processos utilizados no projeto. Também faz parte do alcance do projeto a utilização de PID em microcontroladores - matéria que não foi estudada na graduação. Com o kit será possível entender melhor o comportamento de um algoritmo PID embarcado em uma malha de controle fechada.

A Figura 4 mostra o Projeto a ser desenvolvido com as respectivas atuações explicadas.

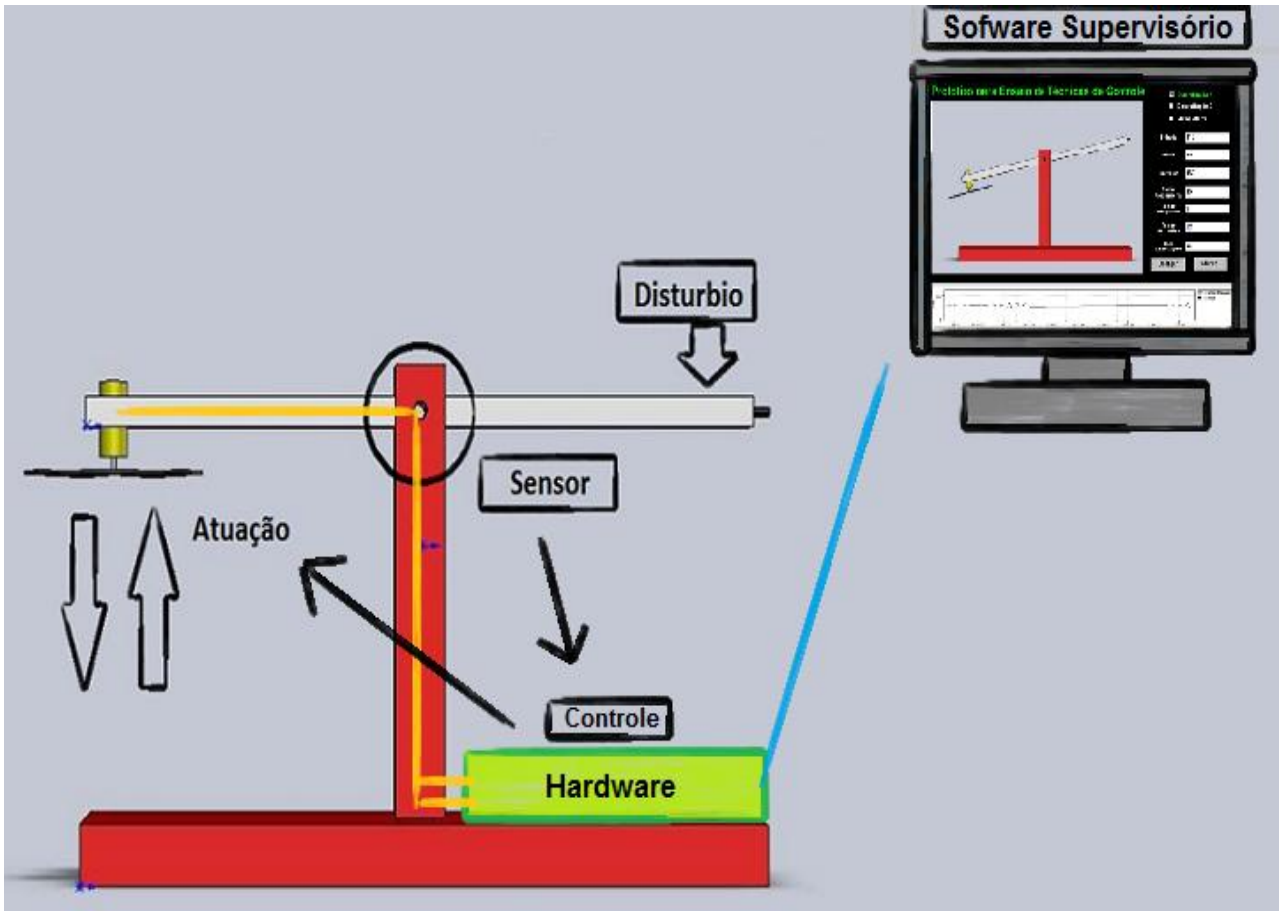


**Figura 4 - Esquemático do Projeto Desenvolvido**
**Fonte: Autoria Própria.**

O controle será feito através de um microcontrolador e fará com que o ventilador equilibre a balança, atuando de maneira proporcional ao distúrbio. O sensor, que é um potenciômetro, será responsável por traduzir de forma correta para o controle a posição da balança. Com a ocorrência de algum distúrbio na outra ponta da balança ela se desestabilizará e dependerá da atuação do controle para buscar o equilibro novamente.

Todo o processo mencionado será transmitido ao operador, o qual estará comandando o *software* supervisório no computador. Este operador observará o desempenho do processo, assim como poderá efetuar mudanças a partir do supervisório. A comunicação será feita por meio do protocolo modbus.

## 1.5 Estrutura do Trabalho

O capítulo 1 traz a Introdução, a qual trata dos aspectos iniciais do trabalho, bem como mostra os objetivos, problemas e justificativas para o desenvolvimento do protótipo. Também na introdução são dadas breves explicações sobre os tipos de malhas existentes e os tipos envolvidos no projeto.

O capítulo 2 mostra a fundamentação teórica do presente trabalho. São vistos conceitos aplicáveis às malhas de controle, tais como a função degrau. Outro aspecto abordado são os componentes do sistema de controle como *setpoint*, erro, variáveis controladas e manipuladas, entre outros. São aspectos que tem relação direta com o tipo de malha utilizada no presente projeto, no caso, a malha fechada. São vistos também os critérios de análise das respostas transitórias do sistema, como o tempo de atraso (Td), o tempo de subida (Ts) e o tempo de pico (Tp), por exemplo, os quais são podem ser observados na curva de resposta e servem de análise para melhorar a regulagem do sistema. Além disso, pode-se observar no capítulo 2 as diferentes ações de controle do PID e os métodos de sintonia de CHR e Ziegler-Nichols, e suas respectivas tabelas para a construção de parâmetros Kp, Ti e Td otimizados. No item 2.6 são apresentados 5 tipos diferentes de arranjos discretizados de PID. São apresentados os diagramas em bloco dos PID e gráficos com as respostas dos arranjos para respostas em degrau e com parâmetros específicos. Além disso, são demonstradas as equações respectivas a cada arranjo e o método utilizado para realizar a discretização. No último tópico do capítulo, é apresentada a comunicação modbus com um breve histórico e uma pequena explicação acerca dos pacotes enviados e recebidos e seus modos de envio e recebimento.

O capítulo 3 contém informações sobre o desenvolvimento do protótipo, abordando itens como: estrutura mecânica, *hardware*, programação, comunicação, *software* supervisório e o funcionamento do projeto, sendo que o *hardware* e a programação possuem diagramas que são complementados pelos apêndices presentes neste trabalho. Este capítulo também conta com a definição e uma breve explicação dos componentes utilizados e suas funcionalidades no projeto. Quanto à comunicação, tem-se sua programação anexada ao apêndice, como também há figuras para melhor entendimento do procedimento de comunicação. O item 3.5 mostra o desenvolvimento

do *software* supervisório e algumas telas para observação e estudo dos gráficos gerados pelo protótipo. O último item do capítulo trata do funcionamento do projeto com sua explanação.

No capítulo 4 encontra-se o levantamento de gráficos de testes de dois tipos diferentes de arranjos de PID e suas análises em relação a dois *setpoint*s diferentes, bem como a análise de respostas de uma malha aberta do sistema. Por fim, os testes visam mostrar diferenças em relação aos PID utilizados.

No capítulo 5 apresenta-se a conclusão do presente trabalho e propostas de possíveis trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo serão abordadas a função degrau e os componentes de um sistema - tais como *setpoint*, erro, variável controlada e variável manipulada, entre outros. Também consta os critérios de análise da resposta transitória, como tempo de atraso e tempo de subida; e os modos de controle do controlador PID. Além disso, são vistos os métodos de sintonia, tais como o primeiro e segundo métodos de Ziegler-Nichols, e o método CHR; os arranjos de PID: controlador PID em arranjo paralelo ideal, controlador PID em arranjo série, controlador PID em arranjo paralelo ideal com filtro derivativo, controlador PI-D em arranjo série com filtro derivativo e controlador PID em arranjo série com filtro derivativo; e a Comunicação modbus RTU e seus respectivos pacotes de envio e recebimento de dados. Entretanto todos esses aspectos serão explanados e detalhados no decorrer do capítulo.

## 2.1 Função Degrau

A função degrau representa a mudança do valor de uma variável, que pode ser para mais ou para menos. A alteração da temperatura desejada em um aquecedor pode ser representada por uma função degrau: Ao mudar de 20ºC para 22ºC teremos um degrau de 2ºC, sendo a função degrau uma transição instantânea e discreta. (Ogata, 1998)

O gráfico da função degrau é demonstrado na figura 5:

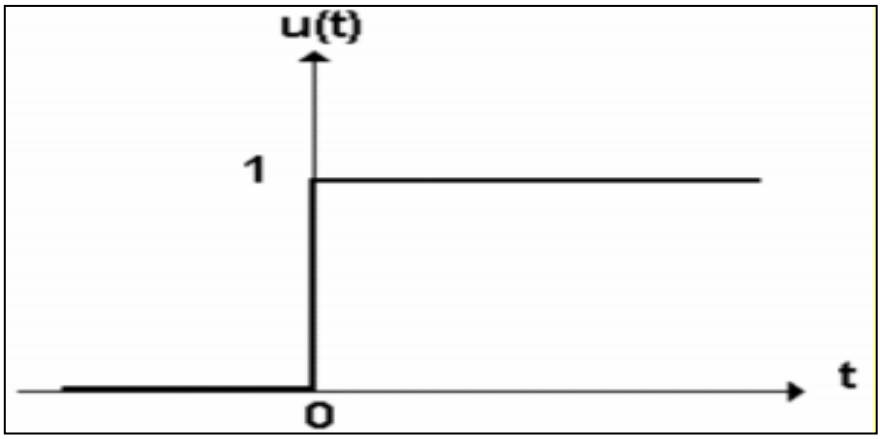


**Figura 5 - Função Degrau**
**Fonte: SCHNEIDER (2011, p. 12).**

## 2.2 Componentes de um Sistema de Controle:

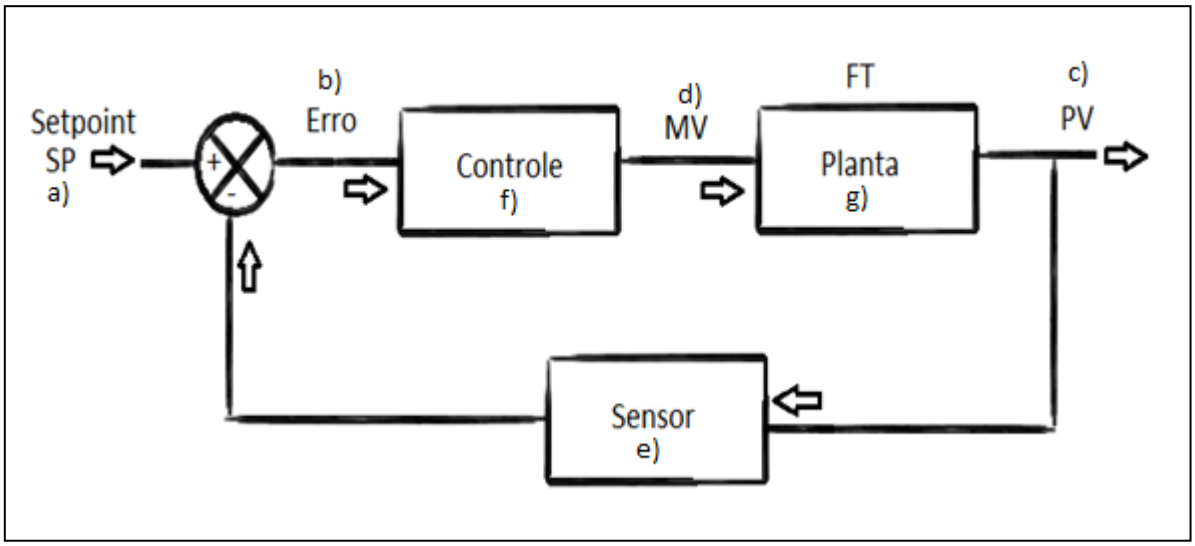


**Figura 6 - Malha fechada sob análise**
**Fonte: Adaptado de OGATA (1998, p. 55).**

Passa-se à análise dos componentes de um sistema de controle como demonstrado na figura 6:

a) *Setpoint* (SP)

O *setpoint* é a variável de entrada, é o valor de referência do controlador, (valor desejado) que é utilizado para o cálculo do erro. Em uma temperatura que possa variar de 0ºC a 200ºC, supondo variações lineares e a variação do *setpoint* de 0 a 100, para cada variação de 1% no *setpoint* teremos 2ºC variados. Iremos obter uma reta partindo da Origem onde SP = 0 e T =0 variando 2ºC a cada 1% aumentado, então se desejarmos uma temperatura de 86ºC, neste exemplo, o *setpoint* será de 43% (OGATA, 1998).

b) Erro

O erro é a variável calculada entre a diferença do sinal referência (*setpoint*) e a saída do sistema. Esse sinal é usado para se determinar como a ação de controle irá atuar no sistema para que variável de saída fique o mais próximo possível da referência. Ainda utilizando o exemplo, caso se deseje 86ºC o *setpoint* será 43% e nesse momento tem-se uma temperatura de 20ºC, ou seja, 10%. O valor do erro será o

valor do *setpoint* subtraído do valor da temperatura e, calculando esse valor, chega-se a 33%. Esse sinal de 33% será usado para gerar uma correção a fim de atingir a temperatura desejada (OGATA, 1998).

c) Variável Controlada (PV)

A variável PV é variável de processo, *process variable* (PV) ou variável controlada, entendida como, por exemplo, a pressão, o nível, a temperatura, vazão, velocidade, posição ou qualquer que seja a variável que se controle. Ela muda de acordo com a variação na entrada e sua representação numérica é usada no cálculo do erro pela formula E= SP-PV ou E=PV-SP dependendo do controle em questão. A PV é o objetivo do controle. No caso do exemplo anterior a PV é a temperatura (OGATA, 1998).

d) Variável Manipulada (MV)

A variável manipulada, *manipulated variable* (MV), é a variável de saída do controlador, é a variável que atua na planta. É chamada de manipulada porque, por exemplo, quando em malha aberta é a única que atua na planta e que é literalmente manipulada, sem a necessidade de prévio processamento ou de ajustes. Portanto é possível perceber que para o controle da variável controlada (controle indireto) é necessário a variável manipulada (controle direto) (OGATA, 1998).

e) Sensor

O sensor é o dispositivo responsável por mensurar a saída e informar de forma precisa qual o valor medido. Essa informação é comparada ao SP gerando o erro que é o sinal processado pelo controle. O sensor, peça fundamental da malha, deve estar sempre calibrado, pois deve enviar o valor medido precisamente (OGATA, 1998).

f) Controle

O controle é o protagonista no sistema e para que este funcione adequadamente são utilizados os controladores industriais. Neste sentido existe o controlador PID que executa ações proporcional, integral e derivativa que serão abordadas na seção 2.2. A sua função está em atuar no processo e buscar a estabilização do sistema (OGATA, 1998).

g) Planta

A planta é o conjunto de equipamentos que fazem parte do processo. Sua representação pode ser feita em uma equação no plano S. Quando a planta recebe o sinal do controlador, esta envia a resposta de acordo com sua equação, a qual é a fiel reprodução do processo e assim pode-se afirmar que é impossível duas equações de plantas serem idênticas devido aos diversos fatores que influenciam na sua composição. A planta é representada por meio de uma função de transferência (OGATA, 1998).

Função de transferência é uma função que representa as relações de entrada e saída dos componentes ou sistemas, que podem ser descritos através de equações lineares - representa a união das equações relativas a cada equipamento do sistema. A função de transferência de um sistema representado por equações diferenciais lineares invariantes no tempo é definida como a relação entre a transformada de Laplace do sinal de saída (função resposta) e a transformada de Laplace do sinal de entrada (função excitação), na hipótese de que todas as condições iniciais são nulas (OGATA, 1998). A título de exemplo, a figura 7 e as equações 1 a 7 apresentam a Função de Transferência de um carrinho, que neste caso será a planta do nosso sistema.

A função de transferência é uma equação que relaciona algebricamente a saída de um sistema à sua entrada. Essa função permite combinar algebricamente representações matemáticas de vários subsistemas para obtenção do sistema total. (NISE, 2000)

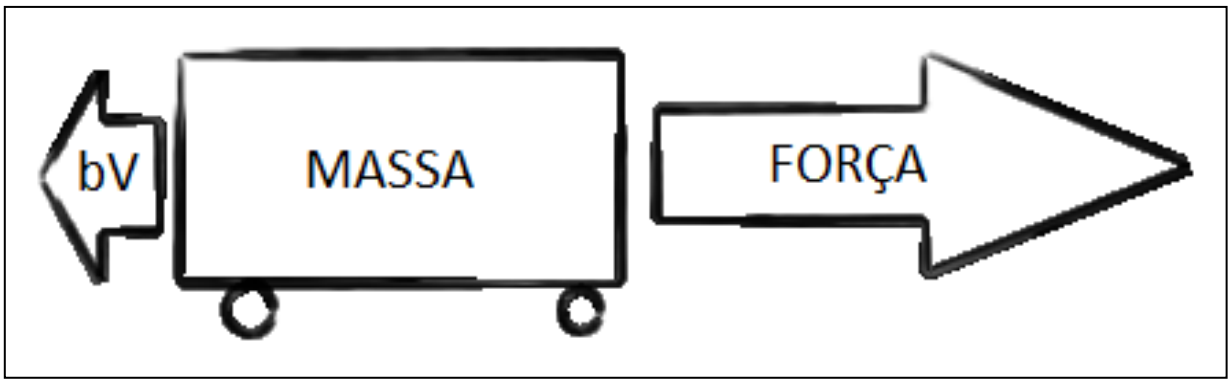


**Figura 7 - Exemplo de uma planta de referência para geração de uma função de transferência – situação real**
**Fonte: Adaptado de SCHNEIDER (2011, p. 12).**

Nesse exemplo, sabe-se que para a locomoção do carrinho é necessária uma força (F) suficiente para movê-lo apesar de sua massa (m) e de sua resistência (b), a aceleração (a) e a velocidade (V) (SCHNEIDER, 2011). Equações (1) a (6) propostas por Schneider para a resolução da planta do carrinho (2011).

Tem-se que:

$$\boldsymbol{F(t) - \ bV(t) = m.a(t)} \quad (1)$$

E sabe-se que:

$$\boldsymbol{a(t) = \frac{dv(t)}{dt}} \quad (2)$$

Então:

$$\boldsymbol{F(t) - bV(t) = m.\frac{dv(t)}{dt}} \quad (3)$$

Transpondo a equação para o plano S:

$$\boldsymbol{\frac{dx(t)}{dt} = s.X(s)} \quad (4)$$

$$\boldsymbol{m.s.V(s) + \ bV(s) = F(s)}$$

Considerando ***F(s)*** a excitação e ***V(s)*** a Resposta:

$$\boldsymbol{H(s) = \frac{V(s)}{F(s)} = \frac{1}{ms+b}} \quad (5)$$

Assim, se a massa for 1000 e a resistência for 50, teremos:

$$\boldsymbol{H(s) = \frac{1}{1000s+50}} \quad (6)$$

A figura 8 mostra a malha fechada (MF) do sistema de controle de velocidade para o “carrinho”. Considerando o carrinho com massa de 1000 kg e um coeficiente de atrito b de 50. PV representa a velocidade do carrinho e MV representa a força. O sensor mede a velocidade do carrinho e realimenta o sistema (SCHNEIDER, 2011).

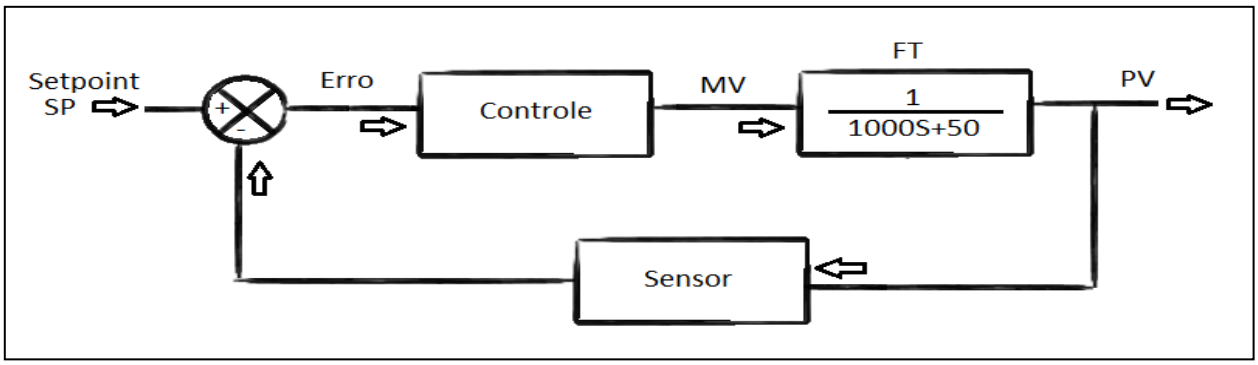


**Figura 8 - Sistema em malha fechada do controle do movimento da massa**
**Fonte: Adaptado de SCHNEIDER (2011, p. 12).**

## 2.3 Critérios de Análise

Para analisar a resposta transitória do sistema, são analisados os seguintes aspectos representados na figura 9. Para a análise, é aplicada uma função degrau como excitação:

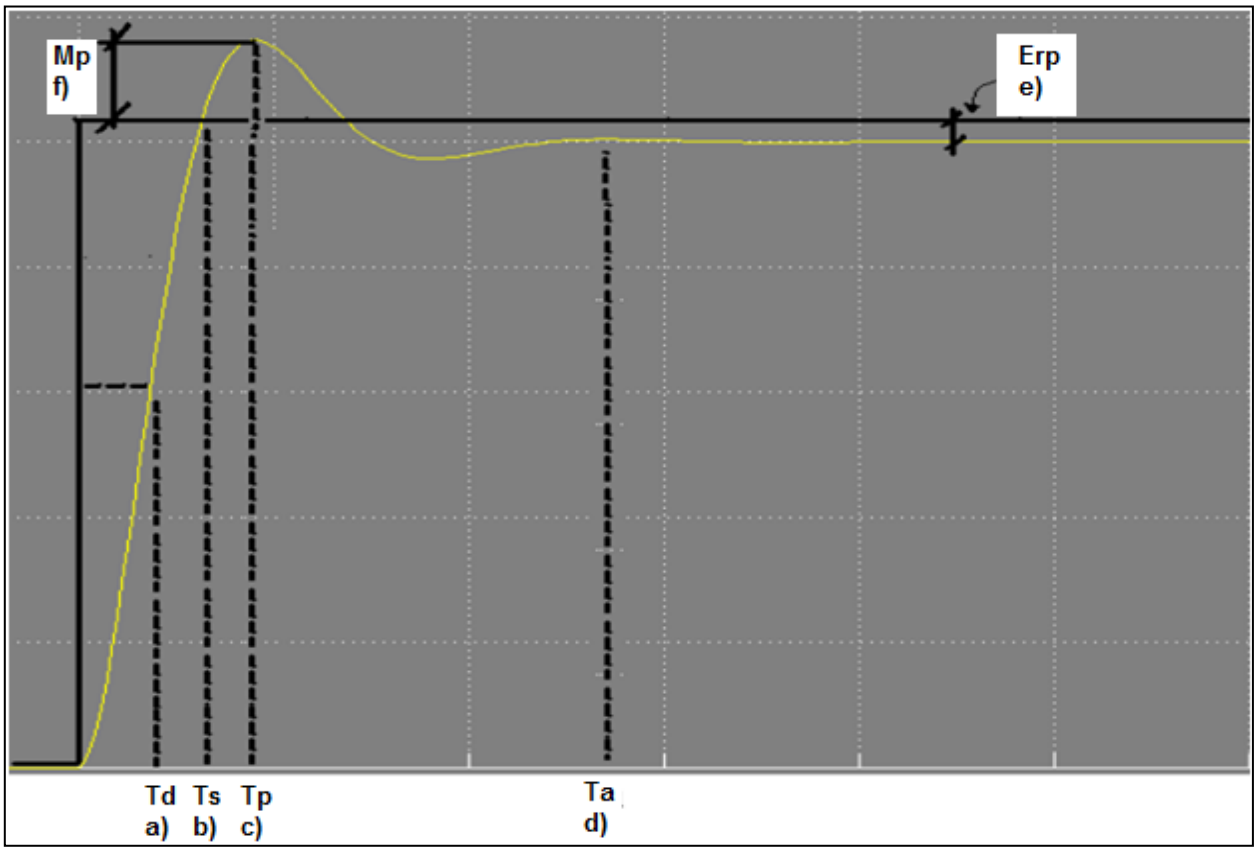


**Figura 9 - Análise da Resposta Transitória**
**Fonte: Adaptado de SCHNEIDER (2011, p. 16).**

a) Tempo de atraso, (Td) (*delay time*).

O tempo de atraso é o tempo necessário para o sistema atingir a metade do valor da sua acomodação final. Se o sistema se estabiliza com o valor de 70, o Td é o tempo necessário para atingir o valor de 35 (OGATA, 1998).

b) Tempo de subida (Ts) ou (Tr) (*rise time*).

O tempo de subida é o tempo necessário para o sistema chegar a 100% do valor da sua acomodação final. (OGATA, 1998).

c) Tempo de pico, (Tp) (*peek time*).

O tempo de pico é o tempo necessário para o sistema chegar ao seu primeiro pico no sobressinal (OGATA, 1998).

d) Tempo de Acomodação (Ta).

O tempo de acomodação é o tempo necessário para que a curva de resposta se estabilize, e pare de oscilar. Pode-se adotar faixas para definir o sistema acomodado, como por exemplo, faixas de 2% a 5% (OGATA, 1998).

e) Erro em Regime Permanente (Erp).

Segundo Cruz (2004), o desempenho dos sistemas de controle pode ser especificado não somente na resposta transitória, mas também no erro estacionário em relação aos sinais de referência como o degrau.

O erro em regime permanente é a diferença obtida entre o momento de estabilidade de duas curvas: a desejada e a obtida. Quando o erro em regime permanente é diferente de 0 conclui-se que o sistema não conseguiu cumprir sua função de atingir o ponto desejado. Assim, devem-se fazer novos ajustes no controle PID (OGATA, 1998).

f) *Overshoot* (Mp)

O *overshoot* Mp ou sobressinal representa o valor ultrapassado, no primeiro momento, do sinal de resposta. Somente é levado em consideração o primeiro

momento porque o sistema tende a se estabilizar, portanto em um segundo momento o *overshoot* poderá ser menor. O sobressinal é indesejado pelo fato de causar um impacto que pode desregular a resposta do sistema e torná-lo instável. (CRUZ, 2004).

## 2.4 Modos de Controle do Controlador PID

As ações do controlador PID são: Ação Proporcional, Ação Proporcional Integral, Ação Proporcional Derivativa, Ação Proporcional Integral Derivativa. A figura 10 apresenta a malha fechada, estando o PID presente no bloco de Controle. As equações dos Blocos representam as ações Proporcional (P), Integral (I) e Derivativa (D).

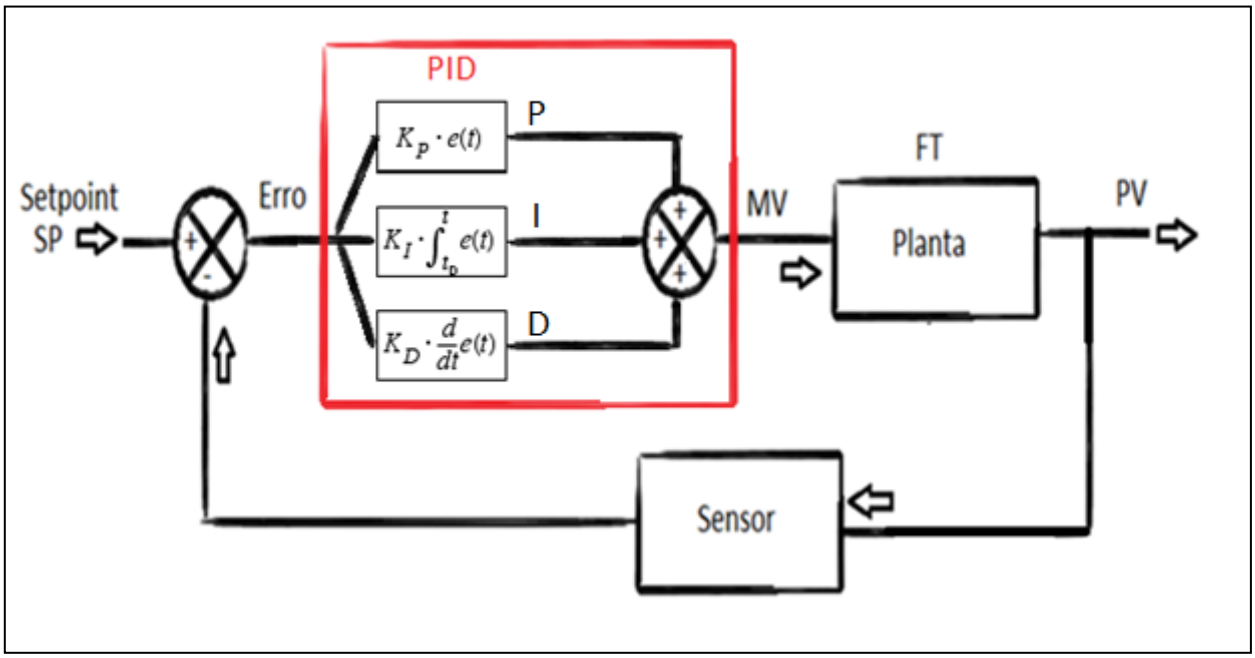


**Figura 10 - Exemplo em Álgebra de blocos da aplicação do controle PID**
**Fonte: Adaptado de OGATA (1998, p. 545).**

a) Ação Proporcional

Segundo Bazanella (2000), a ação proporcional realiza um controle diretamente proporcional ao desvio percebido. Por exemplo, caso o ganho do controle seja 2 a ação duplicará o valor do erro. Assim, uma variação na entrada de 10 resultaria numa variação da saída de 20. Essa ação será dada pela fórmula $c(t) = Po \ +/-Kp\,.e(t)$ em que c(t) representa a correção em função do tempo, e(t) é o erro em função do tempo,

Kp é o ganho proporcional do controle e Po é a variável de processo. Percebe-se nas figuras 12 e 13 que a ação proporcional não elimina o erro em regime permanente. Na figura 12 é utilizado um ganho proporcional baixo. Na mudança de setpoint, a saída fica longe do desejado. Na figura 13 é aplicado um ganho proporcional alto, o sistema responde com um *overshoot* alto e aproxima-se mais do sinal desejado, porém não há a eliminação do erro em regime permanente. A equação (7) representa matematicamente o controle proporcional, sendo (c) a saída do controlador e erro (e)

$$\boldsymbol{Kp} = \frac{\boldsymbol{C(S)}}{\boldsymbol{E(S)}} \qquad \boldsymbol{c(t)} = \boldsymbol{Kp}.\boldsymbol{e(t)} \qquad (7)$$

**Equação do controle Proporcional**

Efeito do controle proporcional em malha aberta:

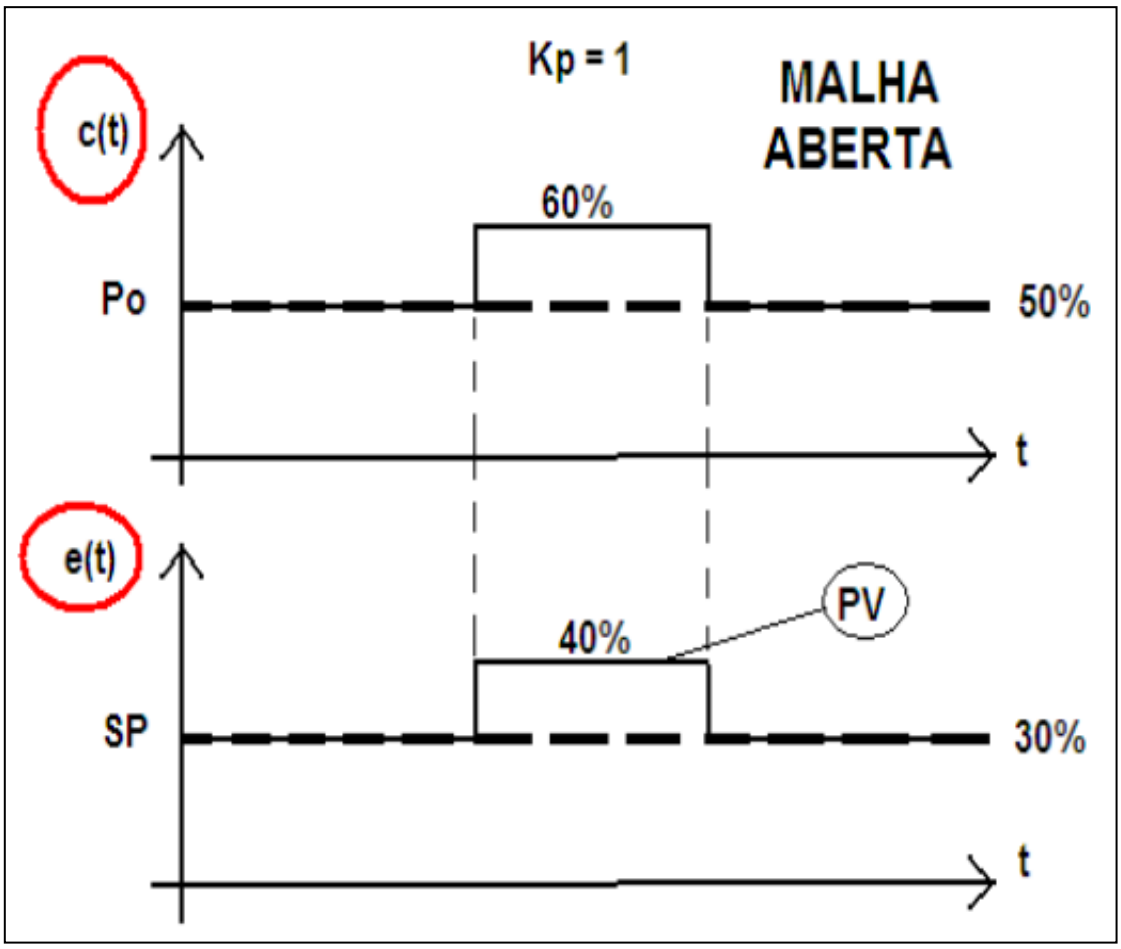


**Figura 11 - Efeito do controle em malha aberta**
**Fonte: SCHNEIDER (2011, p. 26).**

Na figura 11 tem-se a demonstração do efeito de controle em malha aberta onde o SP é aumentado em 10% e concomitantemente a saída também aumenta 10%.

Ação do controle proporcional em Malha Fechada:

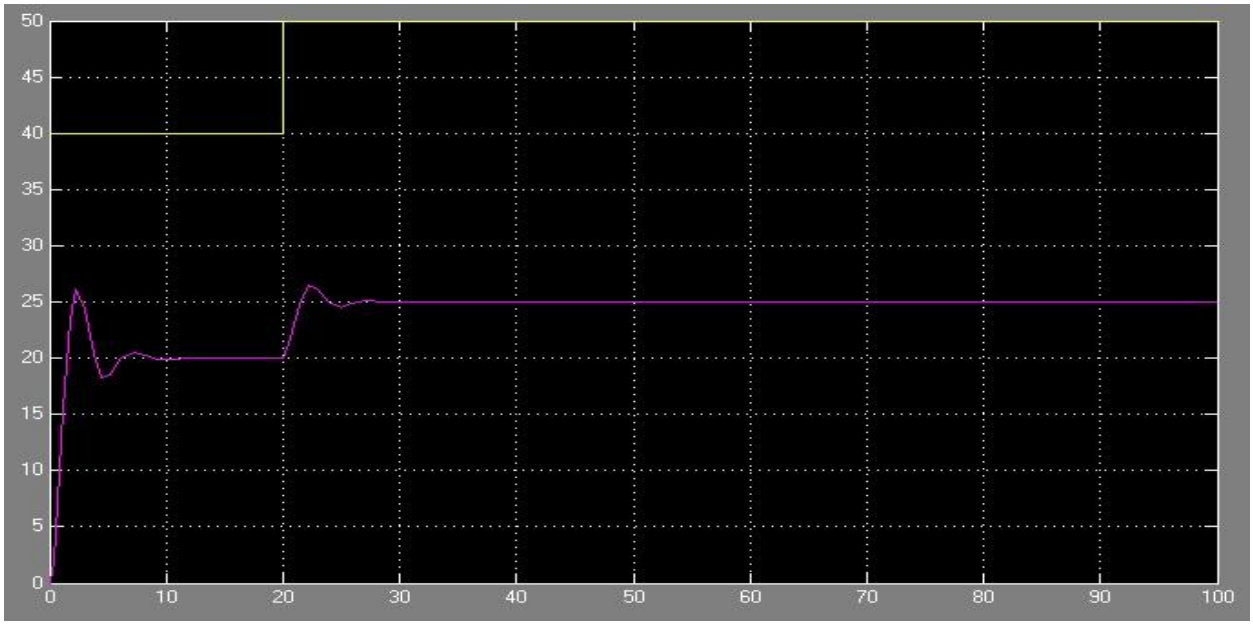


**Figura 12: Controle em Malha Fechada e aplicação do efeito Proporcional com baixo ganho**
**Fonte: Autoria Própria.**

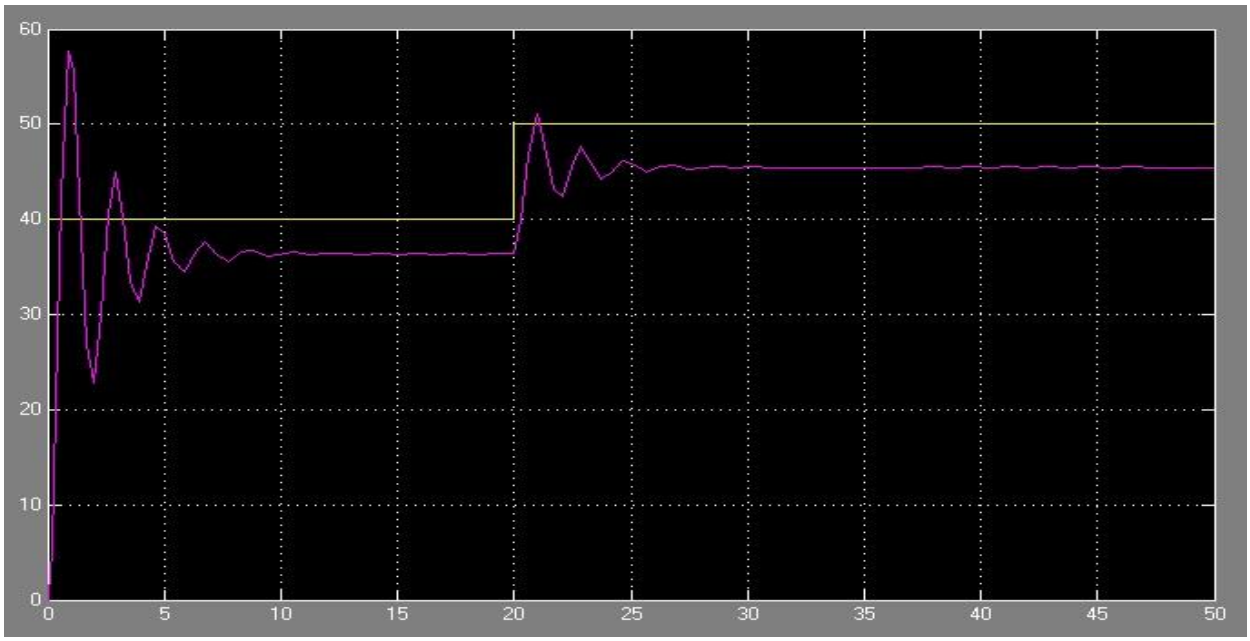


**Figura 13: Controle em Malha Fechada e aplicação do efeito Proporcional com alto ganho**
**Fonte: Autoria Própria.**

b) Ação Proporcional-Integral

A ação proporcional-integral aplica tanto um fator proporcional ao erro quanto um fator integral. O primeiro tende a acelerar a resposta do sistema, e o segundo tende a reduzir ou eliminar o erro em regime permanente (BAZANELLA, 2000). Ti representa o tempo de integração. A relação da equação (8) mostra que se o Kp for 1 e o Ti for 1minuto, a cada 1 minuto de integração, o sinal de saída c(t) variará a porcentagem relativa ao erro (PV - SP). Ou seja, a cada Ti minutos a ação Kp se repete. Percebe-se na figura 14, que a ação PI elimina o erro em regime permanente.

$$\frac{C(S)}{E(S)} = \frac{Kp.(1+Ti.S)}{Ti.S} \qquad c(t) = Kp.e(t) + \frac{Kp}{Ti}\int_0^t e(t)dt \qquad (8)$$

**Equação do controle PI**

Agora há a aplicação do efeito integral e do efeito proporcional. O efeito integral corresponde a uma rampa como demonstrado na figura 14. Essa rampa é a variação do efeito em um determinado período de tempo. Depois de aplicada essa variação, o sistema permanece com o efeito resultante da ação integral que é o responsável pela eliminação do erro em regime permanente como demonstrado na figura 15:

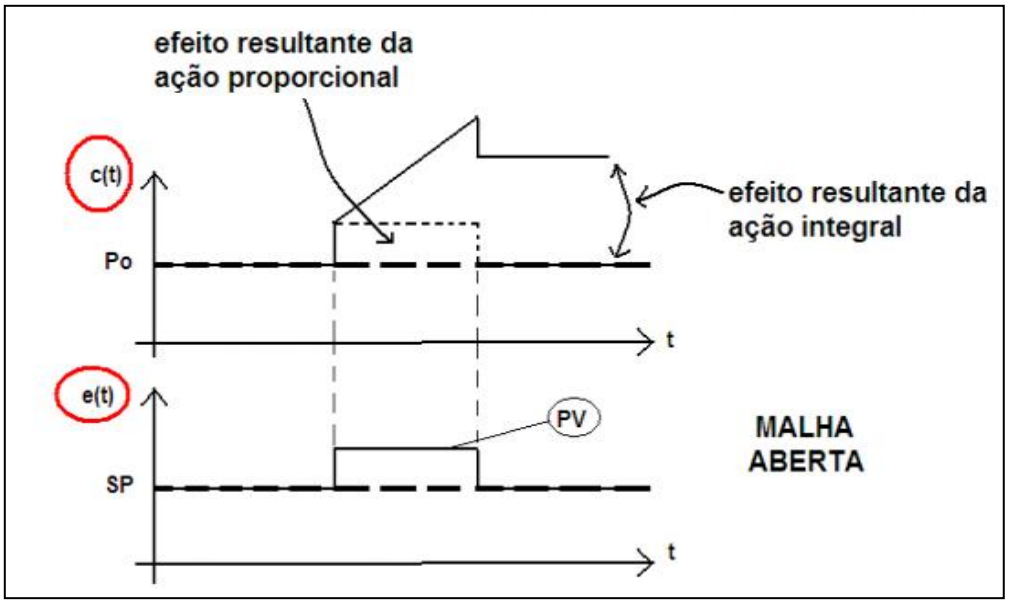


**Figura 14 - Ação de controle Proporcional Integral em Malha Aberta**
**Fonte: SCHNEIDER (2011, p. 27).**

Ação Proporcional Integral em Malha Fechada:

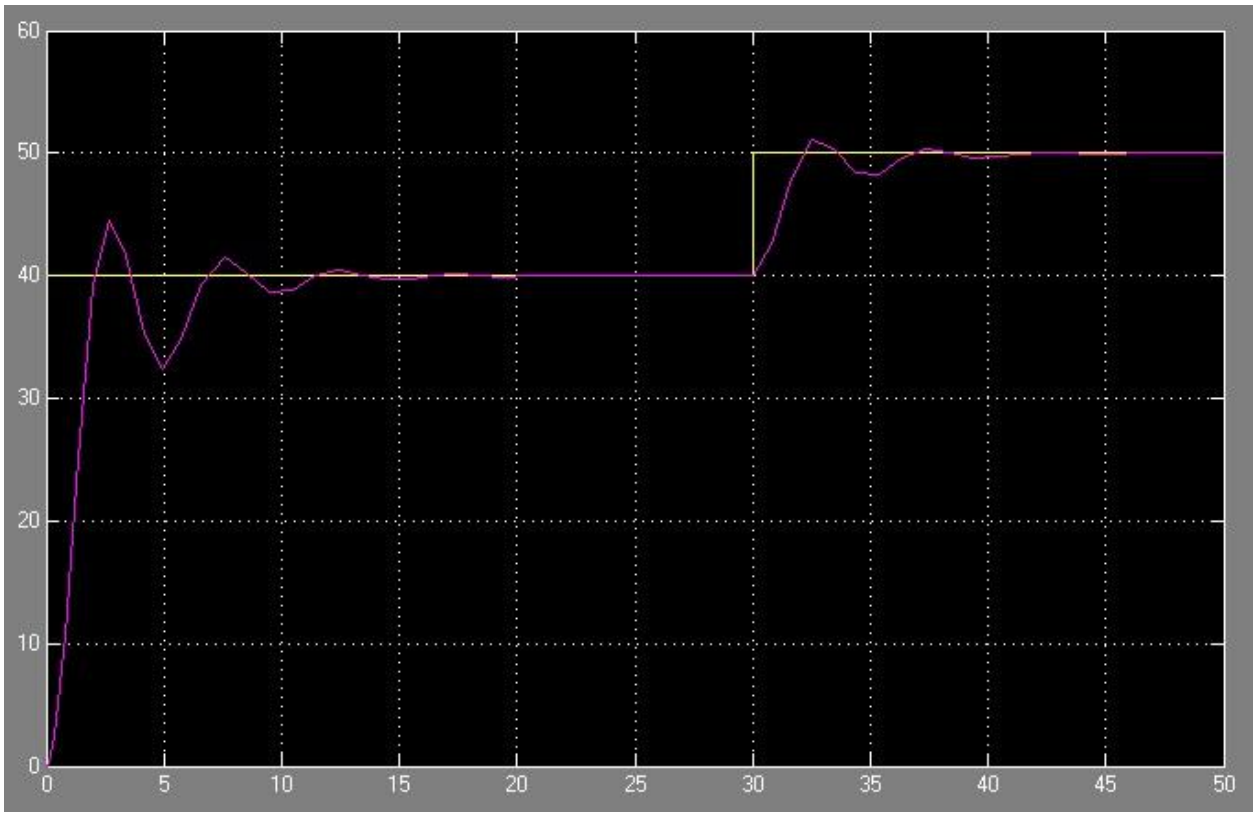


**Figura 15 - Ação de controle Proporcional Integral em Malha Fechada**
**Fonte: Autoria Própria.**

c) Ação Proporcional Integral Derivativa.

É a ação de controle que contém as três ações de controle: Proporcional, Integral e Derivativa. Além dos efeitos das ações P e I, a ação derivativa antecipa o controle em sua resposta, portanto contribui para uma resposta mais rápida com um sobressinal reduzido. A ação integral elimina o erro em regime permanente e a ação proporcional aumenta a velocidade da resposta. O efeito causado pelo controlador PI é eliminado pela ação derivativa. Essa ultima ação aumenta a estabilidade do sistema e torna a resposta mais rápida. Ela causa um efeito antecipatório (BAZANELLA, 2000). É um dos controles mais utilizados nas indústrias. Percebe-se que a ação derivativa diminui as oscilações da curva. A equação de controle PID, sendo Kp o ganho proporcional, Ti o tempo de integração e Td o tempo de derivação, é a seguinte:

$$\frac{C(S)}{E(S)} = Kp.\frac{(1+Ti.S+Ti.Td.s^2)}{Ti.S} \quad (9)$$

$$c(t) = Kp.e(t) + \frac{Kp}{Ti}\int_0^t e(t)dt + Kp.Td.\frac{de(t)}{dt}$$

**Equação do controle PID**

Na figura 16, tem se a ação PID em malha aberta. Nela é possível ver a ação derivativa, a qual faz com que haja uma antecipação no sinal de resposta do controle. Além dessa ação, pode ser observada as outras duas: proporcional e integral.

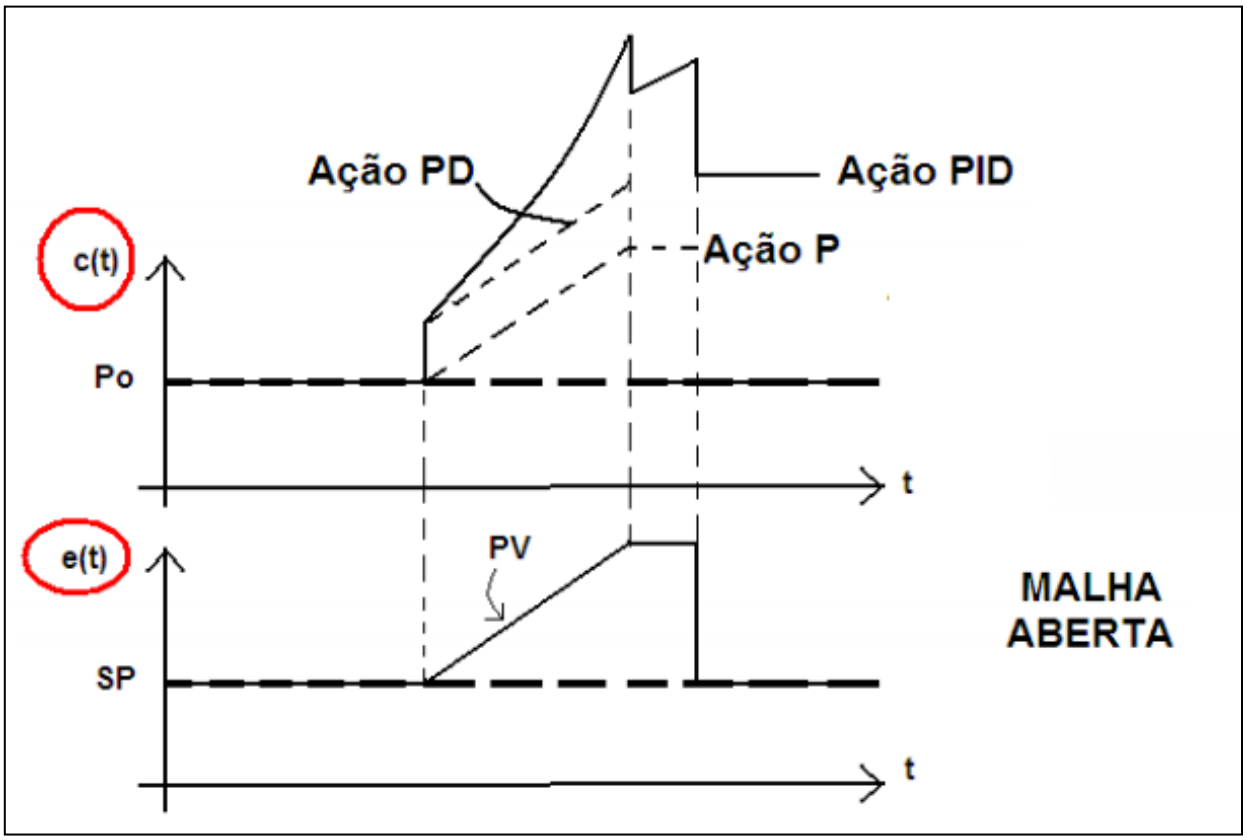


**Figura 16 - Ação Proporcional Integral Derivativa em Malha Aberta**
**Fonte: SCHNEIDER (2011, p. 29).**

Na figura 17, mostra-se a ação do controle PID em malha fechada. Nessa ação não há overshoot nem erro em regime permanente. O primeiro foi eliminado por meio da inserção do controlador derivativo. O segundo foi sanado com a utilização do controle integral.

Ação PID em malha fechada:

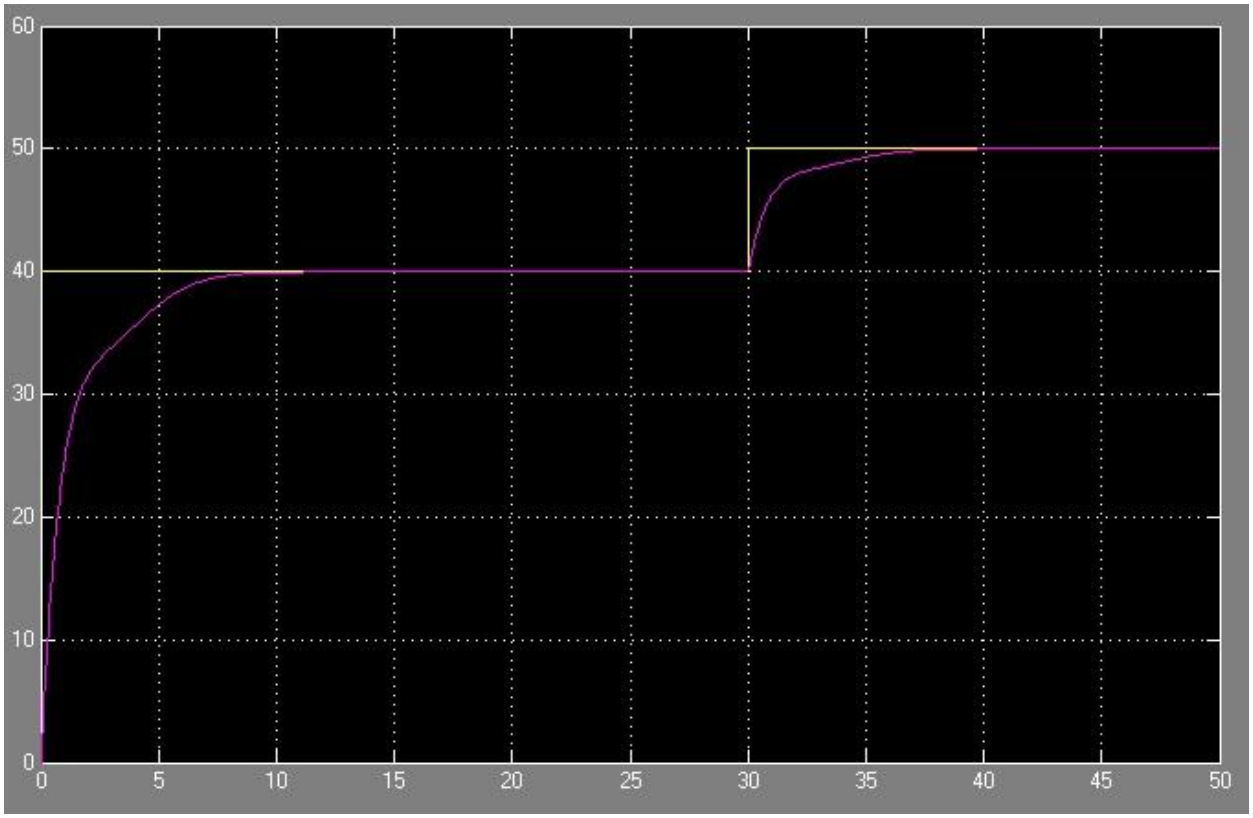


**Figura 17 - Ação Proporcional Integral Derivativa em malha fechada**
**Fonte: Autoria Própria.**

## 2.5 Métodos de Sintonia

Nesse item são analisados os métodos de sintonia do PID. Esses métodos foram desenvolvidos com o objetivo de aperfeiçoar a regulação por meio da aplicação de constantes que multiplicam os parâmetros. Dentre eles estão o primeiro e o segundo método de Ziegler-Nichols e o método CHR.

a) Primeiro Método de Ziegler-Nichols

O método Ziegler-Nichols, segundo Ogata (1998), define uma tabela que permite encontrar o valor de Kp, Ti e Td para o controlador. Para a aplicação da tabela é necessário encontrar o valor do período crítico (Pu), que é obtido depois de achado o ganho crítico (Ku). O ganho crítico é obtido aumentando-se o ganho até que a curva de reposta oscile perfeitamente. Depois de verificada essa condição, é obtido o período crítico, que é o período da oscilação obtida. Aplica-se a função degrau, regula-se o ganho e obtém-se o Ku e o Pu, conforme mostra a figura 18.

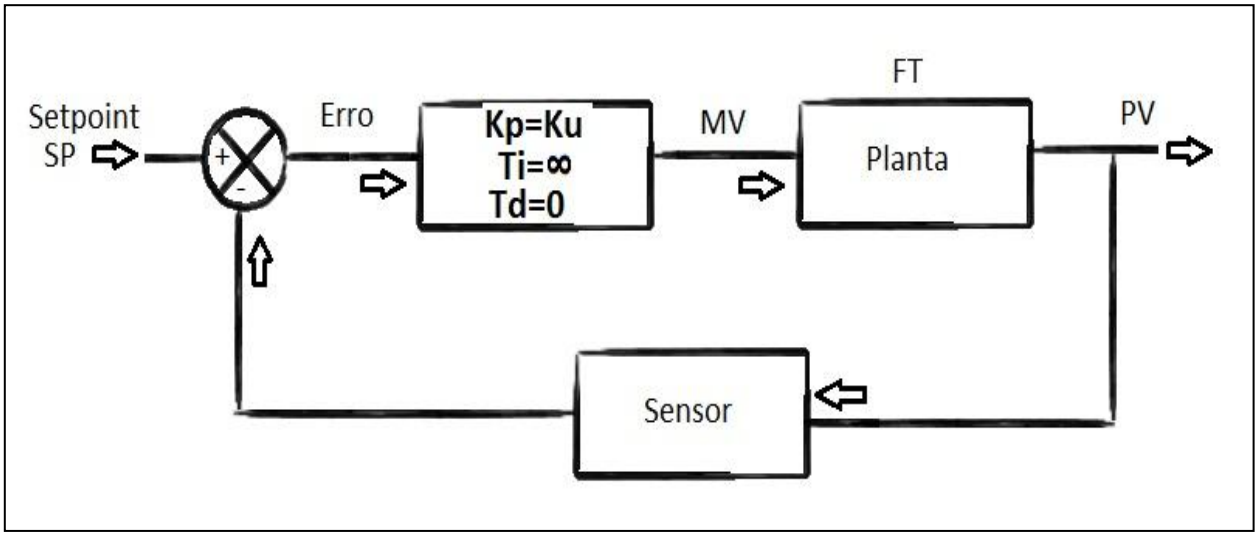


**Figura 18 - Malha Fechada com Ganho Ku**
**Fonte: Autoria Própria.**

Com o Ku aumentado até que se tenha a instabilidade do sistema obter-se-á um gráfico como demonstra a figura 19:

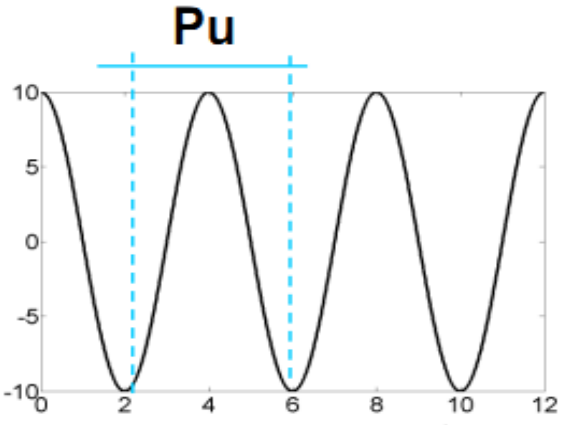


**Figura 19 - Resposta temporal instável com Ganho Crítico**
**Fonte: SCHNEIDER (2011, p. 41).**

Depois de obtidos os valores de Ku e Pu, é possível encontrar os parâmetros do controlador para as ações Kp, Ti e Td conforme demonstra a tabela 1.

**Tabela 1- Cálculo do primeiro método de Ziegler-Nichols.**

| Tipo do controlador | Kp | TI | Td |
|---|---|---|---|
| P | 0,5Ku | ∞ | 0 |
| PI | 0,45Ku | 0,833Pu | 0 |
| PID | 0,6Ku | 0,5Pu | 0,125Pu |

**Fonte: OGATA (1998).**

b) Segundo Método de Ziegler-Nichols

Ziegler e Nichols, segundo Ogata (1998), determinaram regras para se calcular um valor aproximado dos parâmetros PID, (Kp, Td e Ti) a partir de valores da dinâmica da planta. Em seu segundo método é aplicada a função degrau diretamente na planta, obtendo-se assim uma curva. Dessa curva são extraídos dados que serão alocados em uma tabela para a obtenção dos valores de Kp, Ti e Td. Nas figuras 20 e 21 temos L = Tempo morto K= ganho da planta, T = tempo. Na figura 20, há a aplicação de um degrau na função da planta e em seguida há a análise dos gráficos de resposta da planta. Traça-se uma reta com a inclinação da curva de resposta e considera-se o tempo morto a distância da origem do eixo das abscissas até o ponto em que a reta toca o eixo. O tempo T é calculado por meio da subtração do valor correspondente a 63,2% da curva de resposta, menos o valor de L. Também pode-se calcular L e T por meio da utilização de duas formulas de tempo: t1 e t2. O valor de t1 corresponde ao valor de 28,3% da curva de resposta e representa a soma de L com T/3. O valor de t2 corresponde a 63,2% da curva de resposta e representa a soma de L e T. A figura 21 mostra K, L e T.

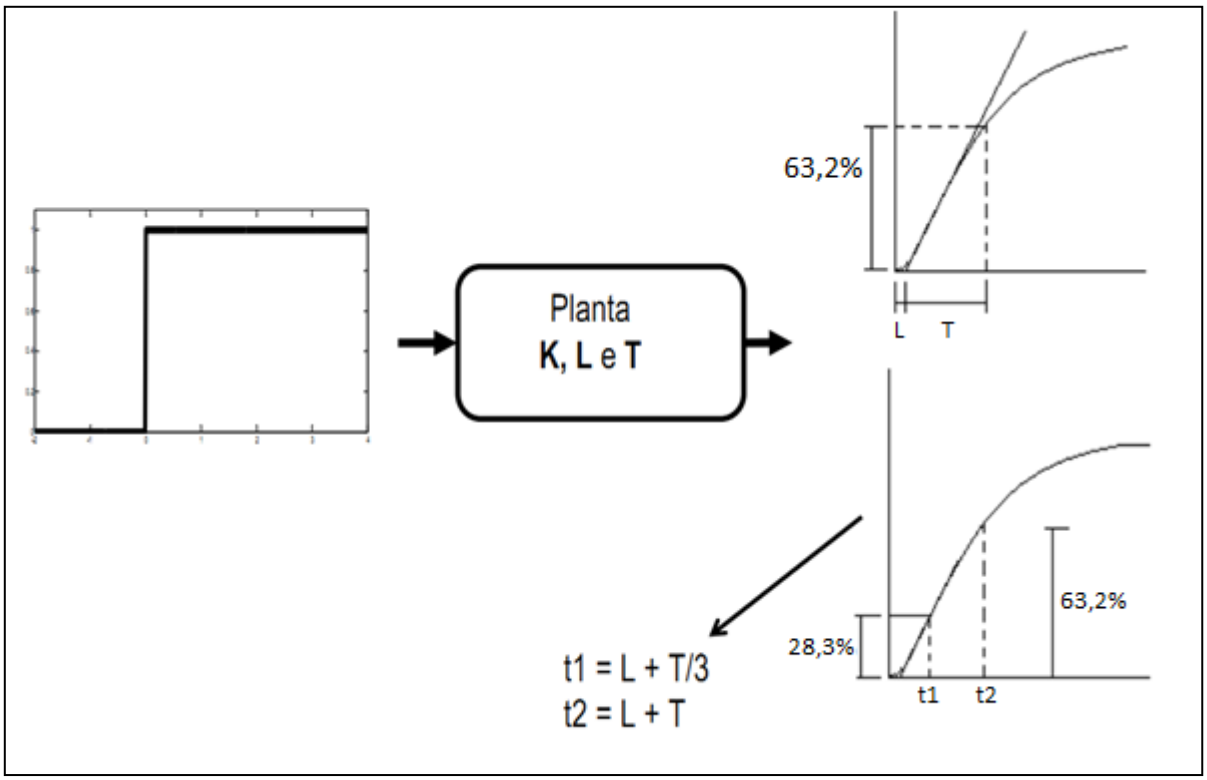


**Figura 20 - Análise do degrau na FT da Planta**
**Fonte: Adaptado de SCHNEIDER (2011, p. 43).**

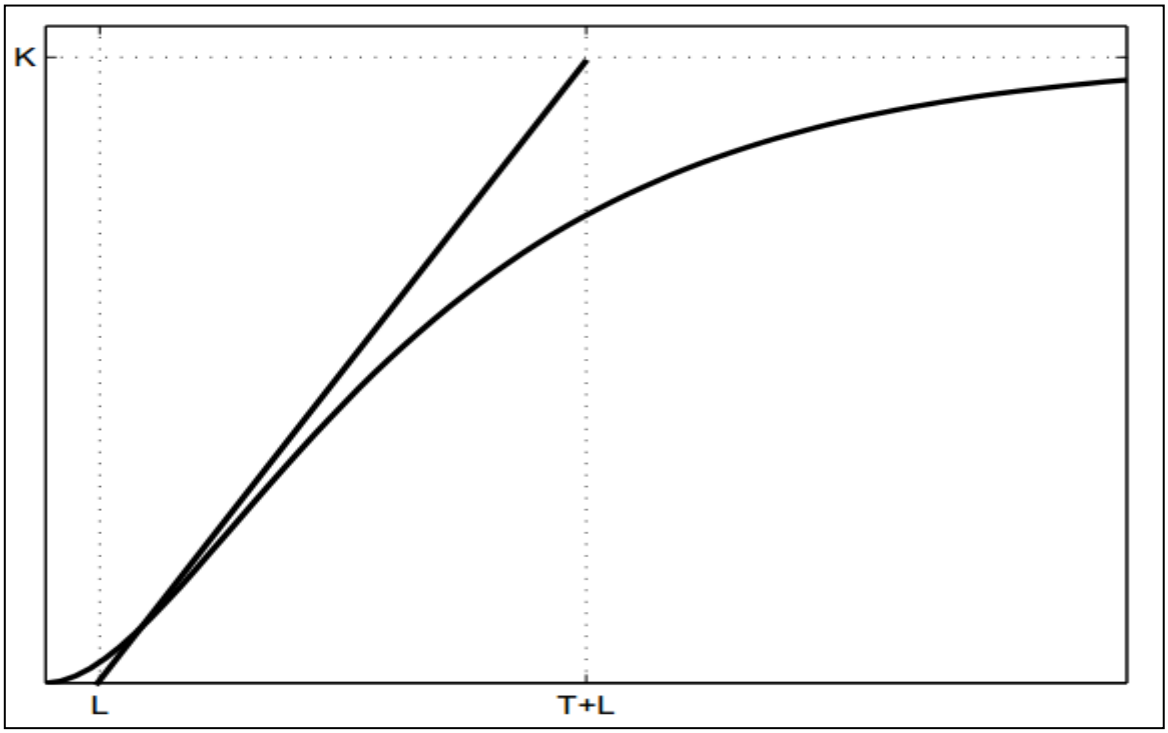


**Figura 21 - Curva representando K, L e T**
**Fonte: MECATRÔNICA (2012, p. 13).**

Observa-se na tabela 2 que os parâmetros de Kp, Ti e Td são definidos pelo tempo (T) e pelo tempo morto (L) no segundo método de Ziegler-Nichols.

**Tabela 2 - Cálculo do segundo método de Ziegler-Nichols.**

| Tipo do controlador | Kp | TI | Td |
|---|---|---|---|
| P | $\frac{T}{L}$ | ∞ | 0 |
| PI | $\frac{0,9T}{L}$ | $\frac{L}{0.3}$ | 0 |
| PID | $\mathbf{\frac{1,2T}{L}}$ | 2L | 0,5L |

**Fonte: OGATA (1998).**

c) CHR

Problema servo é a situação em que o ponto de operação deve seguir uma trajetória, por exemplo: um reator com volume fixo é alimentado, e as temperaturas devem seguir uma trajetória no tempo (variação no processo). Pode ser caracterizado por um degrau no *setpoint* (CAMPOS, 2006). Vide figura 22.

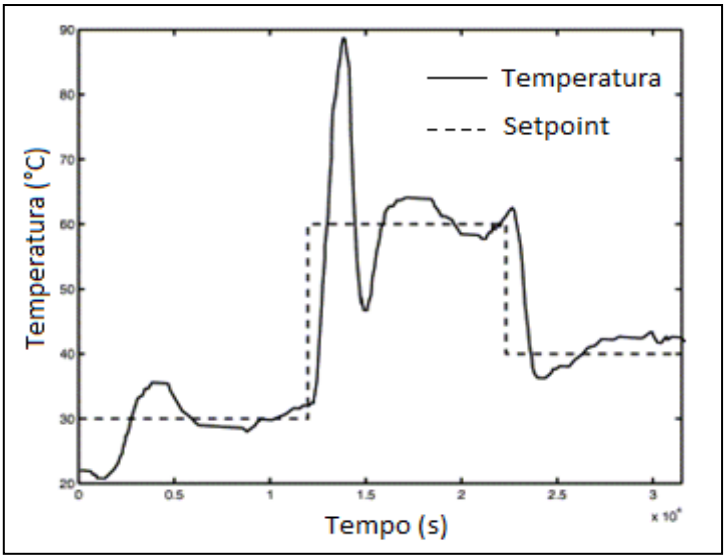


**Figura 22 - Exemplo de problema servo.**
**Fonte: Adaptado de SCIENCE (2012).**

Problema regulatório é a situação em que o ponto de operação (*setpoint*) é fixo e se deseja manter o processo o mais próximo possível deste valor, apesar das perturbações (CAMPOS, 2006). De acordo com a figura 23, o *setpoint* permanece estável e em seguida ocorre um distúrbio no sistema gerando uma perturbação na saída. O problema regulatório pode ser caracterizado pela aplicação de um degrau como distúrbio. de acordo com o demonstrado na figura 24.

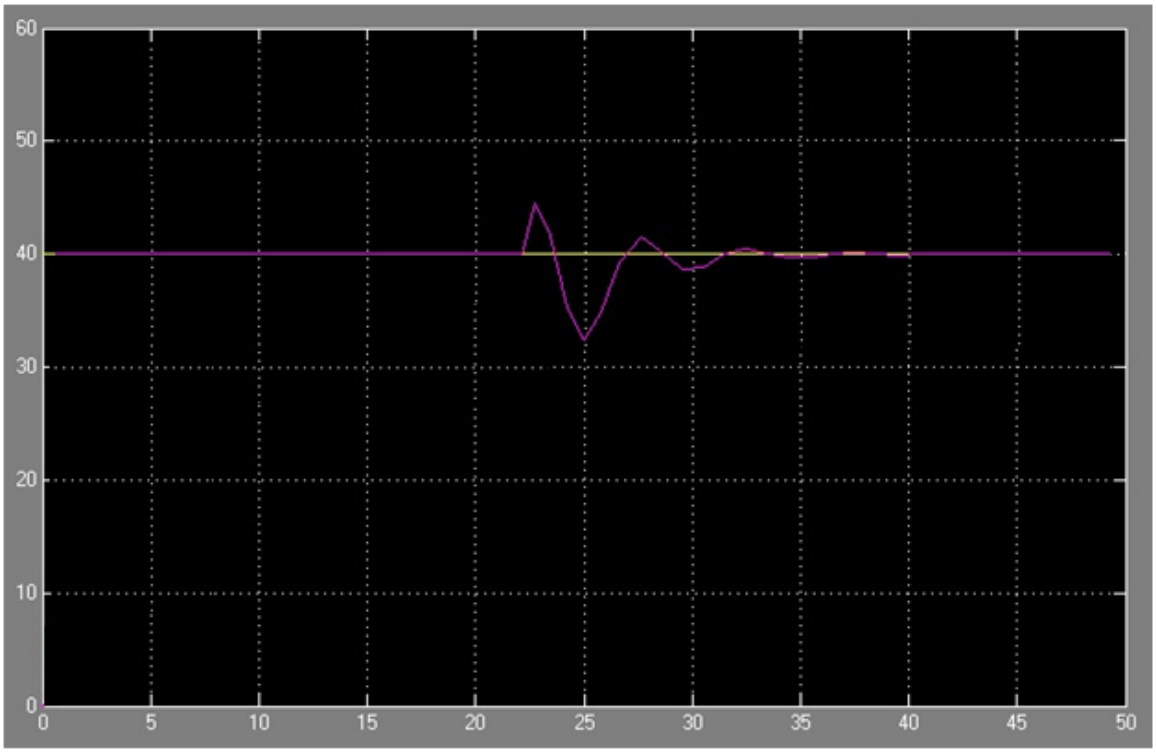


**Figura 23 - Exemplo de problema regulatório.**
**Fonte: Autoria própria.**

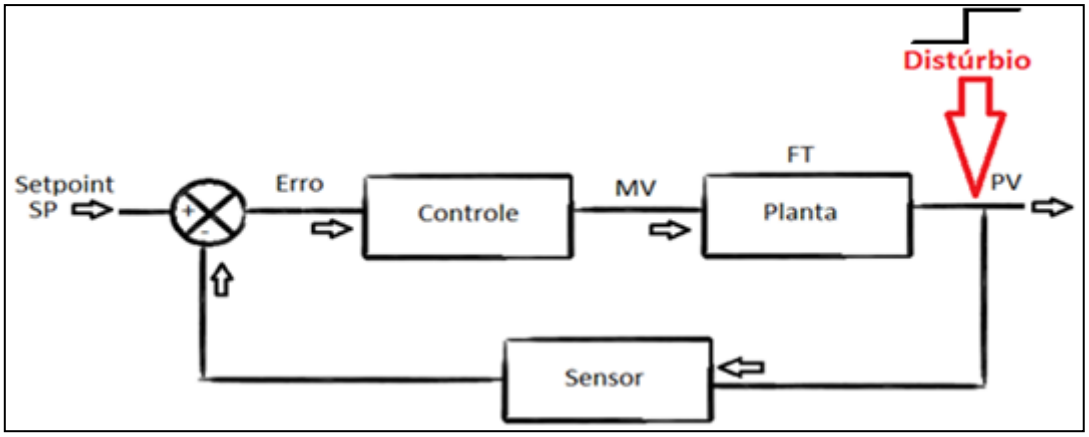


**Figura 24 - Problema Regulatório; Distúrbio.**
**Fonte: Autoria própria.**

O método é baseado na teoria de Chien, Hrones e Reswick, (CHR) que é uma modificação do método de Ziegler-Nichols. O CHR calcula os parâmetros para dois tipos de critérios de desempenho. Examinando o critério de resposta mais rápida sem sobressinal e para problemas do tipo servo, podemos calcular as variáveis do controlador PID como mostra a tabela 3. É também obtido com a função degrau sendo aplicada à Planta. Os valores de K, L e T são obtidos da mesma forma que para Ziegler- Nichols (OGATA, 1998).

**Tabela 3 - Cálculo pelo método de CHR para resposta mais rápida sem sobre valor de problemas do tipo mudança de *setpoint*.**

| Tipo do controlador | Kp | TI | Td |
|---|---|---|---|
| P | $\frac{0,3T}{KL}$ | ∞ | 0 |
| PI | $\frac{0,35T}{KL}$ | $1,16T$ | 0 |
| PID | $\mathbf{\frac{0,6T}{KL}}$ | T | $\mathbf{\frac{L}{2}}$ |

**Fonte: OGATA (1998).**

Caso o problema seja perturbação pode-se usar a tabela 4:

**Tabela 4 - Cálculo pelo método de CHR para resposta mais rápida sem sobre valor de problemas do tipo perturbação.**

| Tipo do controlador | Kp | TI | Td |
|---|---|---|---|
| P | $\frac{0,3T}{KL}$ | ∞ | 0 |
| PI | $\frac{0,6T}{KL}$ | $4L$ | 0 |
| PID | $\mathbf{\frac{0,95T}{KL}}$ | 2,375L | $\mathbf{0,421L}$ |

**Fonte: OGATA (1998).**

A tabela 5 informa como calcular os parâmetros no critério de resposta mais rápida com 20% de sobressinal para problemas do tipo mudança do *setpoint* em degrau (problema servo).

**Tabela 5 - Cálculo do método de resposta rápida com 20% de sobre valor para problemas do tipo mudança de *setpoint*.**

| Tipo do controlador | Kp | TI | Td |
|---|---|---|---|
| P | $\frac{0,7T}{KL}$ | ∞ | 0 |
| PI | $\frac{0,6T}{KL}$ | T | 0 |
| PID | $\mathbf{\frac{0,95T}{KL}}$ | 1,357L | $\mathbf{0,473L}$ |

**Fonte OGATA (1998).**

## 2.6 Arranjos de PID

Nesse item serão vistos cinco arranjos de controladores PID discretizados. Há a demonstração das fórmulas dos arranjos assim como os gráficos e os diagramas de bloco. Os arranjos são: controlador PID em arranjo paralelo ideal, controlador PID em arranjo série, controlador PID em arranjo paralelo ideal com filtro derivativo, controlador PI-D em arranjo série com filtro derivativo e controlador PID em arranjo série com filtro derivativo.

Para a exemplificação das equações tem-se: e = erro; Kp = ganho proporcional, Ti = tempo derivativo, Td = tempo integrativo, Ki = Kp / Ti, Kd = Kp / Td, n = índice de amostragem, T = taxa de amostragem, N = índice de filtro derivativo.

a) Controlador PID em Arranjo Paralelo Ideal

De acordo com a literatura, os controladores paralelos ideais dependentes são denominados dependentes devido aos ganhos integrais e derivativos estarem dependentes da variação do ganho Kp, ou seja, se alterar o ganho proporcional afetará

os demais parâmetros. Esse controlador apresenta a equação (10) (IWASSE, 2009 p.34).

$$u(t) = Kp.\left(e(t) + \frac{1}{Ti}\ .\int e(t)dt + Td.\ \frac{de}{dt}\ (t)\right) +\ Po \qquad (10)$$

Aplicando a Transformada de Laplace:

$$u(s) = Kp.\left(1 + \frac{1}{Ti.s} +\ Td.s\ \right).e(s)$$

Para obter a versão discreta deste controlador poderia se utilizar qualquer um dos métodos para discretização. O método da diferença atrasada (*backward difference*) e a aproximação por Tustin não tendem a gerar instabilidade. Embora a aproximação por Tustin realize um mapeamento mais próximo do ideal entre os planos s e z, preferiu-se utilizar o método da diferença atrasada (*backward difference*), pois se consegue obter um algoritmo mais simples do ponto de vista computacional. Então fazendo uso deste método, por substituição, a função de transferência do controlador passa a ser a equação (11) (MORENO, 2010).

$$u(z) = e(z).\left[\left(Kp + \frac{Ki}{\left[\frac{(z-1)}{T.z}\right]}\right) + Kd.\left(\frac{(z-1)}{(T.z)}\right)\right] \qquad (11)$$

Desenvolvendo a equação (11):

$$u(z).\left[z^2.T - z.T\right] =\ e(z).\left[z^2.(\ Kp.T +\ Ki.T^2 + Kd) - z.(Kp.T\ +\ 2.Kd) +\ Kd\right] \qquad (12)$$

Para não se trabalhar com termos no futuro, divide-se a equação (12) por $Z^k$, sendo k igual ao maior coeficiente positivo de z presente na equação, que neste caso é 2.

$$u(z).[T - T.z^{-1}] =\ e(z).[\ Kp.T +\ ki.T^2 + Kd - z^{-1}.(Kp.T\ +\ 2.Kd) +\ z^{-2}.Kd]$$

Fazendo-se uso da identidade,

$$Z^k.u(z) = u(n+K)$$

Obtém-se:

$$u(n).T - u(n-1).T = e(n).(Kp.T + Ki.T^2 + Kd) - e(n-1).(Kp.T + 2.Kd) + e(n-2).Kd$$

Isolando-se u(n):

$$u(n) = e(n).\left(Kp + Ki.T + \frac{Kd}{T}\right) - e(n-1).\left(Kp + \frac{2.Kd}{T}\right) + e(n-2).\frac{Kd}{T} + u(n-1) \qquad (13)$$

A equação (13), proposta por Moreno (2010) representa o equivalente discretizado do PID paralelo ideal sendo u(n) a variável de saída e e(n) a variável de entrada. A figura 25 mostra o arranjo PID paralelo ideal proposto por Iwasse (2009) e discretizado por Moreno (2010) na equação (13).

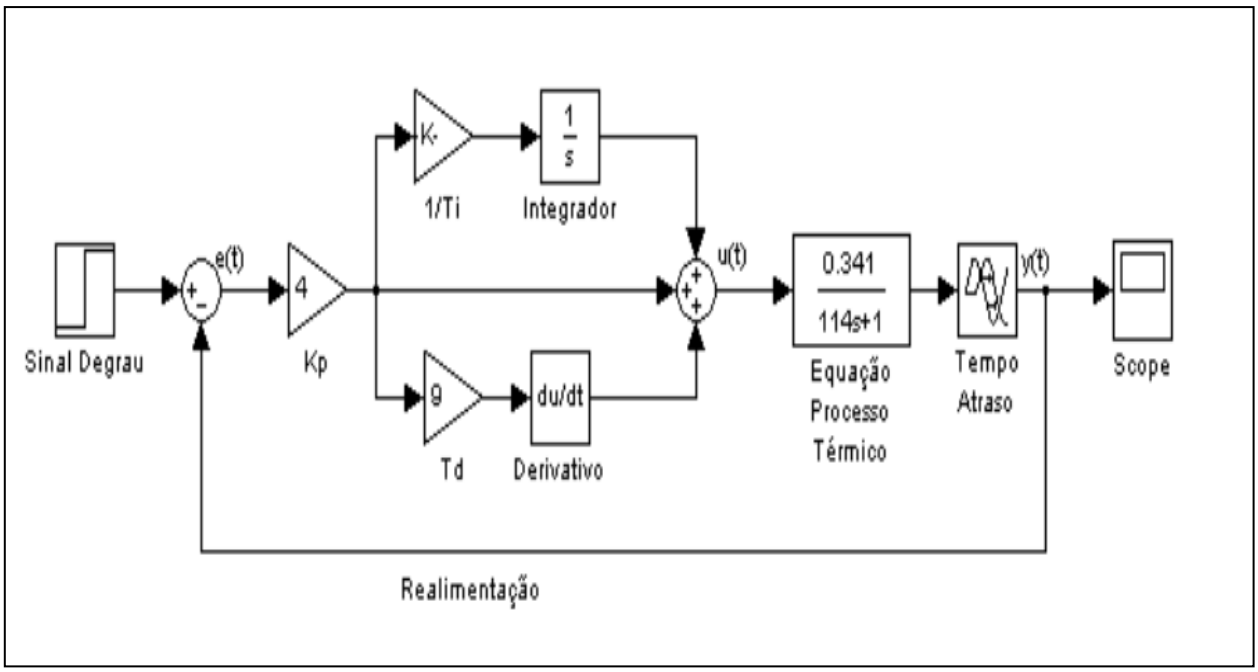


**Figura 25 - Diagrama de blocos da aplicação do controlador PID paralelo ideal**
**Fonte: IWASSE (2009, p. 36).**

b) Controlador PID em Arranjo Série

Esse controlador possui ganhos dependentes, os ganhos Ki e Kd estão relacionados com Kp , ou melhor, os tempos Ti e Td estão relacionados aos termos integrais e derivativos sua equação é definida conforme a equação(14) (IWASSE, 2009).

$$u(t) = Kp.\left(e(t)\left(1 + \frac{Td}{Ti}\right) + \frac{1}{Ti}.\int e(t)dt + Td.\frac{de(t)}{dt}\right) + Po \quad (14)$$

Aplicando a Transformada de Laplace:

$$u(s) = Kp.(1 + \frac{1}{Ti.s})(1 + Td.s)\ .e(s)$$

Para a versão discretizada:

$$u(n) = e(n).\left(Kp + Ki.T + \frac{Kd}{T} + \frac{Kd}{Ti}\right) - e(n-1).\left(Kp + \left(\frac{2.Kd}{T}\right) + \frac{Kd}{Ti}\right) + e(n-2).\frac{Kd}{T}$$

$$+u(n-1) \quad (15)$$

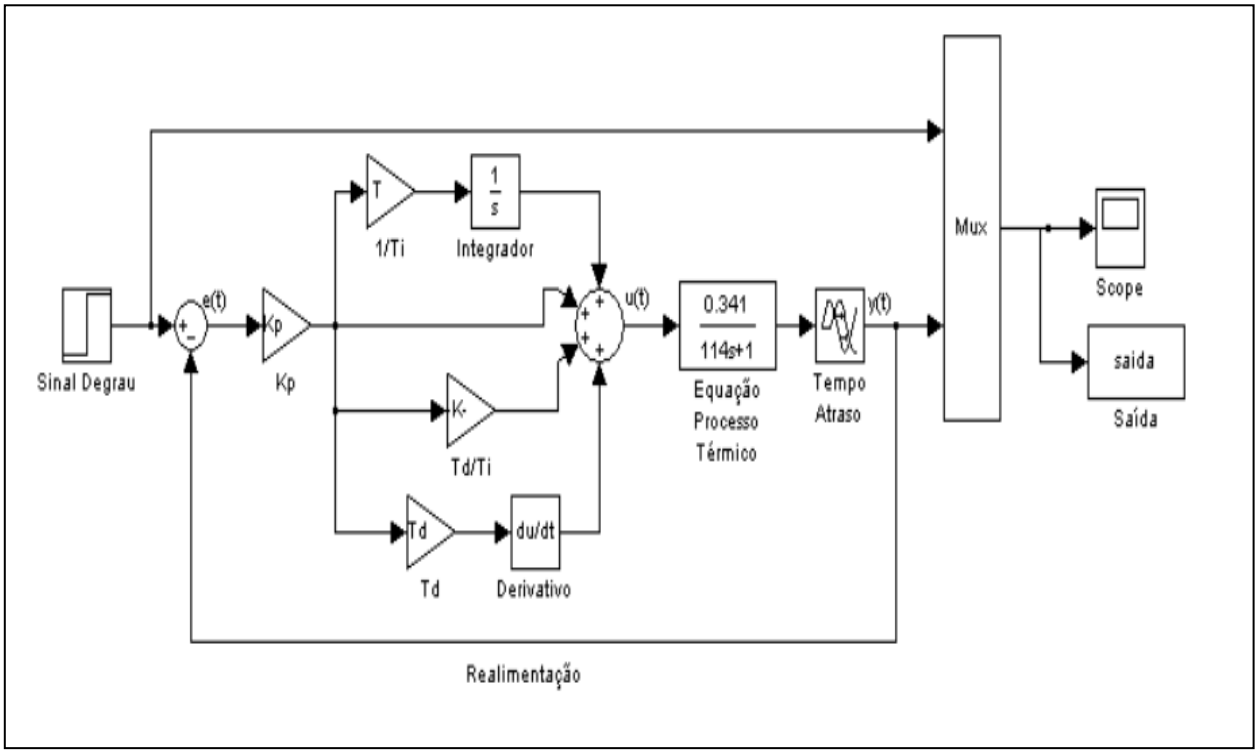


**Figura 26 - Diagrama de blocos da aplicação do controlador PID em série**
**Fonte: IWASSE (2009, p. 38).**

A figura 26 demonstra o diagrama em blocos do PID em arranjo sério proposto pelo Iwasse (2009). A equação (15), proposta por Moreno (2010) demonstra a equação desse diagrama de blocos discretizada.

c) Controlador PID em Arranjo paralelo ideal com filtro derivativo

O controlador paralelo ideal nem sempre é suficiente para controlar alguns sistemas, em alguns casos é adicionado um filtro no termo derivativo suprindo os critérios de projeto do controlador. A equação do PID paralelo com filtro derivativo é definida conforme a equação (16) (IWASSE, 2009).

$$u(t) = Kp.\left(e(t) + \frac{1}{Ti}\,.\int e(t)dt - \frac{N^2}{Td}\,.\int e^{-(\frac{N}{Td}xt)}dt + N.\,\frac{de}{dt}\,(t)\right) + \,Po \tag{16}$$

Aplicando a Transformada de Laplace:

$$u(s) = Kp.\left(1 + \frac{1}{Ti.s} + \frac{Td.s}{1 + \frac{Td}{N}.s}\right).e(s)$$

Para a versão discretizada:

$$u(n) = \left[\frac{2.\left(\frac{Td}{N}\right)+T}{T+\left(\frac{Td}{N}\right)}\right].u(n-1) - \left[\frac{\frac{Td}{N}}{T+\left(\frac{Td}{N}\right)}\right].u(n-2)$$

$$+[(\,Kp.T.\left(1\,+\,\left(\frac{1}{Ti}\right).\left(\left(\frac{Td}{N}\right)+\,T\right)\right)$$

$$+\,Kp.\left(Td + \left(\frac{Td}{N}\right)\right))/(T + \left(\frac{Td}{N}\right))].e(n) \tag{17}$$

$$-\left[\frac{2.Kp.\left(\left(\frac{Td}{N}\right)+Td\right)+Kp.T.\left(1+\left(\frac{1}{Ti}\right).\left(\frac{Td}{N}\right)\right)}{T+\left(\frac{Td}{N}\right)}\right].e(n-1)$$

$$+\,\left[\frac{Kp.\left(Td+\left(\frac{Td}{N}\right)\right)}{T+\left(\frac{Td}{N}\right)}\right].e(i-2);$$

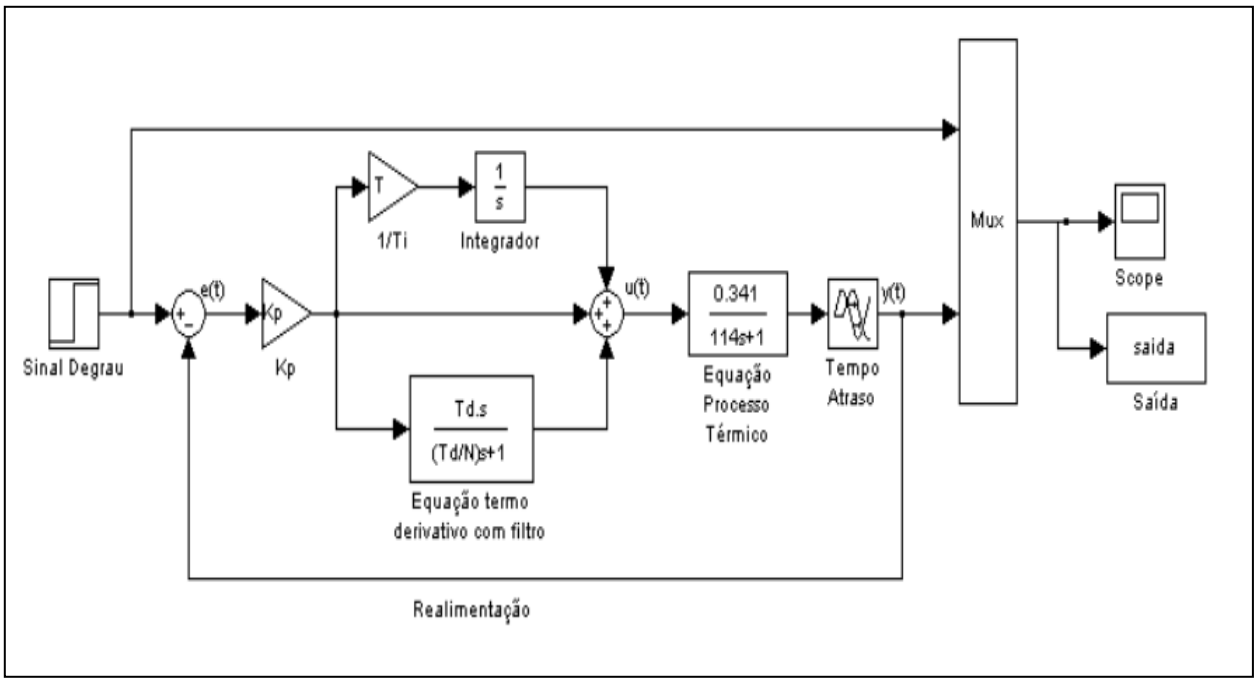


**Figura 27 - Diagrama de blocos da aplicação do controlador PID paralelo ideal com filtro derivativo (FD)**
**Fonte: IWASSE (2009, p. 42).**

A figura 27 mostra o diagrama de blocos do PID em série com filtro derivativo e a equação (17), proposta por Moreno (2010), mostra a discretização desse PID.

d) Controlador PI-D (Arranjo série com Filtro Derivativo)

Esse controlador é conhecido como Controlador Industrial (O'DWYER), sua estrutura é composta de: um filtro derivativo acoplado na realimentação do sistema e os termos: integral e proporcional (PI) estão acoplados na saída da variação do erro. A equação do controlador PI-D é mostrada na equação (18) (IWASSE, 2009).

$$u(t) = Kp.e(t) + \frac{Kp}{Ti}.\int e(t)dt - Kp.\left(N + \frac{1}{Ti}\right).y(t) - Kp.\left(\frac{N-N^2}{Td} + \frac{N-1}{Ti}\right).\int e^{-\frac{n.t}{Td}}y(t)dt \quad (18)$$

Aplicando a Transformada de Laplace:

$$u(s) = Kp.\left(1 + \frac{1}{Ti.s}\right).\left(e(s) - \frac{1 + Td.s}{1 + \frac{Td}{N}.s}.y(s)\right)$$

Para a versão discretizada:

$$u(n) =$$

$$\{ \quad \left(3 * \left(Ti.\left(\frac{Td}{N}\right)\right) + 2.\left(Ti + Ti.\left(\frac{Td}{N}\right) + Kp.Ti.Td\right).T + (Ti + Kp.Ti + Kp.Td).(T^2)\right).u(n-1)$$

$$- \left(\left(Ti + Ti.\left(\frac{Td}{N}\right) + Kp.Ti.Td\right).T + 3.\left(Ti.\left(\frac{Td}{N}\right)\right)\right).u(n-2) + \left(Ti.\left(\frac{Td}{N}\right)\right).u(n-3)$$

$$+ \left(\left(Kp.Ti.\left(\frac{Td}{N}\right)\right).T + \left(Kp.Ti + Kp.\left(\frac{Td}{N}\right)\right).(T^2) + Kp.(T^3)\right).e(n)$$

$$- \left(2.\left(Kp.Ti.\left(\frac{Td}{N}\right)\right).T + \left(Kp.Ti + Kp.\left(\frac{Td}{N}\right)\right).(T^2)\right).e(n-1) \qquad (19)$$

$$+ \left(\left(Kp.Ti.\left(\frac{Td}{N}\right)\right).T\right).e(n-2)) \quad \}$$

$$/\left(\left(Ti.\left(\frac{Td}{N}\right)\right) + \left(Ti + Ti.\left(\frac{Td}{N}\right) + Kp.Ti.Td\right).T + (Ti + Kp.Ti + Kp.Td).(T^2) + Kp.(T^3)\right)$$

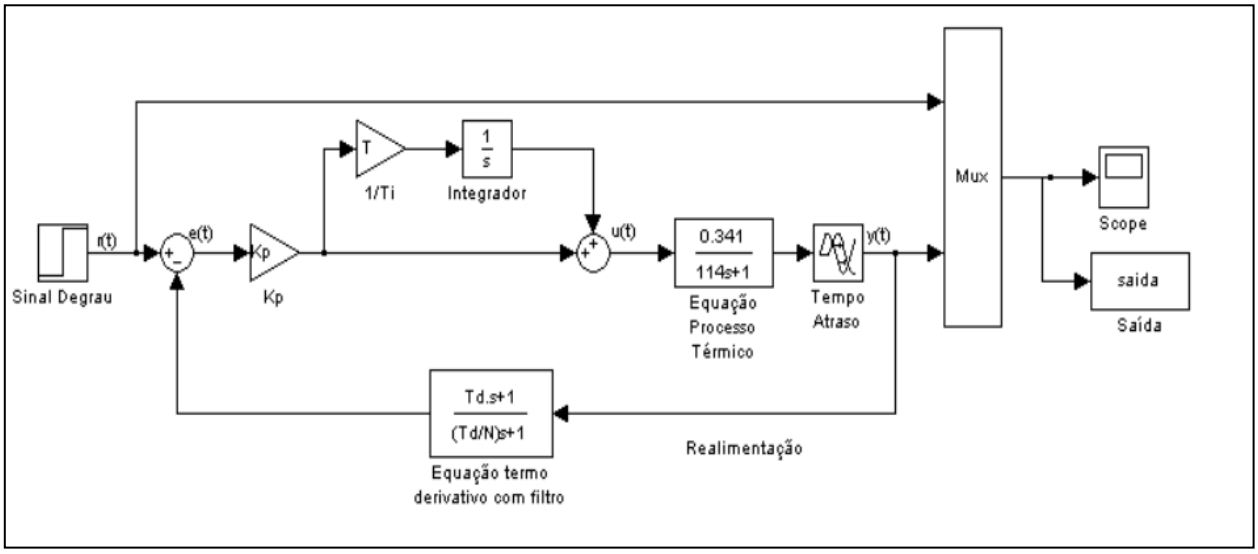


**Figura 28 - Diagrama de blocos da aplicação do controlador PI-D com filtro derivativo**
**Fonte: IWASSE (2009, p. 49).**

A figura 28 mostra o diagrama de blocos do PID em série com filtro derivativo e a equação (19), proposta por Moreno (2010), mostra a discretização desse PID.

e) Controlador PID (Arranjo série com Filtro Derivativo)

Esse controlador é muito utilizado, no entanto sua sintonia é mais difícil, pois todos os ganhos do controlador estão relacionados entre si e possui um filtro no termo derivativo. Esse controlador possui a equação (20) (IWASSE, 2009).

$$u(t) = Kp.\left(N + \frac{1}{Ti}\right).e(t) + Kp.\left(\frac{N-N^2}{Td} + \frac{N-1}{Ti}\right).\int e^{-\frac{n.t}{Td}}\, e(t)dt \qquad (20)$$

Aplicando a Transformada de Laplace:

$$u(s) = Kp.\left(1 + \frac{1}{Ti.s}\right).\left(\frac{1 + Td.s}{1 + \frac{Td}{N}.s}\right).e(s)$$

Para a versão discretizada:

$$\begin{aligned}
u(n) = \{ \quad & \left[T.\left(T^2 + 4.\left(\frac{Td}{N}\right).T + 3.\left(\frac{Td}{N}\right)^2\right)\right].u(n-1) \\
& - \left[T.\left(2.\left(\frac{Td}{N}\right).T + 3.\left(\frac{Td}{N}\right)^2\right)\right].u(n-2) \\
& + \left[T.\left(\frac{Td}{N}\right)^2\right].u(n-3) \\
& + \left[T.(Kp + K.T).\left(T^2 + \left(\frac{td}{N}\right).T + Td.T + Td.\left(\frac{td}{N}\right)\right)\right].e(n) \\
& - [\, Kp.T.\left(T^2 + 2.\left(\frac{Td}{N}\right).T + 2.Td.T + 3.Td.\left(\frac{Td}{N}\right)\right) \\
& + k.T^2.\left(Td.T + 2.Td.\left(\frac{Td}{N}\right) + \left(\frac{Td}{N}\right).T\right)\,].e(n-1) \\
& + \left[kp.T.\left(\left(\frac{Td}{N}\right).T + 3.\left(\frac{Td}{N}\right).Td + Td.T\right) + K.\left(T^2.Td.\left(\frac{Td}{N}\right)\right)\right].e(n-2) \\
& - \left[Kp.Td.\left(\frac{Td}{N}\right).T\right].e(n-3) \quad \} \\
& / \left(T.\left(T^2 + 2.\left(\frac{Td}{N}\right).T + \left(\frac{Td}{N}\right)^2\right)\right)
\end{aligned} \qquad (21)$$

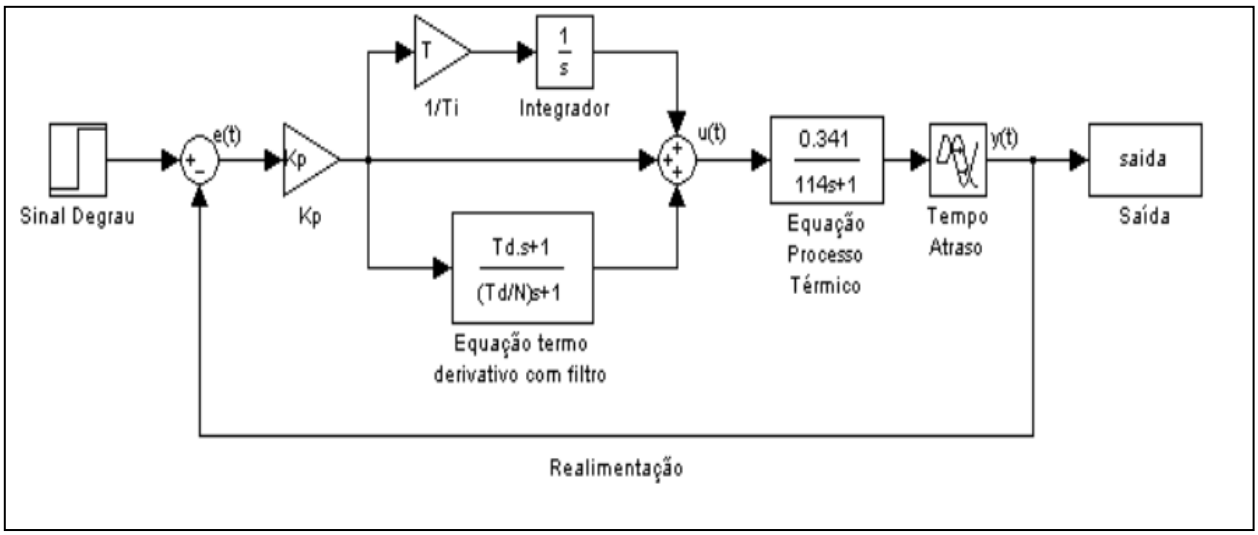


**Figura 29 - Diagrama de blocos da aplicação do controlador PID em série com filtro derivativo**
**Fonte: IWASSE (2009, p. 55).**

A figura 29 mostra o diagrama de blocos do PID em série com filtro derivativo e a equação (21), proposta por Moreno (2010), mostra a discretização desse PID.

## 2.7 Comunicação modbus RTU

Modbus é um protocolo de comunicação de dados utilizado em sistemas de automação industrial posicionado no nível sete do modelo OSI, que fornece comunicação Cliente – Servidor entre dispositivos conectados em diferentes tipos de redes (MODBUS, 2012).

Na figura 30 mostra-se a tabela de aplicações do MODBUS.

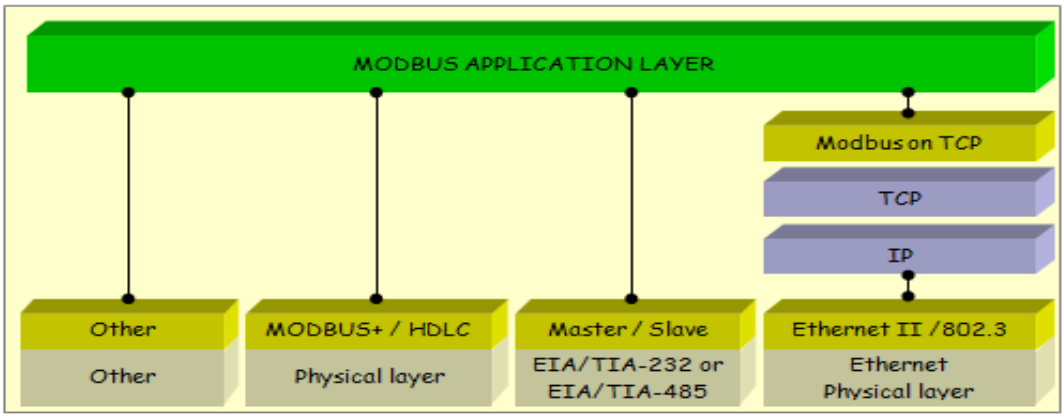


**Figura 30 - Tabela de aplicação do MODBUS**
**Fonte: MODBUS (2012, p. 2).**

O MODBUS vem sendo utilizado desde 1979, foi desenvolvido pela Modicon e continua a permitir que milhões de dispositivos de automação se comuniquem. Hoje, a estrutura do MODBUS continua a crescer. A comunidade pode utilizar o modbus por meio do TCP / IP na porta 502 (MODBUS, 2012). A figura 31 apresenta o formato da mensagem modbus.

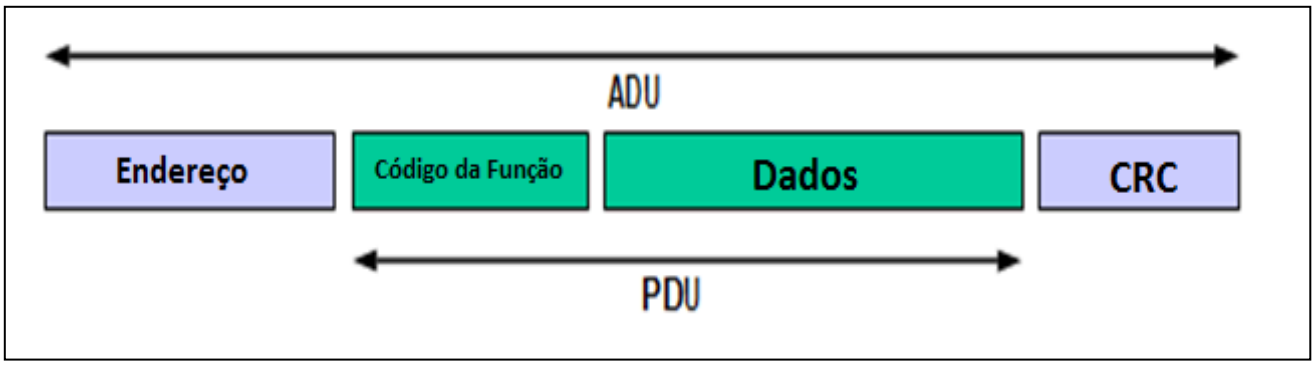


**Figura 31 - Mensagem modbus**
**Fonte: MODBUS (2012, p. 4).**

O Protocolo modbus define uma unidade de protocolo – *protocol data unit* (PDU) - que é independente das outras camadas de comunicação. Essa unidade pode ter campos adicionais chamados de *application data unit* (ADU). O ADU é produzido pelo equipamento escravo e inicia a transcrição modbus. O código da função (*function code*) indica ao mestre qual tipo de ação deverá ser executada. Esse código varia de 1 a 255 na base decimal (os números 128 a 255 são reservados e usados para respostas de exceções). O campo dados da mensagem enviada pelo escravo contém informações adicionais que o mestre utiliza para realizar a ação definida pelo código da função. Essas informações incluem itens como endereços de registro, quantidade de itens a serem manipulados e a contagem de *bytes* o campo. O campo dados também pode ser inexistente (com tamanho zero) em certos tipos de requisições (MODBUS, 2012).

Pacote enviado pelo dispositivo mestre:

O pacote de escrita ou leitura do mestre se resume no quadro 1:

| End. do Dispositivo | Função | End. Registro/Registro Inicial | Quant. de Registros/Dados | CRC |
|---|---|---|---|---|
| 1 *Byte* | 1 *Byte* | 2 *Bytes* | 2 *Bytes* | 3 *Bytes* |

**Quadro 1 - Pacote modbus Mestre**
**Fonte: MODBUS (2012).**

a) Endereço do dispositivo:

É o endereço que foi configurado dentro do dispositivo, não podendo haver dois dispositivos com o mesmo endereço, pois haverá conflito durante a resposta do escravo. Apenas um escravo pode responder por vez. O endereço do dispositivo varia de 1 a 255 podendo ser configurado de acordo com a necessidade do projeto (MODBUS, 2012).

b) Função:

É o comando, ou seja, o que o escravo tem que realizar (MODBUS, 2012). Ver quadro 2.

c) Endereço do Registro/Registro inicial:

É o endereço do registro que se encontra a variável requisitada pelo mestre para a leitura ou a escrita. Quando a leitura é feita em mais de um registro esse campo é definido com o registro inicial (MODBUS, 2012).

d) Quantidade de registros:

É a quantidade de registros a mais que o mestre requisita além dos endereços do campo anterior (Registro Inicial) (MODBUS, 2012).

e) CRC:

Pode ser chamado de “checksum” e representa a soma dos *bytes*. Serve para confirmar que a mensagem que está chegando corretamente (MODBUS, 2012).

| Código do comando | Descrição |
|---|---|
| 01 | Lê um número variável[1] de saídas digitais (bobinas) |
| 02 | Lê um número variável[1] de entradas digitais |
| 03 | Lê um número variável[1] de registros retentivos (saídas analógicas ou memórias) |
| 04 | Lê um número variável[1] de registros de entrada (entradas analógicas) |
| 05 | Força uma única bobina (altera o estado de uma saída digital) |
| 06 | Preset de um único registro (altera o estado de uma saída analógica) |
| 07 | Lê exceções[2] (registros de erro) |
| 08 | Várias funções de diagnóstico |
| 15 | Força uma quantidade variável[1] de bobinas (saídas digitais) |
| 16 | Preset de uma quantidade variável[1] de registros (saídas analógicas) |
| [1] A quantidade de variáveis a ler é definida no frame de solicitação<br>[2] Oito bits previamente configurados. Não é necessário fornecer parâmetros de endereçamento com este comando, pois o escravo vai enviar sempre os oito bits pré-configurados. | |

**Quadro 2 - Tabela de Códigos de Comandos modbus.**
**Fonte: MODBUS (2012).**

O pacote de resposta do escravo está na quadro 3:

| Endereço do Escravo | Código da Função | Dados | CRC |
|---|---|---|---|
| 1 *byte* | 1 *byte* | 0 a 252 *bytes* | 2 *bytes* (CRC-16) |

**Quadro 3 - Pacote modbus Escravo.**
**Fonte: MODBUS (2012).**

a) Endereço do escravo:

É o próprio endereço do dispositivo escravo (MODBUS, 2012).

b) Código da função:

É o mesmo código da função que o escravo recebeu no pacote do mestre, a resposta obrigatoriamente tem que conter o mesmo código (MODBUS, 2012).

c) Dados:

Este campo contém os dados requisitados pelo mestre, valores dos registros, ou uma confirmação de escrita contendo a valor do que foi escrito (MODBUS, 2012).

# 3 DESENVOLVIMENTO

Neste capítulo será abordado o desenvolvimento do protótipo para ensaios de técnicas de controle programadas em microcontrolador, sua estrutura mecanica, o *hardware*, a programação, a comunicação, o *software* supervisório e o funcionamento do protótipo.

Os *software*s utilizados no desenvolvimento do protótipo foram: Elipse E3 para realizar a supervisão dos processos, assim como fornecer gráficos, imagens, valores, medidas e animações em tempo real para o usuário; Labcenter Eletronics Proteus 7.1 para fazer a simulação dos componentes eletrônicos, do *software* no microcontrolador e da placa de circuito impresso; emulador Hércules para realizar testes com a comunicação mediante as portas seriais; Microchip MikroC PRO for PIC, que é um ambiente de desenvolvimento de códigos a serem implementados no PIC; Virtual Serial Ports para criação de portas seriais virtuais para realizar testes de comunicação, com ele foi possível efetivar a comunicação do Proteus com o E3.

## 3.1 Estrutura Mecânica

A mecânica foi desenvolvida com o objetivo de criar uma sustentação e ao mesmo tempo reduzir o peso do protótipo. Devido à potência do ventilador ser limitada, prezou-se por um menor peso dos materiais para que a força do ventilador seja bem aproveitada. O material escolhido foi o *renshape*, comumente usado em protótipos devido a sua leveza e moderada resistência.

A parte mecânica é composta por uma balança, duas hastes de sustentação e um suporte. O ventilador é preso a uma das hastes fazendo com que esta fique mais pesada.

A figura 32 mostra o projeto em desenvolvimento e durante as fases de teste. Nela pode-se observar a haste de suporte, o sensor potenciômetro, o motor e a caixa de suporte do projeto.

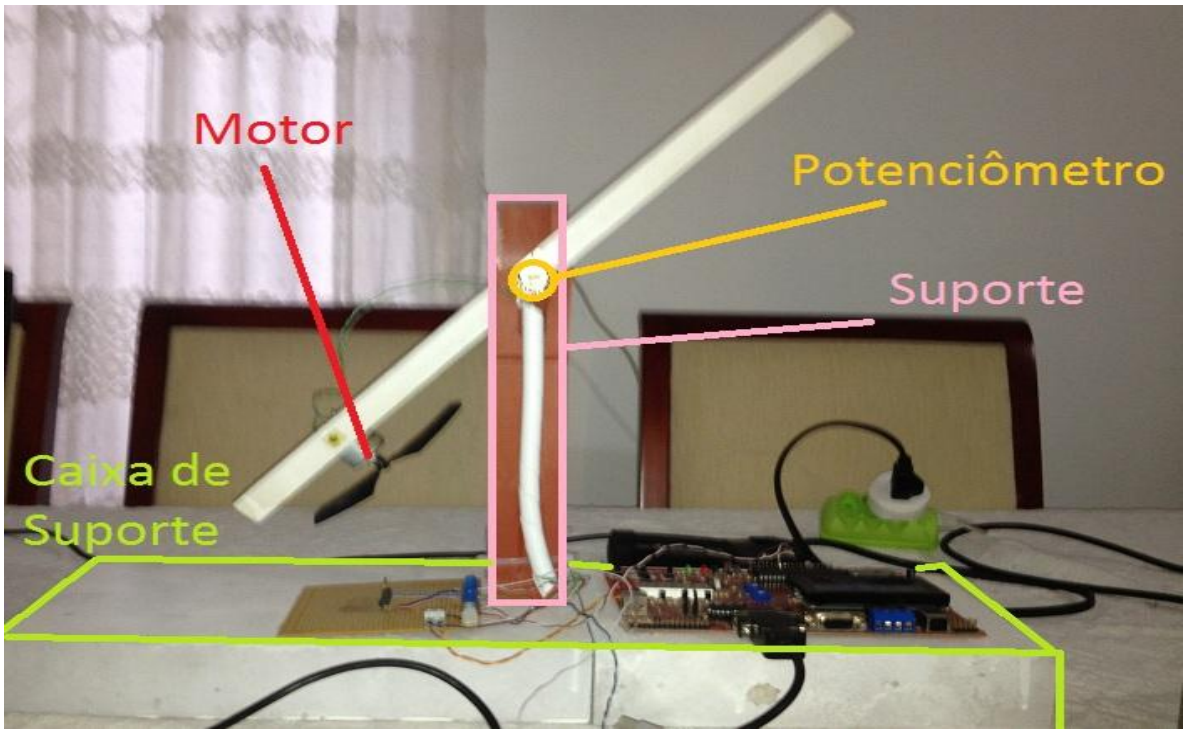


**Figura 32 - Estrutura Mecânica**
**Fonte: Autoria própria.**

## 3.2 *Hardware*

A definição do *hardware* foi peça importante no desenvolvimento do trabalho, pois definiria a base de programação e os métodos de comunicação, além de influenciar na escolha do motor e transdutor angular do protótipo. A equipe usou uma placa fabricada pela empresa Smart Radio, essa PCI (placa de circuito impresso) é própria para desenvolvimento de microcontroladores da famílias PIC e possui vários recursos para desenvolvimento, entre eles: *driver* para motor de passo, duas interfaces seriais RS232, teclado, entrada para teclado de PC, conector de cartão SD, interface USB, LCD gráfico, entre outras. Neste projeto foi utilizado apenas o circuito responsável pela alimentação do PIC e a interface com serial RS232. Utilizamos o PIC 18F4520 pelo custo e pela praticidade do componente. Apesar de pouco utilizado durante o curso, os autores ganharam experiência com o *hardware* através da elaboração de outros projetos que, pertinentes ao curso, influenciaram na decisão a favor do PIC18F4520. Como o projeto necessita de alto processamento, o microcontrolador trabalha a uma frequência de 20MHz. Para realizar o trabalho do sensor de ângulo, escolheu-se o

potenciômetro, o qual está localizado no centro da balança e aufere o ângulo à medida que a balança pende para um dos lados. O sinal varia de 0V a 5V sendo compatível com o limite de tensão da entrada analógica do PIC como receptor do sinal.

Foi utilizado o conversor analógico/digital presente no próprio PIC, o qual possui 10 entradas para os modelos de 28 pinos e 13 entradas para os modelos de 40 a 44 pinos. O PIC 18F4520 pode converter uma entradal analógica para o correspondente a 10 *bits* de um número digital. O conversor é capaz de realizar 100.000 amostras por segundo.

O PIC 18F4520 possui também 1536 *bytes* de memória RAM, memória de armazenamento do tipo FLASH com tamanho de 32KB, quatro timers, sendo 1 de 8 *bits* e 3 de 16 *bits*, EEPROM de dados de 256 *bytes*, alimentação de 2V a 5.5V e dois comparadores. A figura 33 mostra a disposição dos componentes na placa do *hardware*.

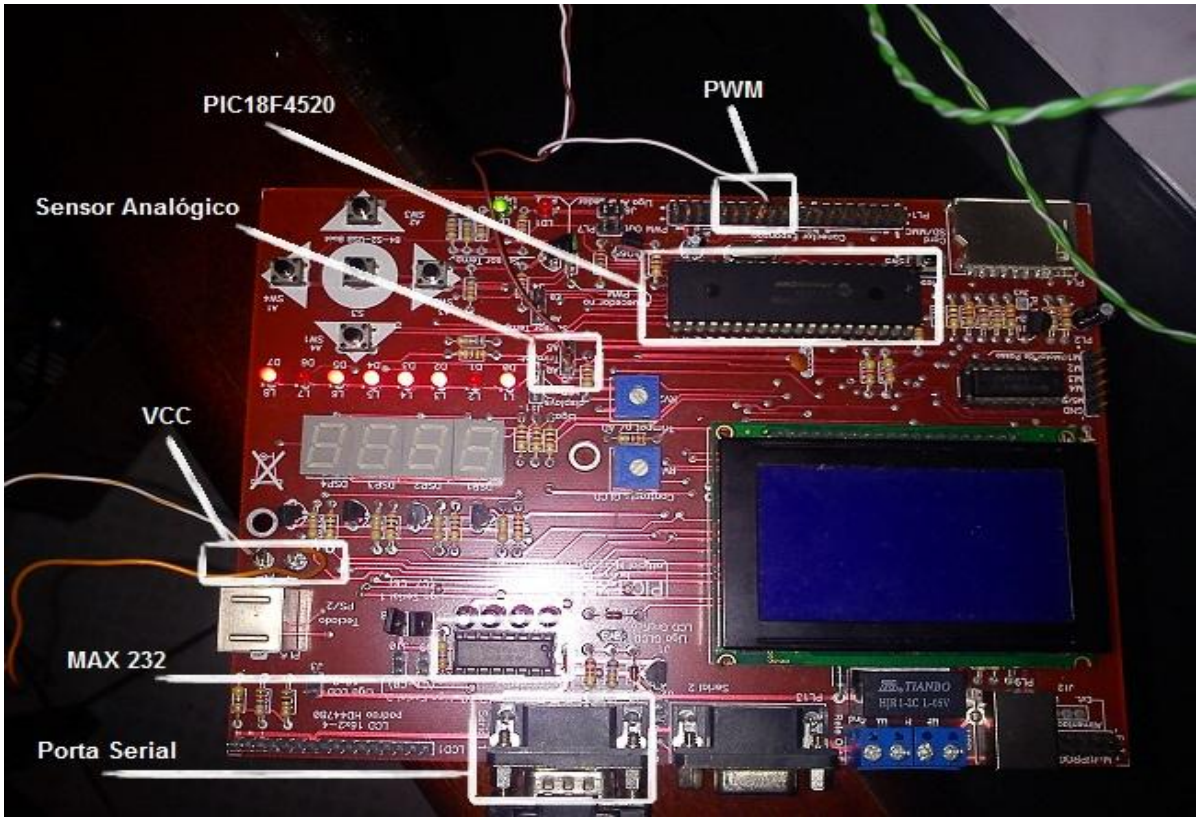


**Figura 33 - Placa do *hardware* durante os testes**
**Fonte: Autoria própria.**

Quanto ao esquemático, presente no apêndice C, tem-se que o motor é acionado pelo o MOSFET IRF 620, que por sua vez é chaveado com um sinal PWM (permite controlar a potência do motor através da largura de pulso) à saída do controlador PID.

O potenciômetro é o sensor para realimentação do sistema e à medida que a balança pende para um dos lados, o mesmo pode aumentar ou diminuir sua resistência.

Quanto ao PIC18F4520, existiram várias características que motivaram sua utilização. Possui frequência de *clock* de até 40 MHz, sendo que no projeto foi utilizado um cristal de 20 MHz; conversores analógicos digitais, sendo somente um usado no projeto; Compatibilidade com a serial RS232, pois essa é a porta serial utilizada no projeto; Níveis de prioridade de interrupção: no projeto são utilizados dois níveis, sendo o estouro do *timer* prioritário sobre o conversor analógico/digital.

Além disso, o PIC possui quatro saídas de modulação em largura de pulso (PWM) sendo que, no projeto, é utilizado um PWM para modular os pulsos que irão aumentar ou diminuir a velocidade do ventilador. São também utilizados dois temporizadores de 16 bits do PIC, para a programação.

Já o MAX232 é um circuito integrado produzido pela MAXIM que converte sinais no padrão RS232, que é de -12V a 12V, para sinais TTL compatíveis com os sinais de circuitos lógicos e microcontroladores. Os sinais de saída podem ser de 0V a 3.3V ou 5V. No projeto é utilizado a tensão de 5V. O próprio MAX232 gera uma tensão mais alta suficiente para o seu funcionamento, não necessitando ser alimentado com uma fonte de +12V e -12V como acontece com outros circuitos integrados de função semelhante.

Para explanação do funcionamento do *hardware*, tem-se a Figura 34, a qual apresenta um diagrama de blocos do circuito. A mensagem vinda do *software* E3 passa pelo *driver* modbus RTU, fornecido pela elipse E3, e chega ao conversor MAX232 por meio de uma porta serial DB9.

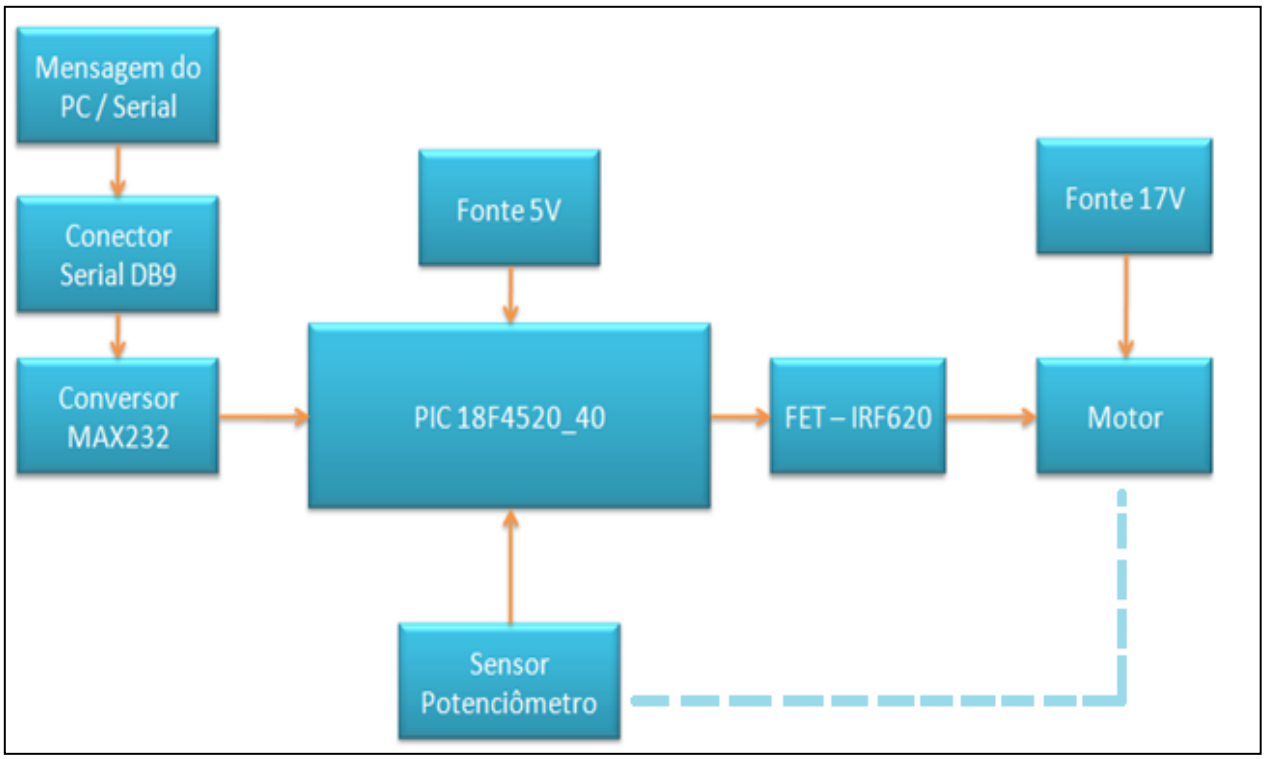


**Figura 34 - Diagrama de Blocos do *Hardware***
**Fonte: Autoria própria.**

O conversor por sua vez, transmite a mensagem ao PIC18F4520_40, sendo este o responsável por efetuar as rotinas e cálculos necessários à continuidade do processo. Há também o valor recebido por meio do sensor potenciômetro, o qual também entra no cálculo do Erro.

Após isso, é enviado um sinal ao MOSFET que regula a tensão aplicada ao motor. Todas as ligações e pinagens, assim como todas as nomenclaturas dos componentes podem ser encontradas no apêndice C.

## 3.3 Programação

A programação foi realizada com o MikroC e baseou-se em dois tipos de discretização, sendo que cada uma apresentou respostas diferentes em relação comportamento da balança. Foram efetuados diversos testes com os dois tipos de discretização, os resultados foram comparados e elaborou-se um gráfico do

desempenho do protótipo em relação à programação. O gráfico de testes pode ser visualizado no capítulo 4. Com esses testes tem-se uma melhor visão do PIC 18F4520 e sua velocidade de resposta em relação às diferentes programações, resultando dessa maneira em inúmeras formas e possibilidades de controle para o processo e efetivando os objetivos da elaboração do protótipo. O operador também pode escolher a função de malha aberta, a qual não possui realimentação e depende de uma calibração prévia.

O programa conta com três funções principais: *Main*, *Interrupt* e PID. A função *Main* é executada desde o inicio do programa configurando o microcontrolador, suas interrupções, conversor analógico/digital e saída PWM. A função PID dispõe os cálculos necessários para a implementação da discretização. Nesta obra são usadas duas discretizações. As equações podem ser observadas no capítulo 2 desta obra.

A função *interrupt* é a função condicionada ao acontecimento de um evento, portanto é ela que efetivamente se comunica com as portas do projeto através da função de comunicação modbus RTU. As funções Main, Interrupt e PID, assim como outras funções necessárias ao funcionamento do PIC podem ser vistas no apêndice A.

Para o desenvolvimento do algoritmo foi de extrema importância o estudo do trabalho do Moreno (2010), no qual são apresentadas equações discretizadas. Essas equações podem ser encontradas na seção 2.6 desta mesma obra. Pelo tempo hábil para realizar o projeto foram feitas as conversões de dois dos cinco algoritmos em linguagem C ficando as outras três podendo ser implementadas em trabalhos futuros. A simulação da comunicação, dos cálculos dos algoritmos e configuração das entradas e saídas foi feita no *software* Proteus e posteriormente foram comparadas com o resultado real. Após os testes, foram integrados os elementos da eletrônica e da mecânica com êxito.

O fluxograma da Figura 35 representa o funcionamento da programação. O código Main, Config PIC, Interrupt, Comunicação modbus e PID podem ser encontrados nos Apêndices A e B. Quando ligado, a primeira tarefa a ser feita é a execução da Função Main. Essa função zera todas as variáveis e em seguida lê o último *setpoint* setado e as últimas configurações de Kp, Ti, e Td que foram setadas. Em seguida, a função Main aciona a função Config PIC.

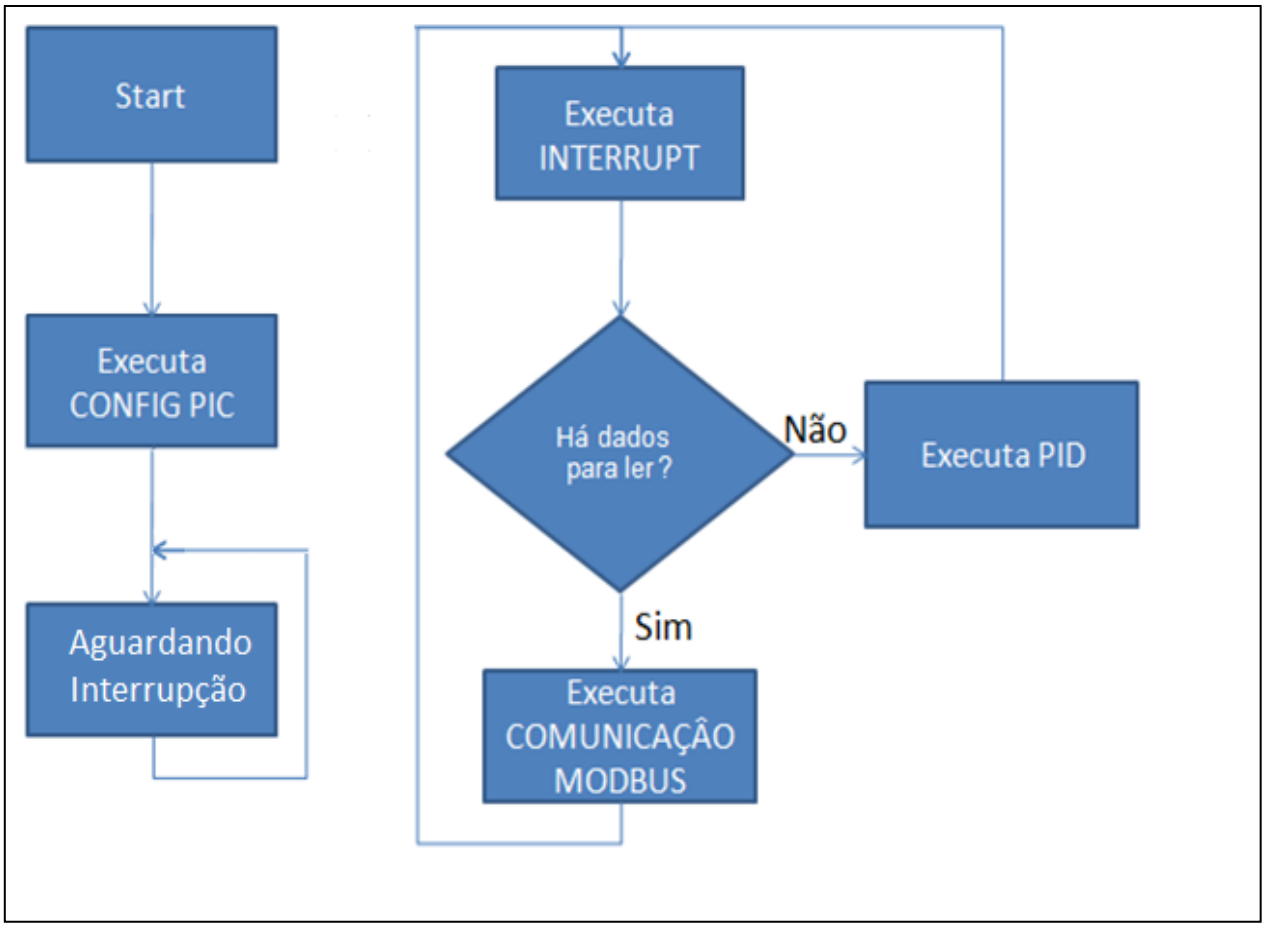


**Figura 35 - Fluxograma de Programação**
**Fonte: Autoria própria.**

A função Config PIC por sua vez aciona todos os comandos para a correta execução do PIC. Nessa rotina é configurado o *timer* em 100Hz para gerar a interrupção, porta A0 como entrada analógica, PWM com frequência de 3khz e a porta serial com *baudrate* de 9600bps. A função Config PIC também configura a função Interrupt que é executada periodicamente enquanto o sistema estiver ligado.

A função *Interrupt* é a função principal de execução do programa. A primeira coisa a ser feita é verificar a recepção de dados na entrada. Caso haja dados, é executada a função comunicação modbus.

Caso não haja dados novos, há a leitura do sensor potenciômetro e o cálculo do erro. Em seguida, é executada a função PID com uma das discretizações que foram

selecionadas e seus respectivos cálculos. Esse valor calculado é transmitido ao PWM, que atuará no Motor.

Em ambos os casos (recebimento ou não de dados), o próximo passo a ser realizado pela programação é o retorno periódico à função *interrupt*. Esse passo se repete infinitamente até o desligamento do sistema.

## 3.4 Comunicação

O desenvolvimento da comunicação se processou em várias etapas. O estudo de viabilidade de protocolos foi a primeira delas. Esse estudo mostrou o protocolo modbus RTU sendo o mais adequado para a utilização devido à maior facilidade em sua programação pelos desenvolvedores do projeto e pela experiência dos executores com o *software* Elipse E3, promovendo assim a eleição do modbus como o protocolo mais apto à situação. Outro processo da comunicação foi o desenvolvimento de uma função no PIC que promovesse a interação do projeto por meio do protocolo modbus RTU. A simulação de comunicação para a realização de testes ocorreu com a utilização de portas virtuais seriais. Foi criado um par de portas seriais virtuais conectadas entre si de modo que o *software* Proteus comunica-se com a porta COM2 e a porta virtual serial redireciona-o para a COM3 que por sua vez se comunica com o E3 tornando assim fácil e rápida a realização dos testes. Posteriormente foi realizado o teste real no projeto que funcionou com êxito. A tabela 8 mostra os endereços utilizados pelos programadores na configuração da comunicação por meio do modbus.

O protocolo de comunicação modbus funciona no modo mestre-escravo, ou seja, o dispositivo mestre requisita informações de um de seus escravos, que o responde. Nesse protocolo a comunicação é somente entre mestre e escravo, não sendo possível a comunicação de escravo para escravo. As interfaces mais usadas são as RS485, RS232 e TCP/IP. As interfaces RS485 e TCP/IP são geralmente usadas para comunicar dispositivos com uma longa distância. No caso deste projet,o preferimos usar a RS232, pois não havia necessidade de longa distância já que é uma conexão simples ponto a ponto.

| Endereço | Tipo | Descrição | Escrita/Leitura |
|---|---|---|---|
| 0 | Byte | Entrada Analógica | L |
| 1 | Byte | SetPoint Desejado | E/L |
| 2 | Byte | Saída PWM | L |
| 3 | Byte | Ganho Proporcional | E/L |
| 4 | Byte | Tempo Integrativo | E/L |
| 5 | Byte | Tempo Derivativo | E/L |
| 6 | Byte | Taxa de amostragem | E/L |
| 7 | Byte | STATUS | E/L |
| 7.0 | bit | Ligar sistema | E/L |
| 7.1 | bit | Saída PWM=0% | L |
| 7.2 | bit | Saída PWM=100% | L |
| 7.3 | bit | Sistema Ligado | L |

**Quadro 4 - Mapa modbus.**
**Fonte: Autoria própria**

Para desenvolver esse protocolo dentro do *firmware* do PIC, foi necessário dividi-lo em algumas funções. Quando o *hardware* recebe algum pacote de dados pela porta serial, ele levanta um *flag (sinalizador)* que ativa a função principal chamada de *comunicacao_modbus().* No supervisório a comunicação é feita com o auxílio do *driver* de comunicação modbus da elipse.

A função de comunicação modbus foi totalmente desenvolvida pela equipe, mediante a utilização do programa MikroC.

A figura 36 apresenta o Fluxograma de comunicação modbus. Quando o Elipse E3, por meio do *driver* modbus envia dados à placa, ocorre a chamada da função

Comunicação modbus. Primeiramente é verificado se o endereço do dispositivo está correto. Caso não esteja, não é feito nada.

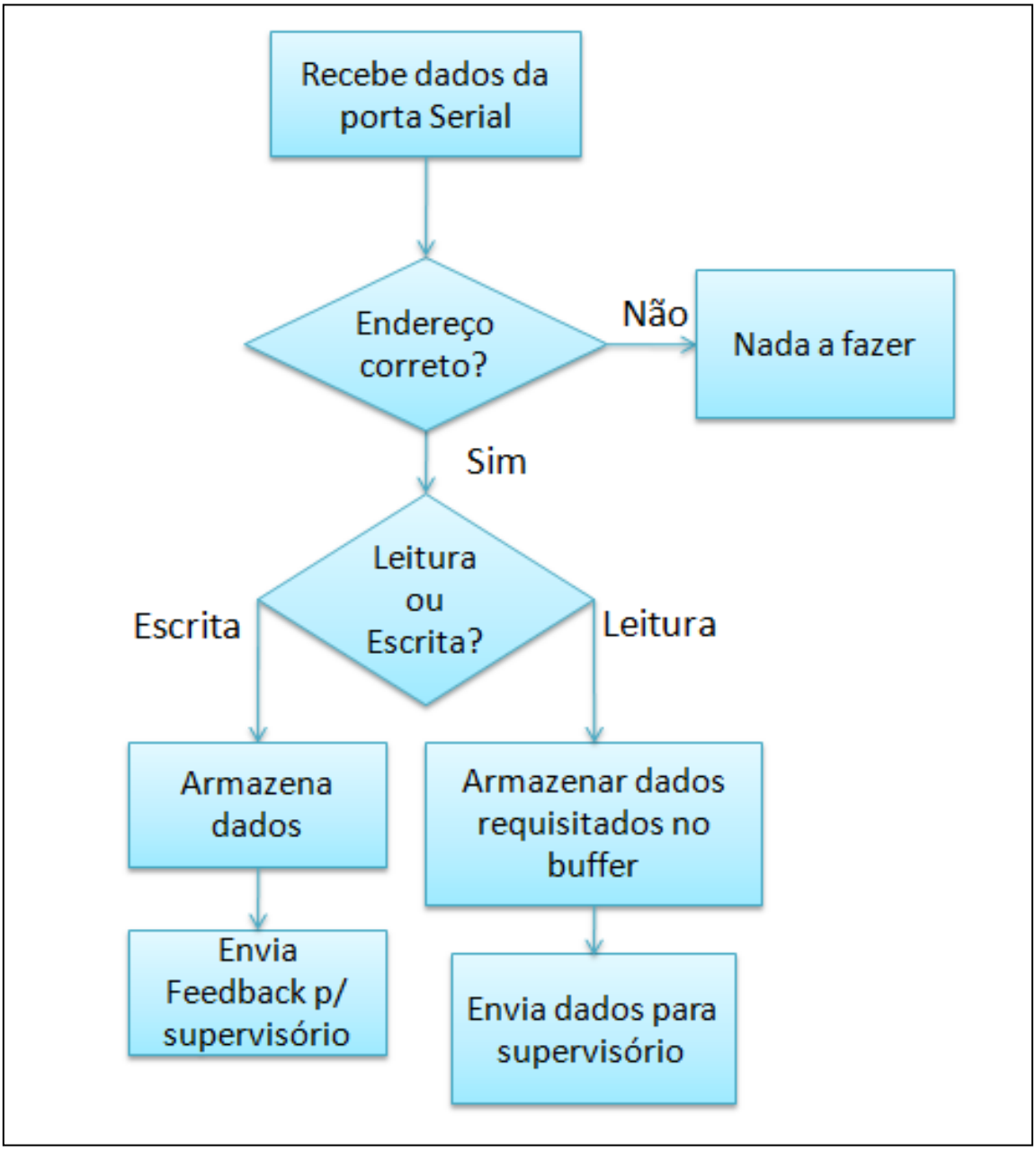


**Figura 36 - Fluxograma de Comunicação modbus**
**Fonte: Autoria própria.**

Se o endereço for correto, o próximo passo realizado pelo programa é a seleção entre Escrita ou Leitura. Caso o Supervisório deseje ler os dados, como por

exemplo, quando o *software* lê a angulação fornecida pelo potenciômetro, a função leitura é selecionada e ocorre o armazenamento dos dados lidos na memória RAM e EEPROM para restaurar os valores caso o sistema desligue. Após isso, os dados são enviados para a tela do supervisório.

Caso o supervisório deseje escrever algo no PIC, é executada a função escrever. Esse momento ocorre, por exemplo, quando ocorrem mudanças do *setpoint* ou das variáveis. O operador escreve a nova variável no *software* supervisório e esse se encarrega de enviá-las ao PIC. Após isso, ocorre uma realimentação realizada pelo PIC, o qual confirma a operação realizada.

Cabe lembrar que o PIC funciona como escravo, ou seja, ele não tem autonomia para requisitar dados ou enviar dados. Todas essas funções são executadas mediante solicitação do supervisório.

## 3.5 Supervisório

A escolha do *software* supervisório Elipse E3 deve-se à necessidade de apresentar várias utilidades que estão sendo amplamente utilizadas no mercado de trabalho. É um *software* mais complexo que o Elipse SCADA e que possui interfaces para desenvolvimento, monitoramento e programação em linguagem C. Possui editores gráficos, links, animações, gráficos, sendo possível a observação em tempo real dos processos assim como das comunicações das medidas recebidas e dos equipamentos/máquinas em utilização.

O desenvolvimento no *software* Elipse E3 se deu em duas maneiras. Primeiro, relativamente às comunicações, foi configurado e escolhido o *driver* correto para propiciar a comunicação entre o E3 e o *hardware*. O *driver* é fornecido pela Elipse e pode ser encontrado no próprio site da empresa. Há a necessidade de um ajuste na comunicação como estabelecimento de parâmetros para o tipo modbus e as configurações das tags de comunicação que permitem ler e escrever. Posteriormente à parametrização do *driver*, passou-se a desenvolver a interface gráfica. A interface possui duas telas, sendo uma a principal e a outra de gráficos. A tela principal contempla a balança em tempo real, sete demonstrações numéricas, sendo elas:

entrada, saída, *setpoint*, ganho proporcional, tempo integrativo, tempo derivativo e taxa de amostragem. Essa mesma tela possui dois botões, sendo eles Liga/Desliga, permitindo o acionamento remoto, e o botão Gráfico. Esse último leva à tela de gráficos, a qual possui a amostragem em tempo real possuindo dois gráficos com três penas (traços), sendo essas: a de leitura analógica, a de leitura do SP e a de leitura da saída. Com a praticidade do *software* a qualquer momento podem ser adicionadas novas "penas" para elaboração de mais gráficos dispostos no mesmo ou em outro quadro.

Tela Principal do *Software* Supervisório:

Nessa tela, o operador entra com os dados de Entrada, Saída, entre outros e, além disso, foram desenvolvidos *checkboxs*, podendo serem selecionados Arranjos de PID 1 e 2 e Malha Aberta. O intertravamento dos botões é feito por meio do Elipse. Na tela é observada em tempo real a balança. A animação foi feita por meio de quadros que mudam conforme o valor indicado pelo sensor. Vide figuras 37 e 38.

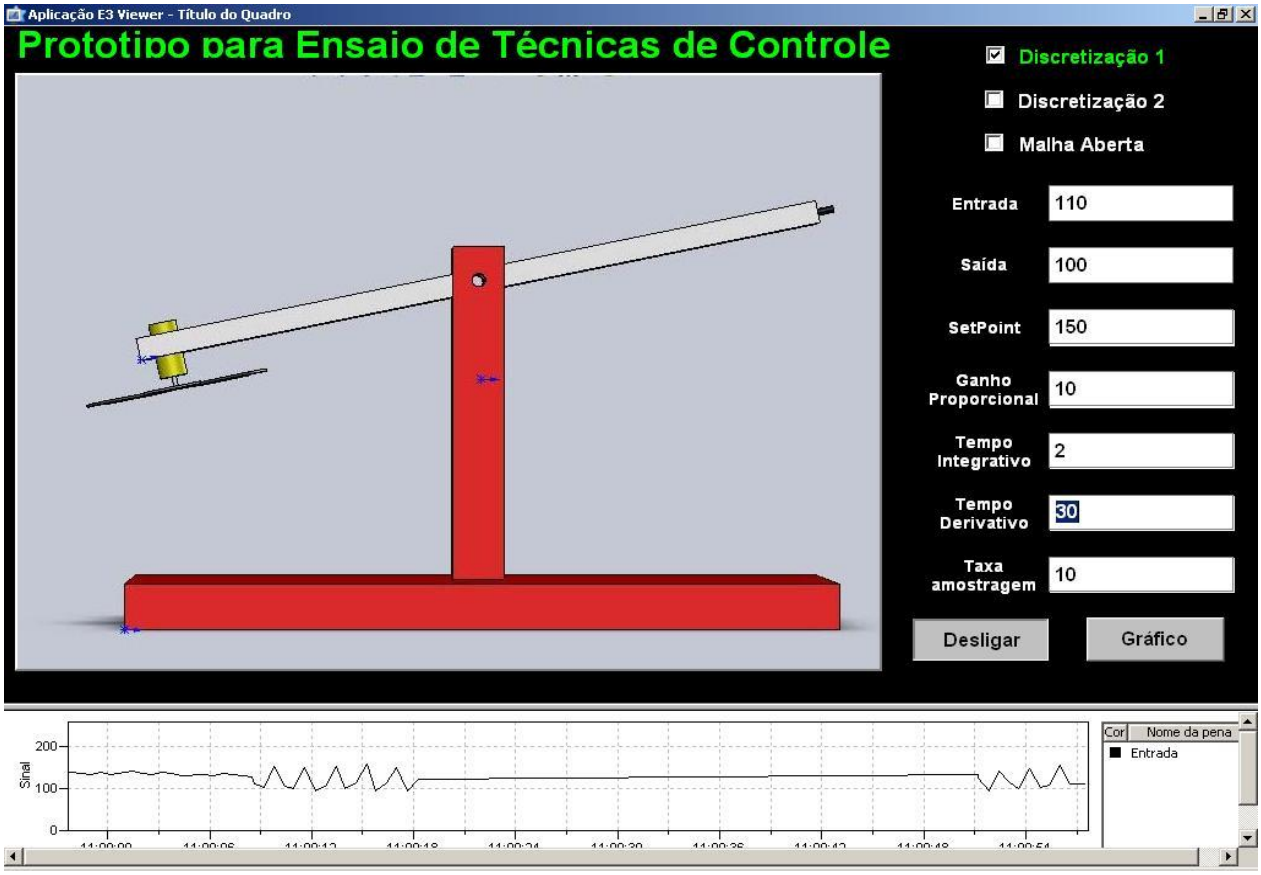


**Figura 37 - Tela principal supervisório**
**Fonte: Autoria própria.**

Tela Gráfico:


**Figura 38 - Tela auxiliar supervisório**
**Fonte: Autoria própria.**

## 3.6 Funcionamento

Para iniciar, deve ser feita previamente ao acionamento do protótipo parametrização dos valores do tempo de integração, tempo de derivação, *setpoint*, ganho proporcional e da angulação inicial da balança. Essas definições são feitas por meio do *software* Elipse E3, no programa desenvolvido pelos executores do projeto, em sua tela principal. A operação tem por objetivo estipular valores para serem enviados ao microcontrolador e posteriormente equilibrarem a balança mediante a aceleração ou

desaceleração do ventilador. Feitos esses procedimentos, o operador deverá observar se os seus parâmetros estão corretos e se a balança está se equilibrando. O algoritmo, que deve ser previamente implementado no PIC, encarregar-se-á de realizar o controle do processo, minimizando o erro e estabilizando a balança novamente. Os procedimentos poderão ser acompanhados através da tela principal do projeto, a qual demonstrará a balança em tempo real e uma tela de gráfico para acompanhamento, presente no programa. Os gráficos do *setpoint*, da saída do controlador e da entrada poderão ser observados na tela auxiliar do programa que será acessada ao clicar-se no botão gráfico, presente na tela inicial. Eles representam as respostas enviadas pelo PIC através da comunicação realizada por modbus RTU mediante a utilização do *driver* fornecido pela Elipse. O operador, então, poderá realizar diversos experimentos, desestabilizando a balança ou criando novas configurações ou novos algoritmos no PID. O programa, desenvolvido no supervisório Elipse, possui também duas opções de equações discretizadas ou a utilização em Malha Aberta. A figura 39 mostra o projeto durante a fase de testes.

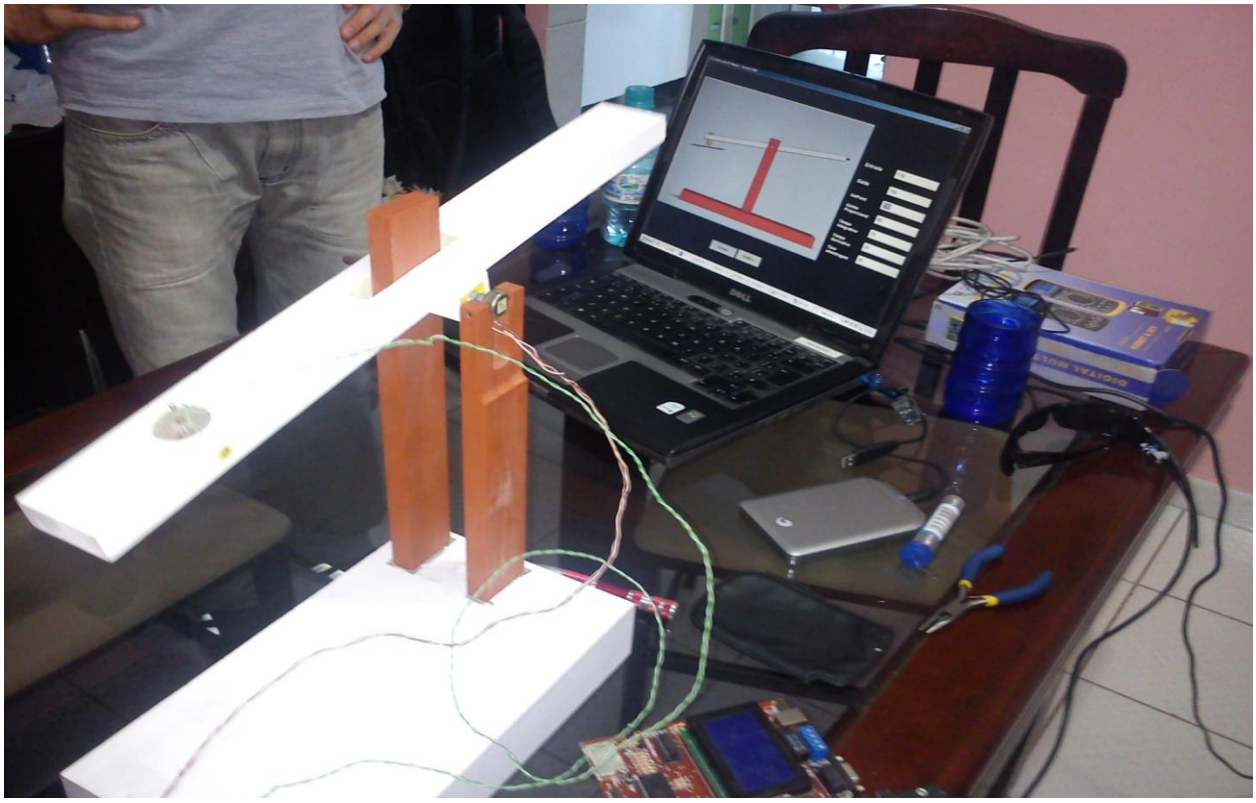

**Figura 39 - Projeto durante os testes**
**Fonte: Autoria própria.**

# 4 TESTES

Esse capítulo visa reportar a realização de testes para comparação entre diferentes arranjos propostos no trabalho em relação a diferentes *setpoint*s estabelecidos. Há também a realização de testes com a malha aberta.

Foram realizados testes com dois tipos de arranjo PID (Controlador PID arranjo paralelo ideal; e Controlador PID arranjo série), e com o modo malha aberta. Para o primeiro arranjo de PID dá-se o nome de PID1 e para o segundo PID2. Quanto aos arranjos, foram feitas duas amostras para cada tipo e foram usados dois tipos de configuração, um para cada arranjo. Quanto à malha aberta, foram feitas duas amostras para dois tipos diferentes de *setpoint*.

O valor de entrada é medido por meio da resposta do potenciômetro. O *setpoint* é o indicado pelo operador, e a saída corresponde à porcentagem de atuação do Motor no processo.

Os valores do *setpoint* e da saída variam entre 80 e 180(valor em 8bits lido do potenciômetro), 80 representa a angulação mínima e 180 representa a angulação máxima possível pela balança, sendo o valor de 130 a 135 o valor em que a balança fica na horizontal. Já o motor é representado no gráfico somente até a escala de 100. Seus valores estão escalonados em porcentagem. A unidade de tempo utilizada é de 100ms.

O PID 1 usou os valores de kp=100, Ti=50, Td=40 e Taxa de amostragem de 20Hz. Esses valores foram auferidos pelos desenvolvedores no método da tentativa e erro. São valores que mostraram boas respostas aos dois tipos de *setpoint*. O primeiro *setpoint* utilizado foi o de 130 como mostra a figura 40.

Observa-se uma acentuada resposta com um grande *overshoot* atingindo a marca de 180. Esse acontecimento resulta da manutenção do ventilador à força máxima, visando à recomposição do valor de entrada ao *setpoint* estabelecido.

Nota-se a redução da força do ventilador e a consequente queda do valor de entrada. No gráfico da figura 40, percebe-se que o sistema consegue eliminar o erro em

regime permanente e consegue se estabilizar por volta do tempo de 10 segundos. Após isso, não há mudanças consideráveis nas variáveis, permanecendo o sistema estável.

Para o segundo teste com o arranjo 1 foi feita uma amostra com o *setpoint* de 150. O resultado é demonstrado na figura 41.

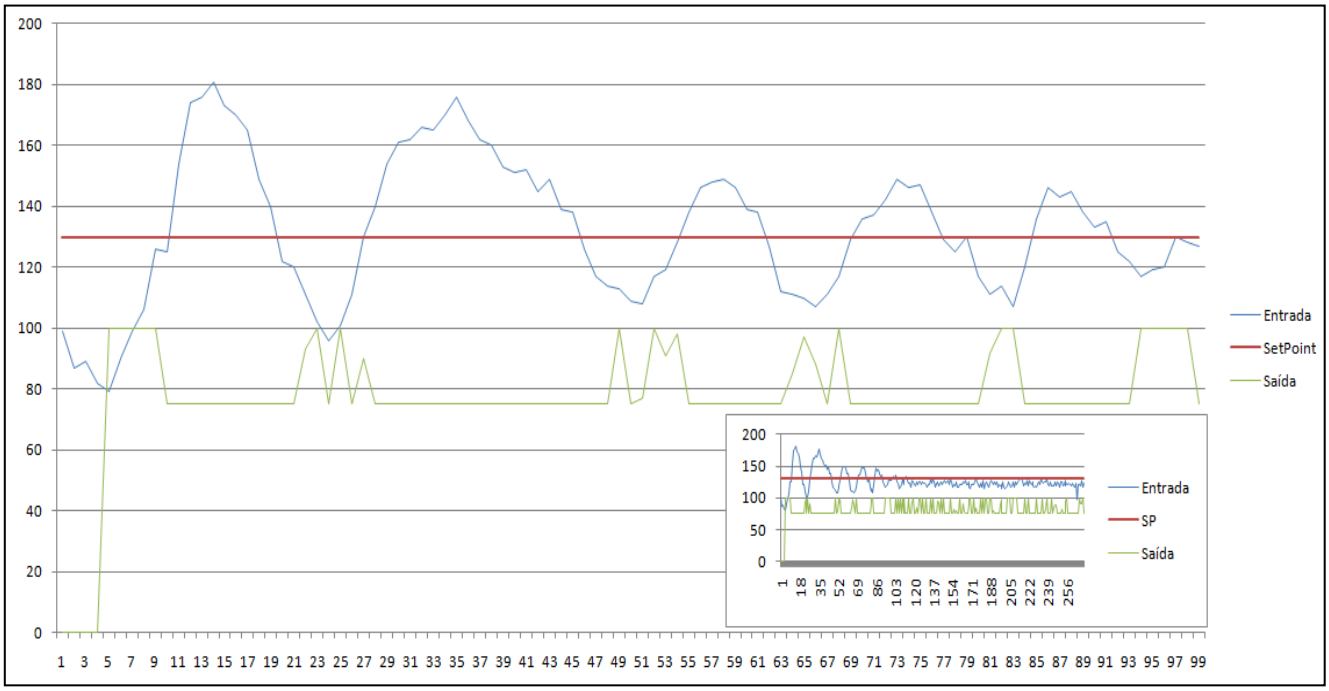


**Figura 40 - Resposta do PID1 ao *setpoint* de 130**
**Fonte: Autoria própria.**

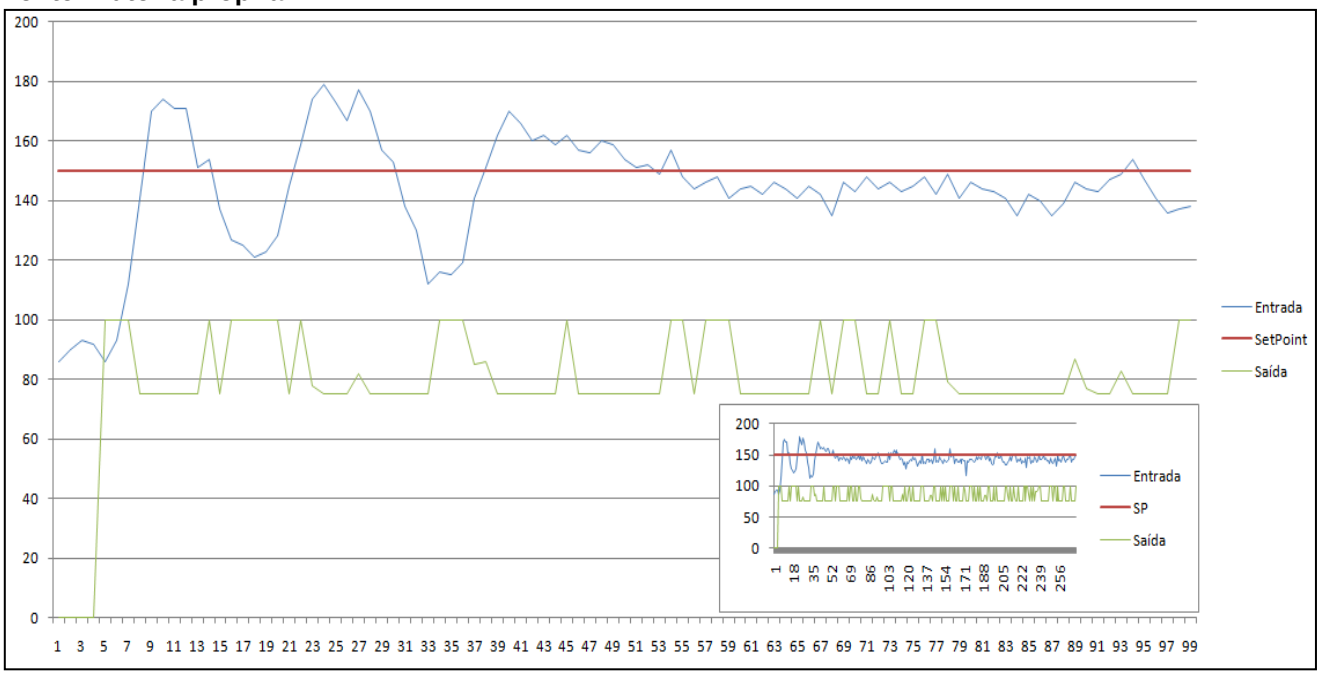


**Figura 41 - Resposta do PID1 ao *setpoint* de 150**
**Fonte: Autoria própria.**

Percebe-se que a configuração proposta para o PID1, de kp=100, Ti=50, Td=40 apresenta uma resposta melhor quando o *setpoint* foi de 150 comparado ao de 130. Ainda assim há um *overshoot*, mas significantemente menor comparado ao experimento anterior. O sistema se mostra estável aos 5 segundos permanecendo assim até o tempo final da amostragem. Não há tanta oscilação quanto no primeiro experimento. A planta tende-se a estabilizar mais facilmente com *setpoint*s mais elevados.

Os mesmos *setpoint*s foram mantidos para a realização dos experimentos com o PID2. O sistema algumas vezes apresenta uma “demora” no cálculo do arranjo 2 o que compromete parcialmente alguns sinais.

Para a utilização do PID2, houve outro ajusto de valores sendo que dessa vez tem-se Kp=170 Ti=60 Td=20 e Taxa de amostragem=50. Os valores foram sintonizados pelo método da tentativa e erro também. Os valores que haviam sido utilizados no arranjo 1 não são compatíveis com o arranjo 2. Vide figura 42:

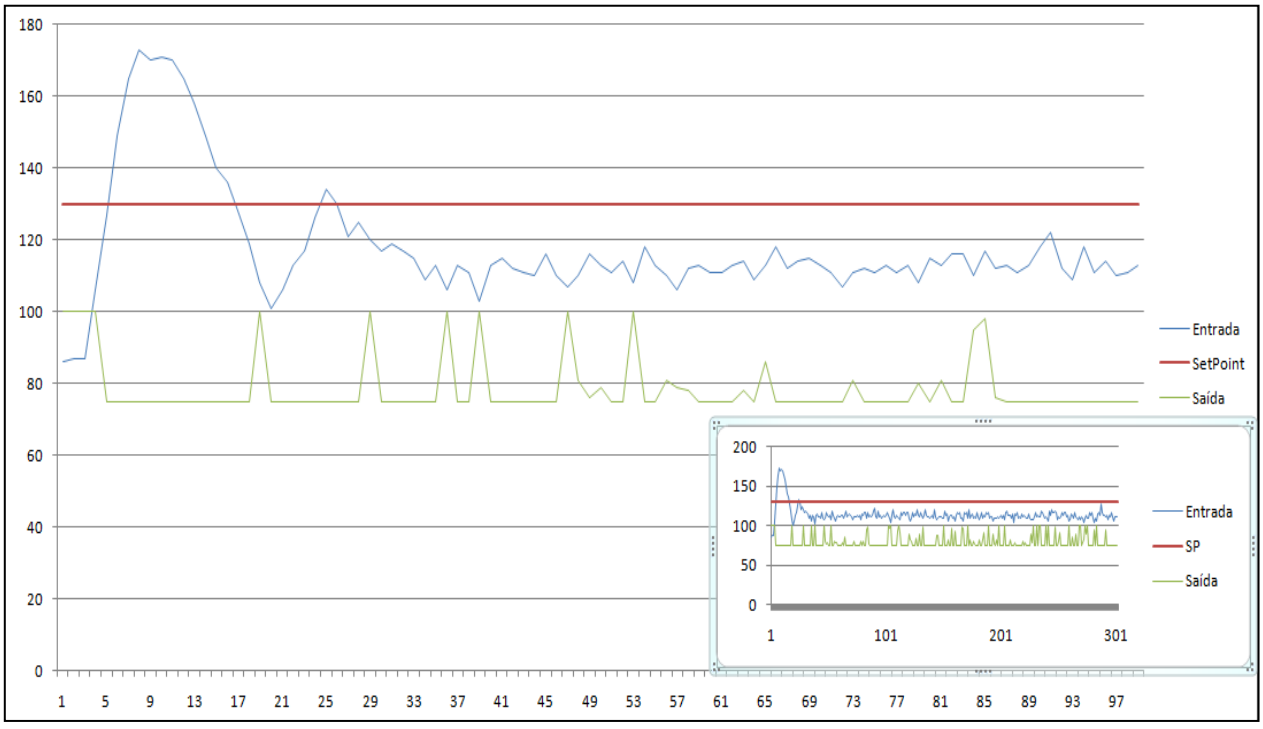


**Figura 42 - Resposta do PID2 ao *setpoint* de 130**
**Fonte: Autoria própria.**

De acordo com o experimento, o PID2, utilizando o *setpoint* de 130, mostrou-se desaconselhável se comparado ao PID1 anteriormente analisado. Observa-se um *overshoot* de 170 e uma oscilação de menor duração. Porém com o PID2 há a presença do erro em regime permanente. O sistema se estabiliza, porém há um erro de mais ou menos 30. O controlador com a configuração citada não conseguiu corrigir esse problema.

A estabilização do controle ocorre por volta dos 4 segundos, mostrando muito mais rápida que a vista no primeiro arranjo. Um possível ajuste pode ser a mudança no tempo de integração visando à diminuição do erro em regime permanente.

Já para o *setpoint* de 150, o sistema ultrapassa o *setpoint* e permanece acima por mais de 2 segundos, devido ao brusco aumento da velocidade, e a leve redução da velocidade quando já ultrapassado o *setpoint*, conforme a figura 43.

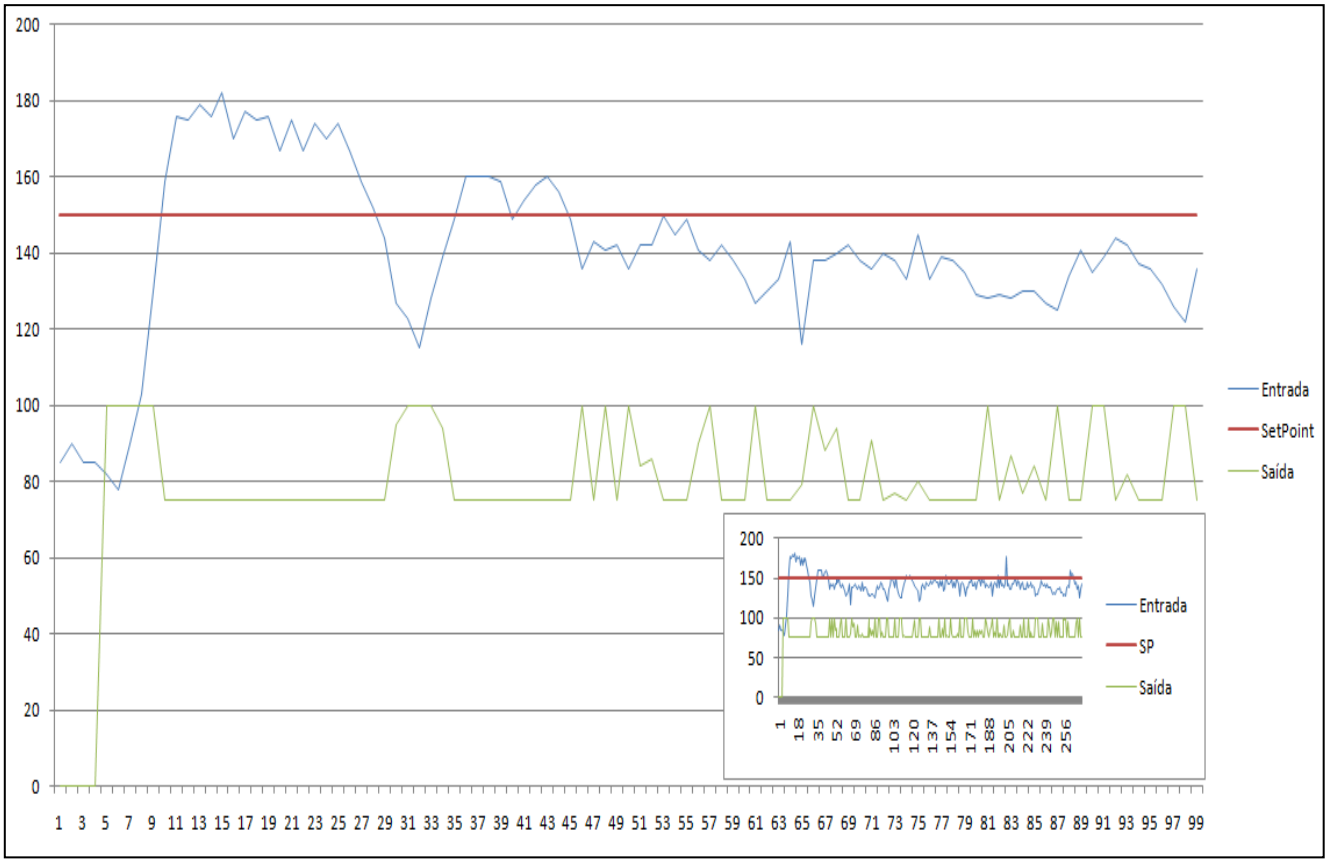


**Figura 43 - Resposta da PID2 ao *setpoint* de 150**
**Fonte: Autoria própria.**

A discretização mostrou-se melhor quando o *setpoint* está em 150. A equipe acredita que a complexidade dos cálculos envolvendo a apuração dos valores no arranjo 2, prejudica o desempenho do sistema de controle.

Foram também realizados dois testes em malha aberta, sendo no primeiro (figura 44) o *sinal* de controle de 78%, e no segundo (figura 45) o *setpoint* de 82%. A malha aberta não possui realimentação, portanto não há a correção dos erros. Não foi possível estabelecer sinais de controle inferiores ou superiores à esses valores. O sistema não responde quando os valores são muito baixos ou responde exageradamente quando são altos.

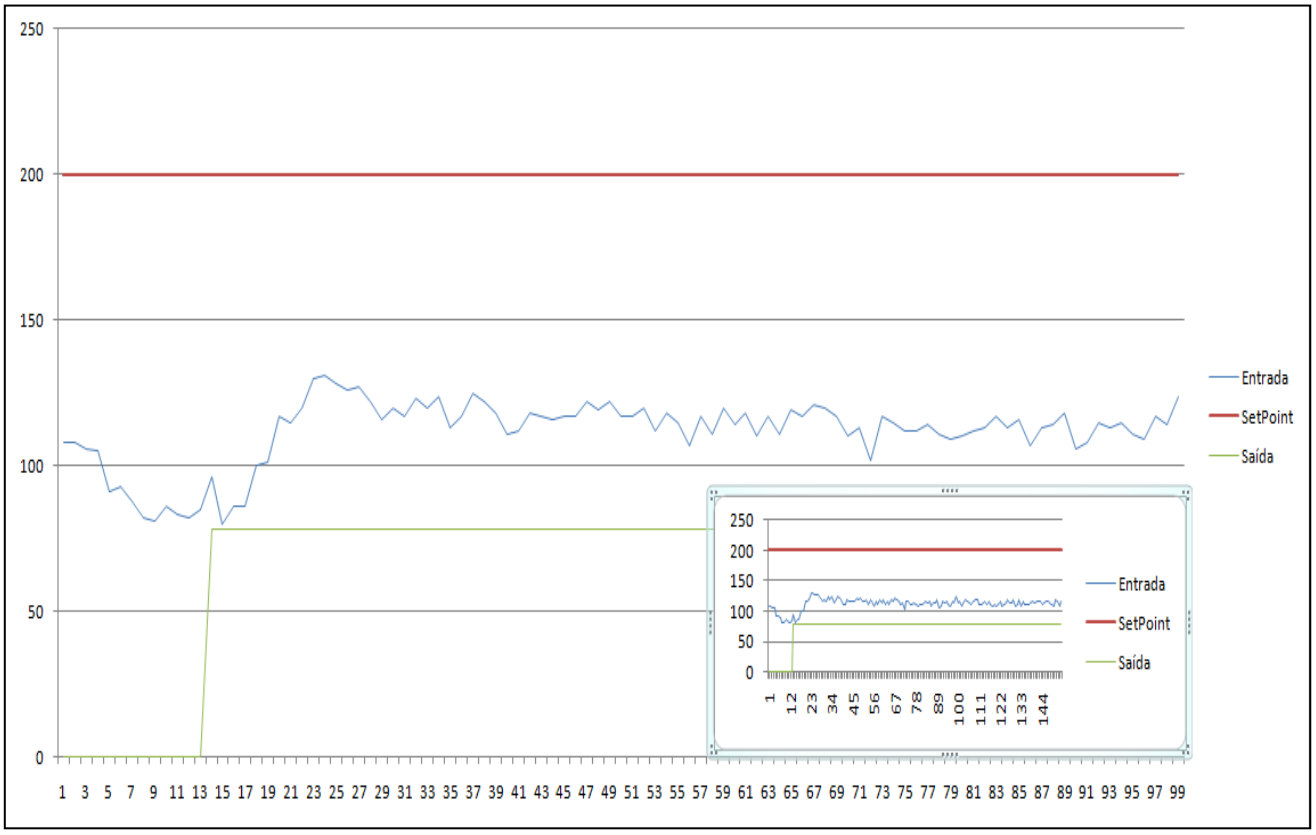


**Figura 44 - Resposta da malha aberta ao *setpoint* de 200**
**Fonte: Autoria própria.**

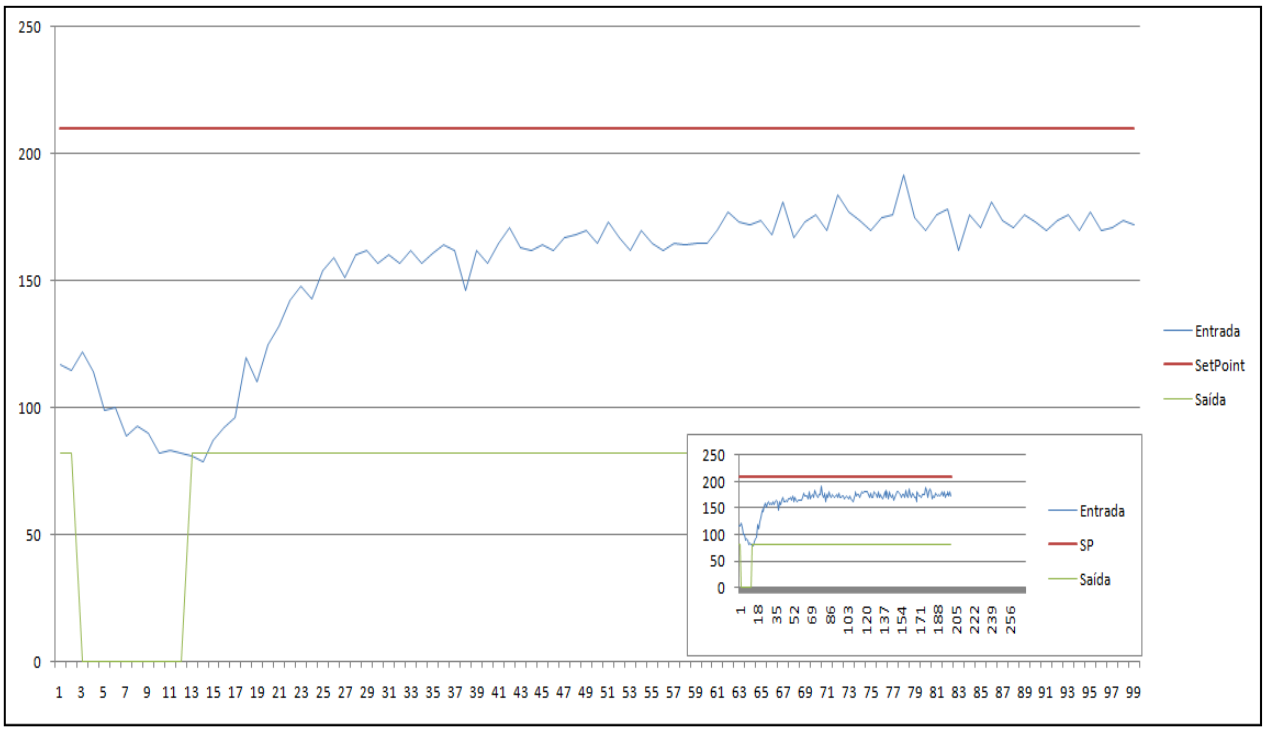


**Figura 45 - Resposta da malha aberta ao *setpoint* de 210**
**Fonte: Autoria própria.**

Observa-se que a saída crava em 76 no primeiro caso de malha aberta, e crava em 82 nos segundo. Esses valores não se alteram, pois não há a realimentação, assim o sistema não sabe que o valor desejado não é o valor obtido.

Para um *setpoint* de 200, o valor de saída estabilizou-se entre 110 e 120, já para o valor de 210, o valor estabilizou-se entre 170 e 180. Essa diferença é percebida porque o controle não reage da mesma maneira para valores diferentes de *setpoint*. Não há a tentativa do sistema eliminar o erro em regime permanente.

Por todo o exposto aconselha-se a utilização do PID1 e de ajustes por sintonia fina quando há casos em que o sistema apresenta uma leve instabilidade, ou uma demora na acomodação da curva. O PID2 pode ser implementado, porém é necessário um ajuste mais preciso. Já a malha aberta torna-se interessante pela demonstração de um controle sem retroalimentação. Nesse caso deverá haver um prévio ajuste para a saída desejada.

# 5 CONCLUSÃO

Conclui-se que o Desenvolvimento de Protótipo para Ensaios de Técnicas de Controle Programadas em Microcontrolador é um projeto viável e com grande potencial educacional, técnico e científico. O projeto foi desenvolvido e testado em cinco meses, funcionou de acordo com o esperado. O presente projeto traz inúmeras possibilidades de utilização em outras operações de controle tais como de temperatura, pressão, vazão e outras. Há um grande estudo sobre os arranjos de PID no que tange à discretização para Microcontroladores. Os trabalhos realizados pelo Moreno (2010) e pelo Iwasse (2009) contribuíram muito para facilitar a discretização e servem como referência de estudo nessa área.

A equipe desenvolveu três tipos de programação para a utilização do protótipo. Pode-se selecionar malha aberta ou uma das duas discretizações propostas. Toda a programação foi realizada com sucesso sendo necessários apenas pequenos ajustes. Os programas posteriormente foram redefinidos para que possuíssem uma forma mais refinada.

A planta foi desenvolvida pela equipe para que pudesse ser versátil e para que possibilitasse testes precisos, assim como comprovação da eficácia do microcontrolador como um PID. Salienta-se que a planta da gangorra utilizada pode ser substituída por qualquer outra desde que sejam feitos os ajustes necessários nos parâmetros e na programação do microcontrolador. Para um caráter didático, o trabalho apresenta explicações detalhadas de todas as funções envolvidas no controle e na programação do microcontrolador, assim como possui duas formas diferentes de discretização e uma forma de funcionamento por malha aberta.

Outro ponto que foi atingido pela equipe foi o desenvolvimento da interface de comunicação do *hardware* com o computador. Essa comunicação foi desenvolvida com sucesso pela equipe por meio de um *software* único que foi feito exclusivamente para propiciar a comunicação modbus. Toda a programação e seu funcionamento encontram-se neste projeto. A cada trecho das linhas de programação há comentários para propiciar maior entendimento ao leitor e garantir a didática do projeto.

O desenvolvimento da balança mostra a variedade de usos do projeto sendo possível o desenvolvimento de outros dispositivos eletromecânicos passiveis de integração com o *hardware*.

Os testes realizados pela equipe demonstram a efetividade do controle por meio de microcontroladores e servem de estudo e análise para futuros projetos. Esses testes eram parte do proposto pela equipe e foram realizados com sucesso como demonstrado no trabalho. A análise crítica visou comparar a efetividade de diferentes programações utilizadas, suas vantagens e seus defeitos. A análise de gráficos é instrumento necessário para comparação dos desempenhos das discretizações propostas.

Por todo o exposto, a equipe demonstra satisfação por ter atingido os objetivos propostos, por ter efetivado o desenvolvimento desse protótipo e por propiciar uma ferramenta que poderá ser utilizada para aprimorar o conhecimento e os estudos dos acadêmicos da Universidade Tecnológica Federal do Paraná.

Sugestões para trabalhos futuros:

- Levantar a FT da planta (modelo matemático) a partir da malha aberta.
- Aplicar os métodos de regulagem CHR, Ziegler-Nichols, entre outros aos diversos arranjos de PID propostos.
- Implementar outros arranjos de PID na planta.
- Implementar sensores mais precisos e com menos ruído.

# REFERÊNCIAS

BAZANELLA, S. Alexandre. **Ajuste de Controladores PID**. Disponível em: <http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/>. Acesso em: 01 out. 2012.

BORBA, Gustavo Benvenutti. **Apostila Modbus.** Disponível em: <http://pessoal.utfpr.edu.br/gustavo/MODBUS_apostila.pdf>. Acesso em: 01 out. 2012.

CAMPOS, Mario C. M. M. de; TEIXEIRA, Herbert C. **Controles típicos de equipamentos e processos industriais**. 1.ed. São Paulo: Editora Edgard Blucher LTDA, 2006.

CAON JR, José Roberto. **Controladores PID industriais com sintonia automática por realimentação por relés**. 1999. 104f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade de São Carlos, 1999.

CRUZ, José Jaime da. **Entendendo e Ajustando Malhas de Controle.** São Paulo: Departamento de Engenharia de Energia e Automação Elétricas GAESI – Escola Politécnica da Universidade de São Paulo, 2004.

IWASSE, Felipe. **Análise Arranjo Estrutural Controladores Comerciais**. 2009. 70f. Monografia (Especialização em Automação Industrial) – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná 2010.

MECATRONICA, disponível em: <http://www.mecatronica.eesc.usp.br/wiki/upload/1/10/Aula5SCI2010.pdf>. Acesso em: 17 jul. 2012.

MICROCHIP, disponível em: <http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010297>. Acesso em: 23 out. 2012.

MODBUS, disponível em: <http://modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf>. Acesso em: 23 out. 2012.

MORENO, Ricardo Humberto. **Estudo para discretização de controladores PID Industriais**. 2010. 103f. Monografia (Especialização em Automação Industrial) – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná 2010.

NISE, Norman S. **Engenharia de Sistemas de Controle**. 3. Ed. Rio de Janeiro: LTC, 2000.

OGATA, K. **Engenharia de Controle Moderno**. 3ª Ed. Rio de Janeiro: Editora LTC. 1998.

SCHNEIDER, Guilherme Alceu. **Sistemas de Controle.** Curitiba: Departamento Acadêmico de Eletrônica – Universidade Tecnológica Federal do Paraná 2011.

SCIENCE, disponível em: <http://www.sciencedirect.com/science/article/pii/S0009250901004390>. Acesso em: 23 out. 2012.

# APÊNDICE A – Funções Interrupt, Main e PIC

```
#define AD_COMPLETO PIR1.F6// conversao a/d esta completa. Limpar bit depois
#define flag_t0 INTCON.F2
#define  END_DISPOSITIVO 2
#define RELE PORTE.F2
#define LIGA_SUPERVISORIO status1.F0
#define FLAG_SERIAL PIR1.RCIF
#define FLAG_TIMER3 PIR2.TMR3IF
#define BOTAO_LIGA PORTB.F0
#define MALHA1 status1.F1
#define MALHA2 status1.F2
#define MALHA_ABERTA status1.F3

unsigned char reg_inicial;
void  CRC16(unsigned int dataLength, char *buffer);
void respota_escrita_md();
void escreve_md(unsigned char byte);
void comunicacao_modbus();
signed int pid();
void config_pic();
void interrupt();
signed int e[3],kp,Ti,Td,u[2],T,sp;
signed int entrada,i=0,j=0,pwm;
unsigned char receive[9];
unsigned char send[30]={0,2,3,8,0},dados[30]={0,2,3,8,0};
unsigned char lowCRC,highCRC,status1,k,l,ref_ma;
signed int v1,v2,v3,kd,ki;


void interrupt(){

if(FLAG_SERIAL){          // flag de sinalização de recepção de dados na seria
l
  comunicacao_modbus();   // executa rotina de comunicação
}
else if(FLAG_TIMER3 ){     // flag de estouro do timer 3

  entrada=ADC_READ(0)>>2;   // le a o canal 0 das entradas analógicas(trasduto
r de ângulo)
  e[0]=e[1];
  e[1]=e[2];
  e[2]=sp-entrada;          // Erro = setpoint - leitura do ângulo

  if(LIGA_SUPERVISORIO==1){       // comando liga do supervisório = 1
   // RELE=1;                   // relé que libera alimentaçaão do motor
    if(MALHA_ABERTA)    // Malha aberta
      pwm=sp;         // saída é igual a referencia setada no supervisório
    else{
      pwm=pid()+235;               // saida pwm do hardware igual a rotina de
execução do PID
      //pwm+=127;
  } }
  else{          // se não estiver ligado executa pwm=0 e sinaliza led desliga
do
    pwm=0;
```

```
    //RELE=0; //desliga alimentação do motor
  }

  PWM1_Set_Duty(pwm);  // executa valor calculado para saida pwm no hardware



  TMR3H = 0x3C;        // preset for timer3 MSB register
  TMR3L = 0xB0;        // preset for timer3 LSB register


  PIR2.TMR3IF=0;       // zera o flag de estouro de timer

}
}
```

```
void main(){
delay_ms(20);

e[2]=0; // zera as variaveis
e[1]=0;
e[0]=0;
u[0]=0;
u[1]=0;

entrada=1;              // seta valor 1 inicial
sp=EEPROM_read(1);      // le o ultimo setpoint setado
delay_ms(20);
kp=EEPROM_read(2);      // le o ultimo kp setado
delay_ms(20);
Ti=EEPROM_read(3);       // le o ultimo Ti setado
delay_ms(20);
Td=EEPROM_read(4);      // le o ultimo Td setado
delay_ms(20);
T=EEPROM_read(5);      // le o ultima taxa de amostragem setado
delay_ms(20);
status1=EEPROM_read(6);    // le bits de status
delay_ms(20);

config_pic();      // executa rotina de configuração dos registros

while(1){
}

}

int pid(){
```

```
u[0]=u[1];



ki=kp/Ti;           // calculo do Ki

kd=kp*Td;           // calculo do Kp

if(MALHA1&&!MALHA2){ // selecionado malha 1
// Vide trabalho Moreno -> u(n) = e(n)*( kp + ki*T + kd/T ) - e(n-
1)*(kp + 2*kd/T) + e(n-2)*kd/T + u(n-1)

  v1=kp+(ki*T)+(kd/T);       // primeira parte da função

  v2=kp+2*kd/T;             // segunda parte da função

  v3=kd/T;                  // terceira parte da função

  u[1]=e[2]*v1-e[1]*v2+e[0]*v3+u[0]; // Juntando as 3 partes armazena em U[1]
}
else if(MALHA2&&!MALHA1){  // selecionado malha 2
//Função PID vide Moreno --> u(n) = e(n)*( kp + ki*T + kd/T + kd/Ti)- e(n-
1)*( kp + ((2*kd)/T) + kd/Ti)+ e(n-2)*(kd/T)+ u(n-1)
  u[1]=(e[2]*(kp+ki*T+kd/T+kd*ki))-
(e[1]*(kp+(2*kd/T+kd*ki)))+(e[0]*kd/T)+u[0]; // MALHA PID adptada em C
}



//



u[1]=u[1]>>2;     // desloca bits em 2 casa para direita
if(u[1]>20)    // caso estourar a varivel 8bits ela não passo de 255
u[1]=20;
else if(u[1]<0)            // se for menor que 0 é igual a 0
u[1]=0;

return u[1];           // retorna o valor calculado na saída do controlador pa
ra atuar no PWM
}

void config_pic(){


ADCON1=0b1110; // AN0 habilitada
TRISA=1;    // AN0 entrada
TRISb=0; // CCP1 PWM SAIDA
TRISD=0;  // Porta D como saida
TRISE=0;   // Porta E como saida
```

```
PIR1.RCIE=1;
//PIR1.TXIE=1; // HABILITA INTER. UART

INTCON.F6=1; // Liga a interrução geral
INTCON.F7=1; // Liga a interrução geral
INTCON.F5=1;  // Liga Timer 0
RCON.IPEN=1;   // habilite prioridade de interrupção
IPR1.ADIP=0;   // prioridade baixa para conversor Analógico-Digital
INTCON2.TMR0IP=1;     // prioridade alta para timer0
IPR1.RCIP=1;          // prioridade alta para comunicação
IPR1.TXIP=1;          // prioridade alta para comunicação
ADCON0.ADON=1;         // habilita conversor Analógico-Digital




// Timer0 // 16-
Bit; Prescaler=1:1; TMRH Preset=84; TMRL Preset=AD; Frequencia=633,49276Hz; Pe
riodo=1,57855 ms
T0CON.TMR0ON = 1;// Liga timer 0
T0CON.T08BIT = 0;// Condifura para 16bits
T0CON.T0CS   = 0;// desliga start externo
T0CON.T0SE   = 0;//
T0CON.PSA    = 0;// desliga prescaler
T0CON.T0PS2  = 1;//
T0CON.T0PS1  = 0;
T0CON.T0PS0  = 0;
TMR0H = 0xB;    // configura registro mais significativo
TMR0L = 0xDC;  // configura registro menos significativo
// Timer3:
// Prescaler=1:1; TMR3 Preset=15536; Frequencia=100,00Hz; Periodo=10,00 ms
T3CON.T3CKPS1 = 0;// Configura Prescaler
T3CON.T3CKPS0 = 0;//
T3CON.T3SYNC  = 1;// Sincronização do timer interna
T3CON.TMR3CS  = 0;// Desabilita FOS/4
T3CON.TMR3ON  = 1;// habilita temporizador
TMR3H = 0x3C;       // configura registro mais significativo
TMR3L = 0xB0;       // configura registo menos signigicativo
//(FONTE: SOFTWARE TIME PIC)

PWM1_Init(3000);  //configura PWM com frequencia de 5kHz
PWM1_Start();     //inicia PWM

UART1_Init(9600);   // Configura porta serial com baudrate de 9600bps

}




void  CRC16(unsigned int dataLength, char *buffer) //CRC 16 do protocolo modbu
s
{
```

```
unsigned int CheckSum;
unsigned int j1;
unsigned short i1;
CheckSum = 0xFFFF;
for (j1=1; j1<=dataLength; j1++)
{
CheckSum = CheckSum^(unsigned int)buffer[j1];
for(i1=1; i1<=8; i1++)
        if((CheckSum)& 0x0001)
        CheckSum = (CheckSum>>1)^0xA001;
        else
        CheckSum>>=1;
}
lowCRC = CheckSum>>8;
CheckSum<<=8;
highCRC = CheckSum>>8;
}
```

# APÊNDICE B – Comunicação modbus RTU

```
void comunicacao_modbus(){

receive[j]=UART1_READ();   //recebe byte da porta serial

if(receive[0]==END_DISPOSITIVO) // Se for igual ao endereço definido avança, s
e não ignora
  j++;
else
  j=0;

if((receive[1]==3&&j>=8)||(receive[1]==16&&j>=11)){      // verificar se é 3(l
eitura) ou 16(escrita)
  j=0;
  if(receive[0]==END_DISPOSITIVO&&receive[1]==3){   // se for leitura, entra n
a rotina de resposta a leitura
                                    send[1]=END_DISPOSITIVO; // responde com
o endereço de dispositivo setado
                                    send[2]=3;   // 3 -
 função leitura registro
                                    dados[4]=0;   // armazena variáveis no ve
tor
                                    dados[5]=entrada;   // armazena variáveis
 no vetor
                                    dados[6]=0;
                                    dados[7]=sp;
                                    dados[8]=0;
                                    dados[9]=pwm;
                                    dados[10]=0;
                                    dados[11]=kp;
                                    dados[12]=0;
                                    dados[13]=Ti;
                                    dados[14]=0;
                                    dados[15]=Td;
                                    dados[16]=0;
                                    dados[17]=T;
                                    dados[18]=0;
                                    dados[19]=status1;


                                    l=4;
                                    reg_inicial=(receive[3]*2)+2; //calcula q
ual é o registro inicial para  o vetor
                                    for(k=reg_inicial;k<=reg_inicial+(receive
[5]*2);k++){  //armazena dados para escrever de acordo com o inical e quantida
de
                                      //send[l-1]=0;
                                      send[l]=dados[k];
                                      l++;
                                    }
                                  escreve_md(receive[5]*2);    // função escr
eve dados


                                  }
```

```
                                    else if(receive[0]==2&&receive[1]==16){
// escrita do supervisório

                                    if(receive[3]==2&&receive[5]==1){        //
 caso escreva setpoint armazena o setpoint e assim por diante com as outras va
riáveis
                                      sp=receive[8];
                                      EEPROM_WRITE(1,sp);
                                      respota_escrita_md();
                                    }
                                    else if(receive[3]==4&&receive[5]==1){
                                      kp=receive[8];
                                      EEPROM_WRITE(2,kp);
                                      respota_escrita_md();
                                    }
                                    else if(receive[3]==5&&receive[5]==1){
                                      Ti=receive[8];
                                      EEPROM_WRITE(3,Ti);
                                      respota_escrita_md();
                                    }
                                    else if(receive[3]==6&&receive[5]==1){
                                      Td=receive[8];
                                      EEPROM_WRITE(4,Td);
                                      respota_escrita_md();
                                    }
                                    else if(receive[3]==7&&receive[5]==1){
                                      T=receive[8];
                                      EEPROM_WRITE(5,T);
                                      respota_escrita_md();
                                    }
                                    else if(receive[3]==8&&receive[5]==1){
                                      status1=receive[8];
                                      EEPROM_WRITE(6,status1);
                                      respota_escrita_md();

                                    }


                                    }


}
}

                                    void escreve_md(unsigned char byte){
                                    send[3]=byte;
                                    CRC16(byte+3,send);
                                    for(i=1;i<=byte+3;i++)    //envia string com
 a resposta

                                      UART1_write(send[i]);
                                      UART1_write(highCRC);     //envia o byte m
ais significativo do CRC

                                      UART1_write(lowCRC);     //envia o byte me
nos significativo do CRC
                                    }
```

```
void respota_escrita_md(){ //responde que a escrita está correta

send[2]=16;
send[3]=0;
send[4]=2;
send[5]=0;
send[6]=1;
CRC16(6,send);
for(i=1;i<=6;i++)
  UART1_write(send[i]);
  UART1_write(highCRC);
  UART1_write(lowCRC);
}
```

## APÊNDICE C – Esquemático *Hardware*