Implementação de controle de posição de motor de corrente contínua com Arduino

Rudi van Els

Universidade de Brasília @ campus Gama

A sabedoria popular diz que depois de uma certa idade a gente começa a ser muito seletivo em escolher os trabalhos e encomendas. No meu caso é também assim. Eu tinha trabalhado no início da minha carreira como engenheiro de controle e automação no desenvolvimento de diversos sistemas e equipamentos e na minha pesquisa de mestrado tinha desenvolvido um sistema de controle de robô digital com microcontroladores. Na época, os microcontroladores eram a novidade da indústria de eletrônica e automação. O estado da arte eram microcontroladores de oito bits, com memórias de programa na ordem de alguns kilobytes, programado em assembly ou em linguagem C. Tinha grande limitação de memória e hardware e cada byte de código era contado para caber dentro da memória RAM ou ROM. O apelido que a gente recebia na época era escovadores de bits. Você não imagino que a gente era capaz de fazer com um microprocessador Z80 ou microcontrolador 8085 com 8kbytes de ROM e 256 bytes de RAM em 1990.

Pois, passaram mais de 20 anos e recebo a demanda de recondicionador um dinamômetro de motor de combustão interno. O dinamômetro fora adquirido do laboratório nos anos sessenta do século passado e tinha um sistema de controle analógica que comandava um freio hidráulico que permitia fazer diversos tipos de ensaios de motores de combustão interno (MCI), com estratégias de controle para manter a carga aplicado ao motor ou a rotação do motor constante. O dinamômetro deve ter dado suporte para o desenvolvimento de dezenas de pesquisas de mestrado e centenas de trabalhos de iniciação científica ou trabalhos final do curso de engenharia mecânica.

Um rápido levantamento na documentação do dinamômetro mostrou que se tratava de uma sistema de controle analógica que comandava um motor de corrente contínua de aproximadamente 50Watts que por sua vez comandava uma borboleta de controle de vazão da água de entrada do freio hidráulico.

ELS, R.H.van. Implementação de controle de posição de motor de corrente contínua com Arduino. 2013, Tutorial publicado na http://fga.unb.br/rudi.van

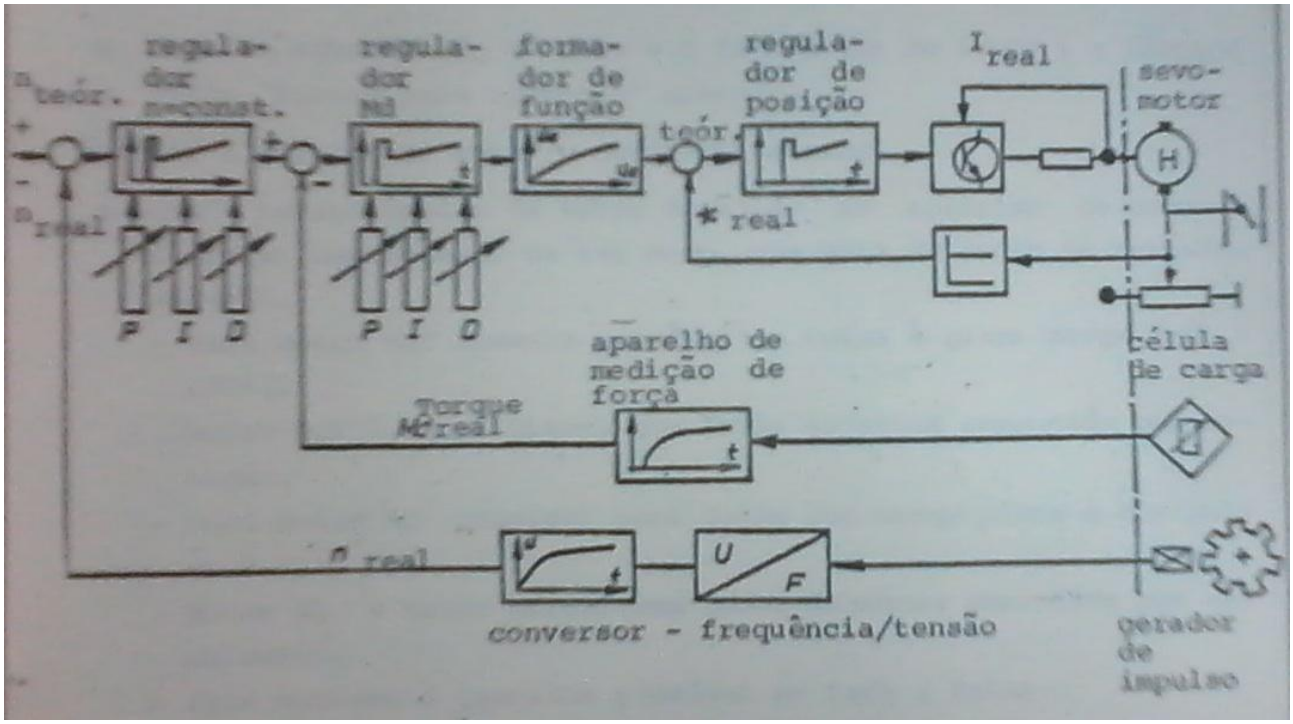

Figura 1: Diagrama de blocos do sistema antigo
§

O sistema de controle fazia a medição da rotação do motor sobre teste, tinha três malhas de controle. Uma malha para fazer o controle de posição do motor de corrente contínua, ou a abertura da válvula de entrada de água no freio. A segunda malha permitia fazer a medição do torque do motor por meio de uma célula de carga e implementar os ensaios de funcionamento com torque constante. A terceira malha de controle media a rotação do motor sobre teste e permitia fazer ensaios de funcionamento com velocidade constante.

O sistema original que permitia o ensaio do MCI mantendo o torque constante não estava mais funcionando. Esse subsistema foi substituído por um medidor digital de torque com célula de carga, ainda nos anos oitenta.
Um análise do estado do dinamômetro mostrou que o controle analógica estava com alguns problemas operacionais, fiação ressecado, mal contato, além de funcionalidade limitada para as novas demandas das pesquisas com MCI. A mecânica do freio, ainda estava num estado razoável, mas precisava de uma limpeza geral e lubrificar a mecânica.

Ficou a seguinte demanda.
Criar um novo sistema de controle para a bancada de ensaio, aproveitando a mecânica bastante robusto e ainda funcional do freio hidráulico e introduzir uma tecnologia nova de controle que permitia customizar as tarefas de ensaio e criar novas ensaios. Sem dúvida, será necessário trocar o sistema analógico por um sistema digital baseado em computador.

Aceitei a demanda.
Primeiro passo foi procurar no mercado algum sistema de controle de posição de motor de corrente contínua que implementava algoritmo de controle PID e com uma interface para computador.

Depois de 3 semanas, procurando no demais laboratórios da Universidade e vasculhando a internet por meio do Google, cheguei a triste conclusão que ia ter que fazer o controlador. Os sistemas disponíveis, ou eram muito caros, muito sofisticados, muito abrangentes ou iam demandar muito tempo para aprender a usa-los.

O segundo passo foi selecionar a plataforma para implementar o sistema de controle da bancada. Depois de algumas conversas com os colegas resolvemos apostar no Labview, embutindo o algoritmo de controle no próprio Labview. Como se tratava de um processo mecânico (MCI) com um tempo de resposta da ordem de alguns décimos de segundos, chegava se a conclusão que qualquer micro computador desktop normal, pode fazer o controle e instrumentação.

Definido isso, sobrou para mi resolver o controlador de posição do motor de corrente contínua. Fui ver nos meus projetos antigos se tinha algo pronto, e também fui ver o que tinha ainda de eletrônica e instrumentação que eu podia aproveitar. Fiquei triste em ver que o hardware que eu dominava e poderia aproveitar para fazer o controlador tinha várias limitações que iam desencorajar os meus estudantes para melhorar o sistema.
O meu hardware baseado em microcontroladores de 8 bits e 11Mhz era da década de noventa e não tinha memória não volátil, nem sistema de desenvolvimento de aplicativas baseados em janelas, limitação de velocidade de comunicação com computador hospedeira. Devo confessar que o meu sistema favorita ficou preso no tempo.
Depois dos anos noventa, não acompanhei a onda dos microcontroladores baseado em tecnologia RISC que deu origem à família dos PICS na virada do século. Tampouco acompanhei a onda de hardware reconfigurável que se popularizou na virado do século. Certamente, percebi que estava na hora de me render a modernidade e usar ferramentas e hardware de desenvolvimento novos, e é bom dizer, mais populares.

Me rendi ao modismo do momento.
Encomendei um kit de desenvolvimento Arduino que comprei pela internet e que me foi entregue em menos de 3 dias. Na minha época, a gente tinha que comprar os componentes e fazer o próprio hardware de desenvolvimento. Os tempos e prazos agora são outros.

Placa Arduino Leonardo com saída PWM e conversar analógico digital (AD). Baixei o IDE do Arduino na internet. Testei a placa com a trivial rotina para piscar um led. Depois de alguns ajustes, consegui colocar meu Mac OS para comunicar com o Arduino e pronto. *Its Alive..! on, off, on, off,* etc.

Figura 2: Arduino.. It's alive

ELS, R.H.van. Implementação de controle de posição de motor de corrente contínua com Arduino. 2013, Tutorial publicado na http://fga.unb.br/rudi.van

O seguinte passo era procurar uma fonte de tensão controlável ou algum driver de potência para motor de corrente contínua. Não achei nada para comprar, mas consegui resgatar uma placa de potência que tinha usado num projeto de controle de cadeira de rodas. Era um placa com ponte H com capacidade de comandar motores de até 5Amperes.
Fiquei animado e fui ver se tinha mais material para rodar uns testes na minha bancada em casa.

Depois de abrir umas caixas com sucata, achei um motor de CC de brinquedo e dois potenciômetros. O motor deve ter sido de um carrinho de controle remoto que foi desmontando ha muito tempo. Fixei um potenciômetro para servir de sinal de referência e ajustei o motor, acoplando ao seu eixo o segundo potenciômetro e .. ualá !
Em menos de duas horas estava com meu protótipo pronto. Placa de controle Arduino, driver com ponte H, motor CC com realimentação de posição e potenciômetro de referencia.

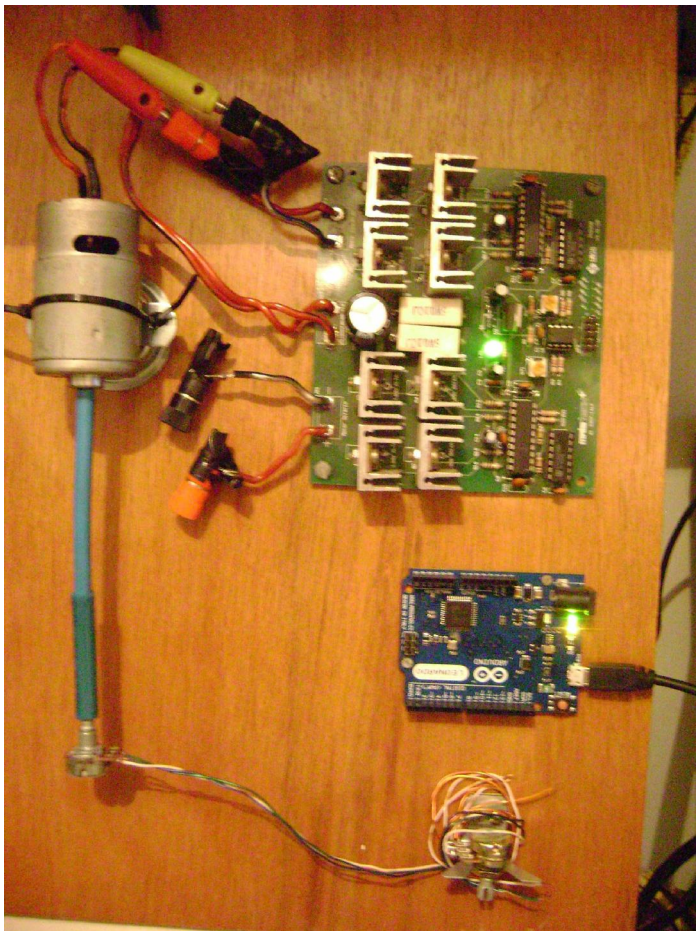

Figura 3: Protótipo inicial

O próximo passo foi ver se o conversor ADC do Arduino estava funcionando. Nos programas exemplos do IDE Arduino achei um programa que lia a porta analógica e mando o valor lido para o PWM e a porta de comunicação serial.

Em menos de cinco minutos consegui fazer com que a velocidade do motor de CC estava sendo

controlado pela potenciômetro de referência com o programa <AnalogInOutSerial.c> disponível no Arduino IDE. O único problema era o som que o acionamento do motor emitia. Suspeito que o PWM estava numa frequência na ordem de alguns centenas de hertz na faixa de audio e o motor CC funcionava como alto-falante. O manual do Arduino confirmou a minha suspeita: a frequência do PWM era 490Hz. Eu sabia que ia ter alguns horas de trabalho para descobrir como mudar a frequencia do PWM na rotina "*analogWrite(analogOutPin, outputValue)*". Mas como isso não era prioritário, passei para a próxima fase.

Segunda fase.

O método científico que mais gosto de usar é a de tentativa e erro. Sempre funciona.
Assim eu comecei a montar aos poucos as rotinas do programa de controle. Como eu não tinha familiaridade com Arduino ou o IDE do fabricante, comecei a fusar para ver como funcionava.
Aproveitei para adaptar o programa exemplo de leitura da porta analógica do IDE para testar os diversos componentes do sistema. O potenciômetro de referência estava funcionando, assim como o potenciômetro do realimentação da posição. Testei o PWM e a mudança de direção de rotação do motor. Assim fui testando e adaptando o código até ter o algoritmo de controle proporcional no laço principal do programa. Obviamente que o funcionamento deixava muito a desejar, pois eu sei que um sistema de controle de posição de um motor de CC com controle proporcional tem um erro em regime que não permite posiciona-lo de forma precisa.

Outra limitação deste programa era que eu não tinha controle sobre o tempo de processamento e amostragem da rotina de controle. A rotina que adaptei fazia dentro do mesmo laço principal a leitura dos dados de posição e referência pelo conversor AD, calculava o erro e o valor do acionamento da saída PWM e mostrava na porta serial os dados lidos e calculados.

```
const int Pino_Posicao = A0;
const int Pino_Referencia = A1;
const int Pino_analogOut = 9;
const int Pino_Direcao = 8;
int Posicao = 0;
int Referencia = 0;
int outputValue = 0;
int Erro = 0;
int Saida =0;

void setup() {
  Serial.begin(9600);
  pinMode(Pino_Direcao, OUTPUT);
}

void loop() {
  Referencia = analogRead(Pino_Referencia);
  Posicao = analogRead(Pino_Posicao);
  Erro = Referencia - Posicao;
  if (Erro < 0) { Saida=-Erro; }
   else { Saida=Erro;}
  outputValue = map(Saida, 0, 1023, 0, 255);
```

```
  analogWrite(Pino_analogOut, outputValue);
  if (Erro < 0) digitalWrite(Pino_Direcao, 0);
   else digitalWrite(Pino_Direcao, 1);
  Serial.print("Referencia  = " );
  Serial.print(Referencia);
  Serial.print("\t Posicao = " );
  Serial.print(Posicao);
  Serial.print("\t Erro = " );
  Serial.print(Erro);
  Serial.print("\t output = ");
  Serial.println(outputValue);
  delay(500);
}
```

Essa maneira de iniciar o trabalho foi muito oportuno, pois aproveitava o código já existente e fui me acostumando aos poucos com a funcionalidade do Arduino. Entretanto, logo comecei a perceber que tinhas alguns limitações.

Fui descobrindo que o Arduino e o seu sistema de desenvolvimento era muito prática, mas escondia todo o potencial do microcontrolador. Parece que o sistema foi feito para leigos. Talvez isso foi a razão da sua grande popularidade e boa penetração no mercado.

Terceira fase

O primeiro desafio relacionado à programação do Arduino era separar a visualização dos valores lidos com a rotina ler os potenciômetros.
Os dados eram monitorados pelo serial monitor do IDE e permitia depurar o código e entender o que estava de fato acontecendo no microcontrolador. Como a nossa velocidade de ler e interpretar esses dados é lento, na ordem de no máximo 3 linhas de dados por segundo, não há necessidade de diminuir o atraso de meio segundo no programa anterior.
Diminuindo esse tempo só enchia a tela com informação e também chegava a sobrecarregar o buffer de leitura do IDE.

Eu também sabia que o tempo de amostragem do caso de um motor de CC com essas características tinha que ser na ordem de milissegundos. Partiu-se então para separar a rotina de amostragem de visualização. A melhor estratégia é de colocar a leitura dos conversores AD e o calculo do algoritmo de controle numa interrupção temporizado.

Foi ai que descobri que o Arduino não tinha nenhum suporte para tratamento de interrupções no IDE. Todo programa desenvolvido no IDE Arduino tinha um rotina de ***setup*** e um ***loop.*** O jeito foi buscar na internet alguns exemplos que envolviam o timer e suas interrupções. Depois de algumas palavras chaves no google achei alguns exemplos e tutorial no site da internet: http://www.engblaze.com/microcontroller-tutorial-avr-and-arduino-timer-interrupts/

Ainda não precisei procurar o datasheet do microprocessador. Só copiei os trechos de códigos no meu primeiro programa e depois de alguns acertos funcionou.

Programei o contador TIMER1 do Arduino de gerar uma interrupção a cada 1 milisegundos e tive que ajustar a rotina de leitura dos potenciômetro com o conversor AD para não fazer duas leituras na mesm interrupção. Suspeitei que a rotina ***analogRead*** tinha poderia forçar a reentrança na interrupção e optei em não fazer conversões em cascata. Uma rápida olhada no manual mostrava que a rotina de conversão demorava em torno de 100 microsegundos ou 0,1 milisegundos.

A estrutura para fazer o algoritmo de controle estava pronto. Iniciei com uma rotina simples de controle proporcional com ganho de 5 e fiz uma rotina na malha principal para mandar os dados do funcionamento para a porta serial a cada décimo de segundo.

O programa de controle proporcional ficou bem compacto e simples.

```
#include <avr/io.h>
#include <avr/interrupt.h>

const int Pino_Posicao = A0;
const int Pino_Referencia = A1;
const int analogOutPin = 9;
const int Pino_Direcao = 8;

int Posicao = 0;
int Referencia = 0;
int Erro = 0;
int Controle = 0 ;
int Saida =0;
boolean select;

void setup() {
  Serial.begin(9600);
  pinMode(Pino_Direcao, OUTPUT);
  cli();
  TCCR1A = 0;
  TCCR1B = 0;
  OCR1A = 15;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << CS12);
  TIMSK1 |= (1 << OCIE1A);
  sei();
}

void loop() {
 Serial.print(Referencia);Serial.print("\t " );
 Serial.print(Posicao);   Serial.print("\t " );
 Serial.print(Erro);      Serial.print("\t " );
 Serial.print(Controle);  Serial.print("\t " );
 Serial.println(Saida);
 delay(100);
}

ISR(TIMER1_COMPA_vect)
{
```

```
  select=!select;
  if (select==1) Referencia = analogRead(Pino_Referencia);
   else
   {
    Posicao = analogRead(Pino_Posicao);
    Erro = Referencia - Posicao;
    Controle =  5 * Erro;
    if (Controle > 0)
     {digitalWrite(Pino_Direcao, 1); Saida = Controle; }
      else { digitalWrite(Pino_Direcao, 0); Saida = - Controle;}
    if (Saida  > 255) Saida = 255;
    analogWrite(analogOutPin, Saida);
   }
}
```

Depois de várias tentativas copiando os dados do serial monitor e fazendo a sua análise numa planilha de cálculo, consegui obter a curva mostrada na figura a seguir.

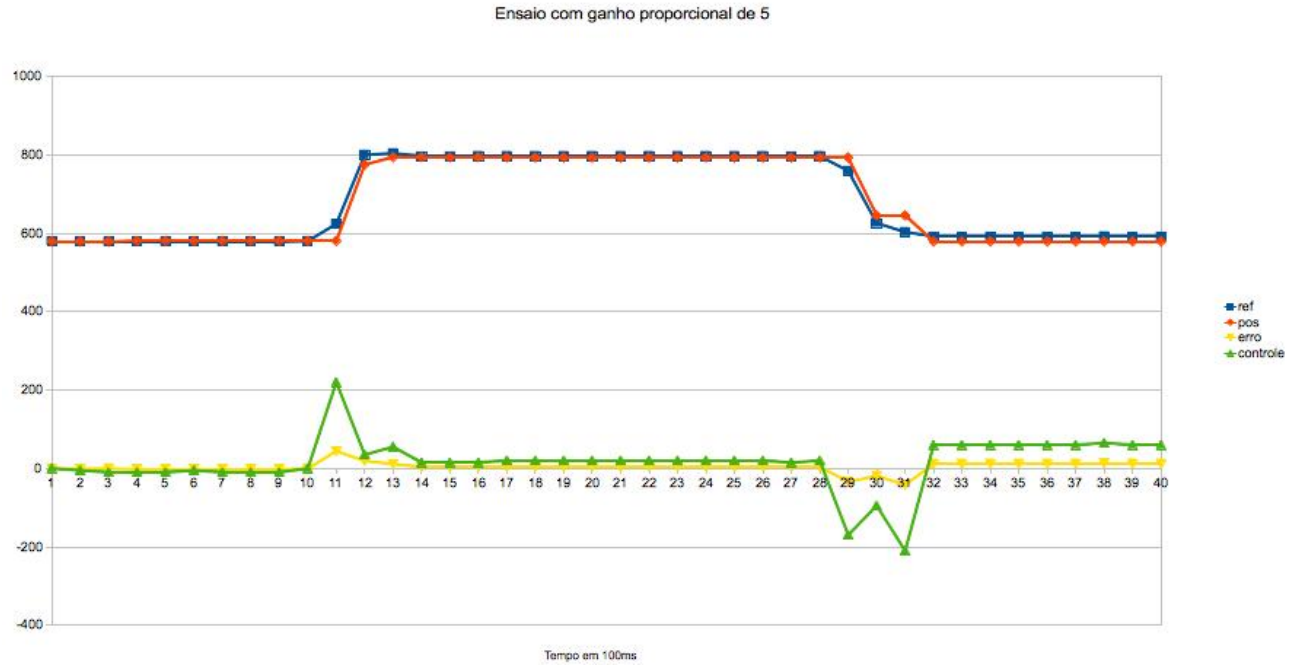

Figura 4: Ensaio com controle proporcional

Aparentemente o sistema ficou muito bom. Entretanto, é necessário olhar com mais detalhe o comportamento dinâmico do sistema, pois a figura não conseguiu captar os trancos que o motor dava ao implementar a rotina de controle.
Vai ser necessário melhorar a captação de dados do sistema e aproveitar melhor as funcionalidade do Arduino. A partir do levantamento do comportamento dinâmico do motor vai ser possível sintonizar o controlador com mais propriedade.

ELS, R.H.van. Implementação de controle de posição de motor de corrente contínua com Arduino. 2013, Tutorial publicado na http://fga.unb.br/rudi.van

25/11/2013
Rudi

Bibliografia.

Claudia Roos. Projeto de Implementação de um Sistema de Controle PID para o Banco de Motores do LEA usando Arquiteturas Reconfiguáveis. 2004. 120 f. Trabalho de Conclusão de Curso. (Graduação em Engenharia de Controle e Automação) - Universidade de Brasília. Orientador: Carlos Humberto Llanos Quintero.