UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE PARA BANCADA EXPERIMENTAL DE MICROTURBINAS

HENRIQUE JOSÉ RIBEIRO ALVES FILHO

ORIENTADOR: TAYGOARA FELAMINGO DE OLIVEIRA

DISSERTAÇÃO DE MESTRADO EM CIÊNCIAS MECÂNICAS

BRASÍLIA/DF: MARÇO - 2020

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE PARA BANCADA EXPERIMENTAL DE MICROTURBINAS

HENRIQUE JOSÉ RIBEIRO ALVES FILHO

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

APROVADA POR:

_____

Prof. Taygoara Felamingo de Oliveira
(Orientador)

_____

Prof. Carlos Alberto Gurgel Veras

_____

Prof. Fábio Cordeiro de Lisboa

BRASÍLIA/DF, 26 DE MARÇO DE 2020

FICHA CATALOGRÁFICA

ALVES FILHO, HENRIQUE J. RIBEIRO

Sistema de Aquisição de Dados e Controle para Bancada Experimental de Microturbinas

138p., 210 x 297 mm (ENM/FT/UnB, Mestre, Ciências Mecânicas, 2020)

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

Departamento de Engenharia Mecânica.

1. Introdução
2. Sistema de Aquisição de Dados e Controle
3. Bancada Experimental
4. Testes e Resultados Experimentais
5. Conclusão

I. ENM/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

ALVES FILHO, HENRIQUE J. RIBEIRO (2020). Sistema de Aquisição de Dados e Controle para Bancada Experimental de Microturbinas. Dissertação de Mestrado em Ciências Mecânicas, Publicação ENM.DM, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF.

CESSÃO DE DIREITOS

AUTOR: Henrique José Ribeiro Alves Filho

TÍTULO: Sistema de Aquisição de Dados e Controle para Bancada Experimental de Microturbinas.

GRAU: Mestre ANO: 2020

Henrique José Ribeiro Alves Filho

SMPW Quadra 17 Conjunto 09 Lote 04 Casa H

71.741-709 Brasília – DF – Brasil

RESUMO

Este trabalho trata do desenvolvimento de um sistema de aquisição de dados e controle para a bancada experimental de microturbinas a gás desenvolvida pelo Laboratório de Energia e Ambiente (LEA) da UnB – Universidade de Brasília. Tal sistema visa permitir a operação segura da bancada e fornecer uma forma de se obter dados experimentas da mesma. O sistema consiste em uma interface de aquisição, composta por hardware e software embarcado, e pelo software de controle. O sistema desenvolvido tem como premissas a segurança do operador, o uso de componentes simples, visando a fácil manutenção do mesmo, e a confiabilidade dos dados obtidos. O sistema foi baseado na solução *open-source* Arduino ATmega2560 e o software de controle foi programado em C#. O sistema desenvolvido mostrou-se adequado para a operação da bancada experimental, reportando dados experimentais confiáveis e permitindo a sua operação segura.

ABSTRACT

This work presents the development of a data acquisition and control system for the experimental bench of gas microturbines developed by the Laboratório de Energia e Ambiente (LEA) of UnB – Univerdade de Brasília. Such system aims to allow the safe operation of the bench and provide a way to obtain experimental data from it. The system consists of an acquisition interface, consisting of hardware with embedded software, and control software. The system developed is based on the operator's safety, the use of simple components, aiming at its easy maintenance, and the reliability of the data obtained. The system was based on the open-source solution Arduino ATmega2560 and the control software was developed in C#. The developed system proved to be suitable for the operation of the experimental bench, reporting reliable experimental data and allowing its safe operation.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASCII – *American Standard Code for Information Interchange*

CI – Circuito Integrado

EMI – *Electromagnetic Interference*

IDE – *Integrated Development Environment*

kSPS – kilo *Samples Per Second*

LED – *Light-Emitting Diode*

PCB – *Printed Circuit Board*

PWM – *Pulse Width Modulation*

UART – *Universal Asynchronous Receiver-Transmitter*

USB – *Universal Serial Bus*

Vcc – *Voltage common collector*

1. INTRODUÇÃO

No atual cenário global a busca por fontes de energia limpas e renováveis é um tópico que ganha cada vez mais destaque. Podemos citar como principais fontes alternativas a energia solar, eólica, fotovoltaica e das marés. Infelizmente, todas essas fontes sofrem de um mal comum: a intermitência na geração devido a mudança das condições climáticas que ocorrem no decorrer do dia (KUANG *et al.*, 2016).

Turbinas a gás são uma excelente alternativa para suprir a intermitência que fontes alternativas de energia possuem. Um exemplo simples citado no artigo escrito por Rathi (2012) sugere a utilização de uma turbina a gás de baixa emissão de poluentes conjuntamente com uma planta de geração solar ou eólica. Turbinas a gás podem ser postas ou retiradas de operação de forma rápida e segura, tornando-se candidatas ideais para auxiliar uma usina solar ou eólica nos períodos de pico de consumo ou mesmo substituir por completo a geração em casos de reduções temporárias dos ventos ou dias nublados.

Turbinas a gás podem ser divididas em 3 categorias: grande porte, utilizadas para geração centralizada e com potência acima de 15MW; de médio porte, que possuem potência entre 1MW e 15MW normalmente utilizadas em plataformas marítimas; e de pequeno porte, com potência abaixo de 1MW (LORA; NASCIMENTO, 2003).

Dentro das turbinas de pequeno porte ainda temos turbinas classificadas como “microturbinas”, que possuem potência entre 20 e 500kW, sendo ideais para uso em geração distribuída ou plantas de cogeração (FERREIRA, 2007).

1.1. MICROTURBINAS A GÁS

Diferentemente das turbinas de médio e grande porte, microturbinas utilizam um compressor centrífugo ou uma combinação de compressor centrífugo e radial e utilizam uma turbina de fluxo radial. Além disso, devido a limitação de temperatura dos gases na entrada da turbina e a baixa eficiência dos componentes utilizados, sua eficiência é inferior. Microturbinas possuem eficiência elétrica entre 15 e 35%, entretanto, o rendimento global do sistema pode chegar a até 85% quando usadas em plantas de cogeração (FERREIRA, 2007).

Com a Resolução Normativa Nº 482 de 17 de abril de 2012, publicada pela Agência Nacional de Energia Elétrica (ANEEL), que trata da geração de energia elétrica distribuída, o uso de microturbinas para este fim também passou a se tornar viável.

Assim como turbinas a gás de grande porte, microturbinas operam seguindo o ciclo Brayton aberto. Na Figura 1.1 são apresentados os diagramas T-s e p-V do ciclo ideal. Neste ciclo, o ar é comprimido pelo compressor através de uma compressão isentrópica de (3) até (4), em seguida segue para a câmara de combustão onde se mistura com o combustível e ocorre uma adição de calor a pressão constante, (4) a (1). Após a câmara de combustão, o fluído de trabalho segue para a turbina, onde ocorre a expansão isentrópica do fluído, (1) e (2). Por fim, ocorre uma rejeição de calor a pressão constante do fluído de trabalho para o ambiente entre os pontos (2) e (3). No ciclo real, o ponto 4s se desloca para a esquerda no digrama T-s, devido às irreversibilidades existentes no processo de compressão. O ponto 2s também se desloca para a esquerda, devido às irreversibilidades no processo de expansão que ocorre na turbina. Devido a estas irreversibilidades, a potência de eixo consumida pelo compressor aumenta e a potência de eixo gerada pela turbina é reduzida.

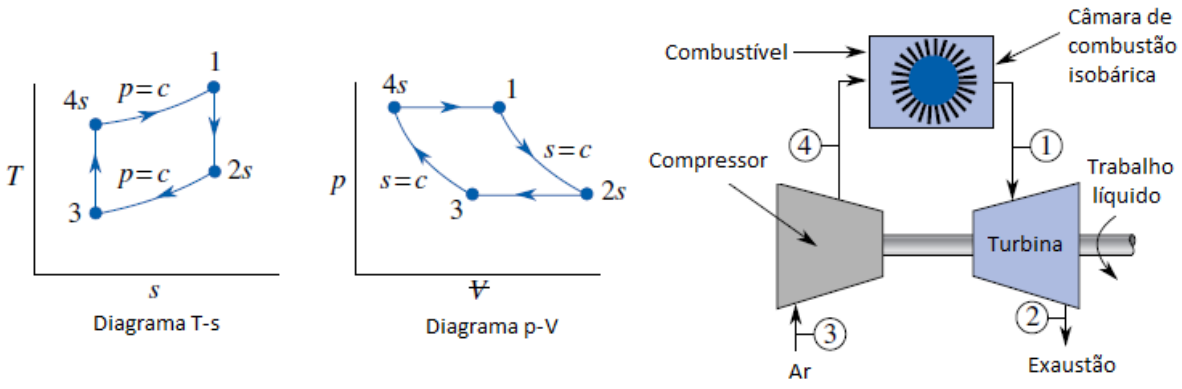


Figura 1.1 – Diagrama ciclo *Brayton* aberto (adaptado de BALMER, 2011).

É possível adicionar regeneradores de calor, fazendo com que a microturbina opere segundo o ciclo *Brayton* regenerativo, aumentando a eficiência do sistema (FERREIRA, 2007).

O modo de operação descrito é conhecido por Operação com Eixo Simples, pois o gerador é conectado ao mesmo eixo em que está conectado o compressor da microturbina. No entanto, também é possível operar no modo Eixo Duplo, onde uma segunda turbina é adicionada na exaustão da turbina que aciona o compressor e o gerador é conectado a esta. A Figura 1.2 ilustra este modo de operação. Neste modo de operação com dois estágios de expansão, todo o trabalho de eixo realizado pela primeira turbina, denominada turbina do gerador de gases, é utilizado

para acionar o compressor, enquanto todo o trabalho realizado pela segunda turbina, chamada de turbina de potência, aciona o gerador. Por utilizar mais de um estágio de expansão, esta configuração se torna mais eficiente.

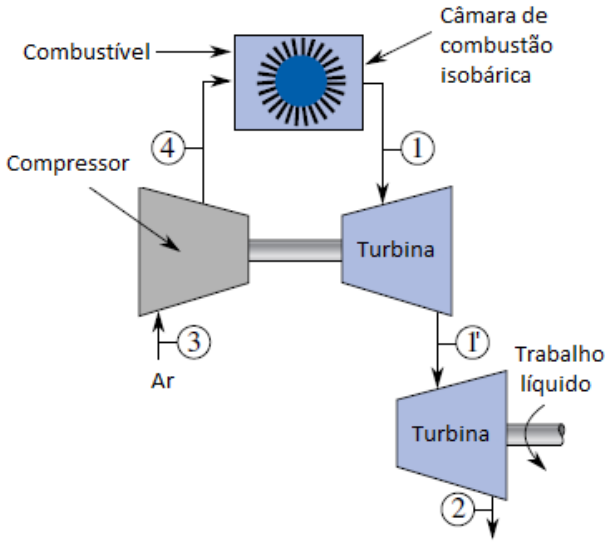


Figura 1.2 – Operação com eixo duplo (adaptado de BALMER, 2011).

As principais fabricantes de microturbinas para geração elétrica são: *AlliedSignal, Elliott Energy System, Ingersoll-Rand Energy System Recuperators & Power Works, turbe, Browman Power* e *ABB Distributed Generation & Volvo Aero Corporation*. Em função de sua simplicidade, as microturbinas apresentam custo de operação e manutenção mais baixos que as turbinas de grande porte, não exigindo ferramental específico e não necessitando de equipamento de controle e monitoramento complexos (FERREIRA, 2007).

Turbinas a gás possuem a capacidade de operar com diversos combustíveis sem que haja a necessidade de modificações extensivas na planta, como exemplificado pelos trabalhos realizados por Corrêa Jr *et al.* (2018) e por Perna *et al.* (2015). Em ambos os trabalhos é discutido o *retrofit* de plantas já existentes para a atualização de gases proveniente de biomassa.

A versatilidade das microturbinas também é descrita no trabalho de Bazzo *el al.* (2013), no qual uma microturbina é utilizada em planta de cogeração para produção de energia elétrica e que utiliza os gases quentes para alimentar um sistema de refrigeração baseado em um *chiller* de absorção modificado.

1.2. AQUISIÇÃO E CONTROLE

Para que uma turbina a gás possa operar de forma segura e eficiente, é necessário um sistema que monitore os parâmetros críticos ao funcionamento. Caso se deseje realizar testes de desempenho, é necessário o monitoramento de diversos parâmetros adicionais (KRAMPF, 1992). Os parâmetros mínimos que necessitam ser monitorados são a temperatura dos gases na saída da câmara de combustão e a rotação do eixo do turbocompressor, de forma a evitar que o mesmo seja danificado por excesso de temperatura ou rotação. Para estudos de desempenho e modelagem termodinâmica do sistema, se faz necessário o monitoramento do estado termodinâmico (temperatura e pressão) do fluído de trabalho nos principais pontos do ciclo.

Um sistema de aquisição de dados moderno é capaz de monitorar com precisão diversas variáveis de interesse do processo, desde que utilizados os sensores corretos e sejam tomadas medidas para evitar ruído nos sinais (MEASUREMENT COMPUTING CORP, 2012).

Os principais componentes de um sistema de aquisição são os sensores, responsáveis por converter a grandeza que será medida em sinais elétricos, condicionadores de sinal, que condicionam o sinal de saída do sensor de forma a permitir que o mesmo seja conectado ao dispositivo de aquisição, conversores analógico-digital, capazes de converter um sinal analógico continuo em um sinal digital discreto, e a interface de aquisição, responsável por interpretar os dados recebidos.

Sistemas de controle podem trabalhar tanto em malha aberta ou malha fechada. Sistemas de controle de malha fechada estabelecem uma relação entre a saída do sistema que está sendo controlado e a entrada de referência deste sistema, utilizando a diferença entre as duas como meio de controle. Em sistemas de controle de malha aberta, o sinal de saída não exerce nenhuma ação de controle diretamente sobre o sistema (OGATA, 1998).

Na operação de uma microturbina é importante que a rotação do eixo seja mantida o mais próximo possível da rotação de referência para evitar oscilações na rede elétrica. Contudo, conforme a carga na rede varia, a carga que o gerador impõe sobre o eixo também varia, fazendo que seja necessário ajustar a potência que a turbina está fornecendo para que a rotação se mantenha constante. A rotação do eixo é proporcional à quantidade de combustível que está sendo fornecido (FLESLAND, 2010), assim o sistema de controle deve ser capaz de atuar sobre este parâmetro.

1.3. CONTROLE DE TURBINAS A GÁS

Em unidades comerciais, o sistema de controle necessário para o correto funcionamento das turbinas a gás é fornecido pelos fabricantes das mesmas, possuindo três funções principais, sendo elas: controle de partida e parada, controle de operação e proteção da turbina. Estes sistemas de controle podem possuir malha de controle aberta ou fechada, sendo mais usual a utilização de sistemas de controle de malha fechada (BOYCE, 2011).

Como dito na seção 1.2, sistemas de controle de malha fechada utilizam parâmetros de saída do sistema controlado para ajustar sua entrada. No caso de controle de turbinas a gás, é possível utilizar sistemas do tipo *feedforward control*, *feedback control* ou uma combinação dos dois. Em um sistema *feedforward*, é possível utilizar a variação da carga medida no gerador para corrigir a injeção de combustível na turbina antes mesmo que a rotação do eixo sofra alguma variação. Já no sistema *feedback*, é necessário que haja uma variação na rotação do eixo para que o sistema atue e corrija a potência fornecida pela turbina. Em um sistema de controle combinado, a correção por parte do *feedforward* irá atuar antes do sistema de controle por *feedback*, reduzindo o esforço deste em manter a rotação constante, garantindo uma maior estabilidade na operação da turbina. O digrama apresentado na Figura 1.3 apresenta a implementação típica dos sistemas de controle.

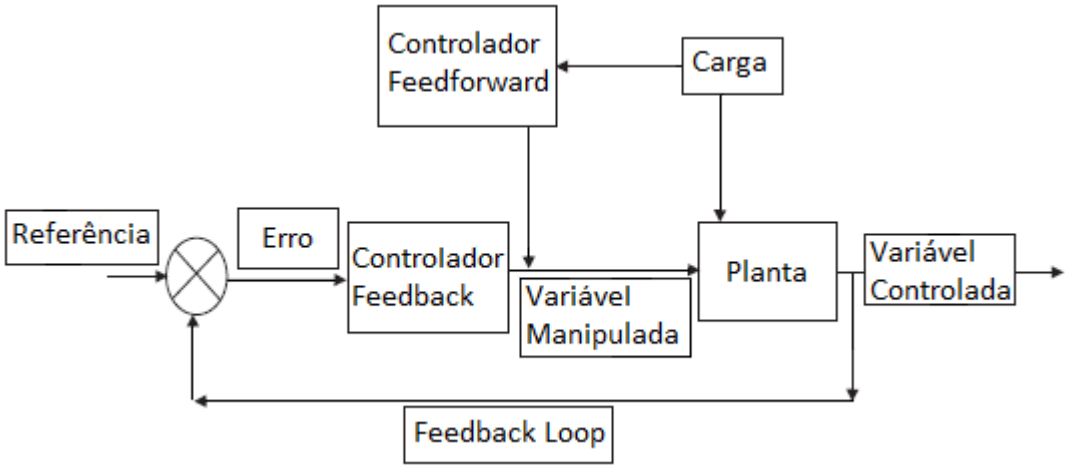


Figura 1.3 – Sistema de controle *feedforward* e *feedback* (adaptado de BOYCE, 2011).

A malha de controle principal de uma turbina a gás é o controle de velocidade, contudo existem duas malhas de controle voltadas para a proteção da turbina que atuam para evitar danos a turbina em situações atípicas, sendo uma malha de controle de temperatura e uma malha de controle de aceleração (CENTENO et al., 2005).

A malha de controle de temperatura atua quando a temperatura dos gases de exaustão ultrapassa o seu valor limite, reduzindo a quantidade de combustível injetada e consequentemente reduzindo a potência da turbina e a temperatura dos gases de exaustão (CENTENO et al., 2005).

A malha de controle de aceleração evita que ocorram variações bruscas na velocidade de rotação do eixo da turbina. Esta malha atua quando o gradiente de aceleração ultrapassa um valor predeterminado, reduzindo a quantidade de combustível injetado e a potência fornecida pela turbina (CENTENO et al., 2005). Tal situação pode ocorrer caso o gerador seja bruscamente desconectado da rede elétrica ou então durante a partida da turbina.

Como as três malhas de controle atuam sobre a quantidade de combustível injetada, as três saídas destas são entradas para um seletor de menor valor e a saída deste seletor é a que controla o sistema de combustível e a potência mecânica que a turbina irá fornecer (MERÍCIA, 2006). A Figura 1.4 mostra um digrama simplificado da interação destas três malhas.

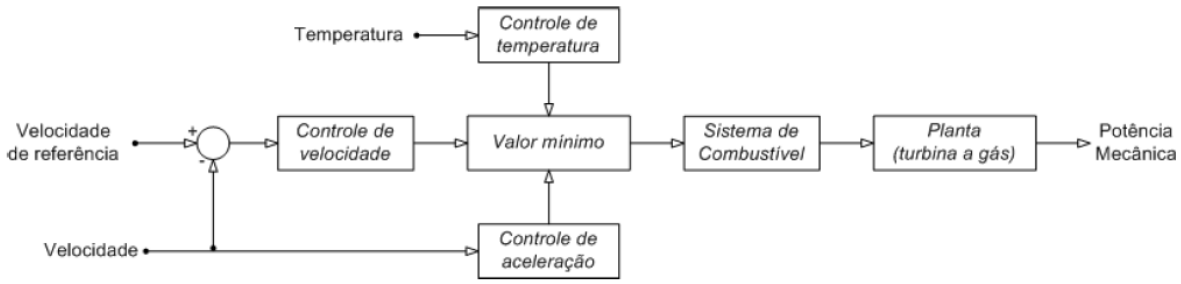


Figura 1.4 – Interação entre as malhas de controle (MERÍCIA, 2006).

O sistema de controle de partida da turbina deve monitorar a aceleração do eixo e a temperatura dos gases de exaustão para garantir a segurança do processo. Caso a temperatura ou a rotação não atinjam os valores mínimos após um tempo pré-determinado, o sistema deverá desligar a turbina e uma nova tentativa de partida só deverá ocorrer após todo o combustível não queimado ser purgado do sistema (BOYCE, 2011). A Figura 1.5 apresenta as curvas características de temperatura e rotação durante a partida.

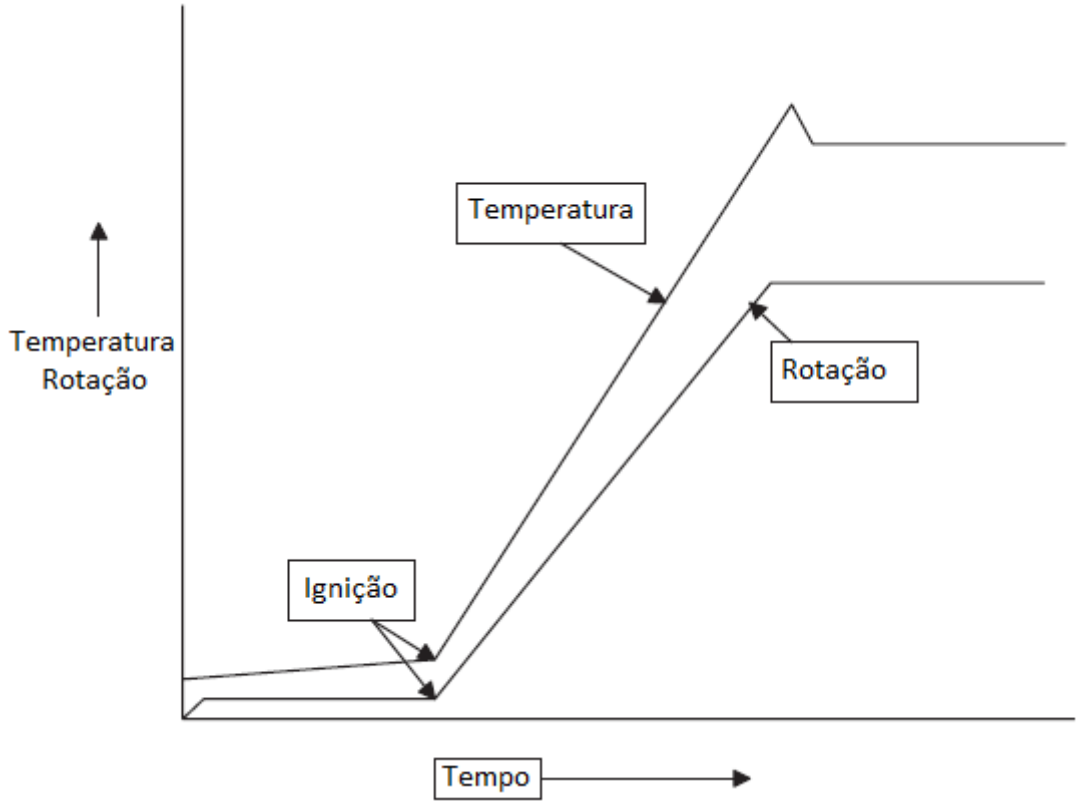


Figura 1.5 – Curvas características de temperatura e rotação durante a partida (adaptado de BOYCE, 2011).

Já para o sistema de parada da turbina, é importante que o sistema de lubrificação possa ser mantido em funcionamento mesmo após o desligamento da turbina, visto que este sistema também é responsável pela refrigeração da turbina, evitando que os mancais e os retentores de óleo superaqueçam devido ao calor acumulado na turbina e na sua carcaça.

1.4. JUSTIFICATIVA E ORGANIZAÇÃO DO TRABALHO

O Laboratório de Energia e Ambiente da UnB (LEA) possuiu uma linha de pesquisa voltada para o controle e a redução de emissões nos processos de combustão, cada vez mais importante para reduzir os efeitos catastróficos do aquecimento global.

O LEA possui uma bancada de ensaios de microturbinas composta por um turbocompressor de uso automotivo, por uma câmara de combustão *DLN* com tecnologia *LPP* desenvolvida por Ferreira (2007) e seus sistemas auxiliares. Esta bancada contava com o sistema de controle desenvolvido por Merícia (2006), contudo, devido a nenhum trabalho ter sido realizado nesta desde o trabalho de Ferreira (2007), a bancada estava sucateada: existiam diversos problemas mecânicos e elétricos na mesma e o sistema de controle havia sido removido.

Este trabalho foi organizado da seguinte forma:

O Capítulo 1 apresenta um resumo sobre microturbinas, sistemas de controle e aquisição de dados e seus principais elementos, e a motivação e objetivos do trabalho. O Capítulo 2 detalha a forma como o sistema de aquisição de dados e controle foi desenvolvido, enquanto o Capítulo 3 trata especificamente da bancada experimental e seus subsistemas. No Capítulo 4 são apresentados os testes e resultados experimentais obtidos. Por fim, o Capítulo 5 apresenta as conclusões do projeto e propõe melhorias que podem ser implantadas no sistema.

1.5. OBJETIVOS

Este trabalho tem como objetivos:

- Desenvolver um sistema de aquisição de dados e controle que permita a operação segura da bancada experimental de microturbinas a gás do LEA-UnB;
- Reativar a bancada experimental, de forma a permitir que a mesma seja utilizada tanto para trabalhos futuros relativos a emissões de poluentes quanto para auxiliar na formação de alunos de graduação na disciplina de Máquinas Térmicas;
- Validar o sistema desenvolvido.

2. SISTEMA DE AQUISIÇÃO DE DADOS E CONTROLE

O sistema de aquisição de dados e controle desenvolvido neste trabalho tem como objetivo instrumentar e controlar a operação de uma bancada experimental para ensaios de microturbinas operando no ciclo Brayton aberto.

Nas próximas seções são apresentadas as premissas do projeto bem como todos os componentes desenvolvidos e selecionados para tal sistema.

2.1. PREMISSAS DO PROJETO

O sistema de aquisição de dados e controle desenvolvido teve como premissas os seguintes pontos:

- O desempenho do sistema deve ser suficiente para que os dados coletados sejam válidos. A taxa de amostragem deve ser suficiente para atender ao teorema da amostragem de *Nyquist–Shannon*, conforme discutido no trabalho de Lévesque (2014). Do trabalho de Cruz (2006), observou-se que a constante de tempo da microturbina é de 5s, assim optou-se por usar uma taxa de amostragem de 20Hz;
- O hardware do sistema deve ser simples e de fácil manutenção, evitando o uso de componentes de difícil aquisição. O hardware também deve priorizar a utilização de sistemas *open source* sempre que possível;
- O hardware deve prever expansão futura para novos sensores e novos módulos, evitando que seja necessário a fabricação de uma nova interface de aquisição e controle para atender as necessidades futuras;
- O software deve ser desenvolvido em uma linguagem amplamente difundida e fácil compreensão;
- O sistema como um todo deve sempre priorizar a operação segura da bancada.

2.2. INTERFACE DE AQUISIÇÃO E CONTROLE

A interface pode ser dividida em duas partes principais: hardware e software embarcado. O hardware, como o próprio nome diz, é a parte física da interface, fornecendo portas para a conexão de sensores e atuadores necessários para a operação da bancada experimental. Já o software embarcado na interface traduz as mensagens recebidas do software de controle e realiza ações sobre os atuadores e/ou coleta dados provenientes dos sensores.

Tanto o hardware quanto o software da interface foram concebidos de forma a permitir uma operação segura da bancada experimental, fornecendo os recursos necessários para tal. Nas próximas subseções temos uma descrição detalhada de ambas as partes da interface.

2.2.1. Hardware

O hardware da interface foi projetado de forma a atender os requisitos estabelecidos. Mesmo todo o hardware estando em uma única placa de circuito impresso, este foi projetado e concebido de forma modular, permitindo que os diversos circuitos desenvolvidos pudessem ser testados em bancada de forma individual, reduzindo o tempo e os custos de desenvolvimento.

A interface se comunica com o computador onde está o software de controle através de uma porta serial virtual, fornecida por uma conexão USB.

Todo o hardware está concentrado em uma PCB de dupla face de aproximadamente 20 x 16 cm. A PCB foi fabricada em FR-4 com 2mm de espessura para garantir uma boa resistência mecânica, visto que está submetida as vibrações da bancada experimental.

Uma atenção especial ao posicionamento dos componentes e design das trilhas foi tomada para minimizar ao máximo o ruído gerado. Capacitores de *bypass* foram posicionados o mais próximo possível dos terminais de alimentação dos CIs, uma das faces da placa foi utilizada como *ground plane*, evitou-se ângulos retos nas trilhas, entre outras boas práticas que foram adotadas (SLATTERY, 1991).

O design da placa foi elaborado utilizando-se o software *Eagle*, desenvolvido pela CadSoft, na versão 7.6.0. Na Figura 2.1 temos uma imagem do layout no software, enquanto na Figura 2.2 vemos uma foto da placa fabricada ainda sem os componentes soldados.

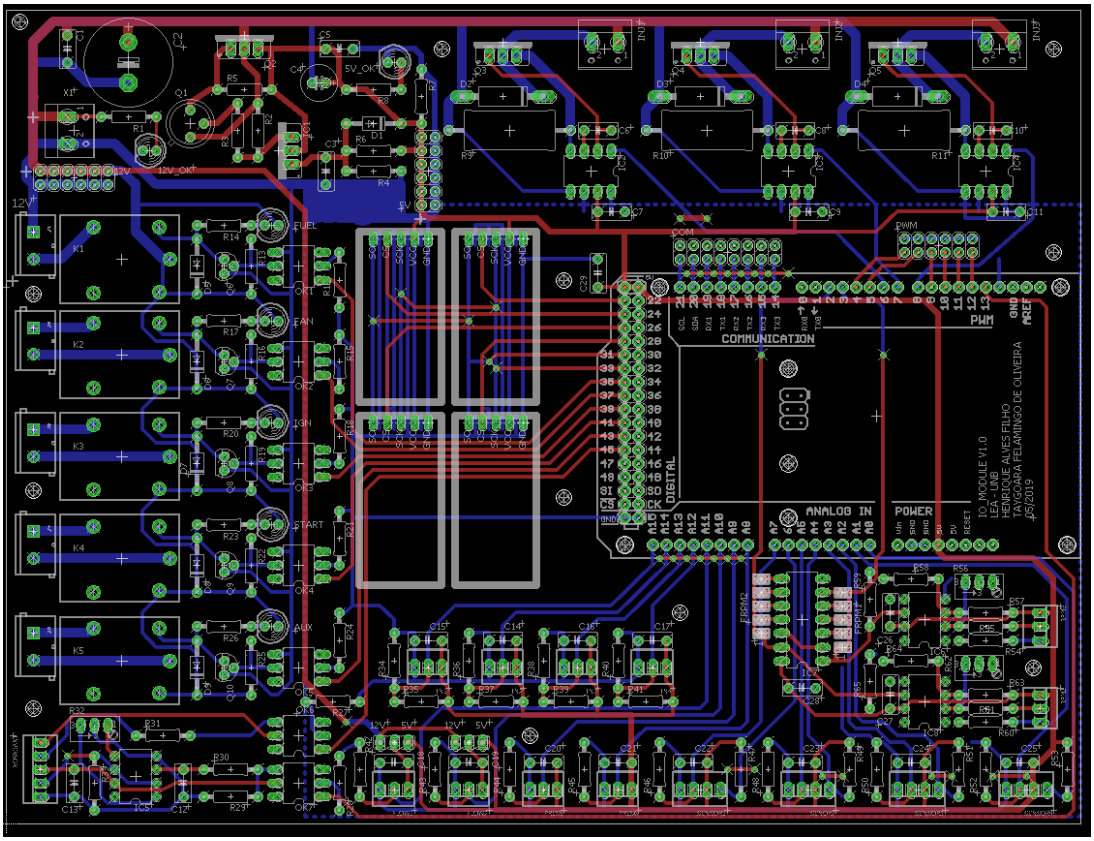


Figura 2.1 – Layout da PCB no software (Elaboração própria).

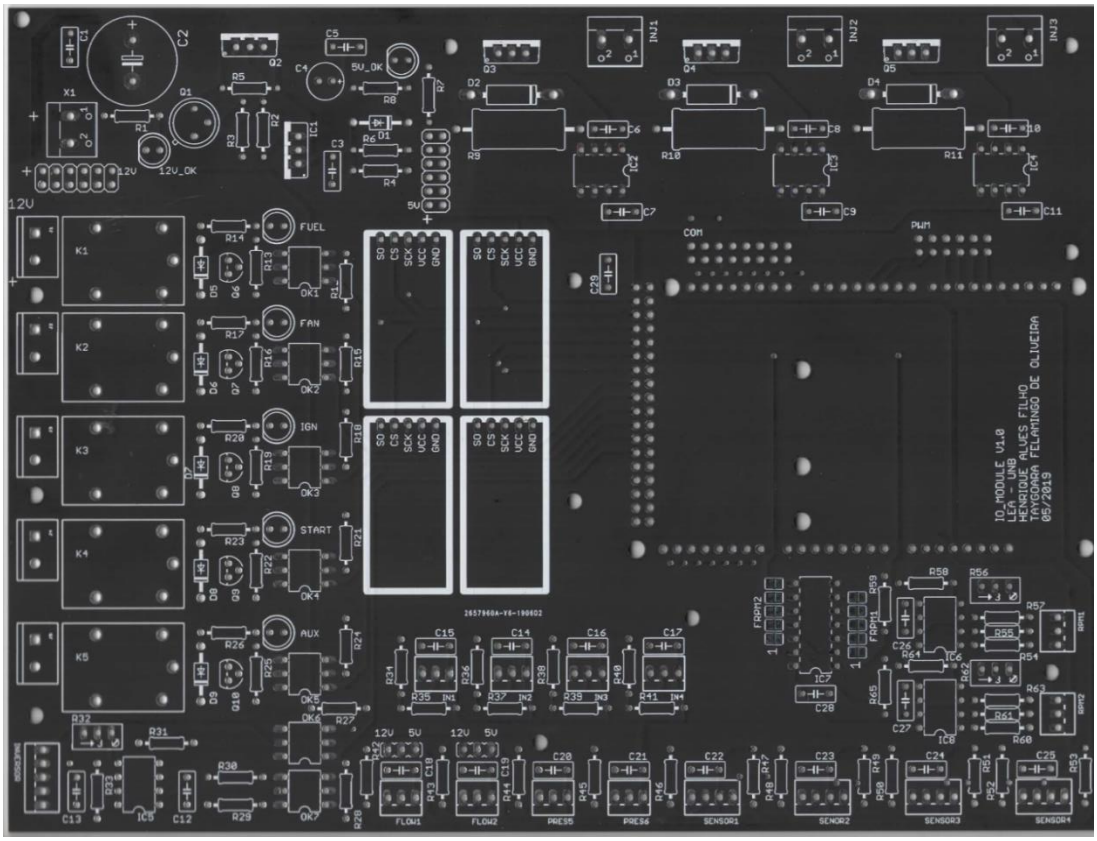


Figura 2.2 – Foto da PCB produzida (Elaboração própria).

O hardware da interface pode ser subdivido nos seguintes módulos:

- Microcontrolador
- Regulador de tensão
- Drivers de Injeção
- Relés
- Controle para Inversor de Frequência
- Entradas analógicas – Genéricas
- Entradas analógicas – Pressão e temperatura
- Entradas analógicas – Pressão
- Entradas analógicas – Fluxo de massa de ar
- Entradas digitais – Rotação
- Termopares
- Botão de emergência

Cada um destes módulos é descrito em detalhes nas próximas subseções.

A interface também conta com diversos conectores para futura expansão. Estes conectores fornecem conexão direta para portas de entrada/saída digitais do microcontrolador, portas seriais do microcontrolador e também alimentação 5VDC e 12VDC.

Na Figura 2.3 temos a localização de cada conector, bem como a indicação de onde está cada módulo de hardware da interface.

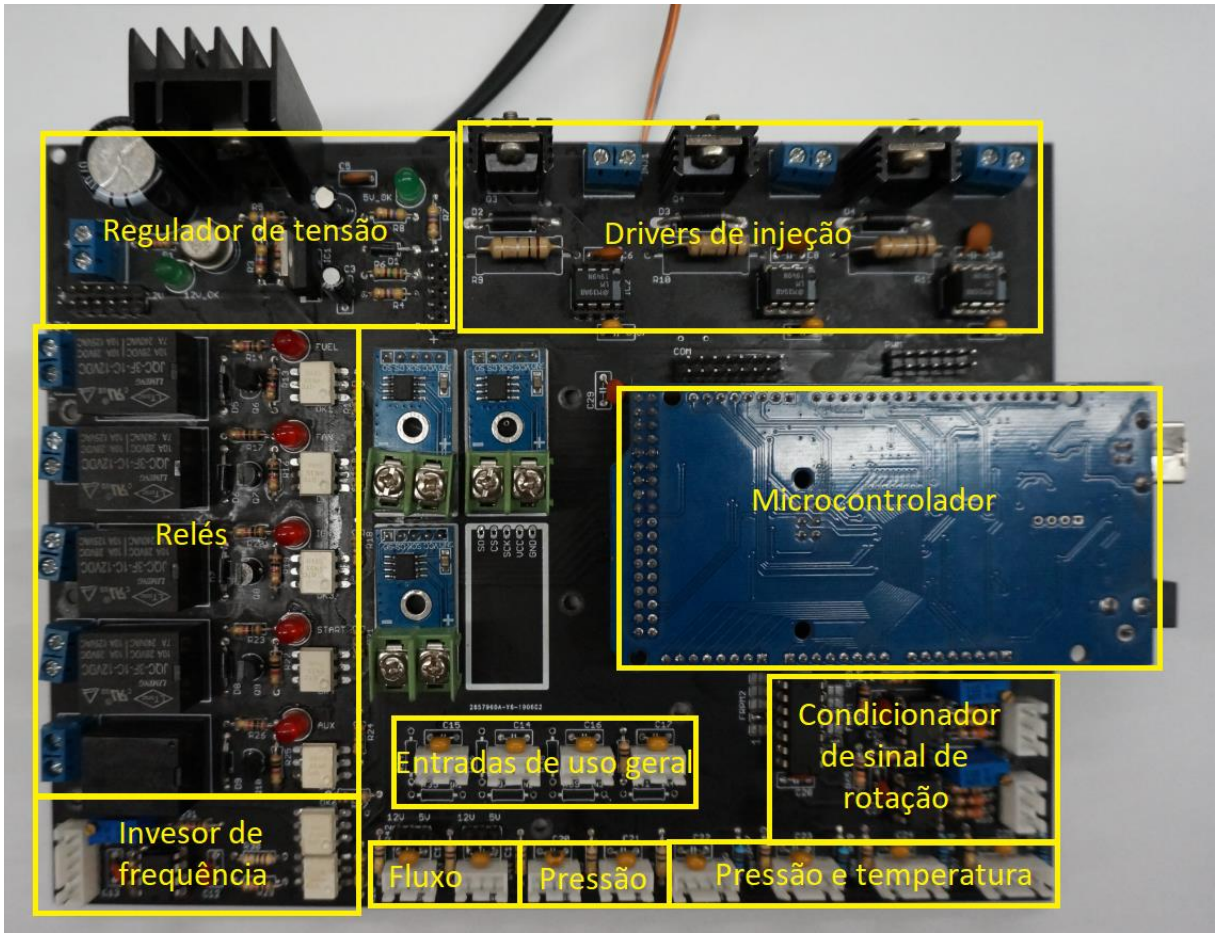


Figura 2.3 – Placa montada (Elaboração própria).

O esquemático completo do hardware desenvolvido se encontra no Anexo I deste trabalho.

2.2.1.1. Microcontrolador

O microcontrolador selecionado para a interface foi o ATmega2560, fabricado pela Microchip. Tal escolha se deu pelo fato do microcontrolador em questão possuir internamente um conversor analógico digital com 10-bit de resolução e 16 portas, possuir 4 *timers* de 16-bit e 2 *timers* de 8-bit, contar com 4 entradas do tipo “Input Capture”, além de possuir capacidade de processamento mais que o suficiente para atingir a performance desejada.

O conversor analógico-digital integrado foi utilizado para todas as entradas de sensores analógicos utilizados. Sua resolução de 10-bit permite medir variações de até 4,9mV nos sinais de entradas. O conversor também possibilita uma taxa de amostragem de até 15kSPS (ATMEL, 2014).

Duas das entradas *Input Capture* juntamente com seus respectivos *timers*, de 16 bit, foram utilizadas para as entradas dos sensores de rotação, deixando livre outras 2 entradas para expansão futura.

Um dos *timers* de 16-bit foi utilizado pelo PWM responsável por controlar os drivers de injeção, enquanto o último *timer* desta resolução foi utilizado para controlar o intervalo entre as leituras dos sensores, de forma a garantir que o tempo entre elas seja o mais constante possível.

Por fim, um dos *timers* de 8-bit foi utilizado para função de *watchdog*, responsável por reiniciar a interface caso esta perda comunicação com o software de controle. O último *timer* de 8-bit foi utilizado pelo PWM utilizado para o controle do inversor de frequência.

Optou-se por utilizar uma placa de desenvolvimento comercialmente disponível que já possui integrada todos os periféricos necessários para o funcionamento e comunicação com o microcontrolador. Esta decisão teve como premissa simplificar o desenvolvimento do hardware da interface e também facilitar a manutenção futura, visto que o micontrolador e alguns componentes necessários para sua implantação só estão disponíveis em pacotes SMD, que exigem uma habilidade muito maior para a sua montagem. Além disso, esta abordagem permite que o microcontrolador seja trocado de forma simples e rápida, facilitando o reparo da interface em caso de danos causados ao microcontrolador. A placa de desenvolvimento escolhida foi a baseada no Arduino ATmega2560, que é *open source*, possuindo versões disponíveis no mercado muitas vezes até mais baratas que adquirir somente o microcontrolador individualmente. Nesta placa de desenvolvimento, o *clock* do microcontrolador é definido por um cristal externo e é de 16MHz.

Na Figura 2.4 temos um esquemático contendo todas as conexões aos pinos do Arduino.

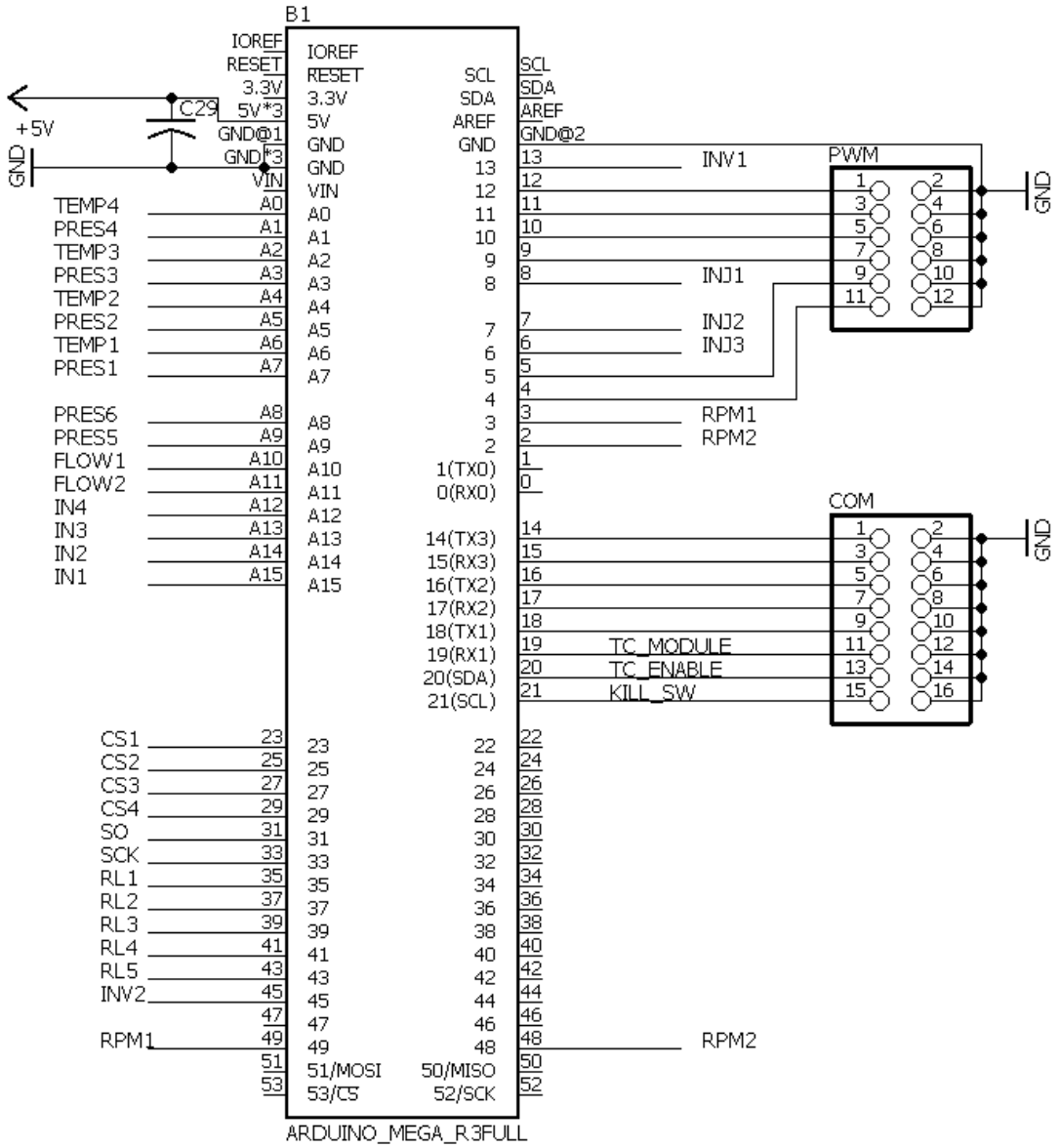


Figura 2.4 – Conexões ao Arduino (Elaboração própria).

2.2.1.2. Drivers de injeção

A estratégia adotada para o controle das solenoides de combustível foi a *Peak-and-hold.* Tal estratégia garante um menor tempo de abertura das solenoides e também um comportamento mais linear em toda a faixa de operação (BECCARI, 2014). Esta estratégia de controle consiste em fornecer as solenoides uma elevada corrente (corrente de pico, ou *peak current*) durante a sua abertura e em seguida uma corrente baixa (corrente de operação, ou *holding current*) durante o tempo em que a solenoide permanecer aberta.

Para tal, desenvolveu-se um *driver* de controle baseado no circuito integrado LM1949, da *Texas Instruments*. Tal CI é especializado no controle de injetores automotivos e possui internamente toda a lógica necessária para este modo de operação, sendo necessário alguns poucos componentes externos para a sua implementação.

Na Figura 2.5 temos o esquemático de um dos 4 drivers utilizados pela interface de controle.

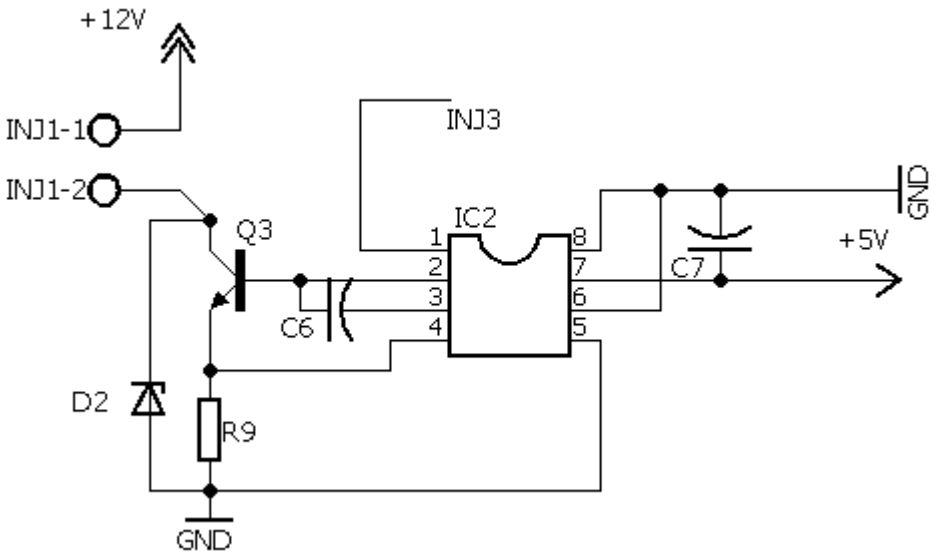


Figura 2.5 – Esquemático do *driver* de injeção (Elaboração própria).

O IC2 é o LM1949, que tem a sua entrada ligada diretamente a saída PWM do microcontrolador da interface. O transistor Q3, do tipo Darlington NPN, é o responsável pelo acionamento da solenoide, e o diodo zenner D2 se faz necessário para fornecer um caminho para a corrente reversa produzida pelo colapso do campo magnético na solenoide quando a mesma é desenergizada ou mesmo para a corrente reversa gerada na transição do estado de *peak* para *hold*, evitando danos ao transistor Q3. O capacitor C7 é um capacitor de *bypass*. Já o capacitor C6 é utilizado para melhorar a estabilidade do LM1949 durante o funcionamento no estado de corrente de operação, seu valor foi retirado diretamente do *datasheet* do CI (TEXAS INSTRUMENTS, 2013).

O resistor R9, ligado em série com a solenoide, tem a função de fornecer o feedback da corrente que está circulando pela solenoide para o LM1949. Este *feedback* é na forma de uma queda de tensão entre os terminais desse resistor, onde o CI espera uma tensão de 385mV para a corrente de pico e uma tensão de 94mV para a corrente de operação. O valor de R9 foi calculado utilizando-se destas tensões esperadas e da corrente necessária para o funcionamento da solenoide. Para garantir que a queda de tensão medida pelo CI seja a mais próxima possível do

valor real, foi dada uma atenção especial as trilhas que conectam os pinos 4 e 5 do LM1949 aos terminais do resistor R9, minimizando ao máximo a queda de tensão nestas.

2.2.1.3. Condicionador de sinal de rotação

Devido ao sinal de saída do sensor de rotação ser analógico, é necessário um condicionador de sinal para que seja possível a identificação dos pulsos pelo microcontrolador. Este condicionador tem como principal função garantir que os pulsos tenham um tempo de subida e descida baixos, evitando situações de metaestabilidade.

A Figura 2.6 apresenta o sinal do sensor antes (em amarelo) e após (em azul) o condicionador de sinal.

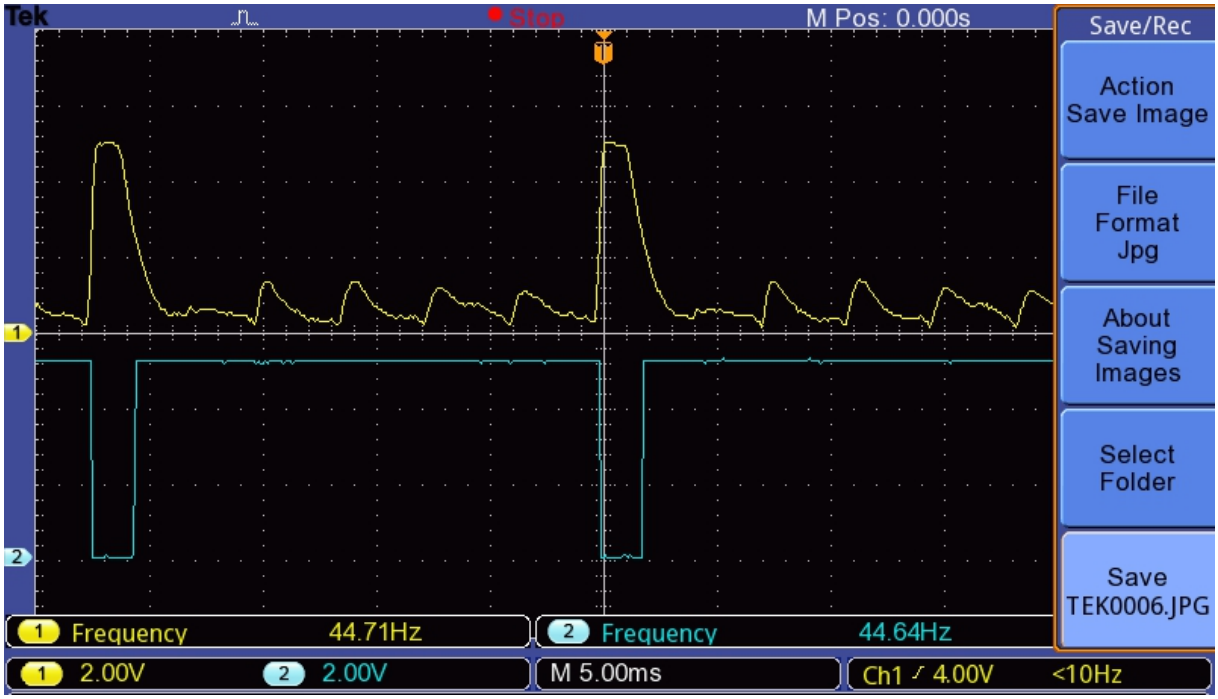


Figura 2.6 – Sinal do sensor antes e após o condicionador de sinais (Elaboração própria).

Prevendo a instalação futura de um segundo conjunto turbocompressor, dois condicionadores de sinais idênticos foram disponibilizados na interface de aquisição e controle. Na Figura 2.7 temos o desenho esquemático de um destes.

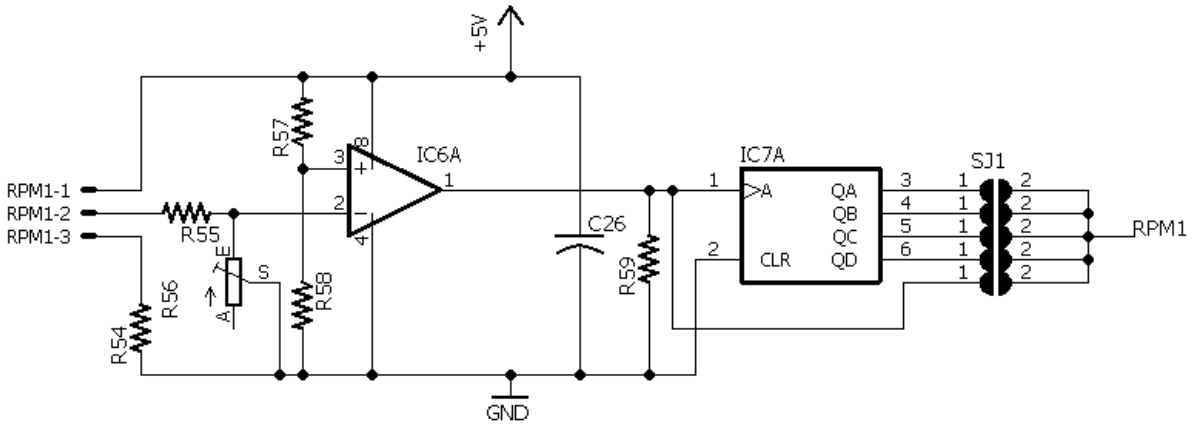


Figura 2.7 – Esquemático do condicionador de sinal de rotação (Elaboração própria).

Este circuito é baseado no amplificador operacional MCP602, fabricado pela Microchip, sendo que somente um dos amplificadores deste CI é utilizado, no esquemático este é representado por IC6. O amplificador está configurado de forma a funcionar como *Schimitt Trigger*, e os valores de R57 e R58 foram calculados de forma que o sinal de saída somente mudará para nível alto caso o sinal de entrada cruze o limite de 1/3 de Vcc e assumirá um nível baixo somente se o sinal cruzar o limite de 2/3 de Vcc.

O sinal de saída do amplificador operacional foi conectado a um dos contadores binários de um 74HC393A, o segundo contador deste CI foi utilizado para o segundo canal de entrada de rotação. No esquemático, IC7A representa este contador. Tal contador foi utilizado para permitir que o tempo entre os pulsos enviados ao microcontrolador seja aumentado em 2, 4, 8 ou 16 vezes. Desta forma, é possível reduzir o uso de processamento necessário para a medição da rotação. Entretanto, a resolução da medição será reduzida proporcionalmente. O *jumper* SJ1 permite selecionar se e qual divisor será utilizado.

O *trimpot* multivota R56, juntamente com o resistor R55, formam um divisor de tensão na entrada do amplificador operacional, permitindo ajustar a sensibilidade de entrada baseada na reflexividade das pás do compressor utilizado.

O resistor R54 limita a corrente que flui através do *LED* infravermelho do sensor óptico. Já o resistor R59 tem a função de garantir que o sinal de saída de IC6 não oscile quando estiver em nível baixo. O capacitor C26 é um capacitor de *bypass* para melhorar a estabilidade do amplificador operacional.

2.2.1.4. Relés

Para controlar dispositivos externos, a interface possui cinco relés, sendo um comutando uma saída 12VDC e outros quatro com os terminais normalmente abertos acessíveis, permitindo a ligação de cargas de até 10A alimentadas por até 250VAC.

Na Figura 2.8 temos o esquemático de um dos módulos de relé.

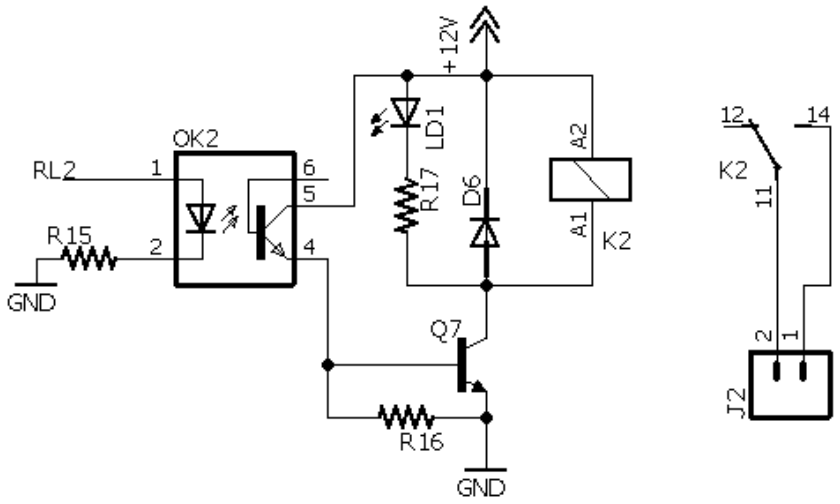


Figura 2.8 – Esquemático do módulo de relé (Elaboração própria).

A fim de garantir a segurança do operador e de evitar danos a interface ou a equipamentos conectados a ela, em caso de curto-circuito todos os relés são isolados opticamente através de um optoacoplador, representado por OK2, do restante do circuito da interface. O resistor R15 limita a corrente que flui através do emissor do optoaclopador. O resistor R16 garante que o transistor Q7, responsável por acionar o relé K2, não irá saturar enquanto OK2 não for ativado. O led LD1 junto com seu resistor limitador de corrente R17 fornecem um indicativo visual do estado do relé. Já o diodo D6 fornece um caminho para corrente reversa, proveniente do colapso do campo magnético da bobina do relé quando este é desligado, fluir sem causar danos a Q7.

2.2.1.5. Saídas para inversor de frequência

A interface desenvolvida também conta com duas saídas para o controle remoto de um inversor de frequência. Uma destas saídas é digital, permitindo o acionamento de uma entrada digital do inversor de frequência, tipicamente utilizada para indicar ao inversor que o motor deve entrar em operação. Já a outra saída é analógica, permitindo o controle da frequência do inversor de forma remota. Ambas as saídas são isoladas através de optoacopladores do restante da interface a fim de minimizar o ruído proveniente do inversor e também evitar danos a interface, visto que

por padrão de indústria, os inversores de frequência trabalham com 24VDC para a alimentação de suas portas. A alimentação da parte isolada dessas saídas deverá ser fornecida pelo próprio inversor.

Mais uma vez temos na Figura 2.9 o esquemático.

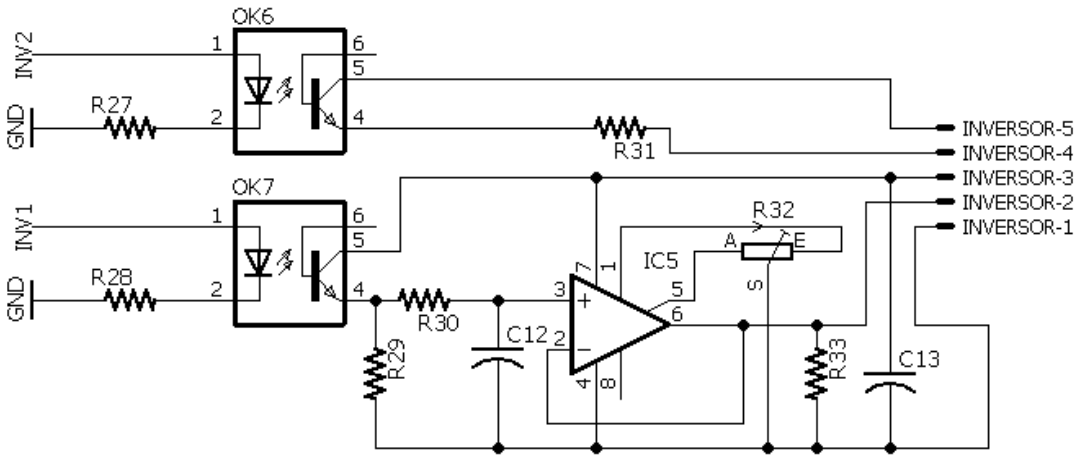


Figura 2.9 – Esquemático *driver* inversor de frequência (Elaboração própria).

O optoacoplador OK6 é responsável por fornecer a saída digital. O resistor R27 limita a corrente que flui através do emissor do optoaclopador. Já o resistor R31 limita a corrente que flui através do fototransistor do optoaclopador.

O optoacoplador OK7 isola a saída analógica do restante da interface, e o resistor R28 limita a corrente que flui através de seu emissor. O coletor do fototransistor é ligado a alimentação positiva fornecida pelo inversor de frequência, enquanto seu emissor é conectado ao terminal negativo da alimentação fornecida pelo inversor através do resistor R29. O emissor também é conectado a um filtro RC passa-baixa composto pelo resistor R30 e pelo capacitor C12, responsável por fornecer uma tensão constante e proporcional ao ciclo de trabalho do PWM que aciona o optoacoplador.

Uma vez que a corrente máxima que pode ser fornecida através do filtro passa-baixa é muito pequena para ser utilizada para o acionamento direto do inversor de frequência, utilizou-se um amplificador operacional LM741, fabricado pela *Texas Instruments*, como driver, representado por IC5. O resistor R33 fornece uma pequena carga na saída do amplificador operacional a fim de melhorar sua estabilidade. O *trimpot* multivolta R32 permite o ajuste da tensão de saída quando a tensão na entrada não inversora do amplificador é zero. O capacitor C13 é um capacitor de *bypass* para o LM741.

2.2.1.6. Entradas analógicas

A interface conta com 4 entradas analógicas para a conexão com qualquer sensor que seja alimentado por 5VDC e que possua sinal de saída de no máximo 5VDC.

Na Figura 2.10 temos o esquemático de uma dessas entradas.

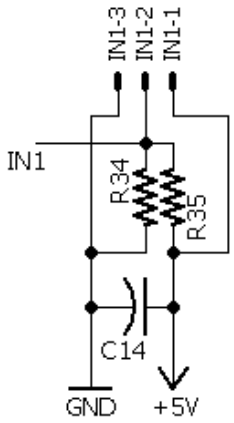


Figura 2.10 – Esquemático para entrada analógica (Elaboração própria).

Cada saída pode ser configurada individualmente como *pull-up* ou *pull-down* através dos resistores R35 e R34, respectivamente. O capacitor C14 é utilizado como *bypass* para o sensor.

2.2.1.7. Entradas sensores de pressão e temperatura

A interface possui 4 entradas analógicas dedicadas para a ligação de sensores de pressão e temperatura conjugados. Cada uma dessas entradas é conectada em duas portas distintas do ADC do microcontrolador.

A Figura 2.11 apresenta um esquemático de uma das entradas.

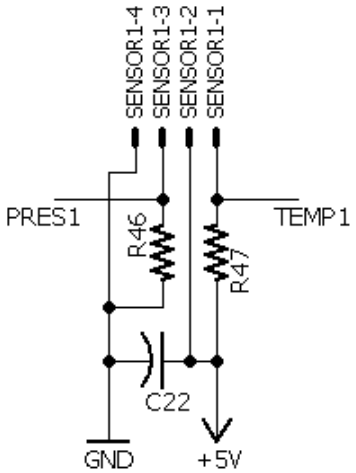


Figura 2.11 – Esquemático para entrada de pressão e temperatura (Elaboração própria).

Cada uma das entradas fornece alimentação 5VDC para o sensor e possuem duas portas para entrada de sinal. A entrada para o sinal de pressão utiliza o resistor R46 como *pull-down*, enquanto a entrada para o sinal de temperatura utiliza o resistor R47 para formar um divisor de tensão junto ao termistor do sensor. Foi utilizada esta configuração pois é a mais usual na indústria automotiva, garantindo a compatibilidade com diversos sensores disponíveis no mercado.

O capacitor C22 é utilizado como *bypass* para o sensor.

2.2.1.8. Entrada sensores de pressão

A interface possui também duas entradas dedicadas a sensores de pressão. Estas entradas são iguais às entradas para sensores de pressão e temperatura, exceto que não possuem a segunda porta para conexão do sinal de temperatura, como pode ser visto no esquemático da Figura 2.12.

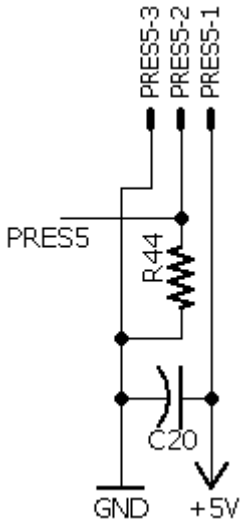


Figura 2.12 – Esquemático para entrada de pressão (Elaboração própria).

O sensor é alimentado por 5VDC. O resistor R44 atua como pull-down para a entrada de pressão, enquanto o capacitor C20 é utilizado como *bypass* para sensor.

2.2.1.9. Entradas para sensores de fluxo de massa de ar

A interface conta também com duas entradas dedicadas para sensores de fluxo de massa de ar do tipo filme quente com condicionador interno de sinal. O sensor pode ser alimentado tanto por 5VDC ou 12VDC, selecionável através de *jumper*, contudo seu sinal de saída máximo deverá ser de 5VDC.

A Figura 2.13 apresenta o desenho esquemático destas entradas.

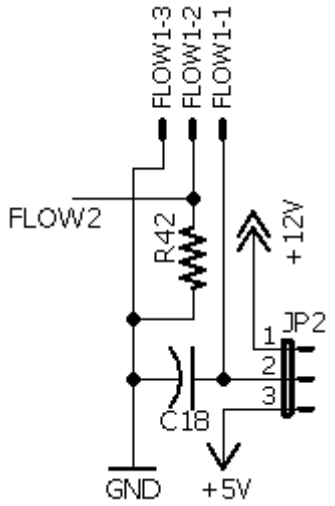


Figura 2.13 – Esquemático para entrada de fluxo de ar (Elaboração própria).

O resistor R42 atua como *pull-down* para a entrada de sinal, enquanto o capacitor C18 é utilizado como *bypass* para sensor.

2.2.1.10. Regulador de tensão

A interface foi concebida para ser alimentada por uma fonte de 12VDC, visto que esta é a tensão utilizada pelas solenoides de controle de combustível selecionadas. Como o microcontrolador utilizado e também os sensores selecionados necessitam ser alimentados por 5VDC, a interface possui um circuito regulador de tensão incorporado a fim de garantir que tensão fornecida esteja livre de ruídos e seja de boa qualidade.

Este regulador foi baseado no CI LM317, fabricado pela Texas Instruments, contudo, para ser capaz de fornecer a corrente necessária para alimentar todos os módulos da interface, utilizou-se um transistor de potência NPN como driver para a carga. Com esta configuração, o regulador consegue fornecer até 4A de forma contínua, contra somente 1,5A caso fosse utilizado somente o LM317.

Optou-se por utilizar um regulador de tensão linear convencional por estes apresentarem uma melhor rejeição de *ripple*, e possuírem um design mais simples e robusto (IVANOV, 2019).

Na Figura 2.14 temos o desenho esquemático do regulador de tensão.

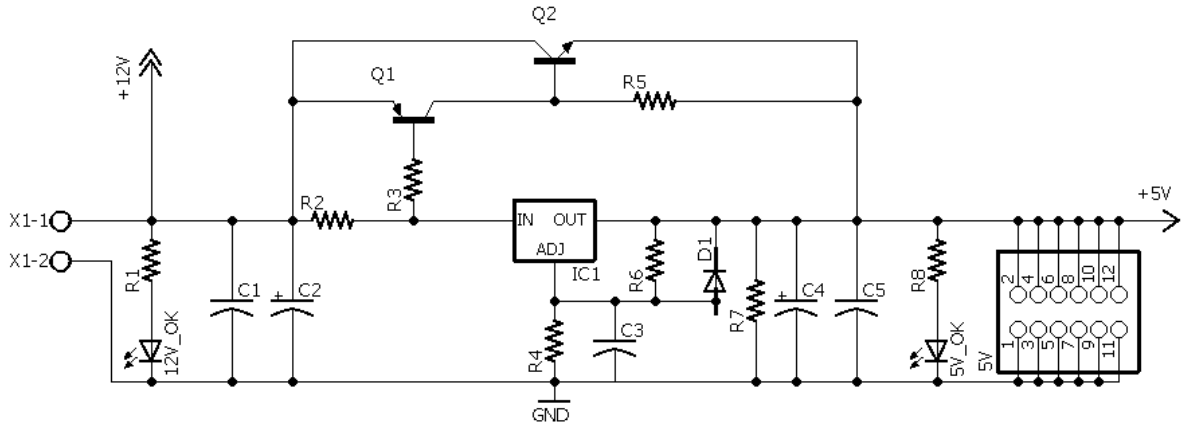


Figura 2.14 – Esquemático do regulador de tensão (Elaboração própria).

A alimentação 12VDC é conectada a interface através do conector X1, onde segue para o regulador de tensão e é derivada para alimentar os módulos de relé, o estágio de potência dos drivers das solenoides e também os sensores de fluxo de massa de ar, se for o caso.

O *LED* 12V_OK, em conjunto com seu resistor limitador de corrente R1, fornece um indicativo visual de que a interface está sendo alimentada. Os capacitores C1 e C2 reduzem o ruído proveniente da fonte de alimentação. Utilizou-se dois capacitores de diferentes tipos, um eletrolítico e outro cerâmico, e diferentes valores para que respondam a uma faixa maior de frequências (SCHMITZ, 2011). O conjunto R2, R3, R5 e Q1 são responsáveis por acionar o transistor de potência Q2, responsável por variar a tensão na saída do circuito IC1 é o LM317, que regula a tensão de saída variando a queda de tensão em R2 de forma a manter a tensão de saída constante. Os resistores R4 e R6 formam o divisor de tensão responsável por definir a tensão de saída do LM317, seus valores foram calculados utilizando a seguinte fórmula:

$$V_{out} = V_{REF} \cdot (1 + \frac{R4}{R6}) \tag{2.1}$$

onde V_{out} é a tensão de saída desejada, no caso 5V, V_{REF} é igual a 1,25V, e os valores de R4 e R6 devem ser expressos em Ohms (TEXAS INSTRUMENTS, 2016).

O capacitor C3 tem como função estabilizar a tensão na entrada de ajuste do LM317, ajudando a reduzir o ruído presente na tensão de saída do regulador. O D1 existe para descarregar C3 caso ocorra um curto-circuito na saída do regulador. O resistor R7 impõe uma pequena carga constante na saída do regulador a fim de melhorar sua estabilidade. Os capacitores C4 e C5 tem a função de estabilizar a tensão de saída, e assim como na entrada do regulador, são de tipos e valores diferentes entre si. O *LED* 5V_OK, e seu resistor R8, indicam que o regulador está

funcionando. O conector 5V foi previsto para fornecer de forma simples e fácil terminais para a conexão futura de outros módulos a interface que necessitem de alimentação 5VDC.

Testes de desempenho do regulador desenvolvido são apresentados na sessão “Resultados e Discussões”.

2.2.1.11. Termopares

Inicialmente, projetou-se a interface com quatro entradas para termopares tipo K, utilizando-se como condicionador de sinal o CI MAX6675, fabricado pela Maxim, e estes sendo lidos diretamente pelo microcontrolador da interface. Em testes iniciais, essa configuração se comportou como o esperado e este design chegou a ser incorporado na PCI da interface e fabricado. Contudo, quando realizados testes com a interface completa, observou-se que esta abordagem não era adequada, fazendo com o que o microcontrolador esperasse um tempo demasiadamente alto, em torno de 150ms por termopar, para obter a medição. Durante todo este tempo, o microcontrolador não era capaz de executar nenhuma outra função, fazendo com que a taxa de amostragem da interface não atendesse às premissas do projeto.

A fim de solucionar essa falha, desenvolveu-se um módulo externo à interface utilizando outro microcontralodar para a leitura dos termopares.

O microcontrolador utilizado foi o ATmega328P, na forma de um Arduino Nano, pelos mesmos motivos de se ter utilizado um Arduino como microcontrolador principal para interface. Os CIs condicionadores de sinal também foram substituídos pele MAX31855, também fabricados pela Maxim, pois seu tempo de conversão é significativamente menor que o MAX6675, 70ms contra 125ms. Optou-se por utilizar módulos comercialmente disponíveis que contêm o MAX31855 e todos os componentes necessários ao seu funcionamento de forma a simplificar a fabricação do módulo.

O funcionamento do módulo é relativamente simples: O ATmega328P aguarda que o estado do pino 5 seja alterado de baixo para alto, quando isso acontece, o mesmo inicia a leitura de cada um dos três termopares através dos três CIs MAX31855 e armazena estes valores em variáveis. Em seguida, verifica se os dados recebidos são números inteiros válidos, caso positivo os valores são separados em dois *bytes* distintos cada para serem enviados através de uma conexão serial para o microcontrolador principal. Também é enviado um sétimo *byte* para *checksum*,

calculado como a soma de todos os 6 *bytes* enviados. Caso o valor da soma ultrapasse 1 *byte* de tamanho, somente o *byte* inferior é enviado.

Na Figura 2.15 – Esquemático do módulo do termoparFigura 2.15 temos o esquemático do módulo desenvolvido. Devido a sua simplicidade, este foi fabricado utilizando uma placa de circuito impresso universal perfurada, decisão que provou não ter nenhum impacto significativo no desempenho. Tal construção pode ser observada na Figura 2.16.

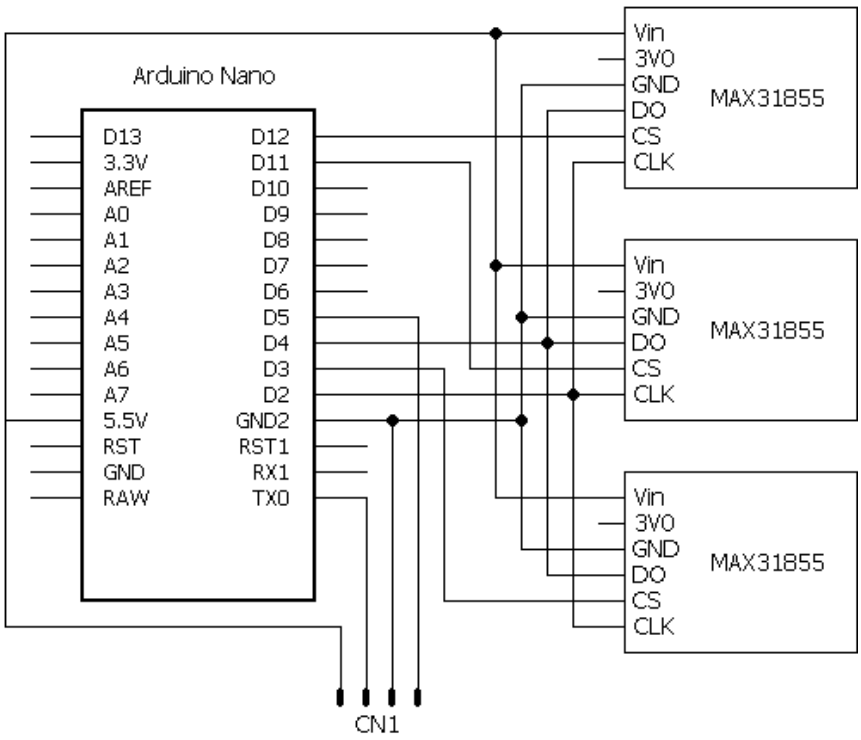


Figura 2.15 – Esquemático do módulo do termopar (Elaboração própria).

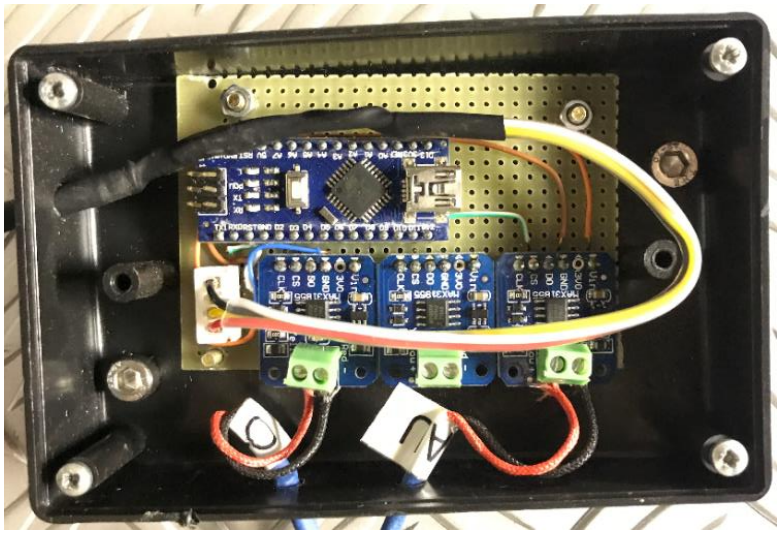


Figura 2.16 – Foto do módulo termopar (Elaboração própria).

O código fonte do módulo está disponível nos Anexo II deste trabalho.

2.2.1.12. Botão de parada

A interface conta também com um botão de parada, garantindo ao operador uma forma de desligar a bancada experimental de forma segura, mesmo que não esteja próximo ao computador que está executando o software de controle.

Foi utilizada uma botoeira de uso industrial CS-102, fabricado pela MarGirius Eletric, para esta função. Na Figura 2.17 temos uma foto da botoeira utilizada.

Figura 2.17 – Botoeira. (Elaboração própria)

O circuito elétrico do botão de parada é bem simples, sendo a botoeira conectada diretamente a um dos pinos de entrada do microcontrolador e seu outro terminal é conectado ao negativo da alimentação, como pode ser visto na Figura 2.18.

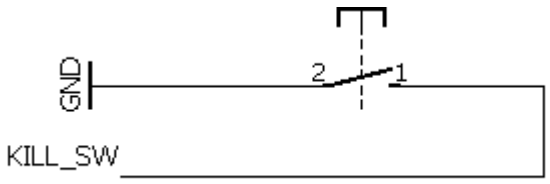


Figura 2.18 – Esquemático botoeira (Elaboração própria).

2.2.2. Software embarcado

O software embarcado na interface foi desenvolvido utilizando a IDE Arduino 1.8 a fim de facilitar os testes e desenvolvimento da interface, visto que esta IDE permite escrever o código, compilar e realizar o upload para o microcontrolador através de um único aplicativo. A IDE

permite que o programa seja escrito diretamente em linguagem C ou C++, facilitando a manipulação de varáveis e a criação de funções com maior complexidade.

O software embarcado é bem simples, visto que todo o tratamento dos dados é realizado pelo software de controle, de forma a reduzir o processamento exigido do microcontrolador.

O código pode ser subdivido em 11 funções principais, sendo elas:

- Setup()
- Loop()
- sendData()
- resendData()
- killFuel()
- ISR(TIMER1_OVF_vect)
- ISR(TIMER2_OVF_vect)
- ISR(TIMER4_CAPT_vect)
- ISR(TIMER4_OVF_vect)
- ISR(TIMER5_CAPT_vect)
- ISR(TIMER5_OVF_vect)

A execução do código se inicia pela função Setup() e segue para a função Loop(). As funções sendData(), resendData() e killFuel() somente são executadas quando chamadas por outra função, já as funções iniciadas por ISR (*Interrupt Service Routine*) são executadas sempre que um evento, *overflow* do *timer* para as funções OVF ou evento na porta Input Capture para as funções CAPT ocorrer. Essas funções são executadas sempre com prioridade acima das outras funções, por isso foram escritas de forma a serem executadas no menor tempo possível, conforme as boas práticas de programação.

O código fonte completo do software embarcado na interface está disponível no Anexo III deste trabalho.

2.2.2.1. Função setup()

Esta função é executa toda vez que a interface é ligada ou reiniciada e é responsável por setar os parâmetros dos *timers* usados pelo software, iniciar as portas UARTs para comunicação entre o computador contendo o software de controle e para comunicação com o módulo de aquisição

dos termopares, setar as portas de entrada e saída e garantir que as saídas utilizadas para comandar os injetores e relés estejam em estado baixo.

2.2.2.2. Função loop()

Esta função é executada de forma contínua durante todo o tempo que a interface está ligada e responsável pela comunicação entre a interface e o software de controle. O fluxograma apresentado na Figura 2.19 detalha a execução desta função.

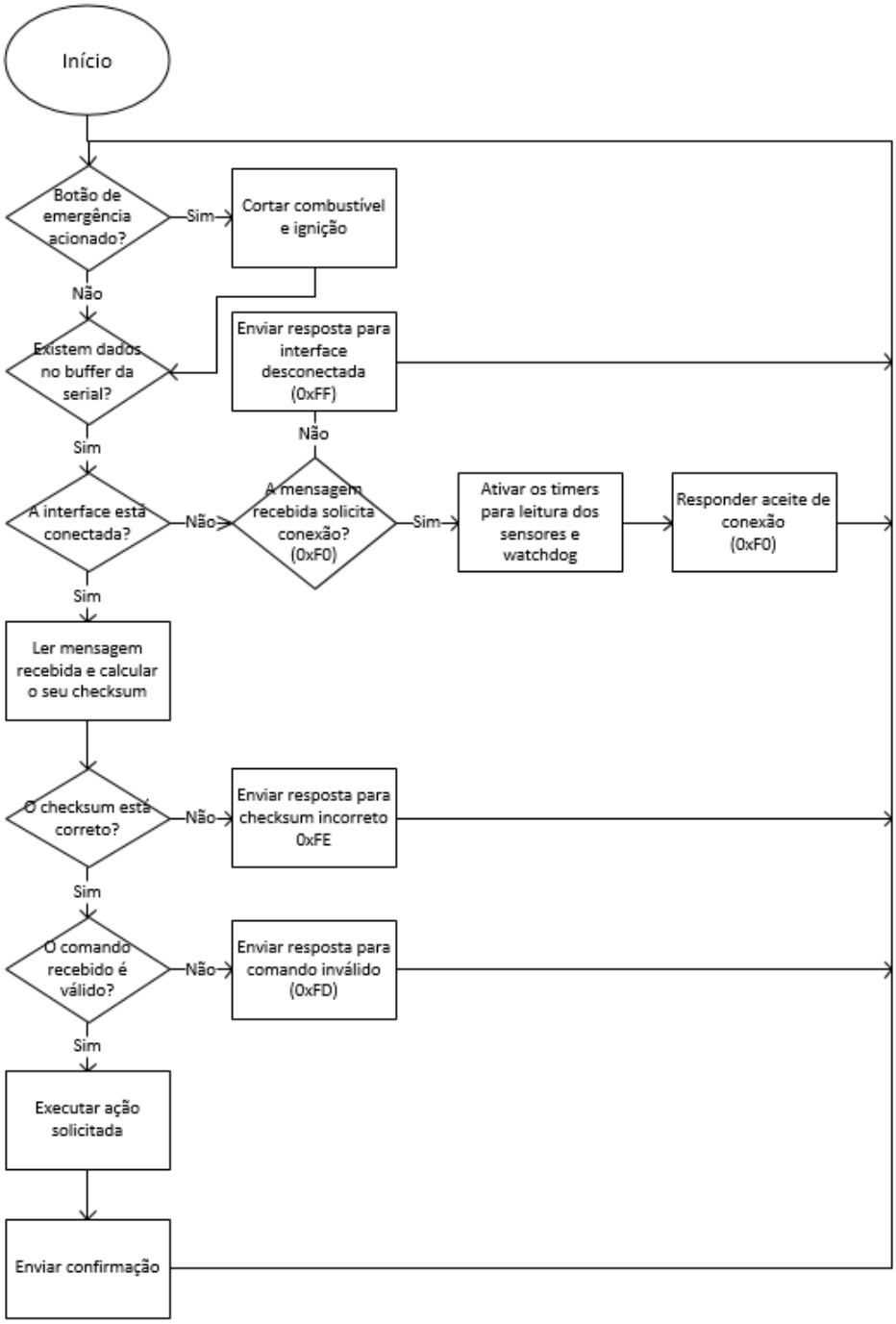


Figura 2.19 – Fluxograma da função loop() (Elaboração própria).

A função sempre é iniciada verificando se o botão de parada está acionado ou não. Em caso positivo, o ciclo de trabalho do PWM dos solenoides de combustível é zerado e a saída de ignição é desligada, através da função killFuel().

Em seguida, a função verifica se há dados recebidos no *buffer* da porta UART conectada ao computador. Caso existam dados, a função verifica se existe uma conexão ativa entre a interface e o software de controle.

Caso não exista a conexão, a função verifica se a mensagem recebida é a mensagem de solicitação de conexão, representada por um único *byte* contendo o valor 0xF0. Caso a mensagem esteja correta, a interface inicia os *timers* para leitura dos sensores e também para a função de *watchdog*, e responde positivamente para o software de controle que a conexão foi bem-sucedida através de um único byte contendo o valor 0xF0. Caso a mensagem de conexão esteja incorreta, a interface somente envia ao software de controle uma resposta negativa para a conexão, representada por um único *byte* com o valor 0xFF.

Vale ressaltar que a partir do início da conexão, todas as respostas enviadas pela interface ao software de controle são através de *strings* ASCII, contudo as mensagens enviadas do software à interface ainda são no formato binário.

Caso a conexão entre a interface e o software de controle já tenha sido estabelecida, a função irá ler a mensagem recebida e calcular o seu *checksum*. Todas as mensagens enviadas a interface após a sua conexão devem seguir o seguinte formato para serem válidas:

- O primeiro *byte* sempre indica o tamanho da mensagem, em *bytes*, e seu valor máximo é 4. Este primeiro *byte* não é considerado para o tamanho nem para o checksum da mensagem;
- O segundo *byte* sempre indica qual comando será executado pela interface;
- Os próximos dois *bytes* indicam os parâmetros para execução do comando e devem ser omitidos para comandos que não necessitam desses;
- O último *byte* sempre é o *checksum* da mensagem, calculado pelo software de controle.

Após a leitura, a função irá calcular o *checksum* da mensagem usando o seguinte algoritmo:

$$Checksum = \left(\sum_{1}^{n-1} byte_n\right) \&\& \; 0xFF \tag{2.2}$$

Essa estrutura para as mensagens, assim como o algoritmo utilizado para o *checksum*, foram baseados no protocolo KWP2000 (ISO 14230-2), amplamente utilizado na indústria automotiva (PÓVOA, 2007).

Caso o *checksum* calculado seja igual ao *checksum* recebido no último *byte* da mensagem, esta é considerada válida e a função segue para a próxima etapa de análise da mensagem recebida. Caso o *checksum* esteja incorreto, a interface responde ao software de controle com uma *sting* ASCII contendo o valor 0xFE.

Com a mensagem validada, a função irá verificar qual o comando enviado e seus parâmetros. Atualmente, a interface conta com os seguintes comandos:

Tabela 2.1 – Comandos da interface (Elaboração própria).

Comando	Ação	Parâmetros Adicionais
0x10	Setar DC PWM	Porta, Valor
0x11	Setar saída digital	Porta, Valor
0x20	Enviar dados	-
0x21	Reenviar dados	-
0x30	Setar PWM injetores radiais	Valor
0x31	Setar PWM injetor piloto	Valor

Caso seja enviado um comando inválido para a interface, a mesma irá responder com uma *string* ASCII contendo o valor 0xFD e nenhuma ação será executada.

Os comandos 0x10 e 0x11 utilizam as funções analogWrite() e digitalWrite(), respectivamente, para executarem suas ações. Os seus parâmetros são os mesmos utilizados por essas funções. Após executada a ação, a interface responde ao software de controle com o mesmo valor do comando recebido através de uma *string* ASCII como confirmação de que a mensagem foi recebida e executada corretamente.

Os comandos 0x20 e 0x21 iniciam a execução de das funções sendData() e resendData(), respectivamente.

Os comandos 0x30 e 0x31 são utilizados para controlar o ciclo de trabalho das solenoides de combustível. Esses comandos também utilizam a função analogWrite() para sua execução. Contudo, antes da sua execução, é checado o status do Botão de Parada, evitando que as

solenoides sejam acionados caso o mesmo esteja ativo. Caso o botão não esteja ativo, após a ação ser executada, a interface responde ao software de controle através de uma *string* ASCII com o mesmo valor do comando recebido. Já se o botão de parada estiver ativo, nenhuma ação é executada e a interface responde ao software de controle através de uma *string* ASCII com o valor do comando recebido mais 1.

Toda vez que um comando válido é recebido pela interface, o contador da função de *watchdog wd* é resetado para 0.

Ao fim, a função Loop() retorna ao seu início e um novo ciclo é executado.

2.2.2.3. Função sendData()

A função sendData() é responsável por enviar os dados coletados dos sensores para o software de controle. Sua execução é iniciada somente quando o comando 0x20 é recebido pela interface.

A função sendData() inicia verificando se existem dados no *buffer* da porta UART utilizada para comunicação com o módulo externo de aquisição dos termopares. Caso existam dados, os mesmos são lidos, validado o seu *checksum* e caso sejam válidos, são formatados e escritos no vetor *sdb*. Caso o *checksum* não seja válido, os valores já armazenados no vetor *sdb* não são modificados.

Em seguida, a função verifica se o *timer* utilizado para medição da rotação estourou desde a última execução da função. Caso positivo, a função considera a rotação como zero e reseta a variável *rpm_of*. Caso negativo, a função copia o valor do vetor *db* para o vetor *sdb*.

A seguir a função desativa o evento de *overflow* do *timer* 1 enquanto copia os dados contidos no vetor *db* para o vetor *sdb*. O *timer* é desativado para evitar que a função ISR(TIMER1_OVF_vect) seja executada durante a cópia, o que afetaria a validade dos dados copiados.

Por fim, a função calcula o *checksum* dos valores contidos no vetor *sdb* e envia esses valores para o software de controle através uma *string* ASCII, separando cada valor através de uma vírgula.

2.2.2.4. Função resendData()

A função resendData() somente é executada quando o comando 0x21 é recebido pela interface. Esta função somente reenvia os dados enviados na última execução da função sendData() para

o software de controle. Esta função foi implementada para ser executada quando o *checksum* da resposta enviada pela função sendData() não for válido. Sua resposta é a mesma da função sendData().

2.2.2.5. Função killFuel()

Está função é executada sempre que o botão de parada estiver acionado ou sempre que o *timer* da função de *watchdog* estourar. Esta função desliga a saída do ignitor da câmara de combustão e também desligada todas as solenoides de combustível, forçando o desligamento da bancada experimental.

2.2.2.6. Função ISR(TIMER1_OVF_vect)

Esta função é executada sempre que o contador TCNT1 estourar, garantindo que seja executada periodicamente de forma precisa. Nas configuração definidas na função Setup(), o *timer* 1 está programado para incrementar seu contador a 1/8 do *clock* do microcontrolador, ou seja, o contador é incrementado a uma taxa de 2MHz. Como este *timer* é de 16 bit, seu valor máximo é de 65.535, o que daria um período de aquisição de aproximadamente 32ms. Contudo, ao fim da execução da função, o valor do contador deste *timer* é setado em 55.535, fazendo com que a função seja executada a cada 5ms, resultando em uma taxa de aquisição de 200Hz.

Esta função é responsável por ler todos os sensores conectados às entradas da interface de aquisição e também por acionar o módulo externo de aquisição dos termopares.

A função inicia lendo os valores do conversor analógico-digital e somando os valores obtidos aos valores já presentes no vetor *db* utilizado como *buffer*. Cada porta do converso analógico-digital possui uma posição específica no vetor. Este vetor também possui uma posição que é incrementada toda vez que a função soma novos valores, permitindo realizar a média dos valores somados.

Em seguida, a função incrementa a variável *tct* que é utilizada como contador para acionar o módulo externo de aquisição dos termopares.

Se o valor de *tct* for igual a 1000 e não existirem dados no *buffer* da porta UART utilizada para comunicação com o módulo externo de aquisição dos termopares, este é acionado através da mudança de estado do pino 20 do microcontrolador de baixo para alto.

Quando *tct* atinge o valor de 1010, o pino 20 tem seu estado alterado para baixo e o contador é resetado para 0. Vale ressaltar que esta implementação para a temporização do acionamento do módulo externo de aquisição dos termopares foi feita via software, portanto este contador não é uma função de interrupção.

Por fim, a função reseta o contador TCNT1 para o valor de 55.535, fazendo que a função seja executada novamente a cada 5ms.

2.2.2.7. Função ISR(TIMER2_OVF_vect)

Esta é a função de *watchdog* da interface, responsável por resetar a interface caso a mesma fique sem comunicação com o software de controle por mais de 10s. Como o *timer* 2 do microcontrolador foi utilizado para esta função, a resolução de 8 bit do contador TCNT2 não é suficiente para contar este tempo, portanto esta função utiliza este *timer* para incrementar a variável *wd* através de software e quando esta variável atingir o valor de 615, a interface será reiniciada. Esta abordagem não causa nenhum efeito negativo no funcionamento e confiabilidade desta função.

2.2.2.8. Função ISR(TIMER4_CAPT_vect)

A medição de rotação do turbocompressor da bancada experimental é realizada utilizando o *timer* 4 operando no modo *Input Capture*. Neste modo, sempre que o sinal da porta alternar do estado baixo para alto, o valor do contador TCNT4 é copiado diretamente pelo hardware do micrcontrolador para o registro ICR4, garantindo que a medição não tenha sua precisão afetada caso o microcontrolador esteja executando alguma outra função de interrupção no momento em que o sinal do sensor acionar a porta.

Assim, essa função se inicia zerando o contador TCNT4, visto que seu valor já está armazenado no registro ICR4. Em seguida, é verificado através da variável *first4* se o contador estourou desde a última execução da função. Caso positivo, a função somente seta a variável *first4* como falsa. Caso negativo, o valor de ICR4 é copiado para o vetor *db*.

2.2.2.9. Função ISR(TIMER4_OVF_vect)

Esta função é executada sempre que o contador TCNT4 estourar e tem como objetivo evitar erros na medição da rotação, permitindo que a função sendData() critique o valor contido em ICR4.

2.2.2.10. Função ISR(TIMER5_CAPT_vect) e Função ISR(TIMER5_OVF_vect)

Estas funções são análogas às funções ISR(TIMER4_CAPT_vect) e ISR(TIMER4_OVF_vect), porém são utilizadas para a segunda entrada de rotação e utilizam o *timer* 5 e seus respectivos contadores/registros.

2.3. SENSORES

Todos os sensores utilizados na bancada experimental foram selecionados de forma a atender às premissas do projeto, garantindo um desempenho adequado ao propósito da bancada.

Os sensores utilizados são destinados a aplicações automotivas ou industriais, estando aptos a operarem em ambientes com grandes gradientes de temperatura e vibração. Optou-se pelo uso de sensores automotivos quando possível pois o custo-benefício proporcionado por esses sensores é imbatível. A principal desvantagem na utilização de sensores automotivos é que sua curva de calibração normalmente não é disponibilizada pelo fabricante, porém esses podem ser calibrados em bancadas ou é possível extrair a curva de calibração do software embarcado na ECU do veículo através de engenharia reversa.

2.3.1. Sensor de rotação

O sensor de rotação utilizado foi o TCRT5000, fabricado pela Vishay. Este sensor é do tipo reflexivo e possui um *LED* infravermelho e um fototransistor montados no seu corpo, como pode ser visto na Figura 2.20.

Figura 2.20 – Sensor de rotação (VISHAY SEMICONDUCTORS, 2009).

Tal sensor não necessita de calibração, uma vez que seu sinal de saída é digital, mudando de estado baixo para alto sempre que uma superfície refletiva se aproxima do sensor.

Para a superfície reflexiva foi utilizada as próprias pás do compressor do conjunto tubocompressor utilizado na bancada experimental.

2.3.2. Sensor de pressão e temperatura

Para a medição de pressão e temperatura do ar foi utilizado o sensor Bosch 0 281 002 576, utilizado por veículos equipados com motores Cummins ISBe6 ou ISBe4, como os caminhões VW Constellation 17.250 e VW Worker 15.190 E Electronic. A Figura 2.21 mostra uma foto do sensor utilizado.

Figura 2.21 – Sensor de pressão (Elaboração própria).

Este sensor é composto por dois elementos: um termistor do tipo NTC para medição de temperatura e um elemento piezelétrico integrado a um condicionador de sinal para a medição de pressão. Este sensor fornece uma medição de pressão absoluta.

A curva de calibração deste sensor foi obtida através do *datasheet* do sensor Bosch 0 281 006 059, que utiliza os mesmos elementos e possui a mesma escala de medição, porém é disponibilizado em outro padrão de fixação.

Para a medida de pressão o sensor fornece um sinal de tensão linearmente proporcional variando entre 0,5V e 4,5V. Desta forma, a curva de calibração será uma equação linear do tipo $P = A \cdot V + B$, como visto no gráfico apresentado na Figura 2.22.

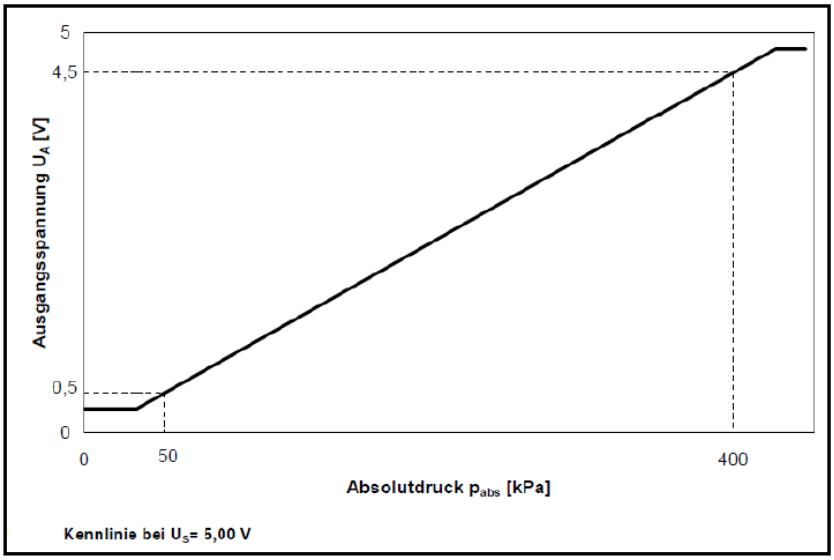


Figura 2.22 – Curva de calibração (ROBERT BOSCH GMBH).

Do gráfico é possível extrair a Equação (2.3).

$$P = 87{,}5 \cdot V + 6{,}25 \tag{2.3}$$

onde P é a pressão medida, em kPa, e V é a tensão de saída, em Volts.

Como a interface envia o valor de saída do conversor analógico-digital para o software de controle, ajustou-se a equação para que esta aceite este parâmetro como entrada, obtendo-se assim a Equação (2.4).

$$P = 0{,}4277 \cdot ADC + 6{,}25 \tag{2.4}$$

onde P é a pressão medida, em kPa, e ADC é o valor de saída do conversor analógico-digital.

Já para a medição de temperatura, a curva de calibração não é linear, como esperado para um termistor NTC. O *datasheet* fornece a Tabela 2.2 contendo a relação de temperatura e resistência elétrica em diversos pontos. De posse dos dados dessa tabela, criou-se uma rotina em *Matlab*, disponível no Anexo IV, para encontrar uma equação matemática que melhor se aproximasse dos valores.

Tabela 2.2 – Relação entre resistência e temperatura (ROBERT BOSCH GMBH).

	Toleranz – Nominalwerte				Toleranz – Prüfwerte	
Temp.	Widerstand R [Ω]			Toleranz	Widerstand [Ω] mit T ± 1K	
T [°C]	nominal	minimal	maximal	[K]	minimal	maximal
-40	45303	43076	47529	± 0,9	40730	50314
-35	34273	32643	35902	± 0,9	30908	37953
-30	26108	24907	27309	± 0,9	23603	28829
-25	19999	19108	20889	± 0,9	18142	22023
-20	15458	14792	16124	± 0,8	14055	16970
-15	12000	11499	12501	± 0,8	10945	13144
-10	9395	9015	9775	± 0,8	8595	10261
-5	7413	7123	7704	± 0,8	6801	8074
0	5895	5671	6118	± 0,8	5420	6403
5	4711	4537	4884	± 0,8	4343	5106
10	3791	3656	3927	± 0,8	3504	4100
15	3068	2962	3174	± 0,8	2842	3310
20	2499	2416	2583	± 0,8	2323	2690
25	2056	1990	2123	± 0,8	1916	2207
30	1706	1653	1760	± 0,8	1591	1827
35	1411	1368	1455	± 0,8	1318	1510
40	1174	1139	1209	± 0,8	1100	1254
45	987.4	959,0	1016	± 0,8	927,0	1051
50	833.8	810,5	857,0	± 0,8	783,1	886,3
55	702.7	683,7	721,7	± 0,8	661,2	746,6
60	595.4	579,7	611,0	± 0,8	561,6	631,4
65	508.2	495,3	521,1	± 0,8	480,2	537,8
70	435.6	424,9	446,4	± 0,8	412,1	460,3
75	374.1	365,2	383,1	± 0,8	354,4	394,9
80	322.5	315,0	329,9	± 0,8	306,0	339,8
85	279.5	273,2	285,8	± 0,8	265,7	294,0
90	243.1	237,8	248,4	± 0,8	231,5	255,4
95	212.6	208,1	217,1	± 0,8	202,7	223,0
100	186.6	182,9	190,3	± 0,8	178,0	195,4
105	163.8	160,3	167,2	± 0,8	156,2	171,6
110	144.2	141,0	147,3	± 0,9	137,5	151,0
115	127.3	124,4	130,1	± 0,9	121,4	133,4
120	112.7	110,1	115,2	± 1,0	107,5	118,0
125	100.2	97,81	102,5	± 1,0	95,55	104,9
130	89.28	87,13	91,43	± 1,1	85,13	93,52
135	79.63	77,67	81,59	± 1,1	75,93	83,45
140	71.18	69,39	72,97	± 1,2	67,88	74,60
145	63.85	62,21	65,48	± 1,2	60,88	66,91
150	57.39	55,89	58,89	± 1,3	54,75	60,15

A melhor aproximação foi obtida com a Equação (2.5), sendo esta uma soma de duas equações exponenciais. O erro entre o valor calculado pela equação e o valor retirado da tabela para cada ponto é apresentado no gráfico da Figura 2.23. Como o intervalo de medição relevante para a bancada experimental está compreendido entre 15°C e 120°C, o erro máximo considerado é inferior a 5%.

$$T = 85{,}94 \cdot \exp(-0{,}0006158 \cdot R) + 102{,}2 \cdot \exp(-0{,}007911 \cdot R) \qquad (2.5)$$

onde T é a temperatura, em ºC, e R é a resistência do termistor, em Ohms.

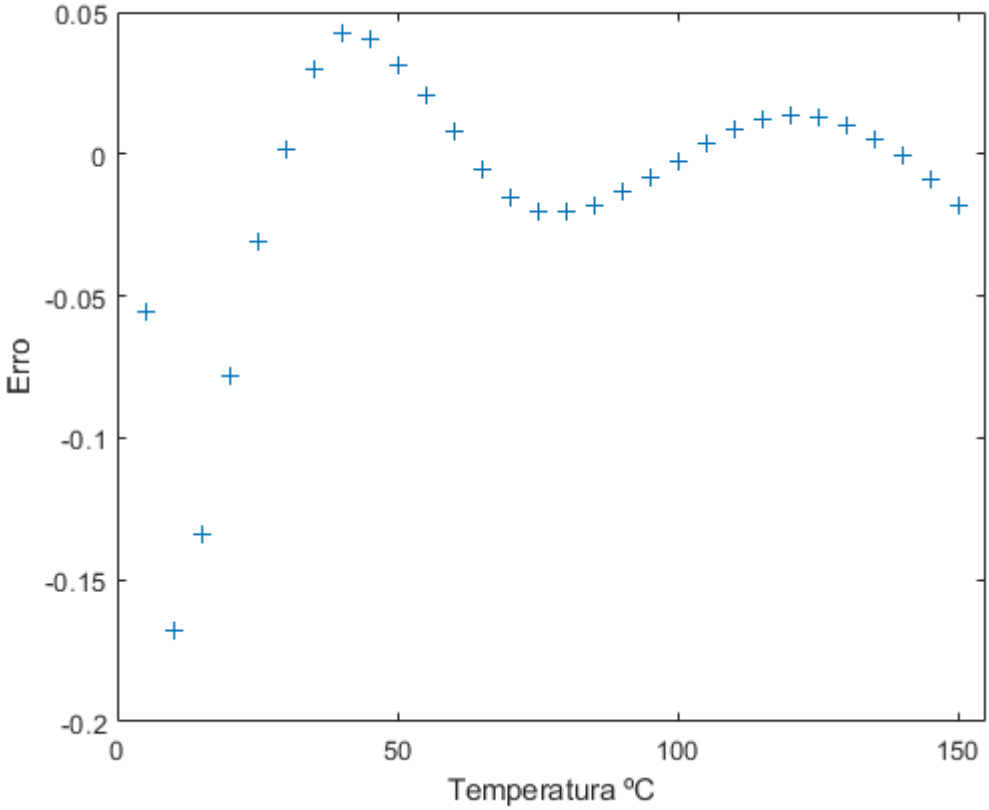


Figura 2.23 – Erro entre valor calculado e valor da tabela (Elaboração própria).

Como o conversor analógico-digital do microcontrolador não consegue realizar a medição direta da resistência do sensor, utilizou-se um divisor de tensão para a obtenção do sinal. O resistor selecionado para o divisor foi do tipo metal-filme, com resistência nominal de 2700 Ohms e 1% de tolerância. O divisor é alimentado com 5VDC. Desta forma, a resistência do sensor pode ser relacionada com a sua tensão de saída de acordo com a Equação (2.6). Por fim a relação entre a temperatura medida e o valor obtido pelo conversor analógico-digital é dada pela Equação (2.7).

$$R = \frac{V \cdot 2700}{5 - V} \tag{2.6}$$

onde R é a resistência do termistor, em Ohms, e V é a tensão de saída do divisor, em Volts.

$$T = 85{,}94 \cdot \exp\left(-0{,}006158 \cdot \left(\frac{\frac{13500 \cdot ADC}{1023}}{5 - \frac{5 \cdot ADC}{1023}}\right)\right) + 102{,}2 \cdot \exp\left(-0{,}007911 \cdot \left(\frac{\frac{13500 \cdot ADC}{1023}}{5 - \frac{5 \cdot ADC}{1023}}\right)\right) \quad (2.7)$$

onde T é a temperatura, em ºC, e ADC é o valor de saída do conversor analógico-digital.

2.3.3. Sensor de fluxo de massa de ar

Para medição do fluxo de massa de ar que é admitida na câmara de combustão, utilizou-se um anemômetro de filme quente de aplicação automotiva. O sensor utilizado foi o Bosch 0 280 218 065, comumente utilizado para medição de fluxo de ar admitido em diversos veículos fabricados pela VW/Audi, como por exemplo o Audi S3 fabricado entre 1999 e 2001.

Este sensor foi selecionado pois fornece uma saída de 0 a 5V proporcional ao fluxo de massa de ar que está sendo medido e por possuir diâmetro compatível com as linhas de ar da bancada.

Não foi encontrado o *datasheet* deste sensor, portanto o mesmo foi calibrado seguindo os procedimentos descritos no trabalho de Alves Filho (2019).

A curva de calibração obtida é apresentada na Equação (2.8).

$$\dot{m} = 0.01274 \cdot V^3 - 0.1019 \cdot V^2 + 0.3174 \cdot V - 0.3289 \quad (2.8)$$

onde $\dot{m}$ é o fluxo mássico de ar, em kg/s, e V e a tensão de saída do sensor, em Volts.

Já a Equação (2.9) apresenta a equação modificada para tomar como entrada o valor de saída do conversor analógico digital.

$$\dot{m} = 1{,}48748 \cdot 10^{-9} \cdot ADC^3 - 2{,}43423 \cdot 10^{-6} \cdot ADC^2 + 1{,}55131 \cdot 10^{-6} \cdot ADC - 0.3289 \quad (2.9)$$

onde $\dot{m}$ é o fluxo mássico de ar, em kg/s, e ADC é o valor de saída do conversor analógico-digital.

2.3.4. Sensor de pressão de óleo

Para o sensor de pressão de óleo foi utilizado o transdutor de pressão Benxu-Tech BX-PTV201-100V. Tal sensor possui a curva de calibração apresentada na Equação (2.10), de acordo com os dados disponibilizados pelo fabricante.

$$P = 25 * X - 12{,}5 \tag{2.10}$$

onde P é a pressão medida, em *psi*, e V é a tensão de saída do sensor, em Volts.

Convertendo a equação para que tenhamos sua saída em kPa e que tome como entrada os valores diretos do conversor analógico-digital, temos a Equação (2.11):

$$P[kPa] = 0{,}84247 \cdot ADC - 86{,}1845 \tag{2.11}$$

onde P é a pressão medida, em kPa, e ADC é o valor de saída do conversor analógico-digital.

2.3.5. Termopar

Os termopares utilizados são do tipo K e devem ser isolados. Para os termopares não é necessário nenhum tipo de tratamento de dados, pois o circuito integrado MAX31855, utilizado para a conexão dos mesmos, já fornece o valor de temperatura diretamente em graus Celsius.

2.4. ATUADORES

A bancada experimental possui somente dois atuadores controlados pelo sistema de aquisição e controle, sendo eles um conjunto de solenoides para o controle da vazão de combustível e um inversor de frequência para o controle da rotação, e consequentemente vazão, da bomba do sistema de lubrificação.

2.4.1. Solenoides de combustível

Para o controle de vazão de combustível, foram utilizados quatro injetores para GNV/GLP de uso automotivo modelo IG1 Apache, fabricados pela RAIL S.r.l. Os injetores são montados sobre uma flauta que distribui o GLP para cada um dos injetores e também serve como base para fixação dos mesmos. Os injetores utilizados possuem bobina do tipo A2, com 2 Ohm de resistência, possuindo um tempo de abertura de 2,8ms +- 10% quando operados com pressão diferencial de 1 bar. A Figura 2.24 mostra uma foto dos injetores utilizados.

Figura 2.24 – Injetores (RAIL S.R.L.).

Segundo o *datasheet* do fabricante (RAIL S.R.L.), os injetores possuem pressão de trabalho entre 0,5 e 2 Bar, usuais em sistemas de GNV/GLP de uso automotivo. Porém, testes experimentais provaram que os injetores conseguem manter a vedação e operar normalmente em pressões de até 8 Bar.

Os injetores possuem um orifício calibrado que pode variar de 1,00 a 3,25 mm, modificando sua vazão. Atualmente os injetores estão utilizando o orifício de 1,00 mm.

2.4.2. Inversor de frequência

Para o controle da rotação da bomba de óleo do sistema de lubrificação e arrefecimento do turbocompressor, utilizou-se um inversor de frequência modelo CFW300, fabricado pela WEG.

O inversor selecionado possui alimentação 220VAC monofásica e é capaz de alimentar um motor trifásico 220VAC de até 1CV de potência. O inversor também possui uma entrada analógica, permitindo que o mesmo seja controlado pela interface de aquisição e controle.

Para reduzir o ruído elétrico que o inversor devolve para a rede, utilizou-se um filtro de EMI 22-RF010-AL, da fabricante Allen-Bradley.

A Figura 2.25 mostra a instalação do inversor e seu filtro.

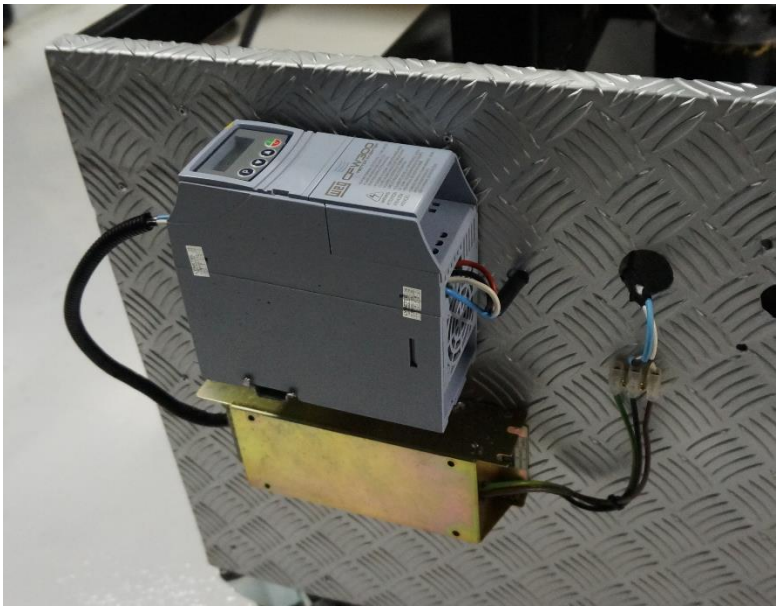


Figura 2.25 – Inversor de frequência e filtro de harmônicos (Elaboração própria).

2.5. SOFTWARE DE CONTROLE

O software de controle é responsável por mostrar em tempo real ao operador os dados obtidos pela interface de aquisição e controle e também permitir que o operador comande a bancada experimental. O software permite ao operador salvar os dados dos sensores em tempo real para um arquivo de forma contínua ou durante um intervalo de tempo. A calibração dos sensores é carregada no software através de um arquivo de texto, permitindo que se utilizem sensores diferentes aos deste trabalho sem a necessidade de alterar o código fonte do software.

O software foi desenvolvido usando a linguagem C# e necessita de um computador com sistema operacional Windows para ser executado. Seu código fonte é disponibilizado no Anexo V deste trabalho.

Assim que executado, o software apresenta sua tela principal, como vista na Figura 2.26.

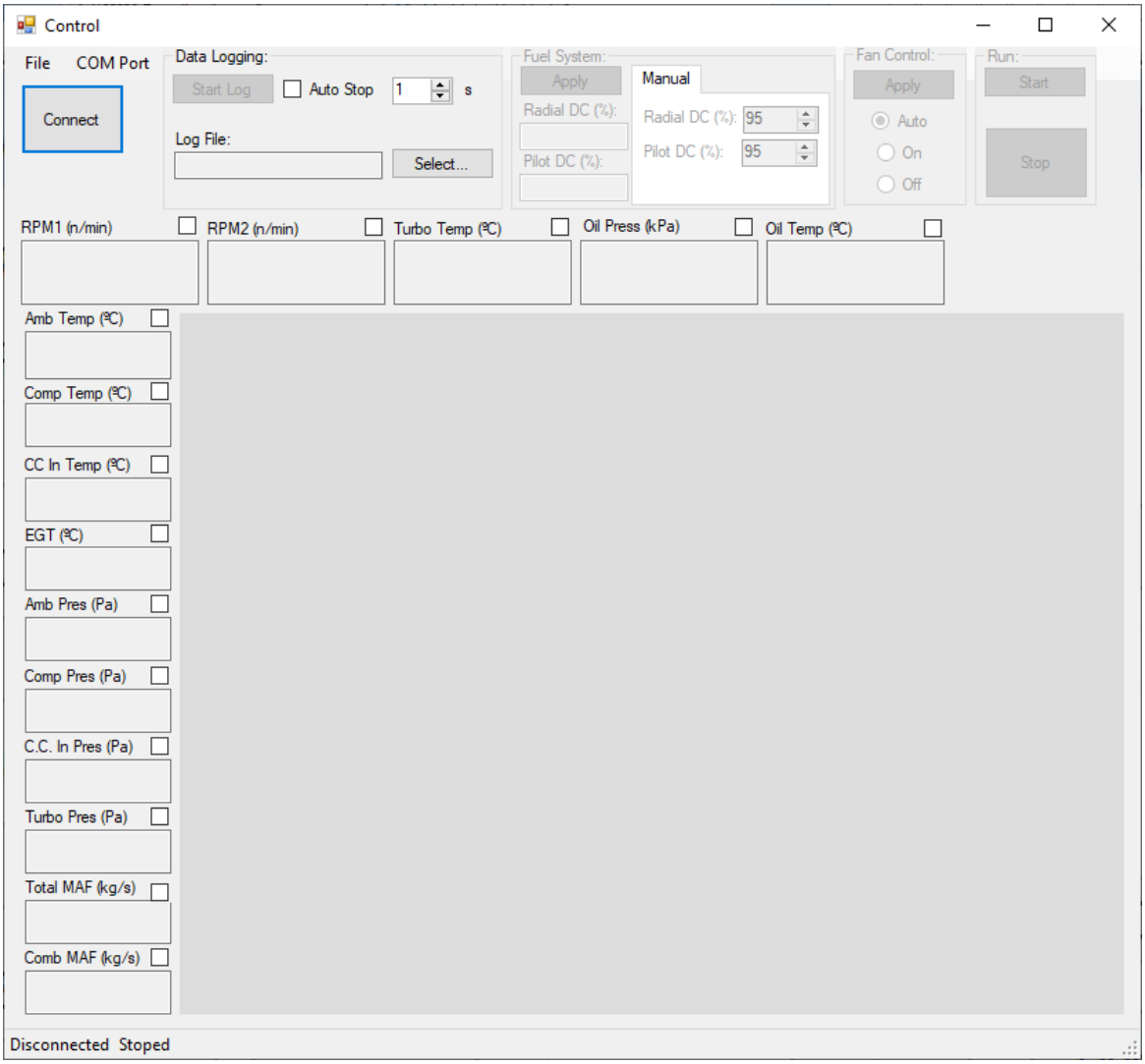


Figura 2.26 – Tela principal (Elaboração própria).

Nesta tela são apresentadas as principais opções para operação da bancada experimental assim como a leitura de todos os sensores conectados na interface de aquisição. A parte superior da tela apresenta os menus e opções para comandar a bancada enquanto a parte inferior é dedicada para visualização dos dados.

Os dados são representados numericamente em seus respectivos campos e caso o *checkbox* deste seja selecionado, os dados deste sensor passam a ser mostrados também através de gráficos de linha.

A parte de controle da tela pode ser dividida em cinco grupos distintos.

O primeiro grupo é composto pelos menus “File” e “COM port” e pelo botão “Connect/Disconnect”. Dentro do menu “File” é possível selecionar o arquivo de calibração dos sensores, abrir o painel de controle manual da bancada, apresentado em detalhes na seção

2.5.1, e exibir a tela de informações do software. Já o menu “COM Port” permite selecionar a porta COM do computador a qual a interface de aquisição e controle está conectada, o software salva a última porta no qual foi conectado, evitando ter de realizar esta operação sempre que o mesmo for executado. O botão “Connect/Disconnect”, como o próprio nome sugere, é utilizada para conectar e desconectar da interface de aquisição e controle. Caso o arquivo de calibração não tenha sido selecionado através da opção “Sensor Calibration Data”, contida no menu “File”, uma tela se abrirá para selecionar o mesmo quando o software tentar conexão com a interface. Detalhes de como editar o arquivo contendo a calibração dos sensores serão discutidos na seção 2.5.2.

O segundo grupo é referente as opções de “Data Loggin”. Dentro deste grupo pode-se iniciar ou parar o log dos dados de forma manual assim como habilitar a parada automática do log e definir o seu intervalo. Também é possível selecionar o arquivo no qual será salvo o log. Toda vez que o log for iniciado, um novo cabeçalho é escrito no arquivo. Caso o arquivo selecionado já exista, o log irá continuar ao final do mesmo e nenhum dado será sobrescrito. O arquivo salvo é um arquivo de texto do tipo CSV que utiliza o caractere “;” como separador. A identificação de cada coluna é apresentada no cabeçalho do log.

O terceiro grupo é composto pelo controle de combustível da bancada experimental. O ciclo de trabalho dos injetores é definido pelas caixas de texto a direita e o botão “Apply” envia estes valores para a interface de aquisição e controle. Os campos abaixo do botão informam o ciclo de trabalho atual dos injetores. O valor máximo que pode ser inserido para o ciclo de trabalho é de 95.

O quarto grupo possui o controle para operação do ventilador do trocador de calor do óleo. As opções disponíveis permitem selecionar o controle automático, onde o ventilador será acionado quando o óleo atingir 110ºC e desligado quando o óleo atingir a marca de 95ºC, manter o ventilador ligado de forma constante ou então desligado.

O último grupo apresenta o controle para partida e parada da bancada experimental. O botão “Start” inicia a sequência de partida enquanto o botão “Stop” é utilizado para parar a operação da bancada. Mais detalhes da sequência de partida são discutidos na secção 2.5.3.

2.5.1. Painel de controle manual

O painel de controle manual permite ao operador comandar de forma manual todas as funções da bancada experimental.

O painel só pode ser acessado com a interface de aquisição e controle conectada e com a bancada experimental parada. Caso a rotina de partida automática seja utilizada para colocar a bancada em operação, o painel não estará mais disponível.

A Figura 2.27 mostra o a tela do painel de controle manual.

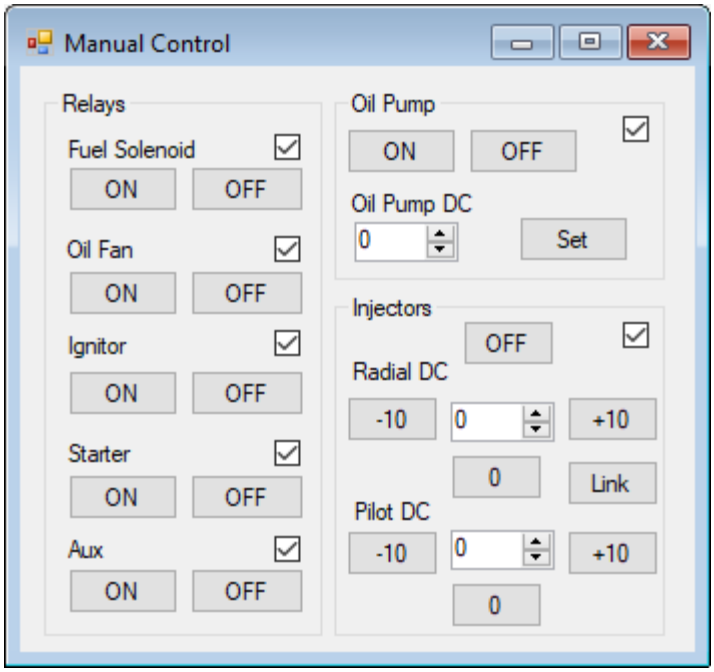


Figura 2.27 – Tela do painel de controle manual (Elaboração própria).

Deve-se selecionar o *checkbox* referente ao campo que se deseja comandar para que o mesmo seja habilitado. Ao clicar em qualquer botão do painel de controle, o comando é enviado instantaneamente a interface de aquisição e controle.

É necessária uma atenção especial ao operar a bancada utilizando o painel de controle manual pois existe o risco de danos a bancada e também risco ao operador.

2.5.2. Arquivo de Calibração dos Sensores

Para permitir que os sensores utilizados sejam trocados por outros, a curva de calibração desses é carregada pelo software de controle através de um arquivo de texto.

O arquivo possui um cabeçalho descrevendo o arquivo, em seguida estão as equações para calibração dos sensores. É importante respeitar o tamanho do cabeçalho de 7 linhas. Também é importante garantir que estejam presentes as dez equações de calibração necessárias para a operação do software. Caso algum sensor não seja utilizado, recomenda-se que sua equação seja substituída por “0”.

A Figura 2.28 apresenta um exemplo de arquivo de calibração.

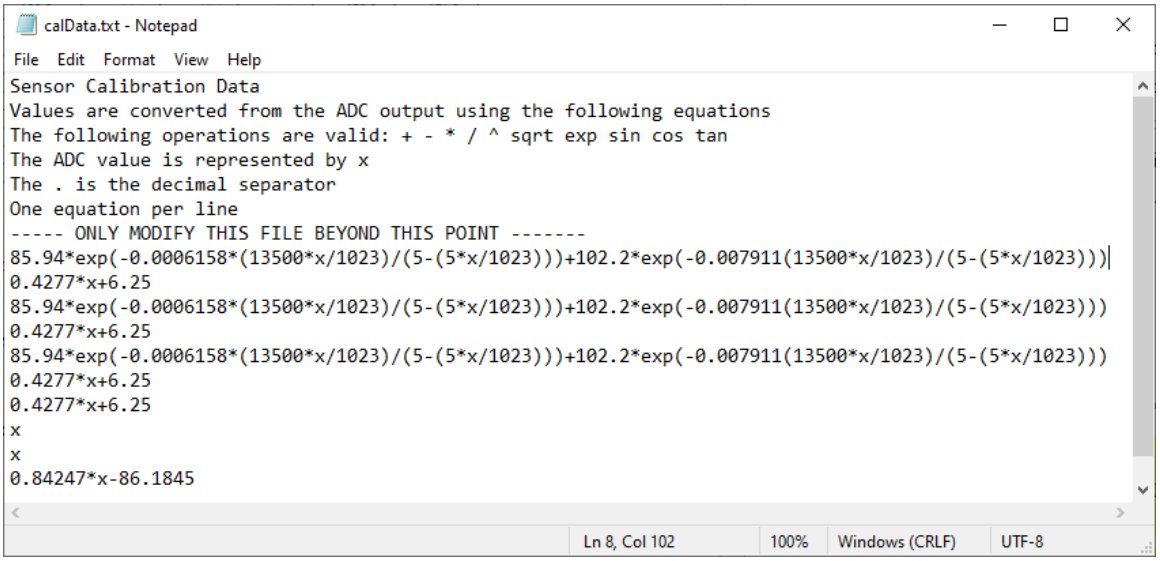


Figura 2.28 – Arquivo de calibração (Elaboração própria).

2.5.3. Rotina de partida

A rotina de partida desenvolvida tem como objetivo iniciar o processo de combustão de forma segura e acelerar o turbocompressor até o ponto em que o processo se estabilize.

A rotina de partida desenvolvida segue o fluxograma apresentado na Figura 2.29.

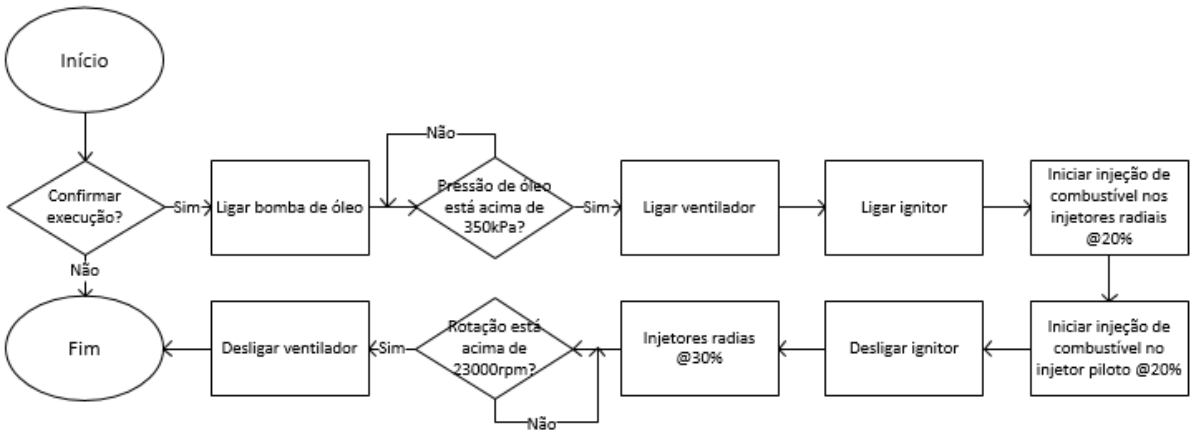


Figura 2.29 – Fluxograma de partida (Elaboração própria).

3. BANCADA EXPERIMENTAL

A bancada experimental utilizada é a mesma do trabalho de Ferreira (2007), porém, devido a nenhum trabalho ter sido desenvolvido nesta durante alguns anos, a mesma se encontrava em péssimo estado de conservação e com muitos componentes faltando. A Figura 3.1 mostra fotos da bancada no estado que se encontrava no início deste trabalho.

Figura 3.1 – Fotos da bancada no início (Elaboração própria).

A estrutura sobre a qual eram montados os diversos componentes da bancada se encontrava com oxidação em diversos pontos e algumas soldas apresentavam trincas. O sistema elétrico havia sido removido da bancada, diversas linhas de óleo apresentavam rachaduras e vazamentos, e todas as solenoides de controle de combustível e linhas de combustível haviam sido removidas. Também não havia mais nenhum sensor para a instrumentação da bancada.

A estrutura da bancada foi reparada e repintada para evitar novos pontos de oxidação. Na Figura 3.3 e Figura 3.3 é possível ver como a bancada se encontra atualmente.

Figura 3.2 – Fotos bancada atualmente (Elaboração própria).

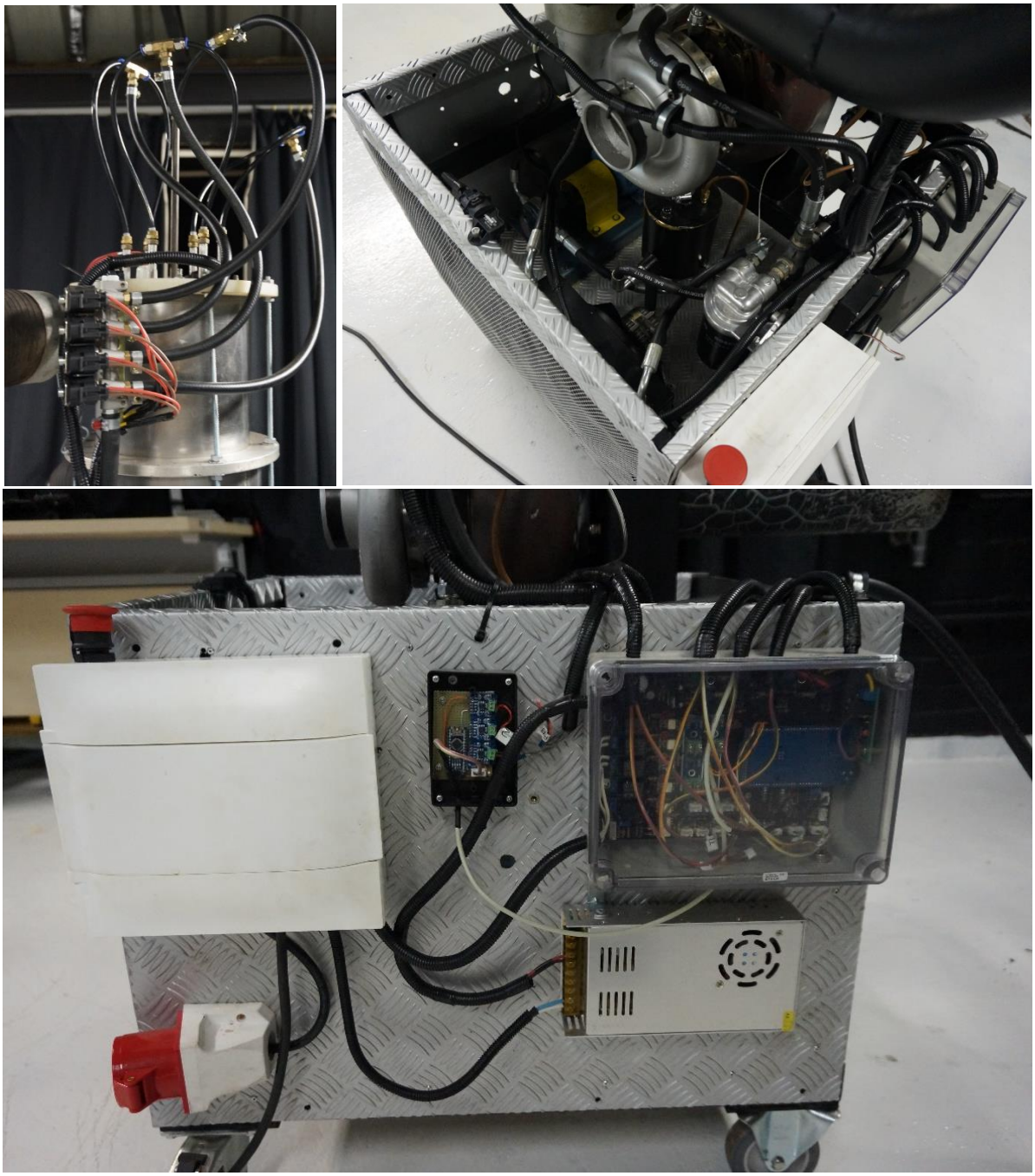


Figura 3.3 – Fotos bancada atualmente (detalhes) (Elaboração própria).

Nas próximas subseções são apresentados os principais componentes da bancada.

3.1. TURBOCOMPRESSOR

O conjunto turbocompressor é o mesmo utilizado por Ferreira (2007). O conjunto apresenta pequenos danos em uma das pás do compressor, porém o seu estado geral é muito bom. O mesmo foi desmontado e limpo para garantir a operação segura da bancada.

O conjunto é formado por uma turbina e um compressor radial conectados através de um eixo apoiado em mancais de deslizamento. A turbina recebe os gases quentes provenientes da câmara de combustão e converte parte da energia desses em trabalho de eixo, que por sua vez aciona o compressor que irá comprimir o ar para alimentar a câmara de combustão. Na Figura 3.4 temos uma vista em corte de um conjunto similar ao utilizado.

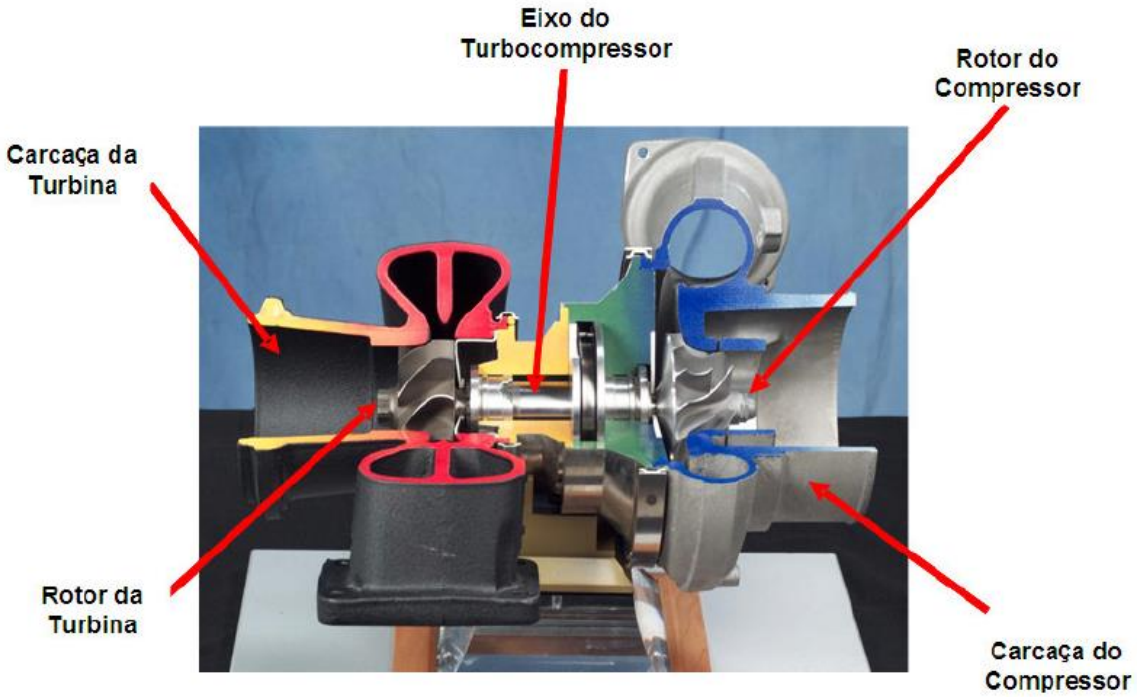


Figura 3.4 – Vista em corte de um turbocompressor (FERREIRA, 2007).

O turbocompressor utilizado é o TV-7704, fabricado pela Garret, e usualmente utilizado para sobrealimentar motores de combustão interna utilizados em caminhões, como o Scania 142. Da forma como este conjunto é construído, não é possível adicionar nenhuma carga externa ao eixo, portanto todo o trabalho realizado pela turbina é consumido pelo compressor. Na Figura 3.5 temos o mapa do compressor.

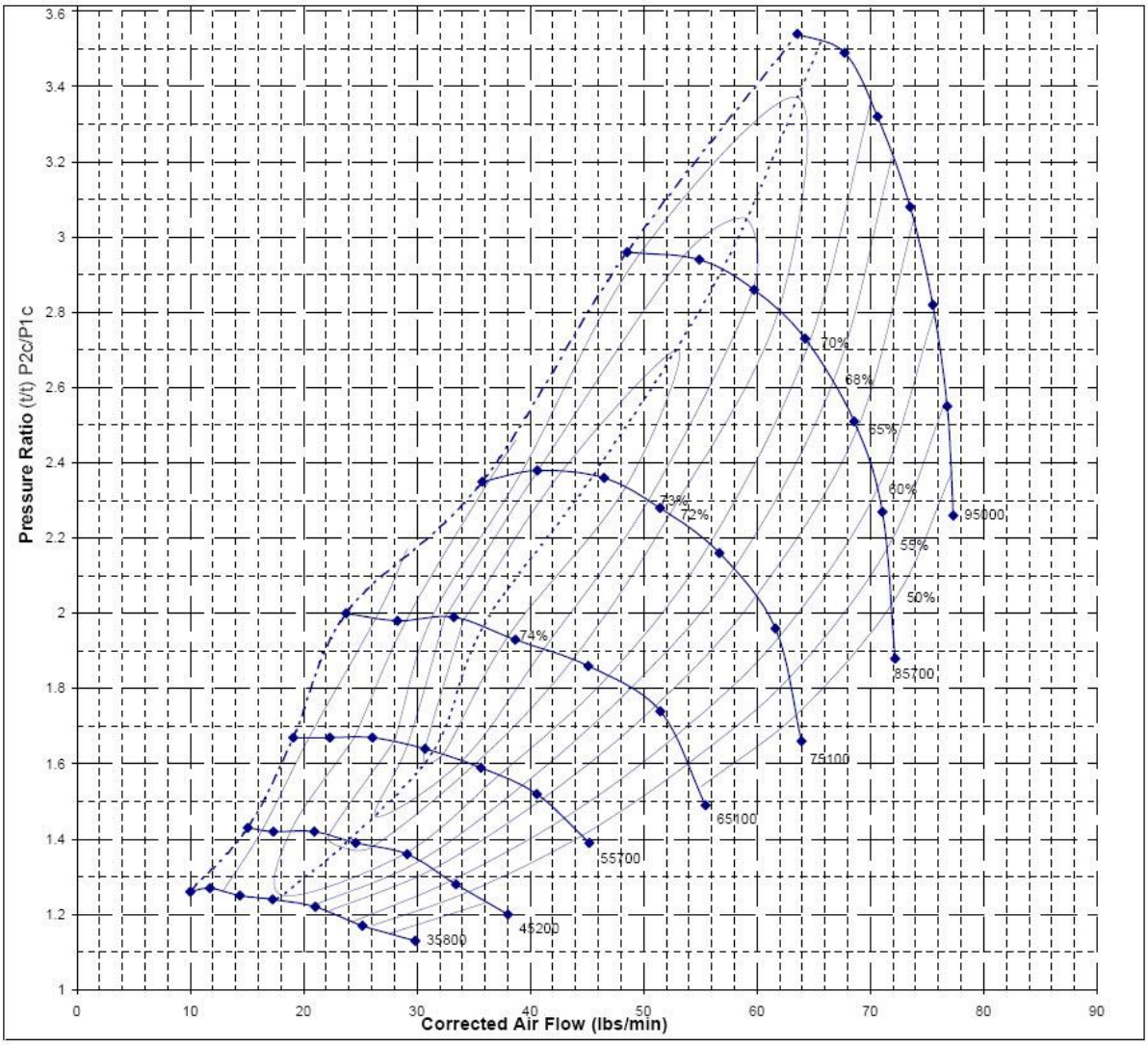


Figura 3.5 – Mapa do compressor (FERREIRA, 2007).

3.2. CÂMARA DE COMBUSTÃO

A câmara de combustão utilizada na bancada experimental é aquela desenvolvida no trabalho de Ferreira (2007). A câmara de combustão não apresentava nenhum componente danificado, sendo esta somente desmontada para limpeza e remontada conforme as informações contidas no trabalho. Na Figura 3.6 temos uma vista em corte da câmara de combustão.

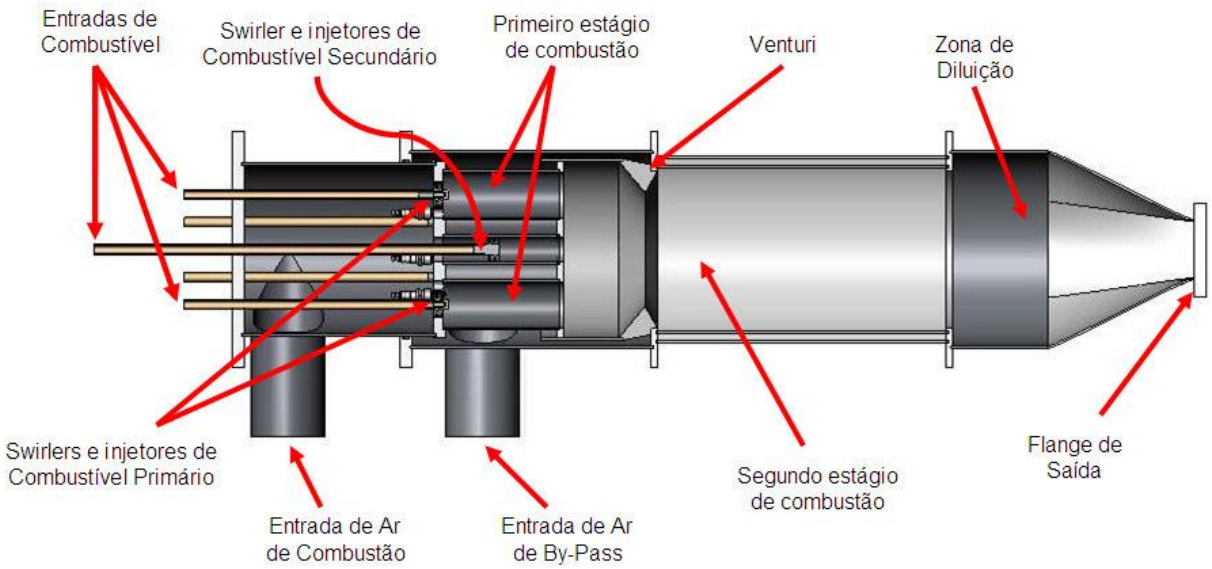


Figura 3.6 – Câmara de combustão (FERREIRA, 2007).

3.3. SISTEMA ELÉTRICO

O sistema elétrico da bancada foi totalmente refeito utilizando fios de bitolas adequadas e seguindo as recomendações da ABNT – NBR5410:2004. Todos os chicotes utilizados na bancada foram cobertos com conduítes corrugados de uso automotivo e retardantes a chama.

A bancada é alimentada por uma única tomada industrial do tipo “Steck 3P + N + T” de 32A. Tal tomada fornece tanto a alimentação trifásica 380VAC necessária para alimentar o ventilador centrífugo utilizado para a partida da bancada quanto a alimentação monofásica 220VAC necessária para os demais componentes da bancada.

A bancada conta com dois disjuntores para proteção dos seus componentes, o primeiro é utilizado como chave geral, já o segundo energiza somente os componentes do sistema de lubrificação e ignição.

Para alimentação dos componentes que necessitam de 12VDC, a bancada conta com uma fonte chaveada capaz de fornecer até 30A de forma contínua.

Foi instalado na bancada um botão de emergência que desliga totalmente a mesma da energia. Tal botão foi posicionado de forma a ser facilmente acessível.

Na Figura 3.7 temos um desenho esquemático de todo o sistema elétrico da bancada.

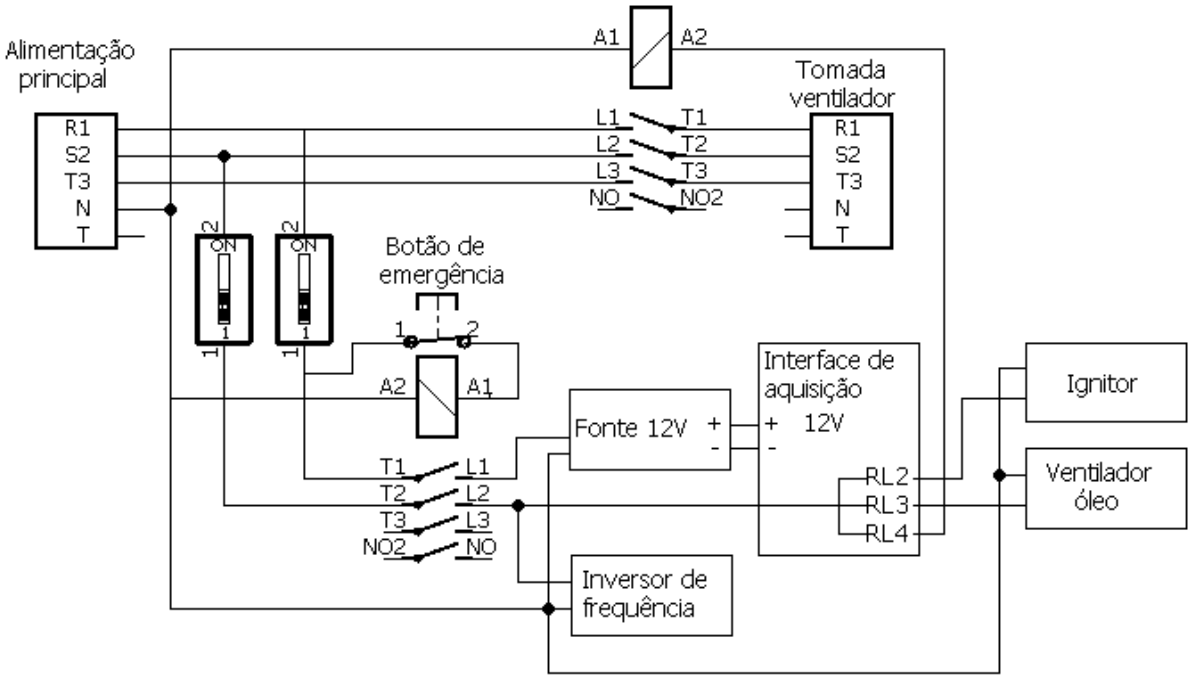


Figura 3.7 – Esquema elétrico da bancada (Elaboração própria).

3.4. SISTEMA DE LUBRIFICAÇÃO E ARREFECIMENTO

O sistema de lubrificação e arrefecimento da bancada é responsável por manter um fluxo de óleo lubrificante constante sobre os mancais de deslizamento do turbocompressor. Este mesmo óleo é usado para refrigerar o conjunto central do turbocompressor, evitando que o mesmo superaqueça durante a operação.

O sistema de lubrificação consiste em uma bomba de engrenagens acoplada a um motor trifásico de 0,5CV, um trocador de calor, um filtro de óleo, uma válvula reguladora de pressão e um reservatório.

A Figura 3.8 mostra um diagrama do sistema.

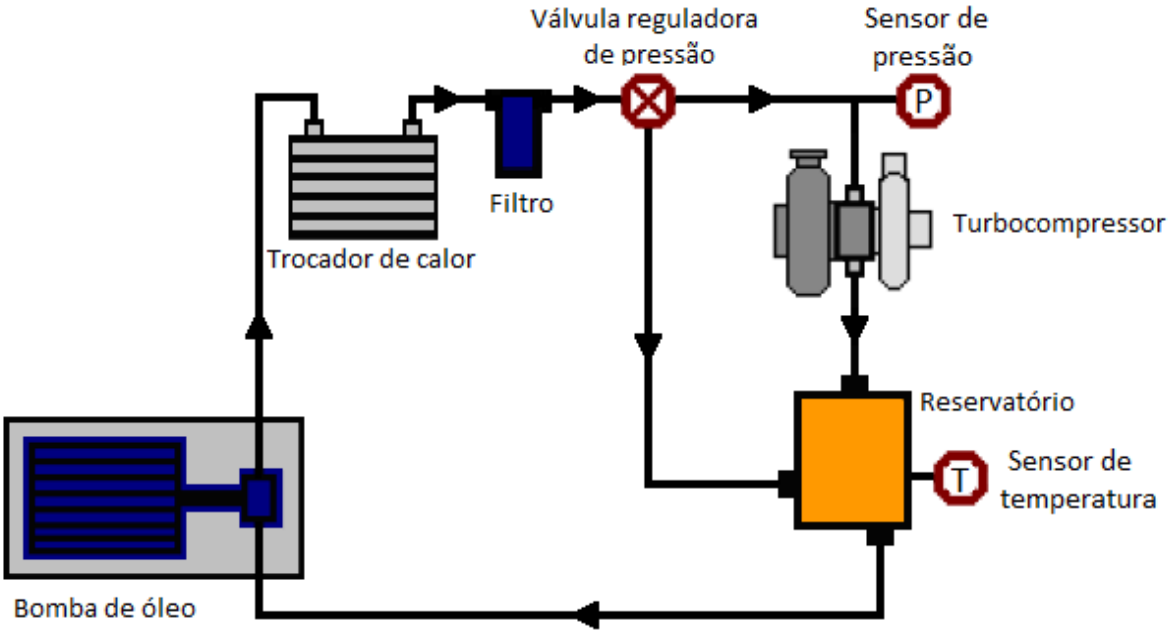


Figura 3.8 – Diagrama do sistema de lubrificação e arrefecimento (Elaboração própria).

A rotação da bomba, e consequentemente sua vazão, são controlados por um inversor de frequência conectado a interface de controle e aquisição. Tal controle é importante pois a viscosidade do óleo muda consideravelmente da temperatura ambiente para a temperatura de operação, fazendo com que, se mantida constante a vazão da bomba, tem-se um excesso de pressão de óleo enquanto o mesmo está frio ou uma baixa pressão quando o óleo está quente. A válvula reguladora de pressão utilizada não é suficiente para controlar a pressão do óleo frio, portanto, essa só deve ser ajustada com o óleo em temperatura de operação para que a pressão na linha se estabilize próximo a 450 kPa.

O trocador de calor conta com um ventilador para aumentar sua capacidade, controlado pelo software de controle de forma a manter o óleo na faixa de temperatura ideal, que é de 90 a 120°C. Operar a bancada com cargas ou rotação elevadas enquanto o óleo não atingir a temperatura de trabalho pode levar ao desgaste prematuro do mancal.

O filtro utilizado é de aplicação automotiva, utilizada em veículo VW Gol equipados com motorização AP.

O óleo a ser utilizado deve atender a especificação API CI-4/SL ou superior na viscosidade 15W-40.

3.5. SISTEMA DE ALIMENTAÇÃO DE COMBUSTÍVEL

O sistema de alimentação de combustível da bancada experimental é bem simples, consistindo em quatro solenoides para controle da vazão montadas sobre uma flauta de distribuição, sete injetores localizados dentro da câmara de combustão, um cilindro de GLP e um manômetro de pressão.

O cilindro de GLP é conectado diretamente na flauta de distribuição, e a saída de três das quatro solenoides de controle, montadas sobre a flauta, é dividida em duas e cada uma dessas saída é conectada a um dos injetores de combustível radial. A quarta solenoide é conectada ao injetor central da câmara de combustão. A Figura 3.9 apresenta uma foto identificando as linhas de combustível.

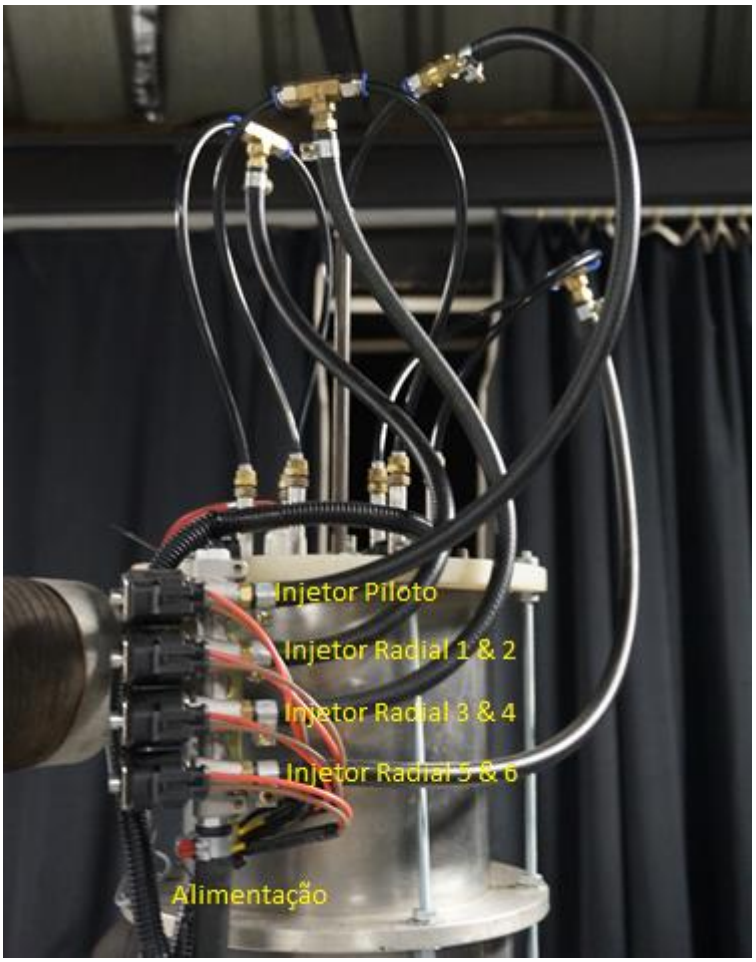


Figura 3.9 – Identificação das linhas de combustível (Elaboração própria).

Na linha pressão que alimenta a flauta de distribuição também é montado um manômetro.

3.6. VÁLVULAS DE CONTROLE DE AR

Para controlar a razão de by-pass da câmara de combustão são utilizadas duas válvulas borboleta, como mostra a Figura 3.10.

Figura 3.10 – Válvulas de controle de ar (Elaboração própria).

Para que o processo de combustão seja estável, é necessário que a válvula de linha de by-pass esteja totalmente aberta e que a válvula da linha de ar de combustão esteja posicionada entre as posições 3 e 6.

3.7. SISTEMA DE PARTIDA

O sistema de partida conta com um ventilador centrífugo, um centelhador e seis velas de ignição.

O ventilador é utilizado para gerar um fluxo de ar inicial no combustor para que juntamente com o combustível disponibilizado pelos injetores, o processo de combustão se inicie. O centalhador e as velas de ignição fornecem a energia inicial para inflamar o combustível. Após

o processo de combustão se estabilizar e a rotação do turbocompressor for suficiente para que o processo se mantenha estável, o ventilador e o centelhador devem ser desligados.

O centelhador utilizado foi um centelhador para fogões com seis saídas, já as velas de ignição são velas utilizadas para o acendimento de fornos a gás.

O ventilador centrifugo utilizado é apresentado na Figura 3.11 e possui um motor trifásico de 4CV.

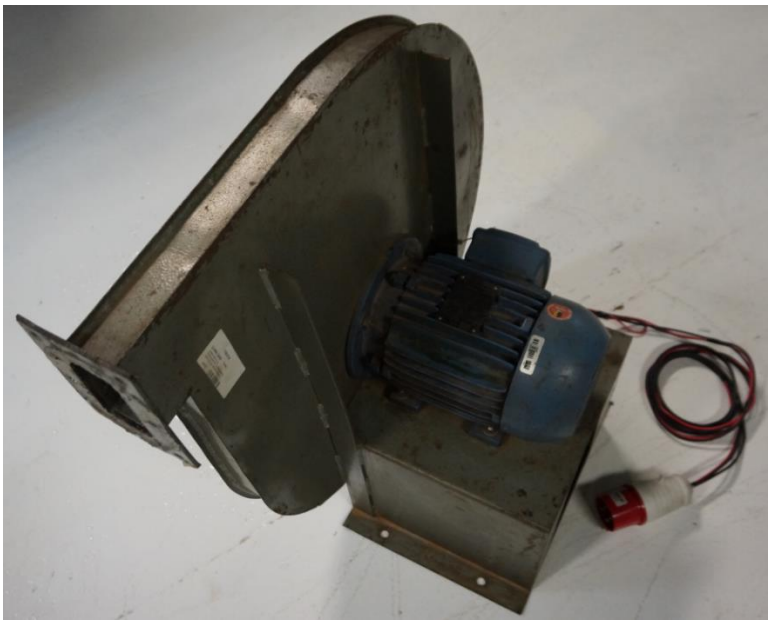


Figura 3.11 – Foto do ventilador de partida (Elaboração própria).

3.8. INSTRUMENTAÇÃO

A instrumentação da bancada experimental utiliza os sensores descritos na seção 2.3.

Tomadas de pressão e temperatura foram instaladas nos pontos 1 a 4 do diagrama apresentado na Figura 3.12. No ponto 5, foi instalada somente uma tomada de temperatura, visto que a pressão neste ponto é igual a pressão do ponto 1.

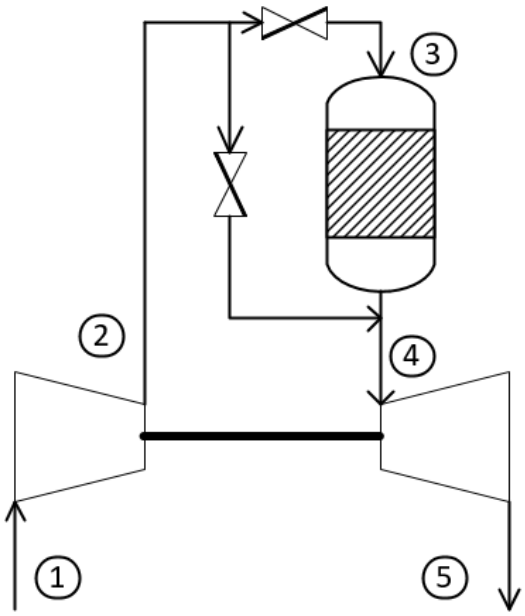


Figura 3.12 – Diagrama de instrumentação da bancada (Elaboração própria).

Para a medida de temperatura nos pontos 4 e 5 foram utilizados os termopares tipo K.

O sensor de rotação foi montado na entrada do compressor, de forma que ficasse o mais próximo possível das pás do mesmo. A fim de melhorar as condições de operação para o sensor, uma das pás do compressor foi pintada de branco, enquanto as restantes foram pintadas de preto, assim o sensor detecta somente um pulso a cada rotação do compressor, fazendo com o que o sinal de rotação medido seja mais estável. A Figura 3.13 mostra a montagem do sensor bem como o detalhe das pás do compressor.

Figura 3.13 – Detalhe do sensor de rotação e compressor pintado (Elaboração própria).

A bancada também conta com uma tomada de pressão de óleo logo após a válvula reguladora de pressão e com uma tomada de temperatura no reservatório de óleo.

4. TESTES E RESULTADOS EXPERIMENTAIS

Nesta seção, são apresentados os testes realizados com o hardware e software desenvolvidos neste trabalho. Tais testes tem como objetivo validar o sistema desenvolvido bem como avaliar sua performance em situações reais de operação.

4.1. REGULADOR DE TENSÃO

É importante que o regulador de tensão mantenha sua saída constante de forma a não influenciar na leitura dos dados dos sensores analógicos. Devido às características construtivas da interface de aquisição e controle, a mesma praticamente não varia seu consumo enquanto em operação. Dessa forma, optou-se por caracterizar o regulador somente em pontos estáticos de carga, desconsiderando os regimes transientes. O consumo de corrente da interface de aquisição e controle em operação é de aproximadamente 2,4A, já considerando todos os sensores utilizados neste trabalho.

Para medir a performance do regulador, realizou-se cinco medições de sua tensão de saída em cinco pontos de carga diferentes utilizando-se um osciloscópio Tektronix TBS1052B, e definiu-se uma janela de 60 segundos para cada medição. A carga foi imposta no regulador usando uma carga eletrônica BK Precision 8500B. Para garantir que o aquecimento do regulador não influenciasse nos resultados, deixou-se o mesmo conectado a carga por tempo suficiente até que se atingisse o equilíbrio térmico no dissipador de calor do transistor de potência.

Os gráficos apresentados nas Figura 4.1 a Figura 4.5 mostram os resultados dos testes.

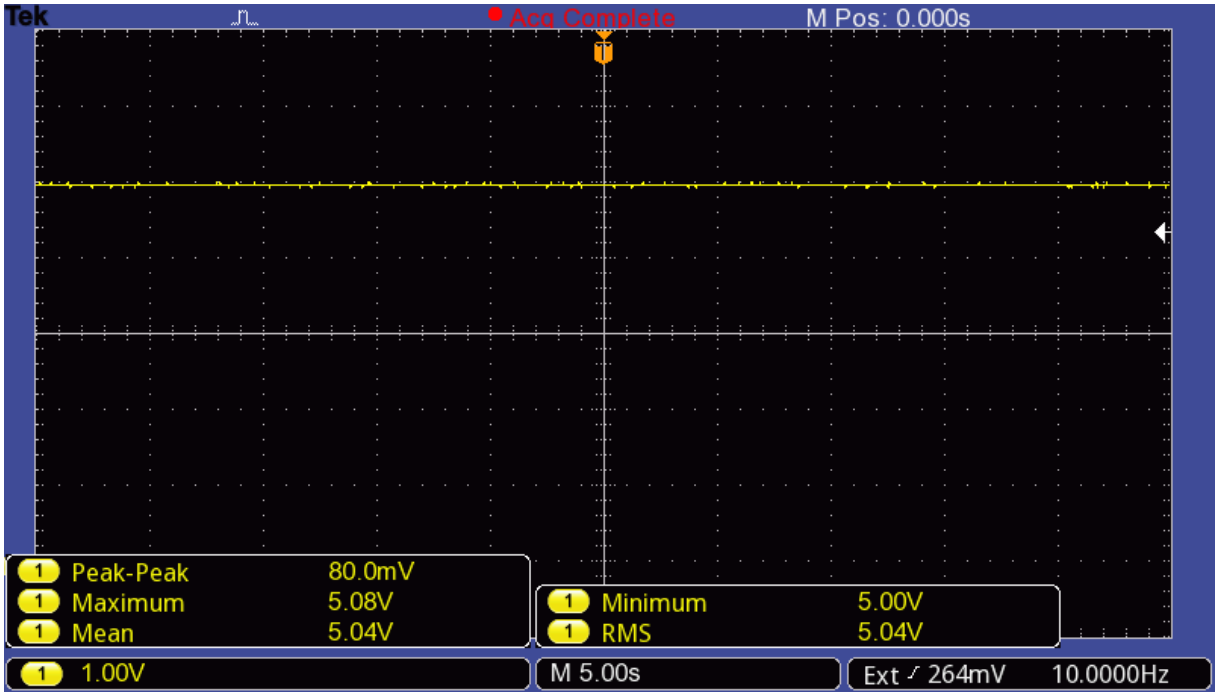


Figura 4.1 – Teste sem carga (Elaboração própria).

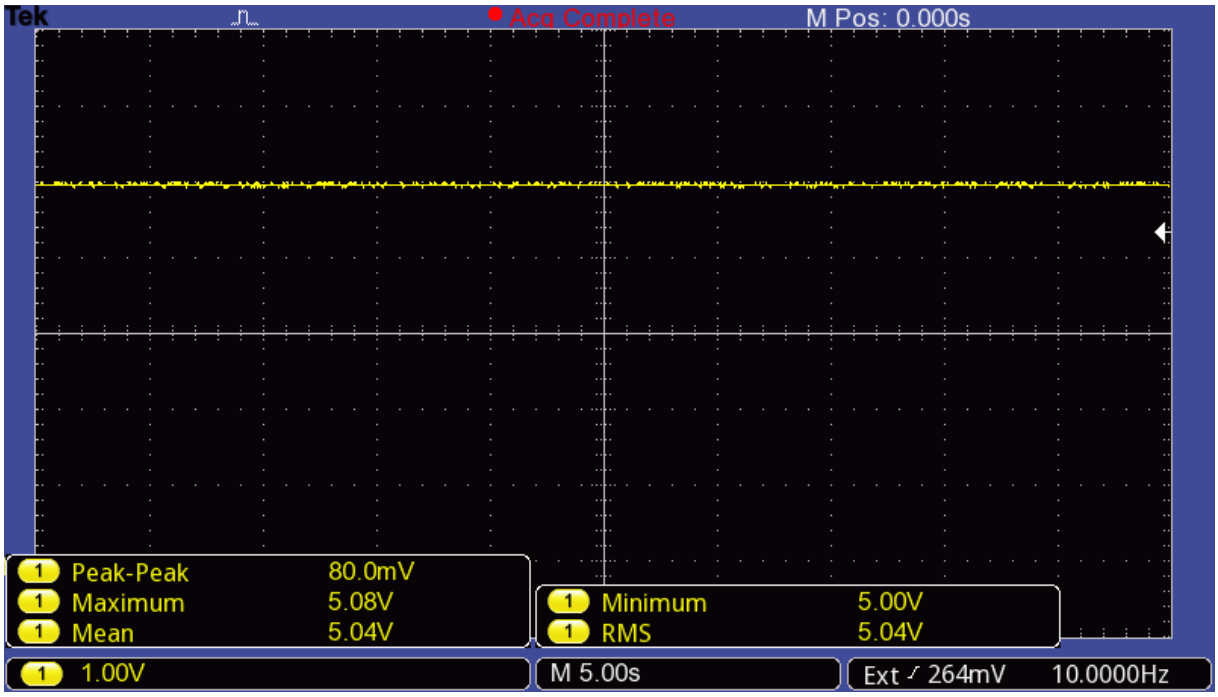


Figura 4.2 – Teste com carga de 1A (Elaboração própria).

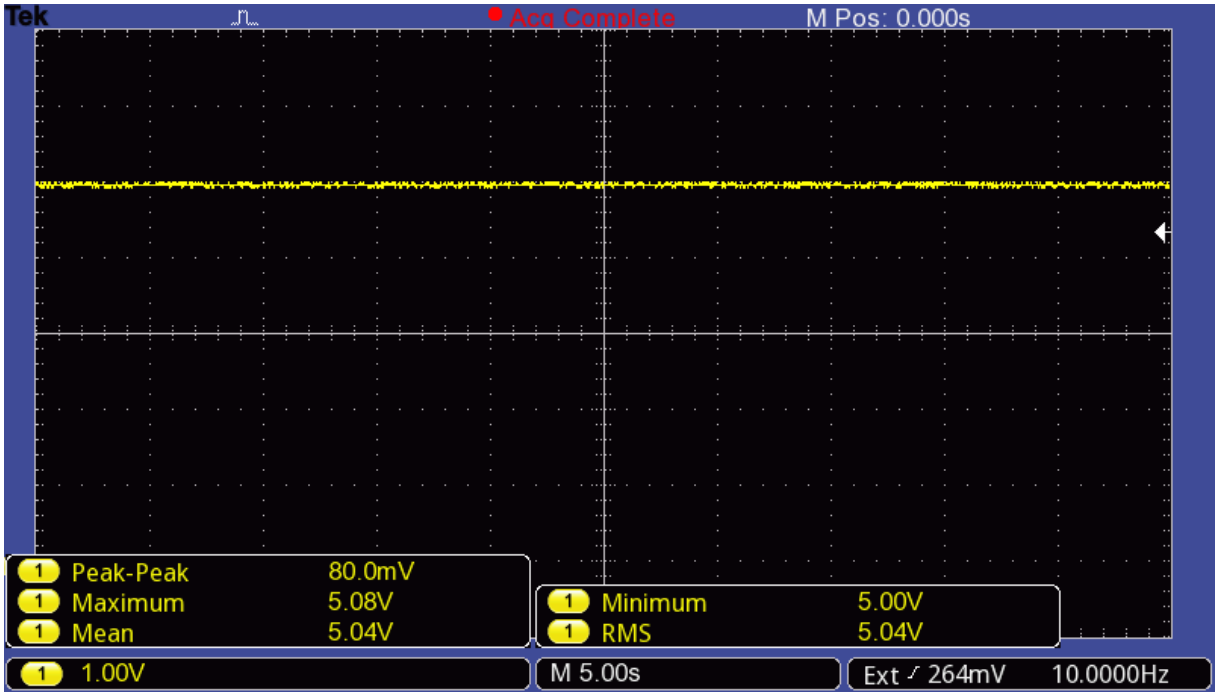


Figura 4.3 – Teste com carga de 2A (Elaboração própria).

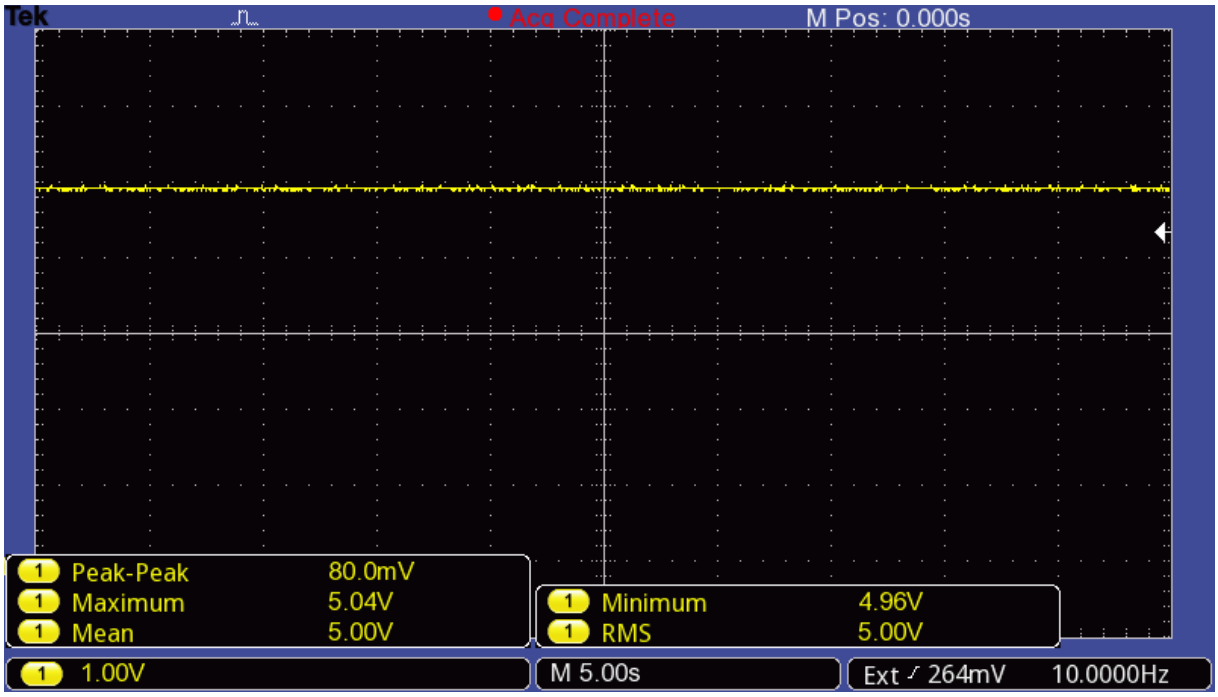


Figura 4.4 – Teste com carga de 3A (Elaboração própria).

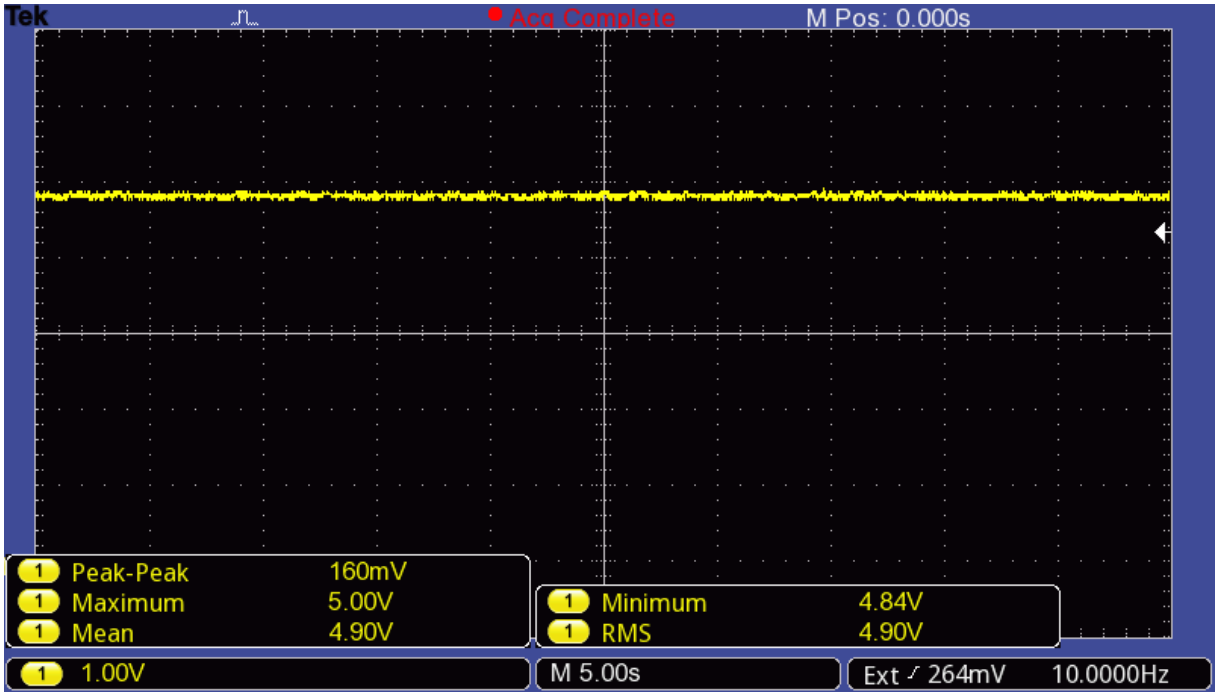


Figura 4.5 – Teste com carga de 4A (Elaboração própria).

Como observado, a variação de tensão é mínima independentemente da carga aplicada, exceto para a carga de 4A, visto que está já está no limite para o qual o regulador foi concebido. Contudo, mesmo havendo uma pequena queda de tensão, a mesma se manteve praticamente constante nesse novo patamar.

Em todos os testes, a variação máxima na tensão fornecida pelo regulador foi inferior a 160mV. Desconsiderando o resultado obtido para a carga de 4A, visto que está é uma situação atípica de operação, a variação máxima na tensão é de 80mV.

4.2. ENTRADAS ANALÓGICAS

Para validar que as entradas de sinal analógico estão apresentando valores reais, utilizou-se um divisor de tensão obtido com o uso de um potenciômetro multivolta para variar a tensão medida. Para alimentar o divisor de tensão, utilizou-se o próprio regulador de tensão da interface de aquisição e controle.

Para medir a tensão real na porta, utilizou-se um multímetro de bancada Fluke 8840A. A fim de verificar se existe histerese presente nas medições, essas foram realizadas de forma crescente e decrescente.

Devido ao conversor analógico-digital possuir uma resolução de 10 bit, a tensão medida varia em passos de 4,887mV, aproximadamente. Desta forma, os valores medidos que se encontram a até 4,887mV do valor real foram considerados como erro zero.

A saída do conversor analógico digital foi enviada diretamente através de uma porta UART para o computador e posteriormente convertido em tensão usando-se a Equação (4.1).

$$V = ADC * \frac{5}{1023} \tag{4.1}$$

Foram realizadas medições em dezenove pontos, variando em intervalos iguais, de 0,5 a 5V. Os resultados obtidos são apresentados na Tabela 4.1.

Tabela 4.1 – Resultado dos testes do conversor analógico-digital (Elaboração própria).

Valor Real (V)	ADC	Valor Medido (V)	Diferença (mV)	Sentido da Medição
0,5012	103	0,503421	-2,221	Crescente
1,0267	210	1,026393	0,307	Crescente
1,5101	309	1,510264	-0,163	Crescente
2,0002	410	2,00391	-3,710	Crescente
2,5167	514	2,512219	4,481	Crescente
3,0134	617	3,01564	-2,240	Crescente
3,5223	721	3,523949	-1,649	Crescente
4,0198	822	4,017595	2,204	Crescente
4,5097	922	4,506354	3,346	Crescente
4,9045	1003	4,902248	2,251	Crescente
4,5034	921	4,501466	1,933	Decrescente
3,9978	818	3,998045	-0,244	Decrescente
3,4812	712	3,479961	1,239	Decrescente
3,0001	614	3,000978	-0,877	Decrescente
2,4909	510	2,492669	-1,768	Decrescente
2,0010	409	1,999022	1,977	Decrescente
1,4892	305	1,490714	-1,513	Decrescente
1,0165	208	1,016618	-0,117	Decrescente
0,4976	102	0,498534	-0,933	Decrescente

Observou-se que não existem diferenças significativas em nenhuma das medições, também não foi observada nenhuma histerese no sistema.

4.3. MEDIÇÃO DE ROTAÇÃO

Idealmente, para a validação das medições de rotação realizadas pela interface de aquisição e controle seria necessário o uso de um sistema de medição totalmente independente da mesma. Contudo, devido às altas rotações em que o turbocompressor opera, suas características construtivas e dificuldade de sincronizar dois sistemas de aquisição distintos, tal abordagem é inviável na pratica.

Desta forma, para garantir que os valores reportados pelo software de controle estão minimamente corretos, utilizou-se o sinal de saída do mesmo sensor utilizado pela interface de aquisição e controle conectado a um osciloscópio Tektronix TBS1052B.

Estabilizou-se a rotação do turbocompressor em diversos pontos de operação, com o osciloscópio mediu-se a frequência do sinal do sensor de rotação e com isso calculou-se a rotação do compressor. Comparou-se o valor obtido com o valor reportado pelo software de controle de forma visual e não se observou nenhuma diferença significativa em nenhum dos pontos.

A Figura 4.6 apresenta uma imagem da tela do osciloscópio durante uma das medições.

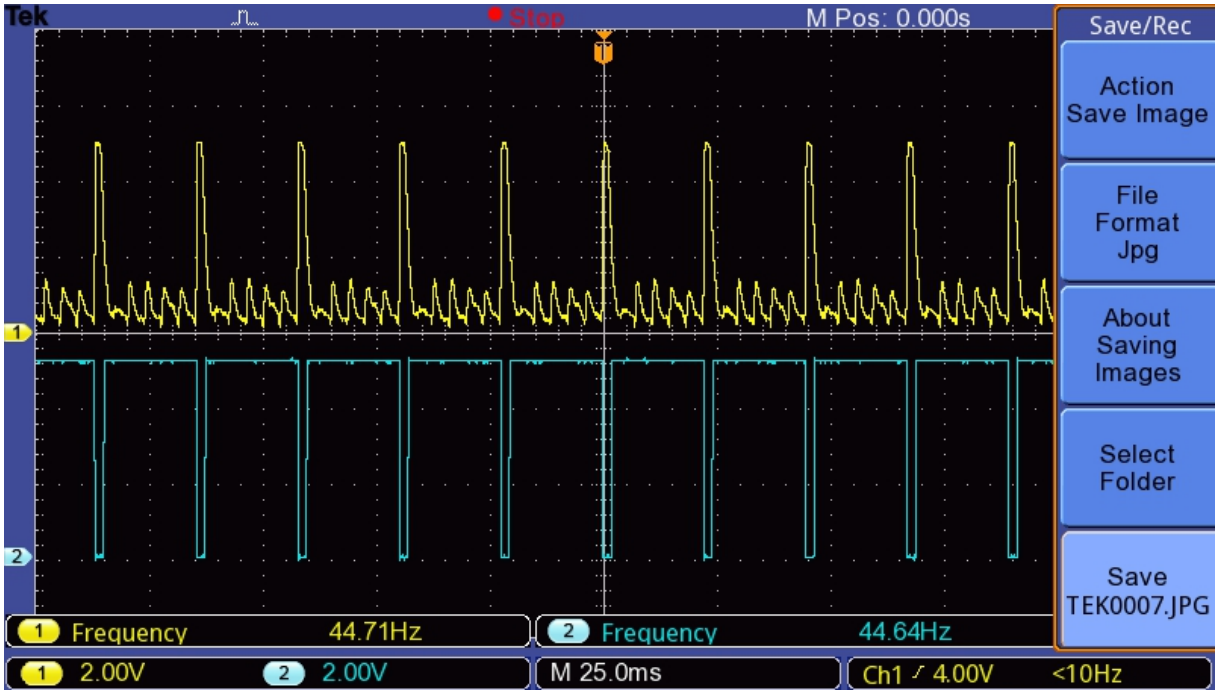


Figura 4.6 – Teste para validação do sensor de rotação (Elaboração própria).

4.4. ROTINA DE PARTIDA

A rotina de partida desenvolvida foi validada realizando-se testes experimentais com a bancada. A Figura 4.7 apresenta um gráfico mostrando todos as etapas realizadas pela rotina até a estabilização do funcionamento da bancada.

O ponto 1 mostra o ponto onde o ventilador centrífugo é ligado, o ponto 2 mostra o momento em que a combustão é iniciada e o ponto 3 mostra o momento em que o turbocompressor começa a ser acelerado. O ponto 4 identifica o ponto em que o sistema atingiu a rotação mínima para se tornar autossustentável e o ventilador centrífugo é desligado e removido da entrada do compressor.

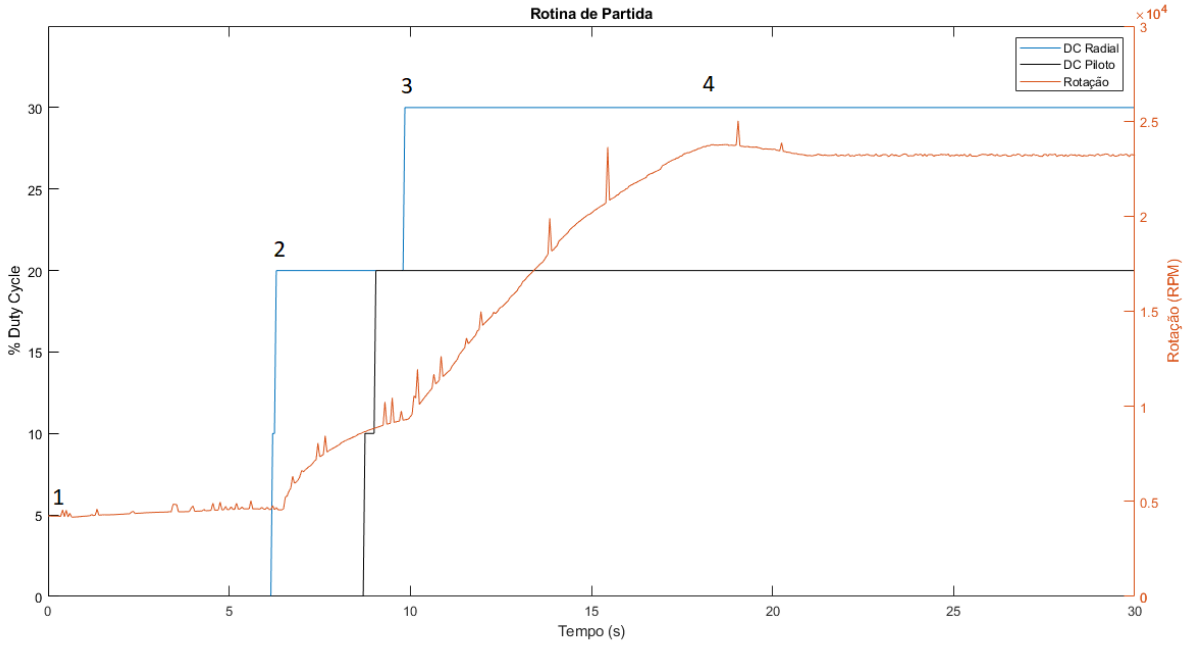


Figura 4.7 – Gráfico de partida (Elaboração própria).

Vale notar que, devido a problemas no inversor de frequência utilizado, o controle da bomba de óleo teve de ser executado de forma manual, diretamente no inversor.

4.5. AQUISIÇÃO DE DADOS

Para validar o sistema de aquisição, os dados dos sensores foram coletados desde a partida da bancada e posteriormente analisados de forma a identificar alguma discrepância nesses.

O sensor de fluxo de massa de ar não foi instalado na bancada experimental pois o mesmo acabou sendo danificado em um dos testes realizados, portanto não há dados do mesmo.

Os termopares utilizados também apresentaram problemas pois o isolamento dos mesmos não resistiu as temperaturas de operação da bancada, fazendo com entrassem em curto com a estrutura da bancada, impedindo que qualquer resultado válido fosse obtido destes.

Iniciou-se a coleta dos dados ainda com a bancada experimental parada e em seguida iniciou-se a rotina de partida. Após o sistema se estabilizar, o mesmo foi mantido neste estado por aproximadamente 12 segundos. Em seguida, acelerou-se o turbocompressor até aproximadamente 48000rpm, onde o mesmo foi mantido operando por mais 15 segundos, aproximadamente. Por fim, executou-se a parada da bancada e aguardou-se até a parada do turbocompressor para se encerrar a coleta dos dados.

Os dados obtidos são apresentados nos gráficos das figuras 4.8 a 4.10, ao final desta seção.

Os dados reportados pelos sensores de pressão e temperatura são condizentes com o que era esperado para os pontos de operação testados. É possível observar que antes da bancada entrar em operação, todas as leituras de pressão são praticamente iguais e levemente superiores a ambiente, devido ao ventilador de partida já estar em operação. Também devido ao aquecimento do ar pelo ventilador de partida, as leituras dos sensores de temperatura localizados nas linhas de ar da bancada reportam valores levemente superiores ao ambiente. Logo após o processo de combustão ser iniciado e o compressor começar a ser acelerado é possível observar que as pressões nas linhas de ar da bancada começam a subir. Quando o processo atinge o equilíbrio e o sistema se estabiliza, a pressão também se mantém constante em todos os pontos. Além disso, é possível observar uma pequena diferença entre a pressão na exaustão do compressor, a pressão na entrada da câmara de combustão e a pressão na entrada da turbina. Esta diferença se deve à perda de carga das válvulas de controle de ar e à perda de carga da câmara de combustão, respectivamente.

Devido à grande quantidade de energia térmica liberada pelo processo de combustão, é possível observar uma rápida elevação da temperatura ambiente, e, consequentemente, as temperaturas nas linhas de ar apresentam o mesmo comportamento, com um delta positivo em relação ao ambiente, devido ao processo de compressão. Este comportamento é observado durante todo o ensaio, visto que a duração do ensaio não foi suficiente para que a temperatura ambiente atingisse um estado de equilíbrio.

Quando o turbocompressor é acelerado, é possível observar que as pressões nas linhas de ar sobem proporcionalmente a rotação. Devido a maior razão de compressão, a perda de carga das

válvulas de controle de ar e da câmara de combustão ficam mais evidentes. Quando a rotação se estabiliza, a pressão volta a se estabilizar. É possível também observar que o delta entre a temperatura ambiente e a temperatura das linhas de ar aumenta, devido a maior razão de compressão.

Quando o processo de combustão é extinto, é possível observar uma queda praticamente instantânea na rotação do turbocompressor e na pressão de ar do sistema.

Na medição de rotação é possível observar alguns pontos de ruído. Porém, esses pontos são facilmente identificados e podem ser removidos em um posterior tratamento dos dados. Devido a forma que o ruído se apresenta, é possível a implementação de um filtro no software de controle ou até mesmo no próprio software embarcado na interface de aquisição e controle.

A medida de pressão de óleo se mantém praticamente constante durante todo o teste, visto que o óleo já estava na temperatura de operação no início do teste.

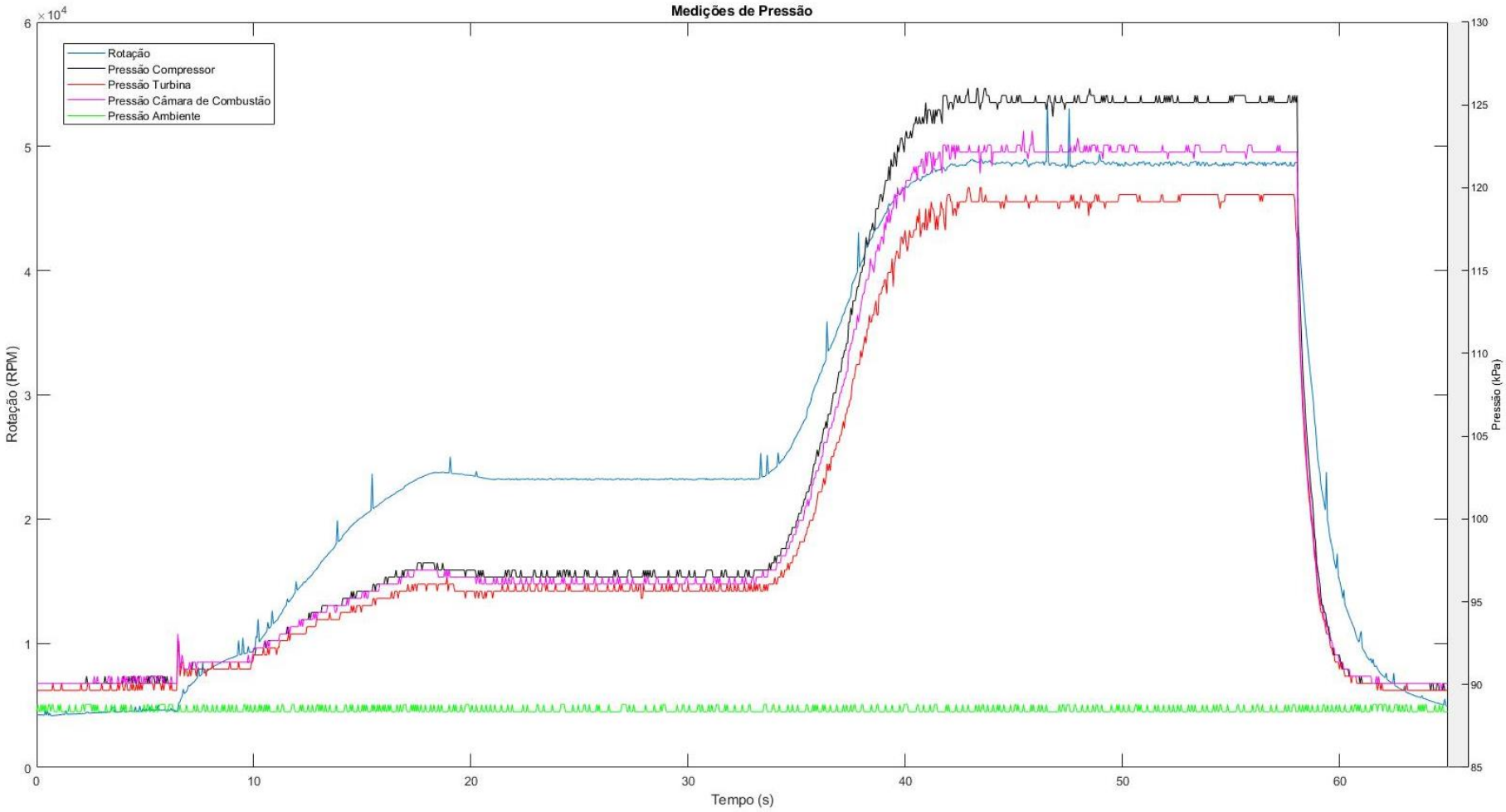


Figura 4.8 – Rotação do compressor e medições de pressão (Elaboração própria).

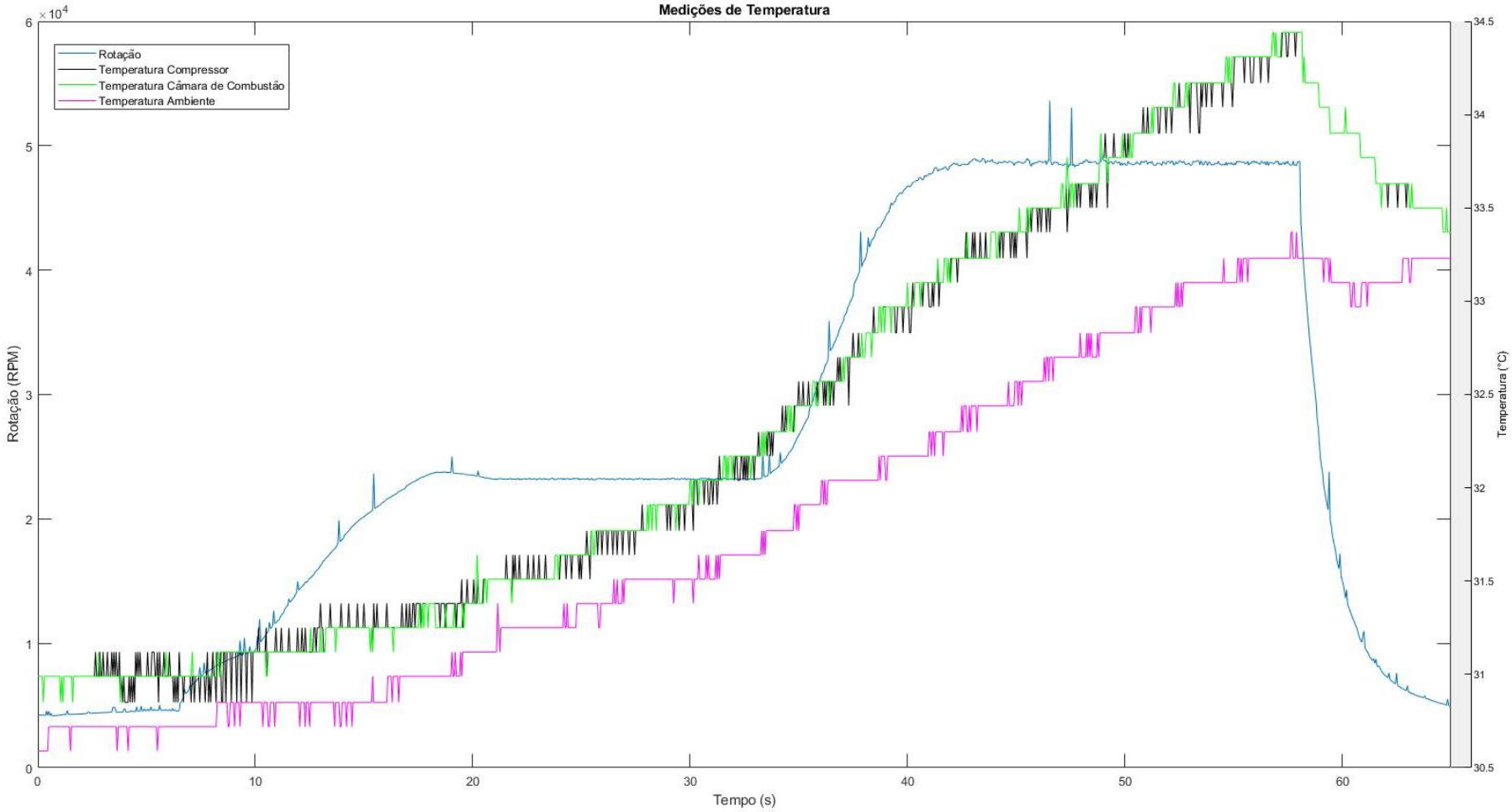


Figura 4.9 – Rotação do compressor e medições de temperatura (Elaboração própria).

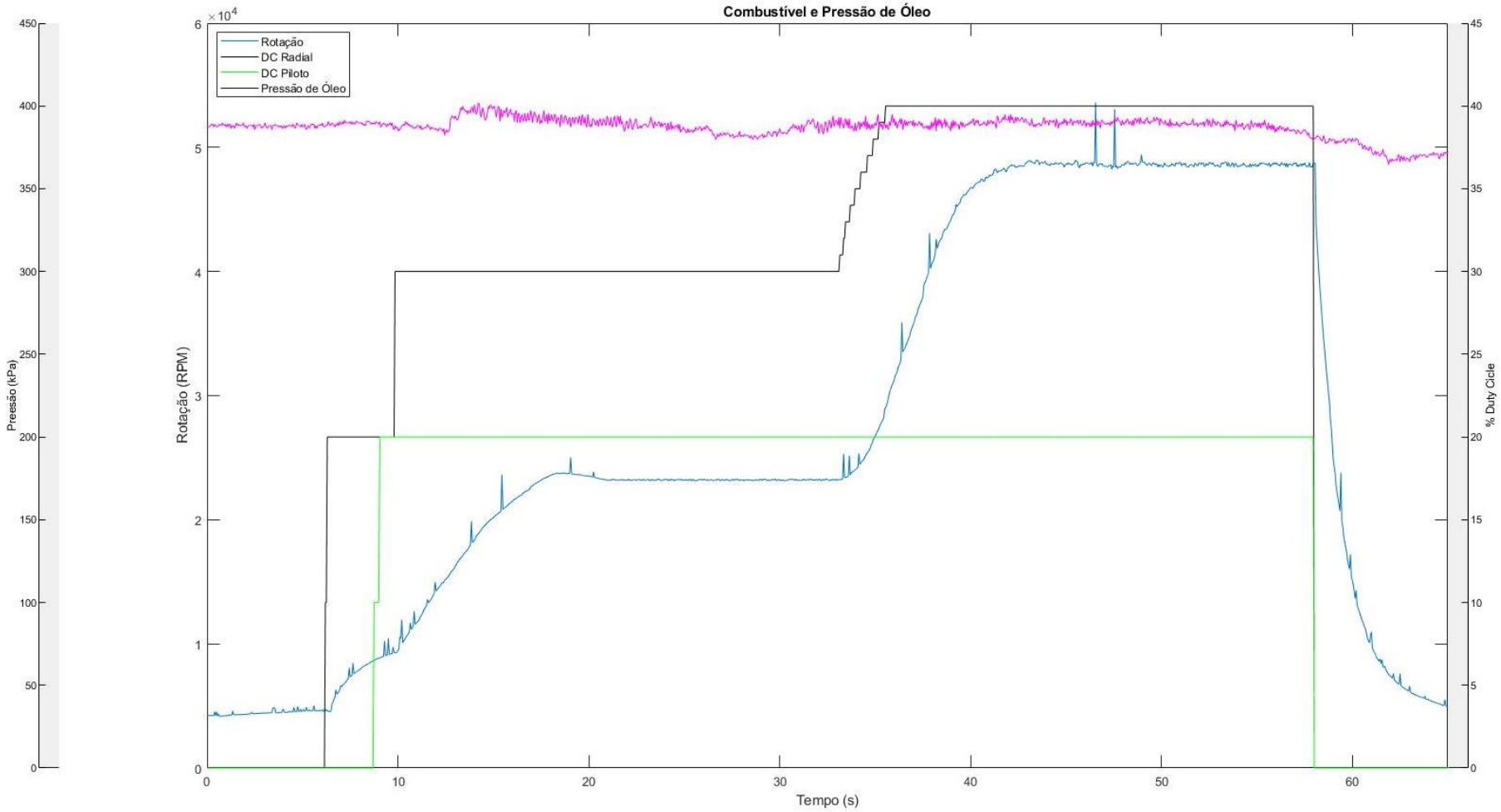


Figura 4.10 – Rotação do compressor e medições de pressão de óleo e injeção de combustível (Elaboração própria).

5. CONCLUSÃO

O sistema de aquisição de dados e controle desenvolvido neste trabalho se mostrou adequado para a operação da bancada experimental de microturbina. O sistema conseguiu obter a maioria dos parâmetros de operação e também permite ao operador iniciar o processo de combustão de forma simples e segura.

Os sensores automotivos utilizados se provaram confiáveis e ideais para esta aplicação. Esses resistiram sem problemas aos diversos testes realizados e não apresentaram nenhum tipo de inconsistência nos dados reportados.

Toda a parte mecânica da bancada experimental se comportou como o esperado, não havendo qualquer sinal de degaste prematuro em nenhum dos seus componentes.

Os termopares utilizados não resistiram às altas temperaturas existentes no sistema e acabaram sendo danificados durante os testes. Contudo, ainda assim foi possível validar o funcionamento desse sistema nos testes iniciais. Recomenda-se a aquisição de novos termopares que tenham uma qualidade construtiva melhor.

O inversor de frequência inicialmente selecionado para a bancada teve de ser substituído pelo modelo descrito neste trabalho por apresentar problemas durante os testes, impossibilitando que o controle da bomba de óleo fosse realizado pela interface de aquisição e controle.

Para trabalhos futuros, ficam as seguintes sugestões para aprimorar a bancada experimental:

- Implementar um filtro para o sinal de rotação;
- Incorporar um meio de aplicar carga na unidade (sugere-se o uso de um segundo turbocompressor ou o uso de uma válvula de freio-motor de caminhão para impor uma restrição na exaustão da turbina);
- Desenvolver uma malha de controle fechada.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES FILHO, H. J. R.; "COB-2019-0179 - Calibration of Automotive Hot-Film Anemometers for Scientific Applications" 25th ABCM International Congress of Mechanical Engineering, Uberlândia, 2019.

ASSOCIAÇÃO BRASILERIA DE NORMAS TÉCNICAS, "ABNT NBR 5410:2004 – Instalações elétricas de baixa tensão", 2ª edição, 2004.

ATMEL, "Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V", Datasheet, 2014.

BALMER, R. T.; "Modern engineering thermodynamics", Academic Press, Estados Unidos, 2011.

BAZZO, E.; CARVALHO, A. N.; MATELLI, J. A..; "Experimental results and thermodynamic analysis of a natural gas small scale cogeneration plant for power and refrigeration purposes", Applied Thermal Engineering, Volume 58, Issues 1–2, Pages 264-272, September 2013.

BECCARI, S.; PIPITONE, E.; CAMMALLERI, M.; GENCHI, G.; "Model-based optimization of injection strategies for SI engine gas injectors", Journal of Mechanical Science and Technology, Vol.28(8), pp.3311-3323, 2014.

BOYCE, M. P.; "Gas Turbine Engineering Handbook", 4th Edition, Butterworth-Heinemann, Reino Unido, 2012.

CENTENO, P.; EGIDO, I.; DOMINGO, C.; FERNÁNDEZ, F.; ROUCO, L.; GONZÁLEZ, M.; "Review of Gas Turbine Models for Power System Stability Studies", Instituto de Investigación Tecnológica (IIT), 9º Congresso Hispano Luso de Engenharia Eletrotécnica, 2005.

CORREA JR, P. S. P.; ZHANG, J.;, LORA, E. E. S.; ANDRADE, R. V.; PINTO, L. R. M.; RATNER, A.; "Experimental study on applying biomass-derived syngas in a microturbine" Applied Thermal Engineering, Volume 146, Pages 328-337, 5 January 2019.

CRUZ, T. V. G.; "Identificação Experimental de um Modelo Dinâmico de uma Microturbina a Gás com Câmara de Combustão com Baixa Emissão de NOx", Dissertação de Mestrado, Departamento de Engenharia Mecânica, UnB, Brasília, 2006.

"Data Acquisition Handbook", 3rd edition, Measurement Computing Corporation, 2012.

FLESLAND, S. M.; "Gas Turbine Optimum Operation"; Master of Science in Product Design and Manufacturing, Department of Energy and Process Engineering, Norwegian University of Science and Technology, Stavanger, 2010.

IVANOV, A. A.; PETROV, V. A.; "Effect of Load Current and Input and Output Capacitors of a Linear Voltage Regulator Chip on the Frequency Dependence of the Ripple Rejection Ratio". Journal of Communications Technology and Electronics, Vol.64 (3), p.198 (5), 2019

KRAMPF, F.,M.; "A Practical Guide for Gas Turbine Performance Field and Test Analysis Data", ASME Paper No. 92-GT-427, 1992.

KUANG, Y.; ZHANG, Y.; ZHOU, B.; CANBING LI, CAO, Y; LIJUAN LI, ZENG, L. A review of renewable energy utilization in islands, Renewable and Sustainable Energy Reviews, v. 59, p. 504-513, 2016.

LÉVESQUE, L.; "Nyquist sampling theorem: understanding the illusion of a spinning wheel captured with a video camera". Physics Education, Vol.49 (6), pp.697-705, 2014.

MERÍCIA, J. G.; "Controle de uma Microturbina a Gás com Câmara de Combustão de Baixa Emissão de NOx"; Dissertação de Mestrado em Ciências Mecânicas, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 2006.

NASCIMENTO, M. A. R.; LORA, E. E. S.; "Geração Termelétrica: Planejamento, Projeto e Operação", vol. 1 e vol. 2, 2003.

OGATA, K.; "Engenharia de Controle Moderno", 3ª edição. Prentice-Hall, Brasil, 1998.

PERNA, A.; MINUTILLO, M.; CICCONARDI, S. P.; JANNELLI, E.; SCARFOGLIERO, S.; "Conventional and Advanced Biomass Gasification Power Plants Designed for Cogeneration Purpose", Energy Procedia, Volume 82, Pages 687-694, December 2015.

PÓVOA, R.; "Aplicação do Protocolo "Kw2000" para Leitura de Dados Disponíveis no Módulo de Controle do Motor de um Veículo Popular", Dissertação de Mestrado, Escola Politécnica da Universidade de São Paulo, São Paulo, 2007.

RAIL S.R.L.; "Technical Data Sheet IG1 Apache", Datasheet

RATHI, C. Will natural gas compete or coexist with renewable power?. Power Engineering, v. 116, p. 14, Nov. 2012.

Resolução Normativa ANEEL Nº 482 de 17 de abril de 2012.

ROBERT BOSCH GMBH, “0.281.006.059_0.281.006.060_TKU_Drucksensor”, Datasheet

SCHMITZ, T.; “Don't Bypass Your Capacitor!”, Electronic Design, Vol.59(13), p.25, Oct 6, 2011.

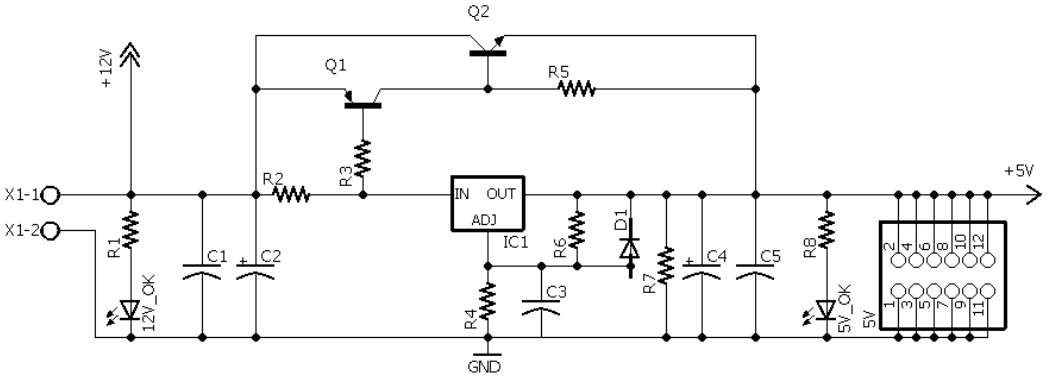
SLATTERY, B.; WYNNE, J.; “Design and Layout of a Video Graphics System for Reduced EMI”, Microprocessors And Microsystems, Vol.15 (2), pp.97-112, 1991.

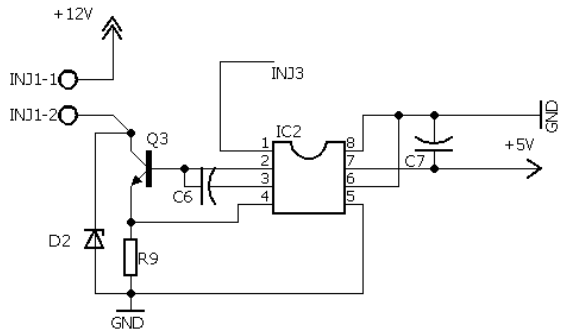
TEXAS INSTRUMENTS, “LM1949 Injector Drive Controller”, Datasheet, 2013.

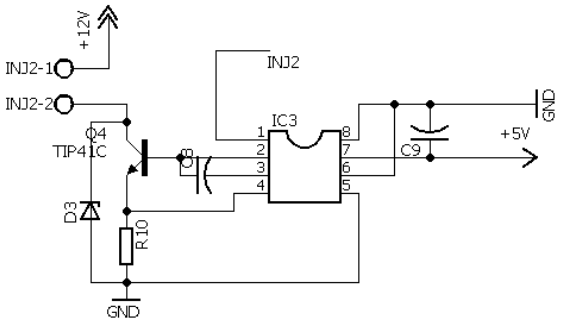
TEXAS INSTRUMENTS, “LM317 3-Terminal Adjustable Regulator”, Datasheet, 2016.

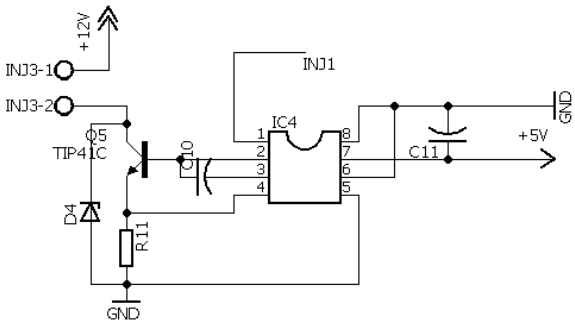
VISHAY SEMICONDUCTORS, “TCRT5000, TCRT5000L”, Datasheet, 2009.

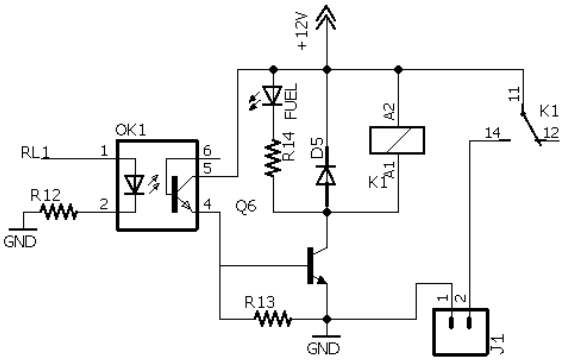
ANEXO I – ESQUEMÁTICO DO HARDWARE

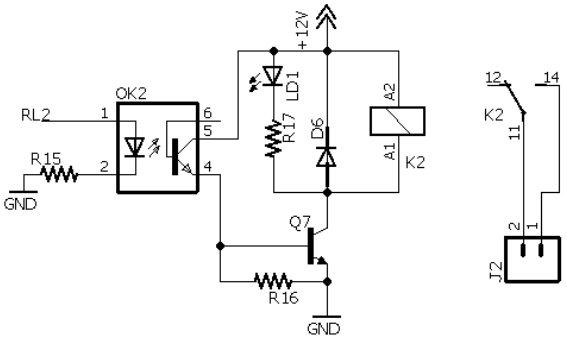


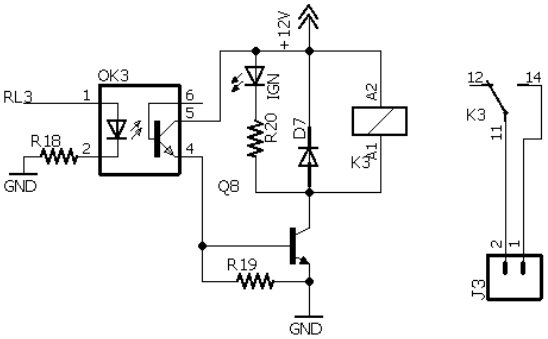


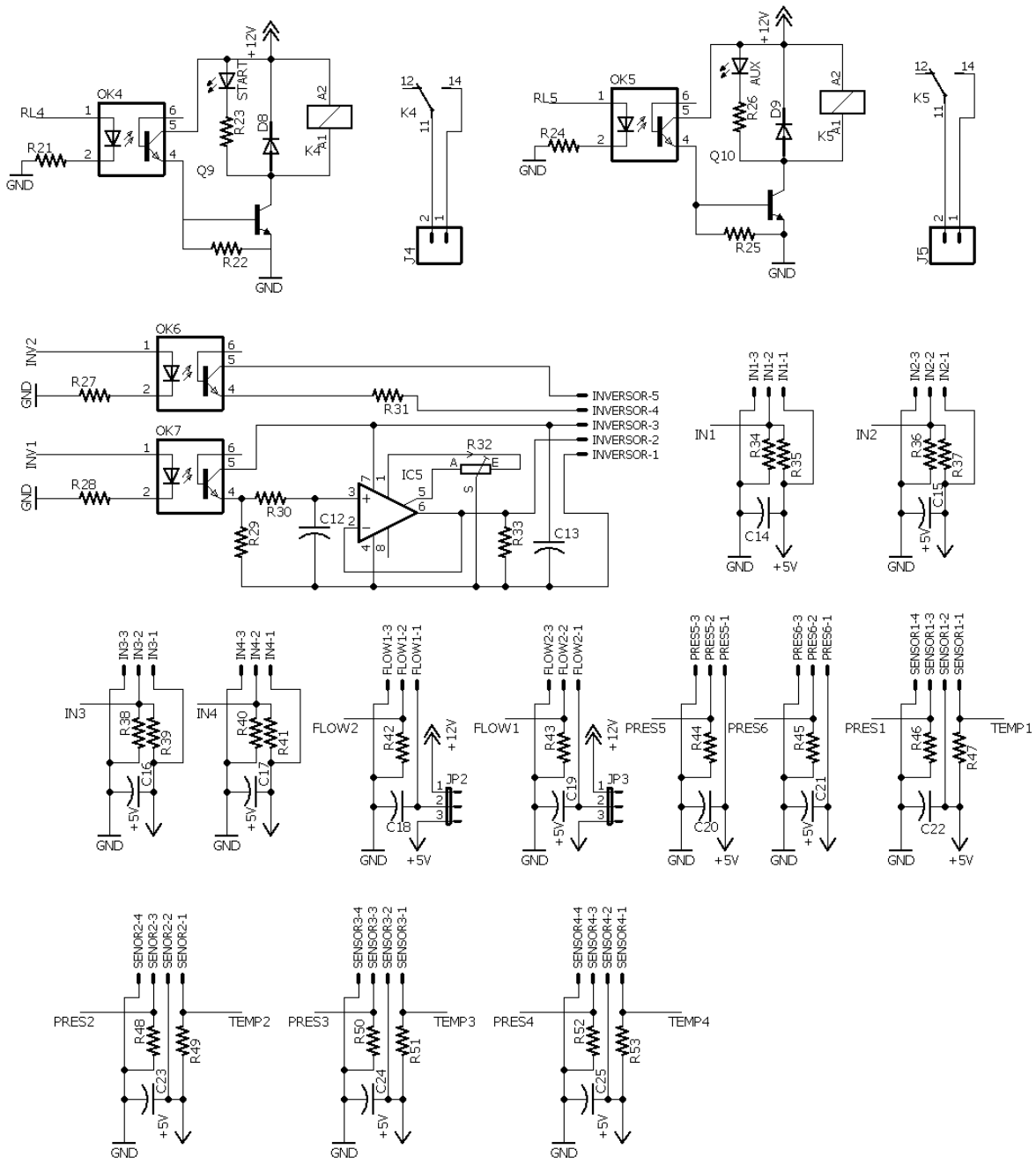


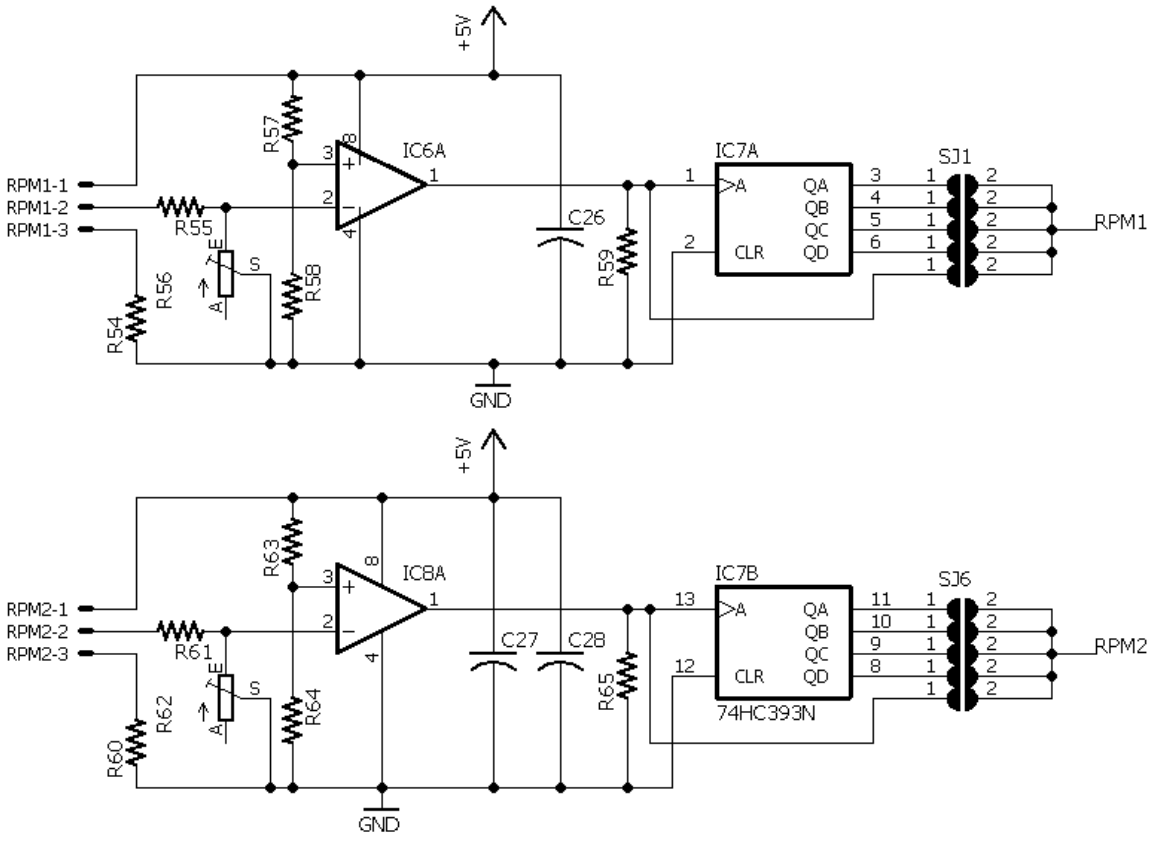


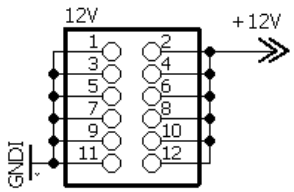


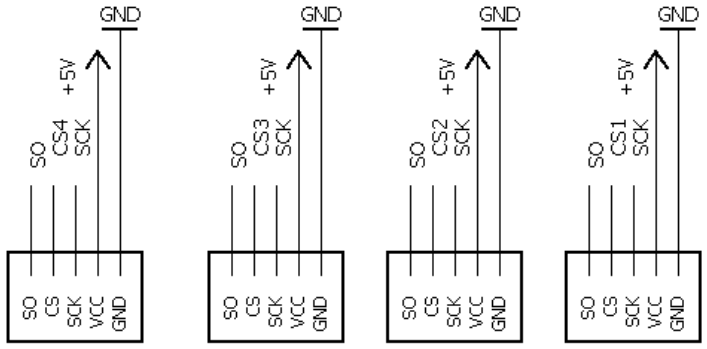


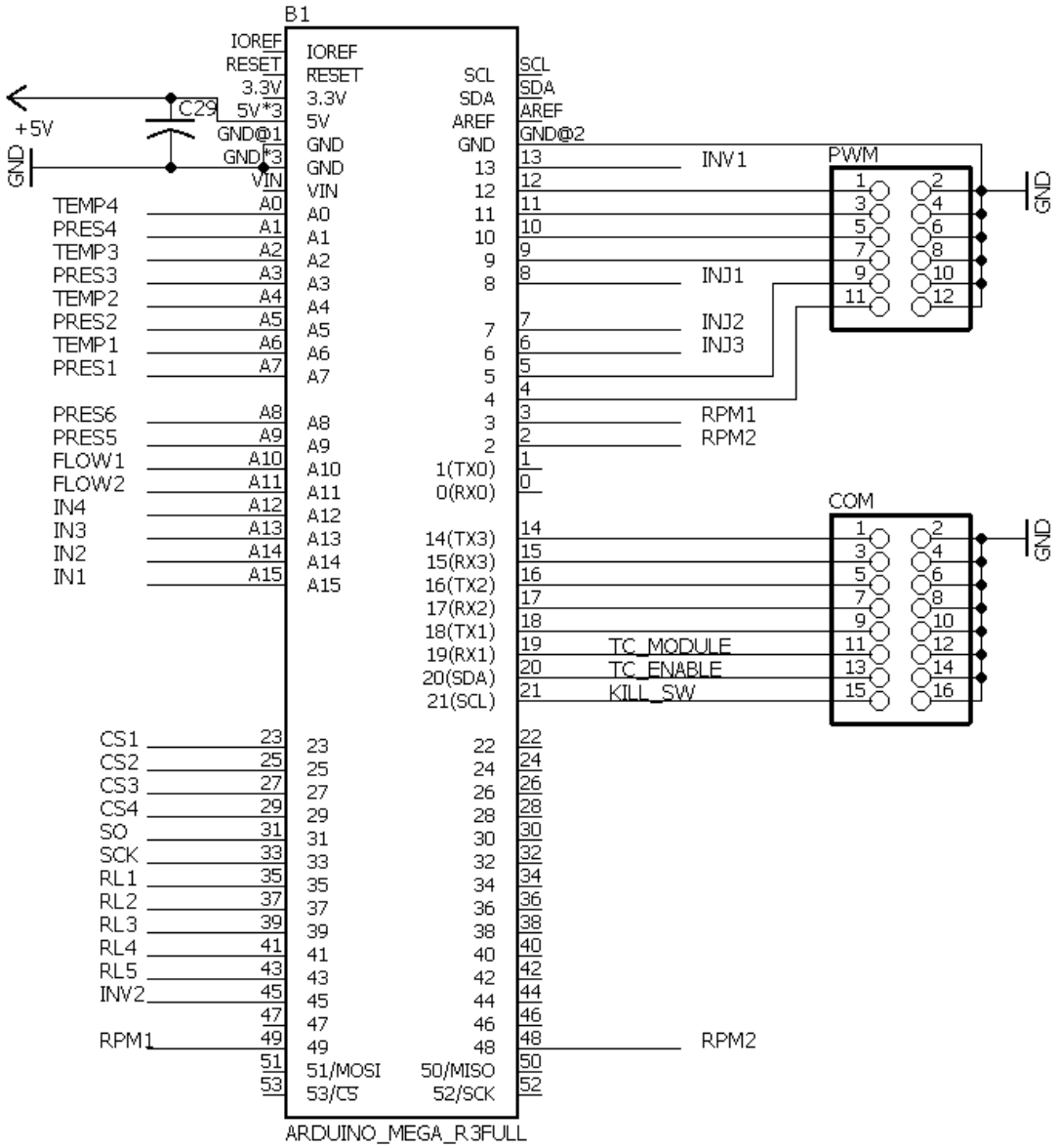



OK4
RL4
R21
GND
Q9
R23
START
D8
+12V
A2
K4
A1
R22
12
14
11
K4
2
1
J4
OK5
RL5
R24
GND
Q10
AUX
R26
D9
+12V
A2
K5
A1
R25
12
14
11
K5
2
1
J5
OK6
INV2
GND
R27
R31
INVERSOR-5
INVERSOR-4
INVERSOR-3
INVERSOR-2
INVERSOR-1
OK7
INV1
GND
R28
R30
C12
IC5
R32
R29
R33
C13
IN1-3
IN1-2
IN1-1
IN1
R34
R35
C14
GND
+5V
IN2-3
IN2-2
IN2-1
IN2
R36
R37
C15
+5V
GND
IN3-3
IN3-2
IN3-1
IN3
R38
R39
C16
+5V
GND
IN4-3
IN4-2
IN4-1
IN4
R40
R41
C17
+5V
GND
FLOW1-3
FLOW1-2
FLOW1-1
FLOW2
R42
+12V
JP2
C18
GND
+5V
FLOW2-3
FLOW2-2
FLOW2-1
FLOW1
R43
+12V
JP3
C19
+5V
GND
PRES5-3
PRES5-2
PRES5-1
PRES5
R44
C20
GND
+5V
PRES6-3
PRES6-2
PRES6-1
PRES6
R45
C21
+5V
GND
SENSOR1-4
SENSOR1-3
SENSOR1-2
SENSOR1-1
PRES1
R46
TEMP1
R47
C22
GND
+5V
SENOR2-4
SENOR2-3
SENOR2-2
SENOR2-1
PRES2
R48
TEMP2
R49
C23
+5V
GND
SENSOR3-4
SENSOR3-3
SENSOR3-2
SENSOR3-1
PRES3
R50
TEMP3
R51
C24
+5V
GND
SENSOR4-4
SENSOR4-3
SENSOR4-2
SENSOR4-1
PRES4
R52
TEMP4
R53
C25
+5V
GND


+5V
RPM1-1
RPM1-2
RPM1-3
R57
R55
R56
R54
R58
IC6A
C26
R59
IC7A
SJ1
RPM1
GND
RPM2-1
RPM2-2
RPM2-3
R63
R61
R62
R60
R64
IC8A
C27
C28
R65
IC7B
74HC393N
SJ6
RPM2


12V
+12V
GND!


GND
+5V
SO
CS4
SCK
CS3
CS2
CS1
CS
VCC



Transistores	
Q1	2N2905A
Q2	TIP122
Q3	TIP121
Q4	TIP121
Q5	TIP121
Q6	BC547
Q7	BC547
Q8	BC547
Q9	BC547
Q10	BC547

Diodos	
D1	1N54007
D2	1N5364B
D3	1N5364B
D4	1N5364B
D5	1N4007
D6	1N4007
D7	1N4007
D8	1N4007
D9	1N4007

Circuitos Integrados	
IC1	LM317
IC2	LM1949N
IC3	LM1949N
IC4	LM1949N
IC5	UA741
IC6	MCP602
IC7	74HC393N
IC8	MCP602

Optoacopladores	
OK1	4N35
OK2	4N35
OK3	4N35
OK4	4N35
OK5	4N35
OK6	4N35
OK7	4N35

Resistores - Valores em Ohms 1/4W 5%, exceto quando especificado					
R1	1k		R34	-	
R2	22		R35	-	
R3	4,7k		R36	-	
R4	470		R37	-	
R5	470		R38	-	
R6	150		R39	-	
R7	470		R40	100k	
R8	470		R41	-	
R9	0,1	5W	R42	100k	
R10	0,1	5W	R43	100k	
R11	0,1	5W	R44	100k	
R12	150		R45	100k	
R13	1k		R46	100k	
R14	1k		R47	2,7k	1%
R15	150		R48	100k	
R16	1k		R49	2,7k	1%
R17	1k		R50	100k	
R18	150		R51	2,7k	1%
R19	1k		R52	100k	
R20	1k		R53	2,7k	1%
R21	150		R54	330	
R22	1k		R55	33	
R23	1k		R56	100k	Trimpot
R24	150		R57	10k	
R25	1k		R58	10k	
R26	1k		R59	10k	
R27	150		R60	330	
R28	150		R61	33	
R29	1k		R62	100k	Trimpot
R30	1M		R63	10k	
R31	1k		R64	10k	
R32	10k	Trimpot	R65	10k	
R33	1k				

Capacitores - Valores em µF Cerâmico 50V, exceto quando especificado		
C1	0,1	
C2	1000	50V, Eletrolítico
C3	10	25V, Eletrolítico
C4	100	16V, Eletrolítico
C5	0,1	
C6	0,01	
C7	0,47	
C8	0,01	
C9	0,47	
C10	0,01	
C11	0,47	
C12	0,1	
C13	0,1	
C14	0,47	
C15	0,47	
C16	0,47	
C17	0,47	
C18	0,47	
C19	0,47	
C20	0,47	
C21	0,47	
C22	0,47	
C23	0,47	
C25	0,47	
C26	0,1	
C27	0,1	
C28	0,1	
C29	0,1	

ANEXO II – CÓDIGO FONTE DO MÓDULO DE AQUISIÇÃO DOS TERMOPARES

```
#include <SPI.h>
#include "Adafruit_MAX31855.h"

Adafruit_MAX31855 TC0(2, 3, 4);
Adafruit_MAX31855 TC1(2, 11, 4);
Adafruit_MAX31855 TC2(2, 12, 4);

int T[3];

byte sb[7];

bool go;

bool CS;

void setup() {
  pinMode(5,INPUT);
  Serial.begin(57600);
}

void loop() {
  if (go == true) {
    T[0] = (int) TC0.readCelsius();
    T[1] = (int) TC1.readCelsius();
    T[2] = (int) TC2.readCelsius();

    for (int i = 0; i < 3; i++) {
      if (isnan(T[i]) || T[i]<0) {T[i]=0;}
    sb[2*i] = lowByte(T[i]);
    sb[(2*i)+1] = highByte(T[i]);
    }

    sb[6] = 0;

    for (int i = 0; i < 6; i++) {
      sb[6] = sb[6] + sb[i];
    }

    for (int i = 0; i < 7; i++) {
      Serial.write(sb[i]);
    }

    if (digitalRead(5) == LOW) {
      go = false;
    }
  }
  else {
    if (digitalRead(5) == HIGH) {
      go = true;
    }
  }

}
```

ANEXO III – CÓDIGO FONTE DO SOFTWARE EMBARCADO

```
/*Serial Messages:
  Disconnected: 0xFF
  Initialized: 0xFE
  Checksum Error: 0xFD
  Unknow Command: 0xFC
*/

//Serial Communication & checksum
byte sb[4];
byte cs;
byte ccs;
byte k;
unsigned long sc;
unsigned long scs;
bool c;
unsigned volatile int wd;

//TC Serial Com:
byte tb[7];
byte tcs;
unsigned volatile int tct;

//RPM measurament
volatile boolean first4;
volatile boolean first5;
volatile bool rpm1_of;
volatile bool rpm2_of;

volatile unsigned int rpm1c;
volatile unsigned int rpm2c;

volatile unsigned int rpm1t[6];
volatile unsigned int rpm2t[6];

volatile bool rpmread;


//Data reading
volatile unsigned long db[16];
unsigned long sdb[17];

//Kill fuel
bool kf;

void setup() {
  //Watchdog timer
  TCCR2A = B00000000;
  TCCR2B = B01000101;

  //Data reading timming
  TCCR1A = B00000000;
  TCCR1B = B01000010;

  //Set PWM Frequency:
  TCCR3B = TCCR3B & B11111000 | B00000101;
```

```
  //RPM1 & RPM2 timer setup:
  TCCR4A = B00000000;
  TCCR4B = B01000010;

  TCCR5A = B00000000;
  TCCR5B = B01000010;

  //Start serial:
  Serial.begin(57600);
  Serial.setTimeout(10);

  Serial1.begin(57600);
  Serial1.setTimeout(10);

  //Set pin mode:
  pinMode(2, OUTPUT); //INJ3
  pinMode(3, OUTPUT); //INJ2
  pinMode(5, OUTPUT); //INJ1
  pinMode(35, OUTPUT); //RL1 SOL
  pinMode(37, OUTPUT); //RL2 FAN
  pinMode(39, OUTPUT); //RL3 IGN
  pinMode(41, OUTPUT); //RL4 START
  pinMode(43, OUTPUT); //RL5 AUX
  pinMode(45, OUTPUT); //Inversor SW
  pinMode(13, OUTPUT); //Inversor PWM
  pinMode(48, INPUT);
  pinMode(49, INPUT);

  pinMode(20, OUTPUT);

  pinMode(21, INPUT_PULLUP);

  digitalWrite(35, 0);
  digitalWrite(37, 0);
  digitalWrite(39, 0);
  digitalWrite(41, 0);
  digitalWrite(43, 0);

  analogWrite(6, 0);
  analogWrite(7, 0);
  analogWrite(8, 0);
}

void loop() {

  //Check for serial data
  if (Serial.available() > 0) {
    if (c == LOW) {
      Serial.readBytes(sb, 1);
      if (sb[0] == 0xF0) {
        Serial.write(sb[0]);
        c = HIGH;
        //TIMSK0 = B00000001;
        TIMSK1 = B00000001;
        //TIMSK2 = B00000001;
        TIMSK4 = B00100001;
        TIMSK5 = B00100001;
      }
      else {
        Serial.println(0xFF);
```

```
      }
    }
    else {

      if (digitalRead(21) == HIGH)
      {
        killFuel();
        kf = true;
      }
      else
      {
        kf = false;
      }

      //Read first byte, representing the lenght in bytes of the message.
THIS FIRST BYTE IS NOT COUNTED!
      Serial.readBytes(sb, 1);

      //Save message lenght
      k = sb[0];
      if (k > 4) {
        k = 4;
      }

      //Read message with k bytes lenght & save to serial buffer(sb)
      Serial.readBytes(sb, k);


      //Checksum from host (cs) is stored on the last byte of the message
      cs = sb[k - 1];

      //Serial.print(cs);

      //Checksum is calculated by adding all bytes in the message and
performing a bitwise logical AND on the result with 0xFF.
      ccs = 0;
      for (int i = 0; i < k - 1; i++) {
        ccs = ccs + sb[i];
      }
      ccs = ccs & 255;

      //If checksum is correct, do things:
      if (ccs == cs) {

        if (sb[0] == 0x10) {
          wd = 0;
          analogWrite(sb[1], sb[2]);
          Serial.println(sb[0]);
        }

        else if (sb[0] == 0x11) {
          wd = 0;
          digitalWrite(sb[1], sb[2]);
          Serial.println(sb[0]);
        }

        else if (sb[0] == 0x20) {
          wd = 0;
          sendData();
        }
```

```
        else if (sb[0] == 0x21) {
          wd = 0;
          resendData();
        }

        else if (sb[0] == 0x30) {
          wd = 0;
          if (kf == false) {
            analogWrite(2, sb[1]);
            analogWrite(3, sb[1]);
            Serial.println(sb[0]);
          }
          else
          {
            Serial.println(sb[0] + 1);
          }
        }

        else if (sb[0] == 0x31) {
          wd = 0;
          if (kf == false) {
            analogWrite(5, sb[1]);
            Serial.println(sb[0]);
          }
          else
          {
            Serial.println(sb[0] + 1);
          }
        }
        else {
          Serial.println("0xFD");
        }

      }

      else {
        Serial.println("0xFE");
      }

    }

  }

}

//Send data to host. Checksum is calculated the same way as the incoming
messages. Data is sent in ASCII with comma separated fields.
void sendData()
{

  if (Serial1.available() > 0) {
    Serial1.readBytes(tb, 7);
    tcs = 0;
    for (int i = 0; i < 6; i++) {
      tcs = tcs + tb[i];
    }
    tcs = tcs & 255;

    if (tcs == tb[6]) {
```

```
      sdb[13] = (tb[1] << 8) | tb[0];
      sdb[14] = (tb[3] << 8) | tb[2];
      sdb[15] = (tb[5] << 8) | tb[4];
    }
  }

  if (rpm1_of == true)
  {
    sdb[10] = 0;
    rpm1_of = false;
  }

  else
  {
    sdb[10]=db[10];
  }


  if (rpm2_of == true)
  {
    sdb[11] = 0;
    rpm2_of = false;
  }

  else
  {
    sdb[11]=db[11];
  }

  TIMSK1 = B00000000;
  for (int i = 0; i < 10; i++)
  {
    sdb[i] = db[i];
    db[i] = 0;
  }
  sdb[12] = db[12];
  db[12] = 0;

  TIMSK1 = B00000001;

  sdb[16] = 0;

  for (int i = 0; i < 16; i++)
  {
    sdb[16] = sdb[16] + sdb[i];
  }

  for (int i = 0; i < 16 ; i++)
  {
    Serial.print(sdb[i]);
    Serial.print(',');
  }

  Serial.println(sdb[16]);

}

void resendData() {
  for (int i = 0; i < 16 ; i++)
  {
```

```
    Serial.print(sdb[i]);
    Serial.print(',');
  }

  Serial.println(sdb[16]);

}

ISR (TIMER2_OVF_vect) {
  wd++;

  if (wd > 615)
  {
    wd = 0;

    killFuel();

    TIMSK2 = B00000000;

    asm volatile (" jmp 0");
  }
}

void killFuel() {
  digitalWrite(35, 0);
  digitalWrite(39, 0);

  analogWrite(5, 0);
  analogWrite(3, 0);
  analogWrite(2, 0);
}

//Read ADC
ISR (TIMER1_OVF_vect) {

  for (int i = 0; i < 6; i++)
  {
    db[i] = db[i] + analogRead(i);
  }

  for (int i = 6; i < 10; i++)
  {
    db[i] = db[i] + analogRead(i + 3);
  }

  db[12]++;

  tct++;

  if (tct == 1000 & Serial1.available() == 0 ) {
    digitalWrite(20, HIGH);
  }

  if (tct == 1010) {
    digitalWrite(20, LOW);
    tct = 0;
  }


  TCNT1 = 55535;
```

```
}


//RPM1 period measurament:
ISR (TIMER4_CAPT_vect)
{
  TCNT4 = 0;

  if (first4 == true)
  {
    first4 = false;
  }

  else {
    db[10] = ICR4;
  }
}

ISR (TIMER4_OVF_vect) {
  rpm1_of = true;
  first4 = true;
}

//RPM2 period measurament:
ISR (TIMER5_CAPT_vect)
{
  TCNT5 = 0;

  if (first5 == true)
  {
    first5 = false;
  }

  else {
    db[11] = ICR5;
  }
}

ISR (TIMER5_OVF_vect)
{
  rpm2_of = true;
  first5 = true;
}
```

ANEXO IV – ROTINA MATLAB

```
clear all;
close all;

R = [5895
    4711
    3791
    3068
    2499
    2056
    1706
    1411
    1174
    987.4
    833.8
    702.7
    595.4
    508.2
    435.6
    374.1
    322.5
    279.5
    243.1
    212.6
    186.6
    163.8
    144.2
    127.3
    112.7
    100.2
    89.28
    79.63
    71.18
    63.85
    57.39];

T = [0
    5
    10
    15
    20
    25
    30
    35
    40
    45
    50
    55
    60
    65
    70
    75
    80
    85
    90
    95
    100
    105
```

```
    110
    115
    120
    125
    130
    135
    140
    145
    150];

f = fit(R,T,'exp2')

 figure;
 hold on;
 plot(R,T);
 plot(f);

 cT=f(R);

 error= (cT-T)./T;

 figure;
 plot(T,error,'+');
 axis([0 155 -0.2 0.05]);
 xlabel('Temperatura °C');
 ylabel('Erro');
```

ANEXO V – CÓDIGO FONTE DO SOFTWARE DE CONTROLE

Arquivo “Form1.cs”:

```
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using FileHelpers;
using org.mariuszgromada.math.mxparser;
using System.Diagnostics;
using System.Timers;
using System.Threading;
using System.Reflection;

namespace Serial

{

    public partial class Form1 : Form

    {
        private static System.Timers.Timer dTimer;

        private static System.Timers.Timer lTimer;

        private static bool _isRunning;

        public static bool isRunning
        {
            get
            {
                return _isRunning;
            }
            set
            {
                _isRunning = value;

            }
        }

        private void EnableControls(bool value)
        {
            gFan.Enabled = value;
            gFuel.Enabled = value;

            if (value == true)
            {
                tRunning.Text = "Running...";
            }
            else
            {
                tRunning.Text = "Stopped";
            }
        }
```

```
        public string selCOM;

        private bool _serialConnected;
        public bool serialConnected
        {
            get
            {
                return _serialConnected;
            }
            set
            {
                _serialConnected = value;
                bSerial.Text = _serialConnected ? "Disconnect" : "Connect";
                tConnected.Text = _serialConnected ? "Connected" : "Disconnected";

                cbSerial.Enabled = !_serialConnected;
                bLog.Enabled = _serialConnected;
                sfCal.Enabled = !_serialConnected;
                tMCtrlP.Enabled = _serialConnected;
                gRun.Enabled = _serialConnected;
                gFan.Enabled = _serialConnected;

                if (_serialConnected == false)
                {
                    isLoggin = false;
                    stopWatch.Stop();
                    bfdLog.Enabled = true;
                }
            }
        }

        public static double[] readData = new double[16] { 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0 };

        Stopwatch stopWatch = new Stopwatch();

        Stopwatch LogStopWatch = new Stopwatch();

        public static string fileLog;

        public bool fLogSel = false;

        private bool _isLoggin;

        public bool isLoggin
        {
            get
            {
                return _isLoggin;
            }
            set
            {
                _isLoggin = value;
                if (InvokeRequired)
                {
                    bLog.Invoke(new Action(() => bLog.Text = _isLoggin ? "Stop Log
" : "Start Log"));
```

```
                }
                else
                {
                    bLog.Text = _isLoggin ? "Stop Log" : "Start Log";
                }

            }
        }

        public string fileCal;
        public bool fCalSel = false;

        public string[] calFunc = new string[10] { "x", "x", "x", "x", "x", "x", "
x", "x", "x", "x" };

        string path;

        public Form1()
        {
            InitializeComponent();


            oilTemp.InitializeBackgroundWorkerOilTemp();

            dataTimer();

            loggerTimer();

            serialConnected = false;
            isLoggin = false;

            string[] coms = SerialPort.GetPortNames();

            ReadDefault();

            cbSerial.Items.AddRange(coms);
        }

        private void ReadDefault()
        {
            path = (Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location
) + "\\default.ini");

            selCOM = File.ReadLines(path)
            .SkipWhile(line => !line.Contains("COM"))
            .TakeWhile(line => !line
            .Contains(Environment.NewLine))
            .ToArray()[0];

        }

        public void loadCalData()
        {
            var engine = new FileHelperEngine<CalData>();

            var result = engine.ReadFile(fileCal);

            int i = 0;
```

```
    foreach (var point in result)
    {
        calFunc[i] = point._function;
        i++;
    }
}

public void dataTimer()
{
    dTimer = new System.Timers.Timer(50);
    dTimer.Elapsed += new ElapsedEventHandler(RequestData);
    dTimer.AutoReset = true;
}

public void loggerTimer()
{
    lTimer = new System.Timers.Timer(50);
    lTimer.Elapsed += new ElapsedEventHandler(StopLog);
    lTimer.AutoReset = false;
}

public void StopLog(object source, ElapsedEventArgs e)
{
    isLoggin = false;
    lTimer.Enabled = false;


    if (InvokeRequired)
    {
        bfdLog.Invoke(new Action(() => bfdLog.Enabled = true));
    }
    else
    {
        bfdLog.Enabled = true;
    }


}

// 0  1  2  3  4  5  6   7   8   9  10  11   12          13  14  15 16
//A0,A1,A2,A3,A4,A5,A9,A10,A11,A12,RPM1,RPM2,ADCCounter,TC1,TC2,TC3,CS
//T1,P1 - A0,A1 - 0,1 - Compressor
//T2,P2 - A2,A3 - 2,3 - Entrada CC
//T3,T3 - A4,A5 - 4,5 - Ambiente
//T4,P4 - TC1,A9 - 13,6 Saída CC
//T5 - TC2 - 14
//MAF1 - A10 - 7
//MAF2 - A11 - 8
//OilP - A12 - 9
//OilT - 15 - TC3
//RPM1 - 10
//RPM2 - 11

public void RequestData(object source, ElapsedEventArgs e)
{
    while (serialCOM.reading == true)
    {
```

```
                Thread.Sleep(1);
            }

            byte[] data = { 0x02, 0x20, 0x20 };

            string ans = serialCOM.sendData(data);

            if (serialCOM.serialTimeOut == true)
            {
                DisconnectSerial();
                MessageBox.Show("System disconnected due to timeout!", "Message",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            string[] rawDataS = ans.Split(',');

            double t = stopWatch.Elapsed.TotalSeconds;
            double tlog = LogStopWatch.Elapsed.TotalMilliseconds;


            int[] rawData = Array.ConvertAll(rawDataS, int.Parse);
            if (rawData.Length != 17)
            {
                MessageBox.Show("Data String is incorrect!", "DS Error!", MessageB
oxButtons.OK, MessageBoxIcon.Exclamation);
                return;
            }

            int csc = 0;

            for (int i = 0; i < 16; i++) { csc = csc + rawData[i]; }

            if (csc != rawData[16])
            {
                MessageBox.Show("Data Checksum Incorrect!", "CS Error!", MessageBo
xButtons.OK, MessageBoxIcon.Exclamation);
                return;
            }

            for (int i = 0; i < 10; i++)
            {
                double rawDataC = rawData[i] / rawData[12];
                Function f = new Function("f", calFunc[i], "x");
                readData[i] = f.calculate(rawDataC);
            }

            for (int i = 10; i < 12; i++)
            {

                if (rawData[i] != 0)
                {

                    readData[i] = 60 * (1 / ((double)(rawData[i]) / 2000000));
                }
                else
                {
                    readData[i] = 0;
```

```
                }
            }

            for (int i = 13; i < 16; i++)
            {
                if (rawData[i] != 0)
                {
                    readData[i - 1] = rawData[i];

                }
                else
                {
                    readData[i] = 0;
                }
            }

            if (isLoggin == true)
            {
                using (StreamWriter sw = File.AppendText(fileLog))
                {
                    sw.Write(tlog);
                    sw.Write(";");

                    for (int i = 0; i < 16; i++)
                    {
                        sw.Write(readData[i].ToString("N"));
                        sw.Write(";");
                    }

                    sw.Write(Fuel.radialDC);
                    sw.Write(";");
                    sw.Write(Fuel.pilotDC);
                    sw.Write(Environment.NewLine);
                }
            }
            UpdateBox(readData, t);
        }


        //GUI Update Functions:
        public void UpdateBox(double[] value, double time)
        {
            if (this.IsDisposed)
            {
                return;
            }

            if (InvokeRequired)
            {
                this.Invoke(new Action<double[], double>(UpdateBox), new object[]
{ value, time });
                return;
            }

            int ix = chart1.Series["SeriesRPM1"].Points.AddXY(time, value[10]);
            chart1.Series["SeriesRPM2"].Points.AddXY(time, value[11]);
            chart1.Series["SeriesOilTemp"].Points.AddXY(time, value[15]);
            chart1.Series["SeriesOilPres"].Points.AddXY(time, value[9]);
```

```
            chart1.Series["SeriesTurboTemp"].Points.AddXY(time, value[13]);


            chart1.Series["Series5"].Points.AddXY(time, value[6]);
            chart1.Series["Series6"].Points.AddXY(time, value[14]);
            chart1.Series["Series7"].Points.AddXY(time, value[0]);
            chart1.Series["Series8"].Points.AddXY(time, value[1]);
            chart1.Series["Series9"].Points.AddXY(time, value[2]);
            chart1.Series["Series10"].Points.AddXY(time, value[3]);
            chart1.Series["Series12"].Points.AddXY(time, value[4]);
            chart1.Series["Series12"].Points.AddXY(time, value[5]);
            chart1.Series["Series13"].Points.AddXY(time, value[7]);
            chart1.Series["Series14"].Points.AddXY(time, value[8]);


            tRPM1.Text = value[10].ToString();
            tRPM2.Text = value[11].ToString("N");
            tOilTemp.Text = value[15].ToString("N");
            tOilPres.Text = value[9].ToString("N");
            tTurboTemp.Text = value[13].ToString("N");
            tTurboPres.Text = value[6].ToString("N");
            tEGT.Text = value[14].ToString("N");
            tAmbTemp.Text = value[0].ToString("N");
            tAmbPres.Text = value[1].ToString("N");
            tCompTemp.Text = value[2].ToString("N");
            tCompPres.Text = value[3].ToString("N");
            tCCTemp.Text = value[4].ToString("N");
            tCCPres.Text = value[5].ToString("N");
            tTotalMAF.Text = value[7].ToString("N");
            tCombMAF.Text = value[8].ToString("N");


            chart1.ChartAreas[0].AxisX.Minimum = double.NaN;
            chart1.ChartAreas[0].AxisX.Maximum = double.NaN;
            chart1.ChartAreas[0].RecalculateAxesScale();




            if (time < 10)
            {
                chart1.ChartAreas[0].AxisX.Minimum = 0;
                chart1.ChartAreas[0].AxisX.Maximum = 10;
                chart1.ChartAreas[0].AxisX.ScaleView.Zoom(0, 10);
            }
            else
            {
                chart1.ChartAreas[0].AxisX.ScaleView.Zoom(chart1.Series["SeriesRPM
1"].Points[ix].XValue - 10, chart1.Series["SeriesRPM1"].Points[ix].XValue);
            }

            chart1.ChartAreas[0].AxisX.LabelStyle.Format = "#";

            tRDC.Text = Convert.ToString(Fuel.radialDC);
            tPDC.Text = Convert.ToString(Fuel.pilotDC);

            if (gFuel.Enabled != isRunning)
            {
```

```
                gFuel.Enabled = isRunning;
            }
            if (gFan.Enabled != isRunning) { gFan.Enabled = isRunning; }
            if (bStop.Enabled != isRunning) { bStop.Enabled = isRunning; }
            tRunning.Text = isRunning ? "Running..." : "Stopped";




        }

        public void UpdateLogFile(string value)
        {
            if (InvokeRequired)
            {
                this.Invoke(new Action<string>(UpdateLogFile), new object[] { valu
e });
                return;
            }
            tLogFile.Text = value;
        }



        //GUI Interactive Functions
        private void BSerial_Click(object sender, EventArgs e)
        {
            bSerial.Enabled = false;

            if (serialConnected == false)
            {

                if (fCalSel == false)
                {
                    fdCal.ShowDialog();
                    fileCal = fdCal.FileName;
                    fCalSel = true;
                }

                loadCalData();

                serialConnected = serialCOM.openSerial(selCOM);

                if (serialConnected == true)
                {
                    dTimer.Enabled = true;

                    bSerial.Enabled = true;

                    stopWatch.Start();

                    int i = 0;

                    List<string> lines = System.IO.File.ReadAllLines(path).ToList<
string>();
                    foreach (string line in lines)
                    {
                        if (line.StartsWith("COM"))
```

```
                {
                    break;
                }
                i++;
            }
            lines[i] = selCOM;
            System.IO.File.WriteAllLines(path, lines);
        }

        else
        {
            bSerial.Enabled = true;
            cbSerial.Enabled = true;
            bLog.Enabled = false;
        }

    }
    else
    {
        DisconnectSerial();
        bSerial.Enabled = true;
    }
}

public void DisconnectSerial()
{
    dTimer.Enabled = false;
    lTimer.Enabled = false;


    serialCOM.closeSerial();

    serialConnected = false;
}

private void Form1_FormClosing(Object sender, FormClosingEventArgs e)
{
    DisconnectSerial();
}

private void SfCal_Click(object sender, EventArgs e)
{
    fdCal.ShowDialog();
    fileCal = fdCal.FileName;
    fCalSel = true;
    loadCalData();
}

private void bfdLog_Click(object sender, EventArgs e)
{
    fdLog.ShowDialog();
    fileLog = fdLog.FileName;
    UpdateLogFile(fileLog);
    fLogSel = true;
}

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

```
            if (isLoggin == false)

            {
                bfdLog.Enabled = false;
                if (fLogSel == false)
                {
                    fdLog.ShowDialog();
                    fileLog = fdLog.FileName;
                    UpdateLogFile(fileLog);
                    fLogSel = true;
                }
                if (fdLog.FileName != null)
                {
                    string time = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss tt");

                    using (StreamWriter sw = File.AppendText(fileLog))
                    {
                        sw.WriteLine();
                        sw.WriteLine(time);
                        sw.WriteLine("Sample Time: 50ms");
                        sw.WriteLine("Time; Comp Temp; Comp Pres; CC Temp; CC Pres
; Amb Temp; Amb Pres; Turbo Pres; MAF; MAF2; Oil Pres; RPM1; RPM2; Turbo Temp; EGT
; Oil Temp; Fuel Radial; Fuel Pilot");
                        sw.WriteLine("ms; ºC; kPa; ºC; kPa; ºC; kPa; kPa; kg/s; kg
/s; kPa; n/min; n/min; ºC; ºC; ºC; DC %; DC %");
                    }

                    if (cBLog.Checked == true)
                    {
                        int s = 1000 * (int)nLog.Value;
                        lTimer.Interval = s;
                        lTimer.Enabled = true;
                    }

                    LogStopWatch.Start();
                }

            }

            else
            {
                bfdLog.Enabled = true;
                lTimer.Enabled = false;
                LogStopWatch.Stop();
                LogStopWatch.Reset();
            }
            isLoggin = !isLoggin;
        }

        private void cbSerial_ValueUpdated(object sender, EventArgs e)
        {
            selCOM = cbSerial.Text;
        }

        private void CBRPM1_CheckedChanged(object sender, EventArgs e)
        {
            chart1.Series[0].Enabled = !chart1.Series[0].Enabled;
```

```
}

private void CBRPM2_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[1].Enabled = !chart1.Series[1].Enabled;
}

private void CBOilTemp_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[2].Enabled = !chart1.Series[2].Enabled;
}

private void CBOilPres_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[3].Enabled = !chart1.Series[3].Enabled;
}

private void CBTurboTemp_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[4].Enabled = !chart1.Series[4].Enabled;
}

private void CBTurboPres_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[5].Enabled = !chart1.Series[5].Enabled;
}

private void CBEGT_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[6].Enabled = !chart1.Series[6].Enabled;
}

private void CBAmbTemp_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[7].Enabled = !chart1.Series[7].Enabled;
}

private void CBAmbPres_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[8].Enabled = !chart1.Series[8].Enabled;
}

private void CBCompTemp_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[9].Enabled = !chart1.Series[9].Enabled;
}

private void CBCompPress_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[10].Enabled = !chart1.Series[10].Enabled;
}

private void CBCCTemp_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[11].Enabled = !chart1.Series[11].Enabled;
}

private void CBCCPres_CheckedChanged(object sender, EventArgs e)
```

```
{
    chart1.Series[12].Enabled = !chart1.Series[12].Enabled;
}

private void CBTotalMAF_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[13].Enabled = !chart1.Series[13].Enabled;
}

private void CBCombMAF_CheckedChanged(object sender, EventArgs e)
{
    chart1.Series[14].Enabled = !chart1.Series[14].Enabled;
}

private void BStart_Click(object sender, EventArgs e)
{
    Start start = new Start();
    start.Show();
}

private void BStop_Click(object sender, EventArgs e)
{
    Stop stop = new Stop();
    stop.Show();
}

private void RefreshToolStripMenuItem_Click(object sender, EventArgs e)
{
    string[] coms = SerialPort.GetPortNames();

    cbSerial.Items.Clear();
    cbSerial.Items.AddRange(coms);
}

private void BFuelApply_Click(object sender, EventArgs e)
{
    if (tcFuel.SelectedTab == tcFuel.TabPages[0])
    {
        Fuel.setRadial(Convert.ToDouble(nRDC.Text));

        Fuel.setPilot(Convert.ToDouble(nPDC.Text));
    }
    else if (tcFuel.SelectedTab == tcFuel.TabPages["Auto"])
    {

    }
    else if (tcFuel.SelectedTab == tcFuel.TabPages["PID"])
    {

    }

}

private void BFanApply_Click(object sender, EventArgs e)
{
    if (bFanAuto.Checked)
    {
        if (oilTemp.autoFan == false)
```

```
                {
                    oilTemp.autoFan = true;
                }
            }
            else if (bFanOn.Checked)
            {
                if (oilTemp.autoFan == true)
                {
                    oilTemp.autoFan = false;
                }

                oilTemp.setFan(true);


            }
            else if (bFanOff.Checked)
            {
                if (oilTemp.autoFan == true)
                {
                    oilTemp.autoFan = false;
                }

                oilTemp.setFan(false);
            }
        }

        private void ToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            ManCtrl manCtrl = Application.OpenForms["ManCtrl"] != null ? (ManCtrl)
Application.OpenForms["ManCtrl"] : null;

            if (manCtrl != null)
            {
                manCtrl.BringToFront();
            }
            else
            {
                manCtrl = new ManCtrl();
                manCtrl.Show();
            }

        }

    }
}
```

Arquivo “Start.cs”:

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;

namespace Serial
{
    public partial class Start : Form
```

```
    {
        public Start()
        {
            InitializeComponent();

            InitializeBackgroundWorker();
        }

        private void InitializeBackgroundWorker()
        {
            backgroundWorker1.DoWork += new
DoWorkEventHandler(backgroundWorker1_DoWork);

            backgroundWorker1.RunWorkerCompleted += new
RunWorkerCompletedEventHandler(backgroundWorker1_RunWorkerCompleted);

            backgroundWorker1.ProgressChanged += new
ProgressChangedEventHandler(backgroundWorker1_ProgressChanged);
        }

        private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            BackgroundWorker worker = sender as BackgroundWorker;


            startSequence(worker, e);
        }

        private void backgroundWorker1_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {

            if (e.Error != null)
            {
                MessageBox.Show(e.Error.Message);
            }
            else if (e.Cancelled)
            {
                MessageBox.Show("Cancelled!!");
                Stop.ExternalStop();
            }
            else
            {
                MessageBox.Show("Sucesso!");
                Form1.isRunning = true;
            }

            this.Close();

        }

        void startSequence(BackgroundWorker worker, DoWorkEventArgs e)
        {
            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            //"Starting Pump...";
            worker.ReportProgress(10);
```

```
            //Start oil pump:
            bool ans = oilPump.setPump(true);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Oil pump switch)", "Connection
Erro!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;
            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            Thread.Sleep(20);

            //Set oil pump DC to 30%:
            ans = oilPump.setDC(30);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Oil pump DC)", "Connection Erro!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;
            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }


            //"Waiting for oil pressure...";
            worker.ReportProgress(20);

            Thread.Sleep(1000);

            double oilPress = Form1.readData[9];

            //Waiting for oil pressure:
            while (oilPress < 50)
            {
                Thread.Sleep(500);
                oilPress = Form1.readData[9];
                if (worker.CancellationPending)
                {
                    e.Cancel = true;
                    return;
                }
            }

            //"Starting fan. Waiting for RPM...";
            worker.ReportProgress(30);
```

```
            //Starter ON:
            ans = Fuel.setStarter(true);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Starter)", "Connection Erro!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;

            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            Thread.Sleep(10000);


            //"Starting combustion...";
            worker.ReportProgress(40);

            //Open fuel solenoid:
            ans = Fuel.setSolenoid(true);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Fuel Solenoid)", "Connection
Erro!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;

            }
            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            Thread.Sleep(20);

            //Start ignitor
            ans = Fuel.setIgnitor(true);


            if (ans == false)
            {

                MessageBox.Show("Incorrect answear (Ignitor)", "Connection Erro!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;
            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }
```

```
            Thread.Sleep(20);

            //Radial DC @20%:
            ans = Fuel.setRadial(20);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Fuel Injector)", "Connection
Erro!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;

            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            double egt = Form1.readData[13];

            //Wating for EGT to rise:
            while (egt < 100)
            {
                Thread.Sleep(1000);

                egt = Form1.readData[13];

                if (worker.CancellationPending)
                {
                    e.Cancel = true;
                    return;
                }
            }

            //Stop ignitor
            ans = Fuel.setIgnitor(false);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Ignitor)!","Connection Erro!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;
            }

            Thread.Sleep(20);

            //"EGT threshold reached. Accelerating to ~25000RPM."
            worker.ReportProgress(50);

            //Pilot DC @20%:
            ans = Fuel.setPilot(20);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Fuel Injector)", "Connection
Erro!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
```

```
                return;

            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            Thread.Sleep(20);

            //Radial DC @30%:
            ans = Fuel.setRadial(30);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Fuel Injector)", "Connection
Erro!", MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;

            }

            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }

            Thread.Sleep(20);

            oilTemp.autoFan = true;

            double rpm1 = Form1.readData[10];

            while (rpm1 < 24500)
            {
                Thread.Sleep(500);

                rpm1 = Form1.readData[10];

                if (worker.CancellationPending)
                {
                    e.Cancel = true;
                    return;
                }
            }

            Thread.Sleep(20);

            //Starter OFF:
            ans = Fuel.setStarter(false);

            if (ans == false)
            {
                MessageBox.Show("Incorrect answear (Starter)", "Connection Erro!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                e.Cancel = true;
                return;
```

```
            }
            if (worker.CancellationPending)
            {
                e.Cancel = true;
                return;
            }
        }

        private void backgroundWorker1_ProgressChanged(object sender,
ProgressChangedEventArgs e)
        {

            if (e.ProgressPercentage == 10) { this.label1.Text = "Starting Pump..."; }
            if (e.ProgressPercentage == 20) { this.label1.Text = "Waiting for oil
pressure..."; }
            if (e.ProgressPercentage == 30) { this.label1.Text = "Starting fan.
Waiting for RPM..."; }
            if (e.ProgressPercentage == 40) { this.label1.Text = "Starting
combustion..."; }
            if (e.ProgressPercentage == 50) { this.label1.Text = "EGT threshold
reached. Accelerating to ~25000RPM."; }

        }


        private void BAbort_Click(object sender, EventArgs e)
        {
            if (backgroundWorker1.IsBusy)
            {
                this.backgroundWorker1.CancelAsync();
            }
            else
            {
                this.Close();
            }
        }

        private void BStart_Click(object sender, EventArgs e)
        {
            bStart.Enabled = false;
            backgroundWorker1.RunWorkerAsync();
        }

    }
}
```

Arquivo “Fuel.cs”:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Serial
{
    class Fuel
```

```
    {
        public static double radialDC = 0.00;

        public static double pilotDC = 0.00;

        public static bool fuelOpen = false;

        public static bool ignitorOn = false;

        public static bool starterOn = false;

        public static bool fuelOff = false;


        public static bool setRadial(double set)
        {

            if (radialDC == set)
            {
                return true;
            }

            byte bRdc = (byte) (255 * set * 0.01);

            //ANALOG, PIN 2, DC
            byte[] data = { 0x03, 0x30, bRdc, 0 };

            data[3] = (byte)((data[1] + data[2]) & 0xFF);

            string ans = serialCOM.sendData(data);

            if (ans == Convert.ToString(data[1]))
            {
                radialDC = set;

                fuelOff = false;

                return true;
            }
            else if (ans == Convert.ToString(data[1]+1))
            {
                MessageBox.Show("Fuel is off", "Message", MessageBoxButtons.OK, Me
ssageBoxIcon.Error);

                fuelOff = true;

                return false;
            }
            else
            {

                return false;
            }
        }

        public static bool setPilot(double set)
        {
```

```
            if(pilotDC == set)
            {
                return true;
            }

            byte bPdc = (byte)(255 * set * 0.01);

            //ANALOG, PIN 3, DC
            byte[] data = { 0x03, 0x31, bPdc, 0};

            data[3] = (byte)((data[1] + data[2]) & 0xFF);

            string ans = serialCOM.sendData(data);

            if (ans == Convert.ToString(data[1]))
            {
                pilotDC = set;

                fuelOff = false;

                return true;
            }
            else if (ans == Convert.ToString(data[1] + 1))
            {
                MessageBox.Show("Fuel is off", "Message", MessageBoxButtons.OK, Me
ssageBoxIcon.Error);

                fuelOff = true;

                return false;
            }
            else
            {
                return false;
            }
        }

        public static bool setSolenoid(bool set)
        {
            if(fuelOpen == set)
            {
                return true;
            }

            byte bFuelOpen = Convert.ToByte(set);

            //DIGITAL, PIN 35, SET
            byte[] data = { 0x04, 0x11, 0x23, bFuelOpen, 0};

            data[4] = (byte)((data[1] + data[2] + data[3]) & 0xFF);

            string ans = serialCOM.sendData(data);

            if (ans == Convert.ToString(data[1]))
            {
                fuelOpen = set;

                return true;
```

```
        }
        else
        {
            return false;
        }

    }

    public static bool setIgnitor(bool set)
    {
        if(ignitorOn == set)
        {
            return true;
        }

        byte bIgnitorOn = Convert.ToByte(set);

        //DIGITAL, PIN 39, SET
        byte[] data = { 0x04, 0x11, 0x27, bIgnitorOn, 0};

        data[4] = (byte)((data[1] + data[2] + data[3]) & 0xFF);

        string ans = serialCOM.sendData(data);

        if (ans == Convert.ToString(data[1]))
        {
            ignitorOn = set;

            return true;
        }
        else
        {
            return false;
        }

    }

    public static bool setStarter(bool set)
    {
        if (starterOn == set)
        {
            return true;
        }

        byte bStarterOn = Convert.ToByte(set);

        //DIGITAL, PIN 41, DC
        byte[] data = { 0x04, 0x11, 0x29, bStarterOn, 0};

        data[4] = (byte)((data[1] + data[2] + data[3]) & 0xFF);

        string ans = serialCOM.sendData(data);

        if (ans == Convert.ToString(data[1]))
        {
            starterOn = set;

            return true;
```

```
            }
            else
            {
                return false;
            }

        }

    }

}
```

Arquivo “ManCrtl.cs”:

```
using System;
using System.Windows.Forms;

namespace Serial
{
    public partial class ManCtrl : Form
    {
        private bool _IJlinked = false;
        private bool IJlinked
        {
            get
            {
                return _IJlinked;
            }
            set
            {
                _IJlinked = value;
                IJlink.Text = _IJlinked ? "Unlink" : "Link";
            }
        }

        public ManCtrl()
        {
            InitializeComponent();
        }

        private void FSon_Click(object sender, EventArgs e)
        {
            Fuel.setSolenoid(true);
        }

        private void FSoff_Click(object sender, EventArgs e)
        {
            Fuel.setSolenoid(false);
        }

        private void OFon_Click(object sender, EventArgs e)
        {
            oilTemp.setFan(true);
        }

        private void OFoff_Click(object sender, EventArgs e)
```

```
{
    oilTemp.setFan(false);
}

private void IGon_Click(object sender, EventArgs e)
{
    Fuel.setIgnitor(true);
}

private void IGoff_Click(object sender, EventArgs e)
{
    Fuel.setIgnitor(false);
}

private void STon_Click(object sender, EventArgs e)
{
    Fuel.setStarter(true);
}

private void SToff_Click(object sender, EventArgs e)
{
    Fuel.setStarter(false);
}

private void AUon_Click(object sender, EventArgs e)
{

}

private void OPon_Click(object sender, EventArgs e)
{
    oilPump.setPump(true);
}

private void OPoff_Click(object sender, EventArgs e)
{
    oilPump.setPump(false);
}

private void OPset_Click(object sender, EventArgs e)
{
    oilPump.setDC(Convert.ToInt32(nPump.Value));
}

private void IJoff_Click(object sender, EventArgs e)
{
    nRadial.Value = 0;
    nPilot.Value = 0;
}

private void IR0_Click(object sender, EventArgs e)
{
    nRadial.Value = 0;
}

private void IP0_Click(object sender, EventArgs e)
{
    nPilot.Value = 0;
```

```
}

private void IRm_Click(object sender, EventArgs e)
{
    decimal value = nRadial.Value - 10;

     if (value < 0)
    {
        value = 0;
    }

    nRadial.Value = value;

}

private void IPm_Click(object sender, EventArgs e)
{
    decimal value = nPilot.Value - 10;

    if (value < 0)
    {
        value = 0;
    }

    nPilot.Value = value;
}

private void IRp_Click(object sender, EventArgs e)
{
    decimal value = nRadial.Value + 10;

    if(value > 95)
    {
        value = 95;
    }

    nRadial.Value = value;
}

private void IPp_Click(object sender, EventArgs e)
{
    decimal value = nPilot.Value + 10;

    if (value > 95)
    {
        value = 95;
    }

    nPilot.Value = value;
}

private void nRadial_ValueChanged(object sender, EventArgs e)
{
    Fuel.setRadial(Convert.ToInt32(nRadial.Value));

    if (Fuel.fuelOff == true)
    {
        nRadial.Value = 0;
```

```
            return;
        }

        if (IJlinked == true)
        {
            nPilot.Value = nRadial.Value;
        }
    }

    private void nPilot_ValueChanged(object sender, EventArgs e)
    {
        Fuel.setPilot(Convert.ToInt32(nPilot.Value));

        if (Fuel.fuelOff == true)
        {
            nPilot.Value = 0;
           return;
        }

        if (IJlinked == true)
        {
            nRadial.Value = nPilot.Value;
        }
    }


    private void CbFS_CheckedChanged(object sender, EventArgs e)
    {
        FSon.Enabled = (cbFS.CheckState == CheckState.Checked);
        FSoff.Enabled = (cbFS.CheckState == CheckState.Checked);
    }

    private void CbOF_CheckedChanged(object sender, EventArgs e)
    {
        OFon.Enabled = (cbOF.CheckState == CheckState.Checked);
        OFoff.Enabled = (cbOF.CheckState == CheckState.Checked);
    }

    private void CbIG_CheckedChanged(object sender, EventArgs e)
    {
        IGon.Enabled = (cbIG.CheckState == CheckState.Checked);
        IGoff.Enabled = (cbIG.CheckState == CheckState.Checked);
    }

    private void CbST_CheckedChanged(object sender, EventArgs e)
    {
        STon.Enabled = (cbST.CheckState == CheckState.Checked);
        SToff.Enabled = (cbST.CheckState == CheckState.Checked);
    }

    private void CbAU_CheckedChanged(object sender, EventArgs e)
    {
        AUon.Enabled = (cbAU.CheckState == CheckState.Checked);
        AUoff.Enabled = (cbAU.CheckState == CheckState.Checked);
    }

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

```
            OPon.Enabled = (cbOP.CheckState == CheckState.Checked);
            OPoff.Enabled = (cbOP.CheckState == CheckState.Checked);
            OPset.Enabled = (cbOP.CheckState == CheckState.Checked);
            nPump.Enabled = (cbOP.CheckState == CheckState.Checked);
        }

        private void CbIJ_CheckedChanged(object sender, EventArgs e)
        {
            IJoff.Enabled = (cbIJ.CheckState == CheckState.Checked);

            IR0.Enabled = (cbIJ.CheckState == CheckState.Checked);
            IP0.Enabled = (cbIJ.CheckState == CheckState.Checked);

            IRm.Enabled = (cbIJ.CheckState == CheckState.Checked);
            IPm.Enabled = (cbIJ.CheckState == CheckState.Checked);

            IRp.Enabled = (cbIJ.CheckState == CheckState.Checked);
            IPp.Enabled = (cbIJ.CheckState == CheckState.Checked);

            nRadial.Enabled = (cbIJ.CheckState == CheckState.Checked);
            nPilot.Enabled = (cbIJ.CheckState == CheckState.Checked);

            IJlink.Enabled = (cbIJ.CheckState == CheckState.Checked);
        }

        private void IJlink_Click(object sender, EventArgs e)
        {
            IJlinked = !IJlinked;
        }
    }
}
```

Arquivo “oilPump.cs”:

```
using System;

namespace Serial
{
    class oilPump
    {
        public static bool On = false;

        public static int DC = 0;

        public static bool setDC(int set)
        {

            if (DC == set)
            {
                return true;
            }

            byte bDC = (byte)(255 * 0.01 * set);

            //ANALOG, PIN 13, DC
            byte[] data = { 0x04, 0x10, 0xD, bDC, 0 };
```

```
            data[4] = (byte)((data[1] + data[2] + data[3]) & 0xFF);

            string ans = serialCOM.sendData(data);

            if (ans == Convert.ToString(data[1]))
            {
                DC = set;

                return true;
            }
            else
            {
                return false;
            }
        }

        public static bool setPump(bool set)
        {
            if (On == set)
            {
                return true;
            }

            byte bOn = Convert.ToByte(set);

            //DIGITAL, PIN 45, SET
            byte[] data = { 0x04, 0x11, 0x2D, bOn, 0 };

            data[4] = (byte)((data[1] + data[2] + data[3]) & 0xFF);

            string ans = serialCOM.sendData(data);

            string a = Convert.ToString(data[1]);

            if (ans == Convert.ToString(data[1]))
            {
                On = set;

                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

Arquivo “oilTemp.cs”:

```
using System;
using System.Threading;
using System.ComponentModel;
```

```
namespace Serial
{
    class oilTemp
    {
        public static bool fanOn;

        private static System.ComponentModel.BackgroundWorker bwOilTemp;

        private static bool _autoFan = false;
        public static bool autoFan
        {
            get
            {
                return _autoFan;
            }

            set
            {
                _autoFan = value;

                if (_autoFan == true)
                {
                    bwOilTemp.RunWorkerAsync();
                }
            }
        }


        public static void InitializeBackgroundWorkerOilTemp()
        {
            oilTemp.bwOilTemp = new System.ComponentModel.BackgroundWorker();
            bwOilTemp.DoWork += new DoWorkEventHandler(bwOilTemp_DoWork);
        }

        private static void bwOilTemp_DoWork(object sender, DoWorkEventArgs e)
        {
            BackgroundWorker worker = sender as BackgroundWorker;


            OilTempMonitoring(worker, e);
        }

        private static void OilTempMonitoring(BackgroundWorker worker, DoWorkEvent
Args e)
        {
            while (autoFan == true)
            {

                if (Form1.readData[15] > 100)
                {
                    if (fanOn == false)
                    {
                        setFan(true);
                    }
                }

                if (Form1.readData[15] < 90)
                {
```

```
                    if (fanOn == true)
                    {
                        setFan(false);
                    }
                }

                Thread.Sleep(5000);
            }
        }

        public static bool setFan(bool set)
        {
            if (fanOn == set)
            {
                return true;
            }

            byte bFanOn = Convert.ToByte(set);

            //DIGITAL, PIN 37, SET
            byte[] data = { 0x04, 0x11, 0x25, bFanOn, 0 };

            data[4] = (byte)((data[1] + data[2] + data[3]) & 0xFF);

            string ans = serialCOM.sendData(data);

            if (ans == Convert.ToString(data[1]))
            {
                fanOn = set;

                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```

Arquvio “serialCOM.cs”:

```
using System;
using System.Windows.Forms;
using System.IO.Ports;
using System.Threading;
using System.Diagnostics;

namespace Serial
{

    class serialCOM
    {
        private static readonly Object serialLock = new Object();

        public static SerialPort serialPort1 = new SerialPort("COM1");
```

```
        public static bool reading = false;

        public static bool serialTimeOut = false;

        static Stopwatch serialT = new Stopwatch();

        public static bool openSerial(string _port)
        {
            serialTimeOut = false;

            serialPort1.PortName = _port;
            serialPort1.BaudRate = 57600;
            serialPort1.Parity = Parity.None;
            serialPort1.StopBits = StopBits.One;
            serialPort1.DataBits = 8;
            serialPort1.Handshake = Handshake.None;
            serialPort1.ReadTimeout = 200;
            serialPort1.DtrEnable = true;
            serialPort1.NewLine = "\r\n";

            try
            {
                serialPort1.Open();
                Thread.Sleep(100);
                serialPort1.DtrEnable = false;
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Message", MessageBoxButtons.OK, Messa
geBoxIcon.Error);
                return false;
            }

            Thread.Sleep(1000);

            byte[] bytestosend = { 0xF0 };
            serialPort1.Write(bytestosend, 0, 1);
            byte[] ans = { 0 };

            try
            {
                serialPort1.Read(ans, 0, 1);
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Message", MessageBoxButtons.OK, Messa
geBoxIcon.Error);
            }

            if (ans[0] == 0xF0)
            {
                return true;
            }

            else
```

```
            {

                serialPort1.Close();
                MessageBox.Show("Incorrect answear", "Connection Erro!", MessageBo
xButtons.OK, MessageBoxIcon.Error);
                return false;
            }
        }

        public static void closeSerial()
        {
            while (reading == true)
            {
                Thread.Sleep(100);
            }

            try
            {
                serialPort1.DtrEnable = true;
                Thread.Sleep(100);
                serialPort1.DtrEnable = false;
                Thread.Sleep(10);

                serialPort1.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Message", MessageBoxButtons.OK, Messa
geBoxIcon.Error);
            }
        }

        public static string sendData(byte[] data)
        {
            if (serialPort1.IsOpen == true)
            {
                if(serialT.IsRunning == false)
                {
                    serialT.Start();
                }

                while (serialT.ElapsedMilliseconds < 20)
                {

                }

                lock (serialLock)
                {
                    reading = true;
                    byte[] bytestosend = data;
                    int lenght = bytestosend.Length;
                    string sb = "0xFE";
                    int i = 0;

                    while (sb == "0xFE" && i < 10 && serialPort1.IsOpen == true)
                    {
                        serialPort1.Write(bytestosend, 0, lenght);
```

```
                        try
                        {
                            sb = serialPort1.ReadLine();
                        }

                        catch (Exception ex)
                        {
                            MessageBox.Show(ex.Message, "Message", MessageBoxButto
ns.OK, MessageBoxIcon.Error);
                        }
                        if (sb == "0xFE")
                        {
                            Thread.Sleep(20);
                            i++;
                        }

                        if (sb == "0xFF" | i == 10)
                        {
                            closeSerial();
                            serialTimeOut = true;
                        }
                    }

                    reading = false;
                    return sb;
                }

            }
            else
            {
                return "0xFE";
            }
        }

        public static bool initialize()
        {
            byte[] bytestosend = { 0xF0 };
            serialPort1.Write(bytestosend, 0, 1);
            byte[] ans = { 0 };

            try
            {
                serialPort1.Read(ans, 0, 1);
            }

            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Message", MessageBoxButtons.OK, Messa
geBoxIcon.Error);
            }

            if (ans[0] == 0xF0)
            {
                return true;
            }

            else
            {
```

```
                serialPort1.Close();
                MessageBox.Show("Incorrect answear", "Connection Erro!", MessageBo
xButtons.OK, MessageBoxIcon.Error);
                return false;
            }
        }
    }
}
```

Arquivo “Stop.cs”:

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Threading;

namespace Serial
{
    public partial class Stop : Form
    {
        public Stop()
        {
            InitializeComponent();

            InitializeBackgroundWorker();
        }

        private void InitializeBackgroundWorker()
        {
            backgroundWorker1.DoWork += new DoWorkEventHandler(backgroundWorker1_D
oWork);

            backgroundWorker1.RunWorkerCompleted += new RunWorkerCompletedEventHan
dler(backgroundWorker1_RunWorkerCompleted);

            backgroundWorker1.ProgressChanged += new ProgressChangedEventHandler(b
ackgroundWorker1_ProgressChanged);
        }

        private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
        {
            BackgroundWorker worker = sender as BackgroundWorker;


            stopSequence(worker, e);
        }

        private void backgroundWorker1_RunWorkerCompleted(object sender, RunWorker
CompletedEventArgs e)
        {

            if (e.Error != null)
            {
                MessageBox.Show(e.Error.Message);
            }
```

```
            else
            {

                MessageBox.Show("Sucesso!");
            }

            this.Close();

        }

        private void backgroundWorker1_ProgressChanged(object sender, ProgressChan
gedEventArgs e)
        {

            if (e.ProgressPercentage == 10) { this.label1.Text = "Cutting fuel..."
; }
            if (e.ProgressPercentage == 20) { this.label1.Text = "Wating for turbi
ne to stop..."; }
            if (e.ProgressPercentage == 30) { this.label1.Text = "Stoping oil pump
..."; }

        }

        private void stopSequence(BackgroundWorker worker, DoWorkEventArgs e)
        {
            Form1.isRunning = false;

            //Cutting fuel:
            worker.ReportProgress(10);

            Fuel.setPilot(0);

            Thread.Sleep(20);

            Fuel.setRadial(0);

            Thread.Sleep(20);

            Fuel.setSolenoid(false);

            Thread.Sleep(20);

            Fuel.setIgnitor(false);

            Thread.Sleep(20);

            Fuel.setStarter(false);

            Thread.Sleep(20);

            worker.ReportProgress(20);

            double rpm = Form1.readData[10];

            while (rpm > 1000)
            {
                Thread.Sleep(1000);
                rpm = Form1.readData[10];
```

```
        }

        if (cbStopPump.Checked)
        {
            worker.ReportProgress(30);

            oilPump.setDC(0);

            oilPump.setPump(false);

            oilTemp.autoFan = false;
        }


    }

    public static void ExternalStop()
    {
        Form1.isRunning = false;

        Fuel.setPilot(0);

        Thread.Sleep(20);

        Fuel.setRadial(0);

        Thread.Sleep(20);

        Fuel.setSolenoid(false);

        Thread.Sleep(20);

        Fuel.setIgnitor(false);

        Thread.Sleep(20);

        Fuel.setStarter(false);

        Thread.Sleep(20);

        //worker.ReportProgress(20);

        double rpm = Form1.readData[10];

        while (rpm > 1000)
        {
            Thread.Sleep(1000);
            rpm = Form1.readData[10];
        }

        //worker.ReportProgress(30);

        oilPump.setDC(0);

        oilPump.setPump(false);

        oilTemp.autoFan = false;
    }
```

```
        private void BCancel_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void BStop_Click(object sender, EventArgs e)
        {
            bCancel.Enabled = false;
            bStop.Enabled = false;
            backgroundWorker1.RunWorkerAsync();
        }
    }
}
```