INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA

DANIEL PEREIRA

DESENVOLVIMENTO DE PLATAFORMAS PARA CONTROLE DO LASER YLPN-1-1x120-50-M

FLORIANÓPOLIS, 2019

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA

DANIEL PEREIRA

DESENVOLVIMENTO DE PLATAFORMAS PARA CONTROLE DO LASER YLPN-1-1x120-50-M

Trabalho de conclusão de curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de engenheiro eletrônico.

Orientador: Me. Samir Bonho

FLORIANÓPOLIS, 2019

Pereira, Daniel
Desenvolvimento de Plataformas Para Controle Do Laser YLPN-1-1x120-50-M/ Daniel Pereira; orientador, Me. Samir Bonho; coorientador, Me. Cláudio Abilio – Florianópolis, SC 2019.
75 paginas

Trabalho de Conclusão de Curso de Engenharia Eletrônica – Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

1. Controle de Laser. 2. Python. 3. Interface humana máquina. 4 Microcontrolador. 5. Raspbery pi I.Me. Samir Bonho. II. Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina III. Plataformas Para Controle do Laser YLPN-1-1x120-50-M

DESENVOLVIMENTO DE PLATAFORMAS PARA CONTROLE DO LASER YLPN-1-1x120-50-M

Daniel Pereira

Este trabalho foi julgado adequado para obtenção do título de Engenheiro Eletrônico em 2019 e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 12 de dezembro, 2019

Banca Examinadora:

Prof. Samir Bonho, Me
Orientador

Prof. Cláudio Abilio, Dr.
Coorientador

Prof. Fernando Santana Pacheco, Dr

Prof. Pedro Giassi Júnior, Dr

AGRADECIMENTOS

Agradeço à minha família: meu pai, por motivar, guiar e servir de inspiração; minha mãe, por todo apoio e por acreditar em meu potencial; e minha irmã por toda ajuda e motivação.

Minha namorada, por crer em meu potencial, dar-me o apoio de que tanto necessitei e ajudar sempre em quaisquer momentos.

Aos meus amigos por todo apoio e suporte que me ajudaram a chegar aonde estou, alcançando meus objetivos.

Ao Instituto Federal de Santa Catarina (IFSC) pelo ensino gratuito e de qualidade, proporcionando-me todos os recursos necessários para alcançar esse nível de formação.

Ao Laboratório de Mecânica de Precisão (LMP) da Universidade Federal de Santa Catarina (UFSC) por permitir o uso seus equipamentos e me oferecer orientação sempre que preciso.

Ao meu orientador e coorientador, Me. Samir Bonho e Me. Cláudio Abílio, pelo apoio nas pesquisas, paciência na hora de auxiliar nas dúvidas, no ensino e em todo suporte recebido.

RESUMO

Lasers são aplicados em muitos processos, como leitura de cds, sensores, equipamentos médicos, cortes de peças e muitos outros. Este trabalho tem como propósito o estudo e compreensão do funcionamento do laser YLPN-1-1x120-50-M da *IPG Photonics*, além de produzir dois sistemas de controle visando facilitar seu uso, possibilitando ao usuário manipular fatores do laser durante seu funcionamento ou antes (de acordo com a plataforma de controle a ser usada). O primeiro sistema tem como foco ajustes rápidos no desempenho e funcionamento do laser enquanto o mesmo está em uso, utilizando o microcontrolador ATmega2560 na linguagem C. Já o segundo sistema, utilizando-se a linguagem *Python* (versão 3.0), é direcionado para uso em um computador com Windows e/ou Linux, provendo uma melhor interface para o usuário e maior número de opções para o controle do laser que podem ser ajustadas antes do mesmo ser ativado. A montagem das plataformas foi efetuada com sucesso, e a utilização de ambas pelo usuário mostrou-se acessível sem a necessidade de conhecimentos avançados sobre o funcionamento do laser.

Palavras-Chave: Laser. Controle. Interface Humano-Máquina. C. Python. Windows. Linux.

ABSTRACT

Lasers are applied in many processes, such as reading CDs, sensors, medical equipment, cutting parts and many others. This work aims to study and understand the operation of the YLPN-1-1x120-50-M laser from IPG Photonics, in addition to producing two control systems to facilitate its use, allowing the user to manipulate laser factors during its operation or before (according to the control platform to be used). The first system focuses on quick adjustments in laser performance and operation while it is in use, using the ATmega2560 microcontroller in C language. The second system, using the Python language (version 3.0), is intended for use in a computer with Windows / Linux, providing a better user interface and a greater number of options to control the laser that can be adjusted before it is activated. The assembly of the platforms was carried out successfully, and the use of both by the user proved to be accessible without the need for advanced knowledge about the laser functions.

Keywords: Laser. Control. Human-Machine Interface. C. Python. Windows. Linux.

LISTA DE FIGURAS

LISTA DE SIGLAS

IEC	"*International Electrotechnical Commission*" ou Comissão Eletrotécnica Internacional
OEM	"*Original Equipment Manufacturer*" ou Equipamento de Fabricação Original.
OS	*Operating System* ou Sistema Operacional.
MCC	Microcontrolador.
LMP	Laboratório de Mecânica de Precisão.

SUMÁRIO

1 INTRODUÇÃO

Atualmente, lasers (*light amplification by simulated emission of radiation* ou amplificação de luz por emissão de radiação) têm se propagado em diferentes áreas devido a sua vasta aplicabilidade como sensores, corte de objetos, manufatura aditiva, entre outros.

Devido às suas diversas aplicações, o Laboratório de Mecânica de Precisão (LMP) da Universidade Federal de Santa Catarina (UFSC) foca na utilização dos lasers para experimentos como cortes, soldagens, marcações, texturizações e preparo de amostras utilizando metais e polímeros. Os lasers são produzidos com especificações individuais, dependendo do seu uso e aplicação, podendo ser adquiridos já completos e funcionais, incluindo um sistema de controle e seus suportes, ou apenas o próprio cabeçote e sua fonte, deixando o restante para ser montado ou comprado separadamente.

Este trabalho tem como foco a construção de dois sistemas de controle de fácil utilização para o laser de modelo YLPN-1-1x120-50-M do fabricante IPG Photonics que se encontra no laboratório LMP da Universidade Federal de Santa Catarina (UFSC). Sendo um dos sistemas de controle destinado para ajustes rápidos (onde será utilizado para alterar parâmetros do laser durante seu uso) e para controle completo, permitindo controle completo de todas as funções do laser e seus ajustes antes de cada uso.

1.1 Justificativa

O LMP faz uso de lasers de vários tipos e potências diferentes para realizar experimentos e preparar amostras de metais ou polímeros. Poder utilizá-los com precisão e facilidade é especialmente importante para garantir bons resultados. Dentre todos os lasers presentes no laboratório o mais novo é o modelo YLPN-1-1x120-50-M do fabricante IPG Photonics.

Em vista da grande aplicabilidade dos lasers de alta potência e sua importância para o laboratório, fazendo com que seu desempenho, facilidade e acessibilidade muito relevante. Sendo que o modelo estudado nesse trabalho não veio com interface de controle dos parâmetros do laser acoplada, é esperado que o usuário seja capaz de equipá-lo com uma, montando ou a adquirindo separadamente.

Este laser tem sido usado a partir de uma conexão adaptada a um computador que utiliza Windows 7 através de uma combinação de dois programas: um chamado *Inkspace ("Tutoriais | Inkscape", 2019),* utilizado para interpretar arquivos de imagens e pareá-los com a movimentação do cabeçote do laser; e outro conhecido como *Thermite ("Termite: a simple RS232 terminal", 2017)* que é responsável pela comunicação serial regulando, assim, vários aspectos do funcionamento do sistema.

No entanto, *Thermite* é de difícil uso devido ao fato de ele requerer que seus comandos sejam feitos em hexadecimal, seguindo o exato protocolo solicitado pelo sistema com o qual se deseja comunicar manualmente. Para que isso ocorra, é necessário que haja conhecimento intermediário na área de programação e grande conhecimento dos protocolos de comunicação do laser.

1.2 Definição do problema

A utilização do laser é de grande importância no LMP, possibilitando a realização de muitos experimentos, e para isso, um meio de controle fácil e otimizado se mostra necessário, porém, o custo de uma unidade de controle é alto. Deste modo, a implementação de um sistema mais barato que satisfaça as mesmas funções é de grande interesse, já que as soluções utilizadas atualmente não são otimizadas e não possuem uma plataforma de ajustes rápidos para o funcionamento do laser sem a obrigatoriedade de se utilizar diretamente o computador conectado no sistema.

1.3 Objetivo geral

Projetar dois sistemas de controle dos parâmetros de funcionamento da fonte do laser YLPN-1-1x120-50-M (fabricante IPG Photonics), um para ajustes rápidos que poderá ser utilizado enquanto o laser estiver em funcionamento e outro para controle total das funções do equipamento que apenas pode ser utilizado antes de se iniciar testes, assim regulando seu desempenho e facilitando seu uso.

1.3.1 Objetivo específico

Dentre os objetivos específicos, podem ser elencados:

a) Realizar o estudo do funcionamento do laser;

b) Analisar o protocolo de comunicação do laser;

c) Projetar e aplicar um sistema de controle, para ajustes rápidos que podem ser realizados com o laser em funcionamento, utilizando um microcontrolador, programado na linguagem C;

d) Criar e aplicar uma IHM (interface humano-máquina) multiplataforma (Windows e Linux), voltada ao controle completo do laser.

2. FUNDAMENTOS TEÓRICOS E CONCEITOS

Neste capítulo serão explorados os conceitos teóricos, funcionamento e montagem de todos os equipamentos e teorias relevantes para este trabalho.

2.1 Lasers

Laser, da sigla *light amplification by simulated emission of radiation* ou amplificação de luz por emissão de radiação, é um dispositivo que emite um feixe de luz monocromático (possui comprimento de onda bem definido), coerente (os fótons que compõe o feixe de luz estão em fase) e altamente direcional (sofre baixa difração). Atualmente lasers estão presentes em várias áreas cientificas (EICHLER, 2018) (SINGH, 2012).

Na medicina onde a radiação emitida pelos lasers de baixa potência tem demonstrado efeitos analgésicos, anti-inflamatórios e cicatrizantes, sendo, por isso, bastante utilizados no processo de reparo tecidual (LINS, 2010) (SCHULZE, 2010).

Para usinagem é utilizado feixes de laser em um processo de remoção de material com base em energia térmica assim sendo amplamente utilizados para corte, perfuração, marcação, soldagem, sinterização e tratamento térmico (DUBEY, 2008) (SCIAMANNA, 2015).

Diodos a laser têm inúmeras aplicações, incluindo imagens, sensores, comunicações por fibra óptica e espectroscopia. Inicialmente, visando fornecer uma potência de saída constante, os diodos a laser são hoje comumente usados para produzir pulsos ópticos curtos periódicos a altas taxas de repetição (SCIAMANNA, 2015).

Lasers são classificados, por razões de segurança, de acordo com seu potencial de causar danos aos olhos e à pele. De acordo com a IEC eles podem ser separados da forma vista na tabela 1 (LINS, 2010).

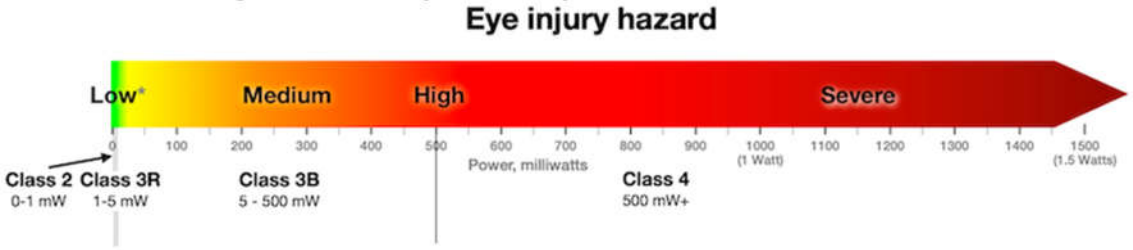
Tabela 1 – Classificação de lasers

Classe 1	Seguro.
Classe 1M	Providenciar óculos de segurança.
Classe 2	Laser visível, seguro se exposição for menor que 0,25 s.
Classe 2M	Laser visível, seguro se exposição for menor que 0,25 s, recomendado equipamentos de segurança visual.
Classe 3R	Não seguro, baixo risco
Classe 3B	Perigoso, porém reflexão do laser não apresenta riscos
Classe 4	Perigoso, reflexão do laser apresenta riscos. Risco de incêndio

Fonte: IEC 60825-1 (2019)

O potencial de causar danos é diretamente relacionado a potência do laser como pode ser visto na figura abaixo (LINS, 2010).

Figura 1 – Relação entre potência e classe do laser



Fonte: IEC 60825-1 (2019)

2.1.1 Lasers de estado sólido

Lasers de estado solido são um tipo específico de laser que utiliza um componente solido como meio ativo (material onde se realiza a amplificação do feixe de laser) ao invés de gases ou líquidos. O primeiro laser deste tipo foi criado em 1960 pelo físico Theodore Harold Maiman (ZERVAS, 2014).

A maioria dos lasers de estado sólido utilizam metais de transição, como Ti (titânio), Cr (crômio) e Co (cobalto) ou terras-raras, como Nd (neodímio), Ho (hólmio), Er (érbio), Tm (túlio) ou Yb (itérbio). Os matérias são escolhidos devido a sua

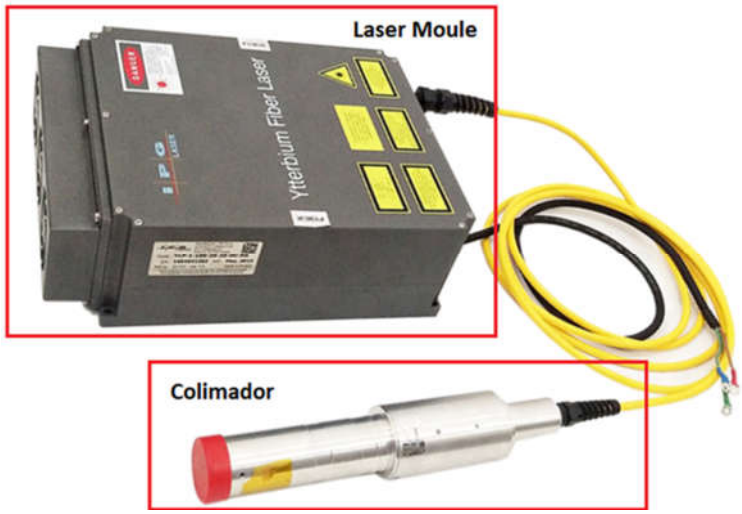
capacidade de continuarem estáveis em altos níveis de energia (ZERVAS, 2014) (EICHLER, H. J., 2018).

2.2 Laser industrial

Lasers industriais são aqueles de classe superiores a 2, utilizados em ambientes industriais para realização de cortes, soldas e escoriações. Considerados perigosos demais para usos domésticos (SVELTO, 2010).

Em sua maioria, os lasers industriais são compostos por dois módulos conhecidos como laser *module* e colimador (figura 2) (SINGH, 2012).

Figura 2 – Modelo de Laser Industrial Utilizado para Corte de Ferro



Fonte: YLPN-1-1x120-50-M Datasheet (2015)

2.2.1 Laser Module

O laser *module* (figura 3) é o dispositivo ligado diretamente ao colimador sendo responsável por sua alimentação e modulação de sinal. Apesar de pôr si só não realizar o controle do laser, é neste módulo em que a comunicação com outros dispositivos ocorre assim possibilitando, através dos comandos recebidos, que o controle seja realizado. O laser module também é responsável por todo o sistema de disparo onde, através de modulação por largura de pulso (PWM), controla o índice de carga e descarga, assim variando sua potência e modo de disparo. *(SINGH, 2012)* (YLPN-1-1x120-50-M Datasheet, 2015).

Figura 3 – Laser Module

Fonte: YLPN-1-1x120-50-M Datasheet (2015)

2.2.2 Colimador

No colimador (figura 4) está presente um sistema de lentes com o intento de focar o feixe a ser expelido em um único ponto, assim garantindo que a potência total do laser não seja desperdiçada atingindo áreas onde não é necessário *(SINGH, 2012).*

Figura 4 – Colimador

Fonte: YLPN-1-1x120-50-M Datasheet (2015)

2.2.3 Sistema de disparo

O laser funciona controlando o ciclo de carga e descarga através da modulação da largura dos pulsos da alimentação de um diodo de alta potência, assim regulando sua frequência de disparo. O feixe de luz oriundo do diodo então irá passar pelo processo de *laser pumping,* onde é intensificado e expelido pelo colimador, cujas

lentes efetuam o foco do laser em um ponto específico. Sua frequência de disparos é controlada pela variação da duração do pulso do laser assim sendo capaz de utilizar feixes mais intensos (para cortes) ou de menor frequência para utilização em marcações (figura 5) *(SINGH, 2012).*

Figura 5 – Sistema de disparo do laser

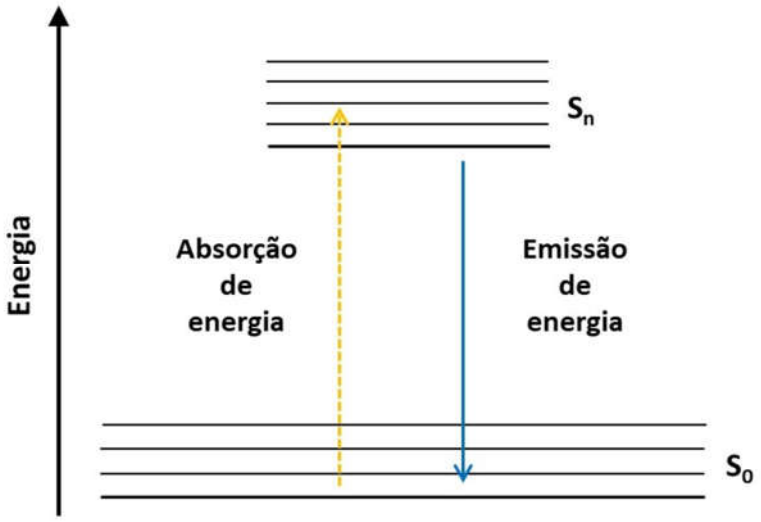
Fonte: Produção própria (2019)

2.2.4 Laser Pumping

Laser *pumping* é o nome dado ao processo de injeção de energia no material emissor do laser (os quais são escolhidos por continuarem estáveis em altos níveis de energia), de maneira a fazer com que os elétrons deste material passem de um nível (S0), de baixa energia, para um estado excitado (Sn), de alta energia. Uma das maneiras de excitar os elétrons para um estado de mais alta energia é a partir do bombardeamento com fótons (que podem ser, por exemplo, provindos do feixe de um laser), e este estado excitado, por ser instável, faz com que os elétrons tenham uma tendência de retornarem ao seu estado fundamental, liberando assim a energia acumulada. Esta energia pode ser liberada na forma de fótons, que, por sua vez, podem atingir outros elétrons e o processo então continuará em cascata. Assumindo que todos os elétrons sejam excitados para um mesmo nível energético eles irão, portanto, liberar energia em um mesmo frequência (a diferença entre os níveis de energia de um átomo ou molécula é quantificada, assim como a energia dos fótons), fazendo com que o feixe do laser resultante desta operação permaneça em um mesmo comprimento de onda (SVELTO, 2010) (THYAGARAJAN; GHATAK, 2010).

No gráfico presente na figura 6, está uma representação da transição de um elétron para uma camada superior (estado excitado "S_n") quando este absorve energia. Quando a energia é liberada este elétron então ira para um estado mais estável (representado por "S_0").

Figura 6 – Transição de elétrons

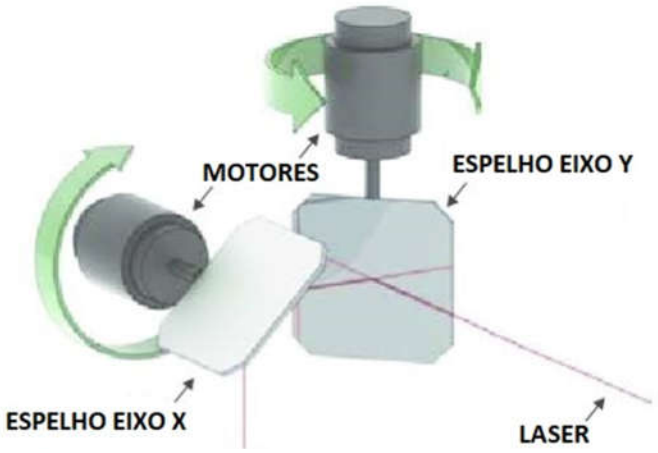


Fonte: Produção própria (2019)

2.2.6 Escâner de varredura óptica

O escâner de varredura óptica funciona refletindo o laser em dois espelhos (com um no eixo x e o outro em y) e, através da movimentação destes se direciona o feixe para o local desejado como mostrado na figura 7. O escâner de varredura óptica é utilizado para direcionar o feixe do laser, de forma eficiente, possibilitando o laser cobrir uma certa área abaixo de si sem a necessidade de sistemas para a movimentação do mesmo (AGV-14HPO Datasheet, 2014).

Figura 7 – Funcionamento dos espelhos



Fonte: Produção própria (2019)

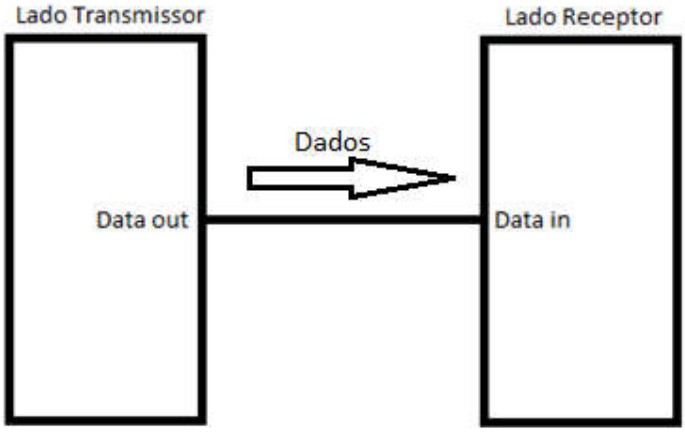
2.3 Comunicação

Dispositivos eletrônicos, para transmitirem dados entre si necessitam se comunicar de alguma forma. Dos vários modos de conexões utilizadas, as mais relevantes para este trabalho serão seriais e paralelas devido a serem as únicas compatíveis com laser estudado (JONES, 2012).

2.3.1 Comunicação Serial

Comunicação serial é o nome dado ao processo de comunicação onde se envia um bit por vez sequencialmente através de um único canal de comunicação (figura 8). Este processo é um dos utilizados por microcontroladores para se comunicarem com sensores e outros dispositivos (JONES, 2012).

Figura 8 – Exemplo de Comunicação Serial.



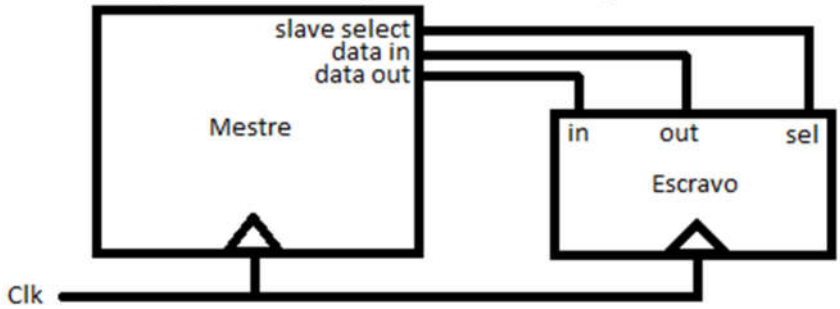
Fonte: Produção Própria (2019)

A velocidade deste tipo de comunicação, conhecida como *baud rate*, é dada em bits por segundo e varia de sistema para sistema (sabendo que alguns sistemas, como Arduinos, de acordo com suas configurações, podem aceitar diferentes velocidades) (BARRETT; PACK, 2006).

Existem diversos protocolos diferentes para este tipo de comunicação, porém o mais relevante para este trabalho, por ser a compatível com o laser, é aquele conhecido como PCI (Peripheral Component Interconnect ou Interconector de Componentes Periféricos). Seu funcionamento consiste de quatro cabos, dois para a

transferência de dados (onde um é utilizado apenas para o recebimento e o outro para envio), um para o sinal que determina a frequência (*clock*) e o último é conhecido como *select* que possibilita um dos componentes (chamado de mestre) selecionar qual outro sistema ligado a ele (escravo) deseja se comunicar naquele determinado momento (figura 9). (JONES, 2012) (BATES, 2008)

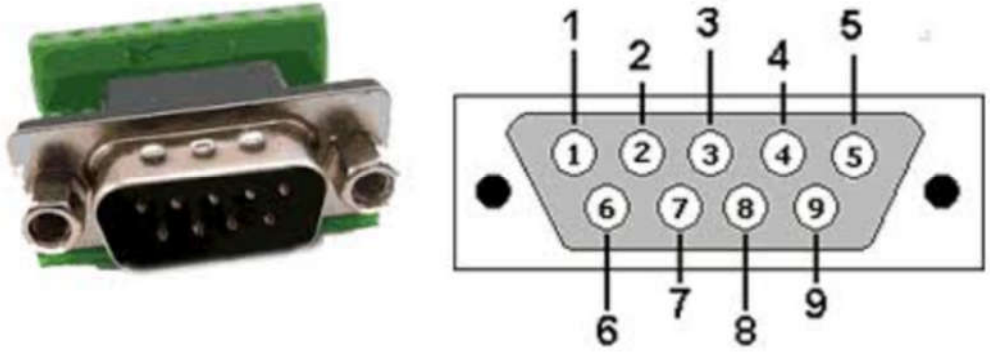
Figura 9 – Exemplo de comunicação PCI.



Fonte: Produção Própria (2019)

Para comunicação serial são mais comumente utilizados cabos do protocolo RS-232 utilizando um conector do tipo DB9 (figura 10).

Figura 10 – Conector DB9



Fonte: Produção própria (2019)

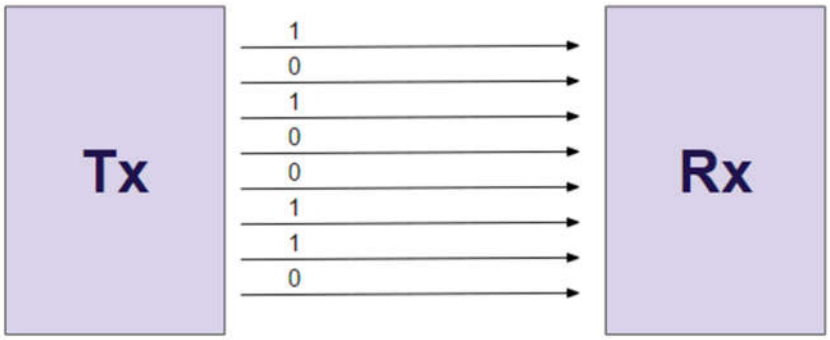
2.3.2 Comunicação Paralela

Comunicação paralela é um método de transmissão de dados que é capaz de enviar uma grande quantidade de *bits* simultaneamente e ao contrário da serial que envia apenas um bit por vez (GEHANI, 1994).

Dispositivos de comunicação paralela possuem um grande número de pinos (figura 11) com a finalidade de transmitir mais dados por vez (onde um conector de 8 bits paralelo possuirá 8 pinos para comunicação de dados), dependendo do protocolo utilizado, também irá possuir um pino para *clock* o qual será responsável por comandar a taxa de transferência de dados ou *latch* que define os momentos em que os dados são transmitidos (JONES, 2012).

Circuitos integrados, dispositivos de memória e peças de computadores costumam possuir este tipo de comunicação devido a maior velocidade de transmissão. (BATES, 2008)

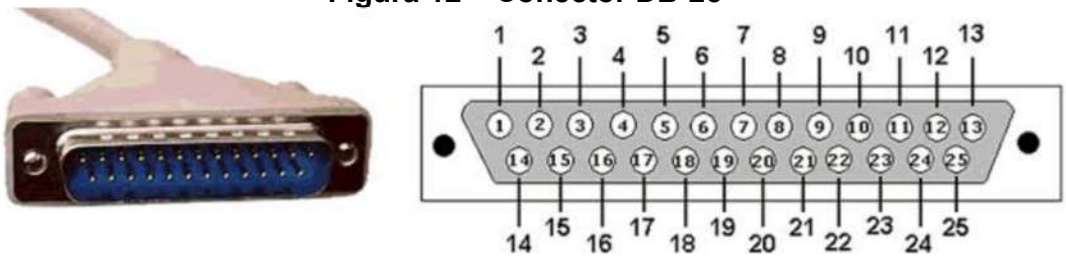
Figura 11 – Exemplo de Comunicação Paralela



Fonte: Produção Própria (2019)

Um dos conectores utilizados para comunicação paralela é o padrão DB-25 como mostrado na figura 12.

Figura 12 – Conector DB-25



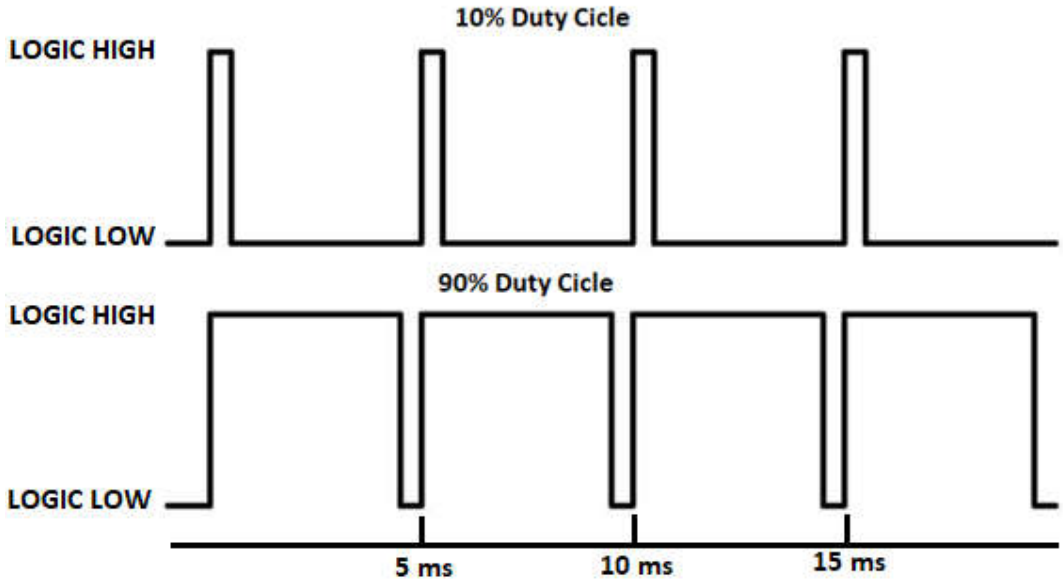
Fonte: Produção própria (2019)

2.4 Modulação por Largura de Pulso

Modulação por largura de pulso (*Pulse Width Modulation*) é o ato de controlar o quanto um sinal permanece em alta (*logic high*) ou em baixa (*logic low*) dentro de um período, através da modulação da razão cíclica (*Duty Cicle*), como demonstrado pela figura 13 e sendo calculado através da equação 1. Podendo ser utilizado para transportar informação, auxiliar na geração de um sinal analógico ou, para casos mais específicos, controlar a razão cíclica do sinal de carga de um laser efetivamente controlando seu disparo (POMILIO, 2014).

$$Duty\ Cicle = 100 * \frac{Largura\ do\ Pulso}{Periodo\ da\ Onda} \qquad (1)$$

Figura 13 - Exemplo de PWM



Fonte: Produção Própria (2019)

2.5 Programação

Para a produção de software, programação de microcontroladores e utilização de sistemas embarcados, programação é uma etapa vital. De acordo com a

necessidade, familiaridade do programador e utilidade pode-se escolher diferentes linguagens de programação (LUTZ, 2011).

2.5.1 Linguagem C

C é uma linguagem de programação para computadores concebida em 1972 para aplicações no sistema operacional Unix (um dos primeiros sistemas operacionais implementados em uma linguagem diferente de *assembly*). Foi criada pelo cientista Dennis Ritchie como uma melhora da linguagem de programação B. No ano de 1973 a versão 4 de Unix teve seu *kernel* (função em computadores que permite a interção entre *softwares* e *hardware*) implementado, em maioridade, na linguagem C. Em 1978 a primeira edição do livro The C Programming Language (escrito por Brian Kernighan e Dennis Ritchie) foi publicado, disseminando o conhecimento e tornando a utilização de C mais comum (KERNIGHAN; RITCHIE, 1998).

Entre as ferramentas disponíveis para a linguagem estão comandos fixos (*if, else, for, while, switch*), operadores aritméticos (+, +=, ++), operadores lógicos (&, ||), capacidade de criar funções (etapas de código destinadas a realizar uma tarefa específica podendo ou não receber dados de entrada e emitir uma saída), definição de componentes, etc. (KERNIGHAN; RITCHIE, 1998)

Sendo uma linguagem versátil ela possui uma grande quantidade de usos que variam da criação de programas para uso em computadores, programação de microcontroladores, até mesmo criação de jogos, etc. (BATES, 2008)

2.5.2 Linguagem Python

Criada em 1989 no Instituto de Pesquisa Nacional Holandês para Matemática e Ciência da Computação (CWI) por Guido van Rossum, Python tem como base a linguagem ABC (linguagem de programação de alto nível cuja principal aplicação é direcionada ao ensino devido a sua simplicidade) e possui muitas similaridades com a sintaxe utilizada em C, utilizando-se por exemplo dos comandos if, else, for, while porém contendo vários próprios para si e derivados destes como elif (LUTZ, 2011).

Sua versão 1.0 foi lançada em janeiro de 1994 contendo instruções como lambda, map e filter. O *upgrade* para 2.0 incluiu compreensão de listas e a sintaxe utilizada para esta versão é bastante similar a *Haskell* (linguagem de programação

funcional). Em dezembro de 2008 a versão 3.0 (mais atual) foi lançada visando corrigir falhas presentes em suas iterações anteriores, certas mudanças na sintaxe o que afetou a compatibilidade com as versões anteriores, necessitando adaptar códigos caso necessário (PADMANABHAN, 2016).

Sendo uma linguagem de alto nível (ao contrário de linguagens como *assembly)*, Python apresenta um grande grau de abstração, isto é, não está diretamente relacionada à arquitetura do computador, possuindo instruções abstratas (instruções que não são convertidas diretamente para a linguagem de máquina, sendo compostas por várias outras instruções menos complexas em sequência, facilitando o ato de programação). Por ser de alto nível a utilização desta linguagem para a criação de códigos mais extensos apresenta menos complexidade que fazer o mesmo utilizando outras linguagens como C ou *assembly*, por exemplo (SUMMERFIELD, 2015).

2.6 Microcontroladores

Microcontroladores podem ser compreendidos como circuitos integrados que funcionam essencialmente como pequenos computadores, capazes de serem programados para realizar as funções específicas pelas quais serão utilizados. Eles são chips que contem memória, interfaces periféricas e um processador (SUSNEA; MITESCU, 2006).

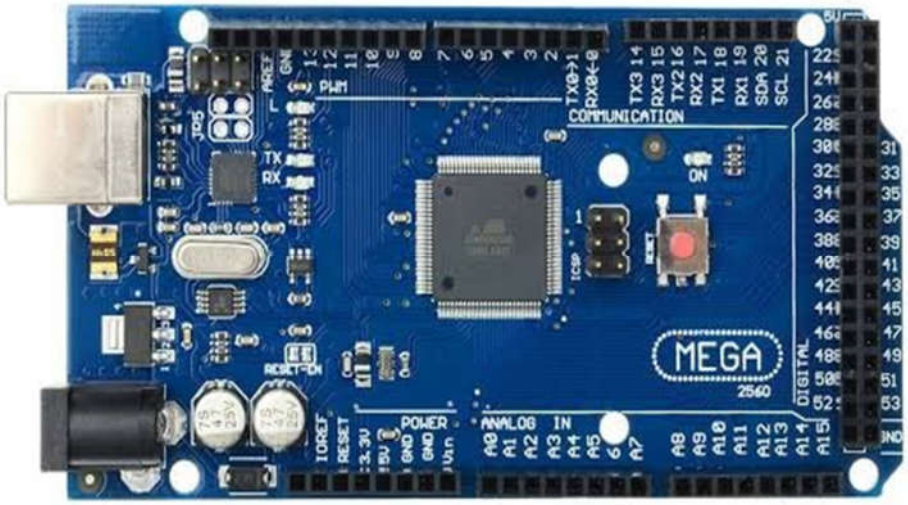
2.6.1 Arduino

O conceito de Arduino foi criado em 2005 com o objetivo de originar um dispositivo de baixo custo, funcional (para propósitos didáticos, prototipagem ou aplicações específicas) e acessível tanto para projetistas quanto amadores (figura 14). As placas Arduino adotam o conceito de hardware livre, significando que a partir do mesmo conceito básico de montagem é possível fazer personalizações e melhorias de acordo com o uso desejado (BATES, 2008).

A placa utiliza microcontroladores (no caso do usado neste trabalho, aquele da empresa *Atmel*), os quais necessitam linguagens baseadas em C/C++ para programá-los. A programação é pode ser feita utilizando a IDE (ambiente de desenvolvimento

integrado) oficial do Arduino, *Atmel Studio*, entre outros, necessitando apenas de um meio para transmitir o código para a placa, mais comumente se utilizando um cabo USB (*universal serial bus* ou porta universal) conectado ao computador onde a programação foi realizada (SUSNEA; MITESCU, 2006).

Figura 14 – Arduino *Mega 2560*



Fonte: Arduino Mega 2560 datasheet (2008).

3. METODOLOGIA

Primeiramente foi realizado o estudo dos protocolos de comunicação e funcionamento interno do laser YLPN-1-1x120-50-M. Visando os objetivos deste trabalho e o estudo realizado se projetou dois sistemas de controle, sendo um dele para ajustes rápidos (pode ser realizados enquanto o laser está em funcionamento porem controla um número limitado de fatores de funcionamento) e outro para controle completo (que tem domínio sobre todos as capacidades do laser porem só não pode ser utilizado enquanto o mesmo está em funcionamento).

3.1 Equipamentos

Neste trabalho foram utilizados:

- Laser YLPN-1-1x120-50-M: Laser classe 4, modelo com fibra de itérbio pulsado do tipo MOPA (*master oscillator power amplifier*) de baixa manutenção;
- Arduino Mega 2560: Arduino com processador ATmega2560, *clock* de 16 MHz, memória EEPROM 4kB, SRAM 8kB e 256 kB de *flash;*
- LM016L: Display digital 16x2 (possui duas linhas compostas de 16 caracteres)

3.2 Plataforma Arduino Mega 2560

O laser foi conectado ao Arduino Mega 2560, utilizando-se o cabo DB-25 (comunicação paralela), como mostrado na figura 15, onde os pinos foram ligados através da seguinte configuração:

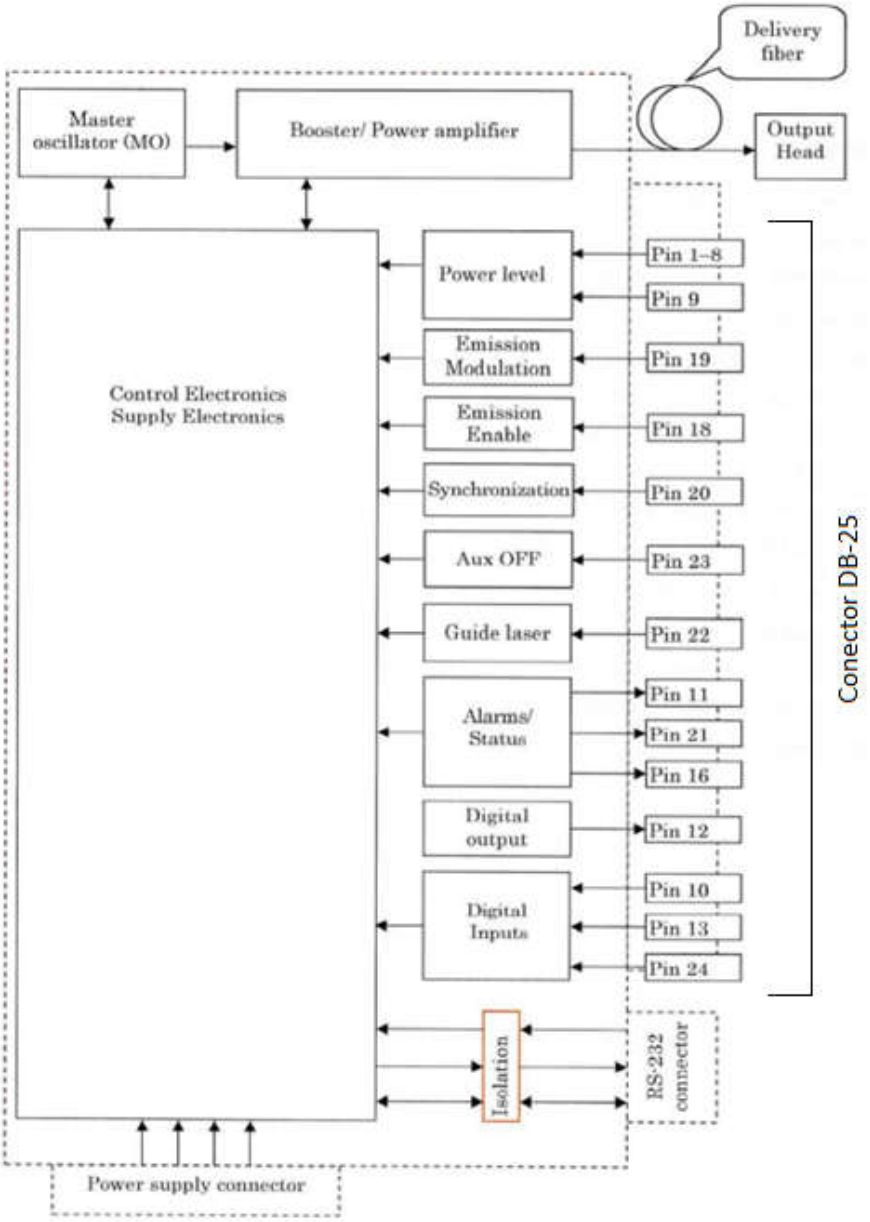
- Pinos 1-9: Responsáveis pela potência do laser, demonstrado por um bit que habilita o envio de um novo valor de potência e os outros 8 pinos representam um número binário o qual representando a potência desejada, onde 00000000 é 0% e 11111111 é 100%;
- Pinos 10, 13 e 24: Responsáveis por receber os comandos do usuário;
- Pinos 11, 16 e 21: Representam o status atual do laser;
- Pino 12: Envia em um pino serial a resposta ao sistema quando necessário;
- Pino 18: Habilita a emissão;
- Pino 19: Ativa a modulação de emissão;

• Pino 20: Utilizado para realizar a sincronização (*Baud Rate*);

• Pino 22: Ativa um laser de emissão vermelha (*laser guide*) que demonstrará a localização de onde o feixe laser está apontando (utilizado para como ponto de referência);

• Pino 23: Desligamento emergencial.

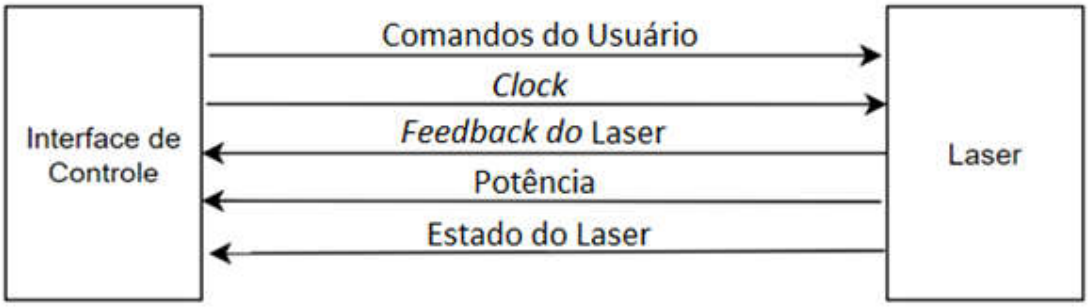
Figura 15 – Esquemático do Laser Module



Fonte: YLPN-1-1x120-50-M Datasheet (2015)

O programa foi desenvolvido na linguagem C, de forma a definir o *clock* do laser, receber os comandos do usuário, enviá-los comandos ao laser e sempre requisitando os valores de potência, estado do laser e *feedback* (figura 16).

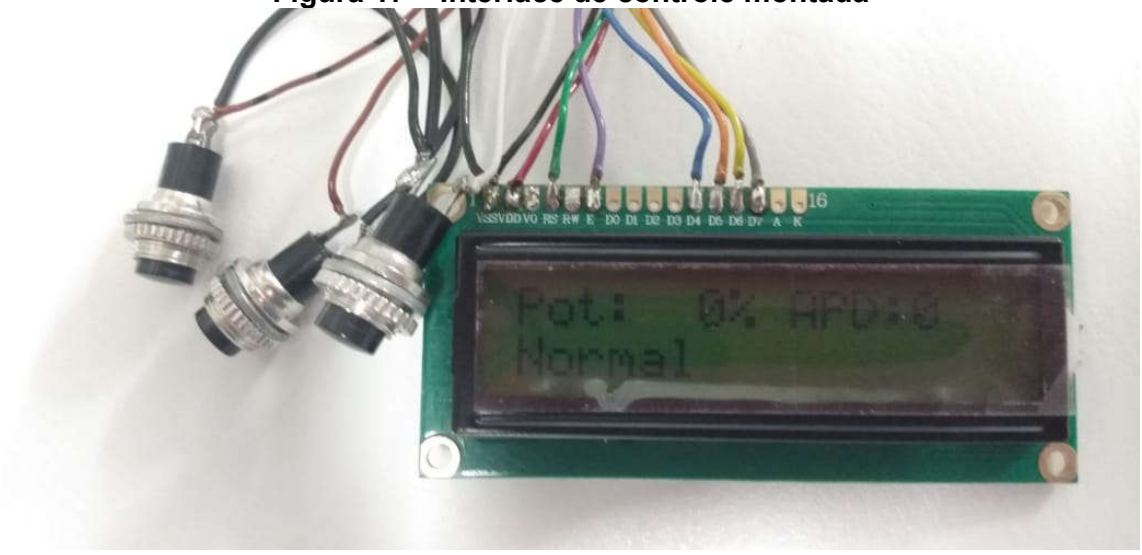
Figura 16 – Diagrama do *P*rograma



Fonte: Produção própria (2019).

Conectou-se o Arduino ao *display* 16x2 (LM016L), e aos *push buttons* (figura 17), para então possibilitar ao usuário interagir com o sistema.

Figura 17 – Interface de controle montada



Fonte: Foto retirada no laboratório (2019)

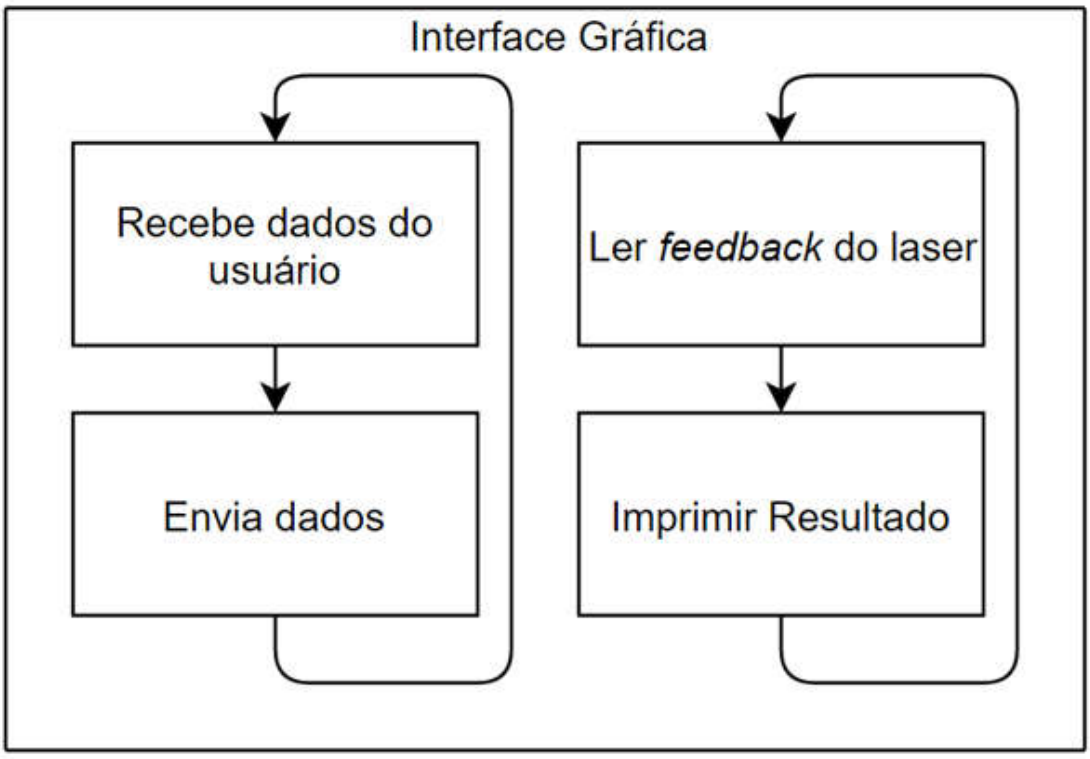
3.3 Interface Humano-Máquina Multiplataforma

A interface humana-máquina, para o controle completo do laser, foi obtido utilizando a linguagem Python com o auxílio das bibliotecas tkinter e *threading*.

- tkinter: biblioteca *Python* utilizada para produzir interfaces gráficas, possuindo suporte para botões, interfaces de entrada, dimensionamento, movimentação e centralização de janelas.
- *threading:* biblioteca com suporte para a execução de funções paralelas dentro de um programa.

O programa realiza constantes leituras daquilo enviado pelo laser em paralelo da etapa do programa responsável pela aquisição de dados do usuário, para então retornar o *feedback* do laser, o qual varia de acordo com o comando enviado pelo usuário, sendo usualmente uma confirmação que o comando foi recebido (figura 18)..

Figura 18 – Diagrama da interface gráfica



Fonte: Produção própria (2019).

4. ANÁLISE E DISCUSSÃO DOS RESULTADOS

Nesta etapa do trabalho, serão analisados e descritos os procedimentos demonstrados na seção anterior e serão apresentados os resultados obtidos. Será feita a análise do funcionamento do laser no qual o trabalho se baseia e serão apresentados os procedimentos utilizados na concepção das plataformas de ajustes rápidos e controle completo.

4.1 Laser YLPN-1-1x120-50-M

Por ser o único modelo no LMP sem um sistema de controle dedicado foi utilizado o laser classe 4 modelo YLPN-1-1x120-50-M do fabricante IPG Photonics (figura 19), da série com fibra de itérbio pulsado do tipo MOPA (*master oscillator power amplifier*) de baixa manutenção com entrada de 24 VDC e 9 A.

O controle do laser é realizado a partir de um cabo de conexão paralela a um Arduino Mega 2560 (para ajustes rápidos) e uma serial conectada a um computador (controle completo do laser).

Para mover o feixe laser, liga-se o colimador a um escâner de varredura óptica montado em um suporte capaz de se mover no eixo Z acima de uma plataforma móvel (eixos X e Y). Esse método foi escolhido por prover maior eficiência e cobertura para o laser.

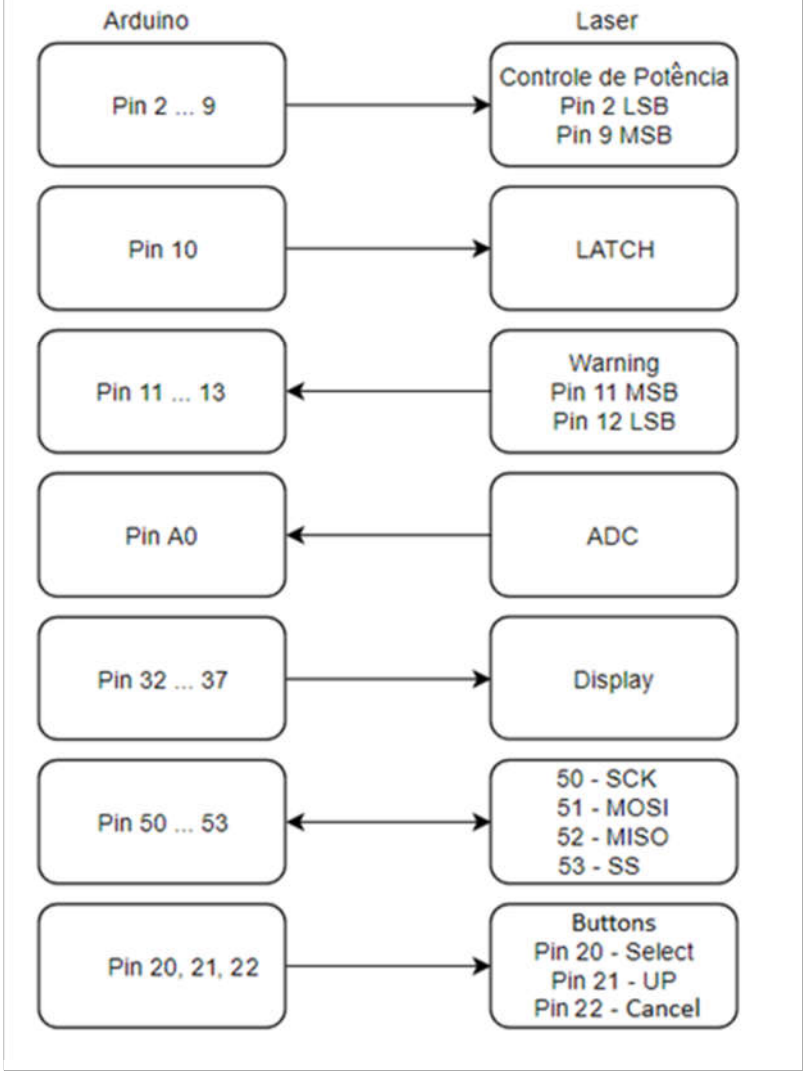
Figura 19 – Laser montado sobre bancada

Fonte: Arquivos LMP (2019).

4.1.1 Comunicação paralela

A comunicação é feita através de um cabo de comunicação paralela DB-25, e cada um dos pinos é conectado a diferentes entradas do Arduino Mega 2560 de acordo com o esquemático demonstrado na figura 20.

Figura 20 – Esquemático da ligação do Arduino - Laser



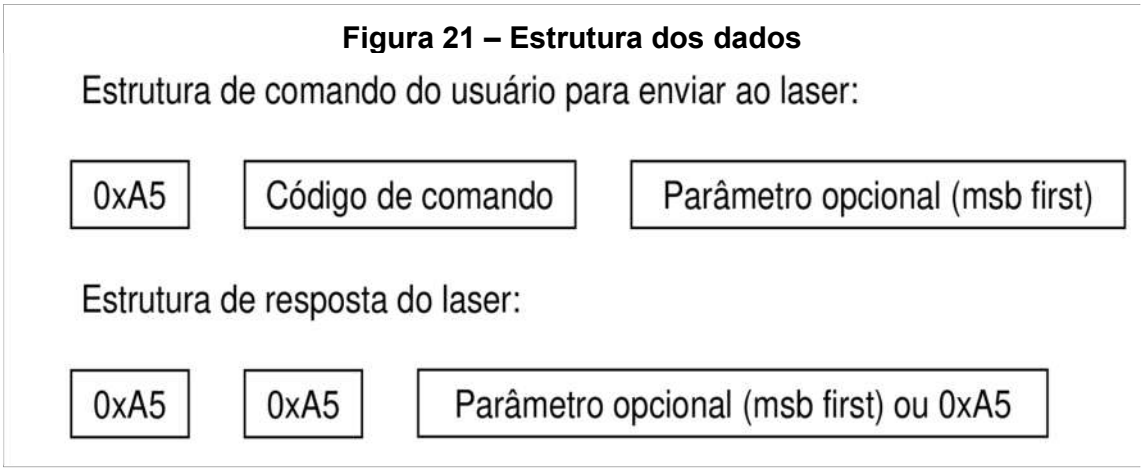
Fonte: Produção Própria (2019)

Os protocolos do laser definem que os pinos de habilitar emissão, ativar modulação de emissão e ativar o marcador do laser são pinos que transmitem informações binárias (sim ou não) e não são dependentes de *clock* ou qualquer forma de controle, já o estado atual do laser e sua potência são representados por 3 pinos

(comunicação paralela), não sendo vinculado a qualquer tipo de sinal de controle porém o monitoramento da potência necessita da ativação do pino de *latch* para que o valor emitido pelo ATMega2560 seja recebido pelo laser. O pino 23 não foi utilizado.

Os pinos 11 e 20 do laser são aqueles responsáveis pelo envio dos comandos através de um sinal serial e o *clock* (escolhido pelo usuário contando que esteja na faixa aceita pelo laser descrita no *datasheet*) respectivamente. Devido a configurações internas do laser, a comunicação serial ocorre em uma frequência de 20 kHz e os comandos enviados devem possuir uma estrutura de envio e recebimento onde primeiramente é enviado 0xA5, que sinalizará ao laser para receber comandos do usuário. Em seguida, se envia o código de comando e qualquer parâmetro necessário, o laser por vez responderá com duas instâncias seguidas de 0xA5 e então o *feedback* caso necessário, do contrário, enviará 0xA5 novamente (Figura 21).

Figura 21 – Estrutura dos dados



Fonte: YLPN-1-1x120-50-M Datasheet (2015)

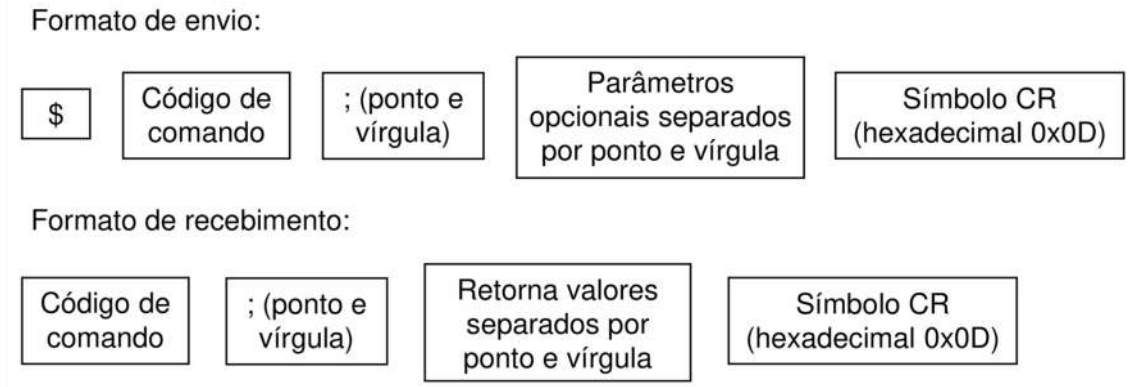
4.1.2 Comunicação serial

Para se conectar ao computador onde está instalada a interface, se utilizou o conector serial com o cabo DB9 do laser através de um conversor USB. Para a inicialização da comunicação, a partir de análises do *datasheet* do laser, estabeleceu-se um *baud rate* de 57600 bits por segundo, desabilitar os bits de paridade e controle de fluxo e definir os bits de *start/stop* para 1.

Todo comando a ser enviado deve seguir o padrão estabelecido pelo fabricante, o primeiro sinal a ser enviado é “$”, iniciando a comunicações e para finalizá-las “0x0D”. Todo comando que necessita de parâmetros deve ser separado deles por “;”.

A resposta do laser é construída enviando primeiramente o comando recebido, caso existam outros valores de retorno irá ser enviado um “;” anterior aos mesmo e para sinalizar que a comunicação acabou “0x0D” como mostrado na figura 22.

Figura 22 – Formato de envio e recebimento de dados

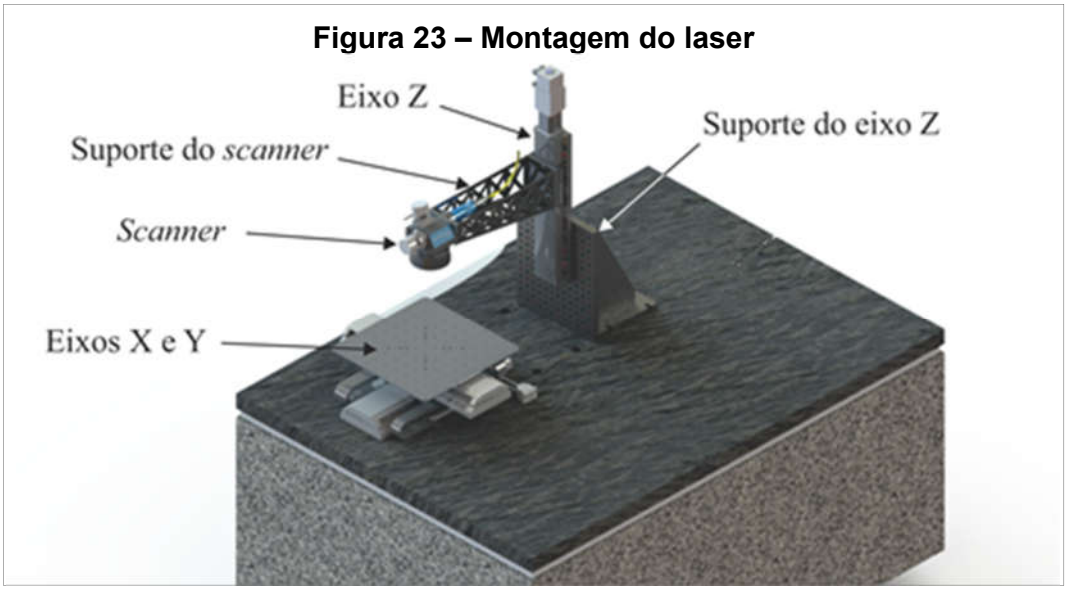


Fonte: Produção própria (2019)

4.1.3 Movimentação

Para movimentar o escâner, utiliza-se um suporte capaz de se movimentar verticalmente (considerado como eixo z) afastando ou aproximando o cabeçote de onde o laser é emitido assim da base onde estará o objeto a ser afetado pelo feixe de luz (o eixo z é utilizado de forma a otimizar a movimentação do laser onde em objetos menores o laser é colocado próximo e para corpos de testes maiores o cabeçote é afastado), abaixo do laser onde há uma plataforma móvel capaz de movimentar os eixos x e y (figura 23). A junção de todos esses sistemas de deslocamento mais a capacidade do escâner de variar a localização do feixe do laser provê uma estrutura altamente eficiente para o uso do equipamento.

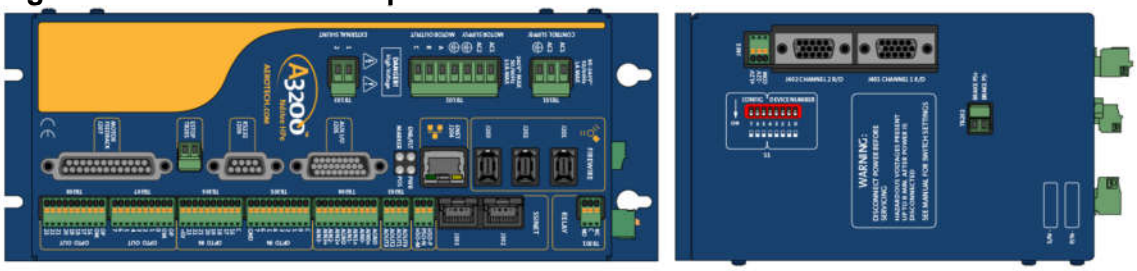
Figura 23 – Montagem do laser



Fonte: Arquivos LMP (2019)

Presente no sistema, para o controle dos eixos X, Y e Z, há três *drivers A3200 Ndrive Hpe da AEROTECH* (figura 24), estes são amplificadores de alta performance que provém comportamento determinístico, auto identificação e conexão serial possibilitando comunicar-se com software (consegue averiguar a posição atual do laser e quando esta precisa ser modificada irá interpretar o comando dado e o traduzira para os motores responsáveis).

Figura 24 – A3200 Ndrive Hpe



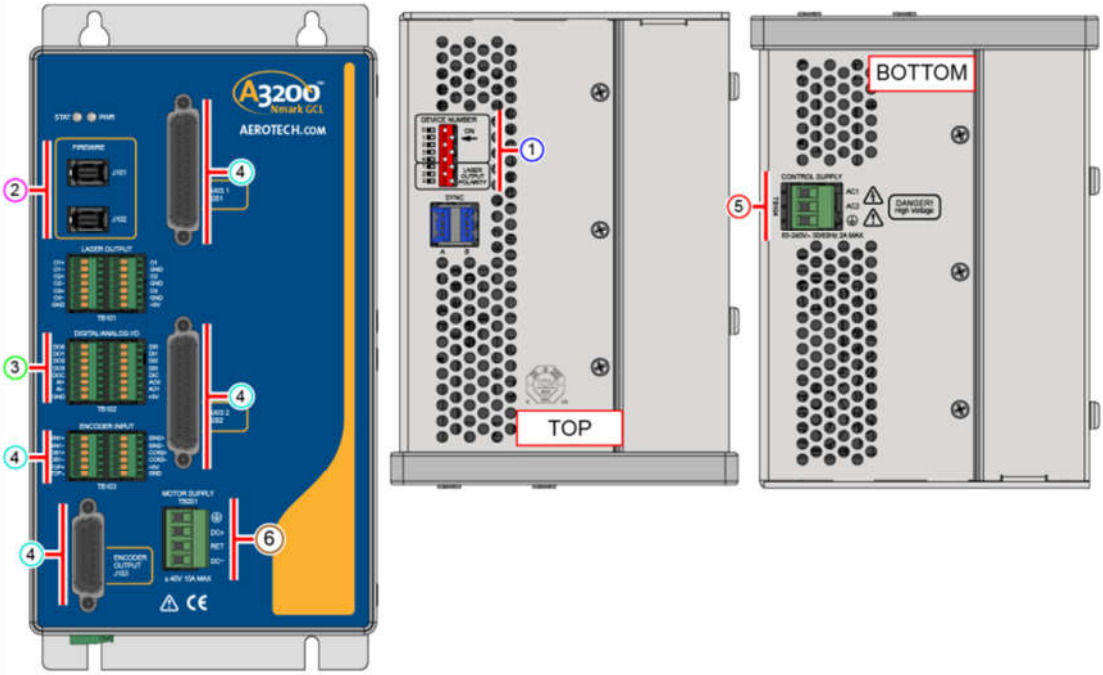
Fonte: A3200 Ndrive Hpe Datasheet (2017)

O controle é feito através da coordenação de 3 *drivers*, onde cada um é responsável por receber comandos relacionados a apenas um dos eixos. No caso, um controlará o movimento do laser no eixo X, outro para Y e o último, o *driver* Z. O *A3200 Ndrive Hpe* então codificará esses dados de posição em sinais elétricos que poderão

ser interpretados e traduzidos com a movimentação do suporte vertical (eixo z) e da plataforma (x e y).

O módulo *A3200 Nmark GCL* (figura 25) é um *driver* que permite controle direto dos servo motores presentes dentro do escâner, sendo efetivamente responsável pelo controle do feixe de laser.

Figura 25 – A3200 Nmark GCL



Fonte: A3200 Nmark GCL Datasheet (2017)

A coordenação desses quatro *drivers* é responsável por toda movimentação do sistema, eles são ligados um ao outro e a um computador pelo protocolo conhecido como *fire wire* (um tipo de padrão de comunicação serial) e coordenados através de um programa denominado A3200.

4.1.4 Alimentação

No sistema estão presentes quatro fontes de alimentação (figura 26), sendo elas:

- Uma de 24 VDC da marca Microtécnica Sistemas de Energia (MCE), responsável por alimentar os relés, contatores, iluminação e microcontrolador.

• Duas fontes de 40 V em série provendo alimentação simétrica para o *driver Nmark.*

• A última fonte de 24 V e 40A da marca Phoenix, alimenta o laser diretamente.

Figura 26 – Fontes Utilizadas

Fonte: Foto retirada no laboratório (2019)

4.1.5 Montagem do laser

A montagem do laser foi feita em dois módulos diferentes. A “torre” onde está localizada toda a parte elétrica do laser e aquele feito em cima da bancada onde está o sistema de disparo do laser em si.

Por motivos de segurança, foi montado ao redor do laser um enclausuramento feito de aço para proteger o usuário de possíveis reflexões do feixe, e um sistema de segurança, que impede ligar do laser caso as portas estejam abertas (figura 27). A organização interna da torre, onde estão presentes os *drivers* e fontes (figura 28), foram posicionados visando melhor utilização do espaço e coordenação com o laser (figura 29).

Figura 27 – *Enclausuramento* de aço

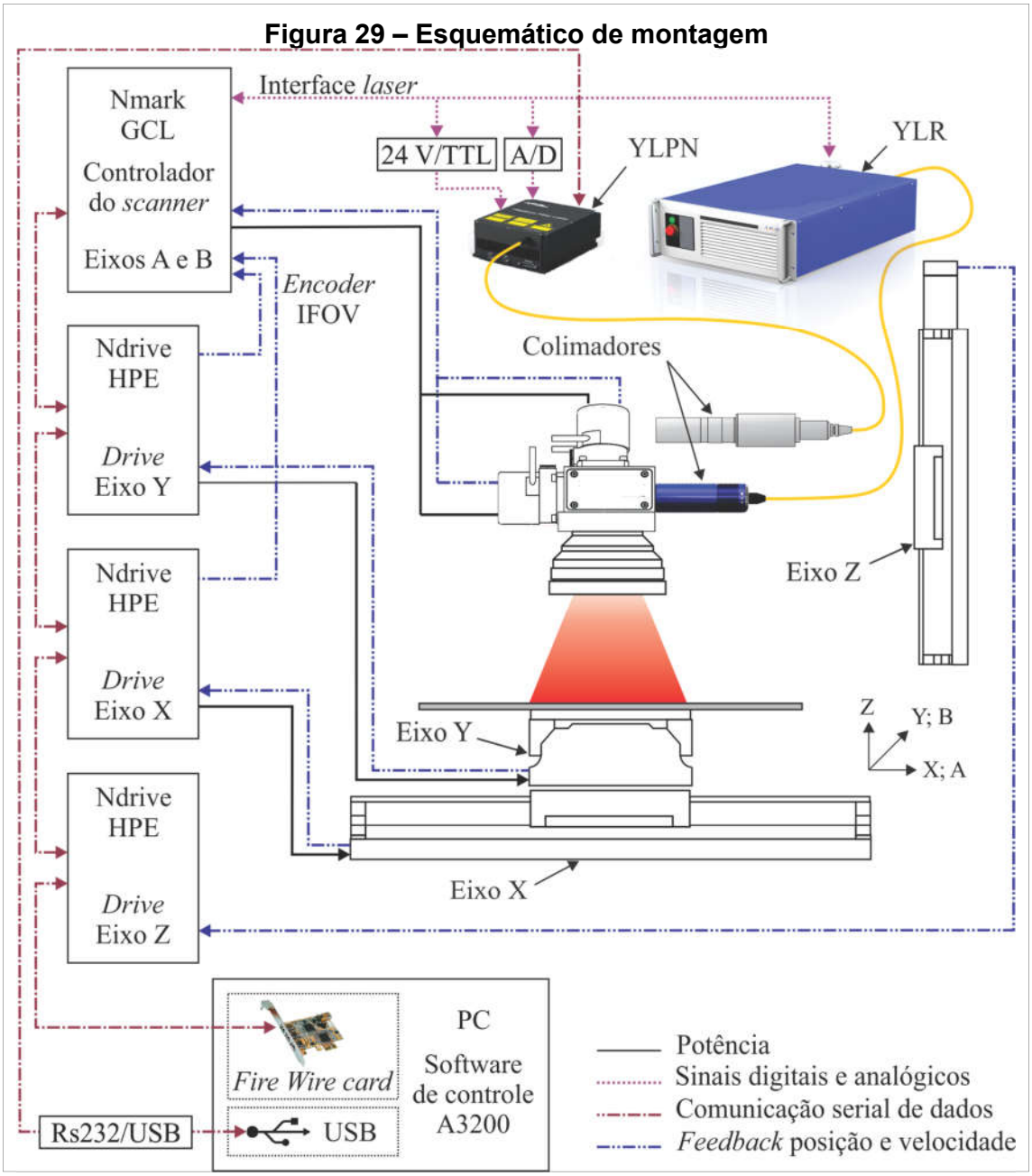
Fonte: Foto retirada no laboratório (2019)

.

Figura 28 – Torre onde estão os sistemas do laser

Fonte: Arquivo LMP (2019)

Figura 29 – Esquemático de montagem



Fonte: Arquivo LMP.

4.2 Plataforma de ajustes rápidos

Esta plataforma será montada no exterior da torre, provendo a possibilidade de realizar ajustes rápidos no funcionamento do laser sem a necessidade de se utilizar o computador ali conectado.

4.2.1 Análise dos comandos disponíveis

Primeiramente estudou-se quais comandos representados, por valores hexadecimais, o laser reconhece através da comunicação serial.

O controle de APD (*Adjustable Pulse Duration*) define qual dos modos pré-determinados pelo fabricante será escolhido para uso, os quais se diferenciam no ajuste de pulso aplicado ao laser, responsáveis pelo índice disparo do laser.

Prepump é a variável utilizada pelo sistema para controlar a intensidade do efeito de laser *pumping* aplicado pelo sistema (tabela 2).

***Tabela* 2 – Comandos para interface serial**

Tipo	Comando	Código de comando	Parâmetros em valores de retorno	Descrição / Parâmetros
Ler	Obter índice máximo do modo APD	0x05	Binário, um byte	Obtenha o índice máximo para o modo APD / *Integer*, faixa 0 ... 15
Escrever	Definir índice do modo APD	0x06	Binário, um byte	Definir modo APD por índice
Ler	Obter índice máximo do modo APD	0x07	Binário, um byte	Obtenha o modo APD atual por índice, que foi definido pelo comando 0x06 ou $69 (RS-232)
Escrever	Definir pré-bomba	0x10	Binário, dois bytes	Definir pré-bomba manual / *Integer*, faixa 0 ... 10000
Ler	Obter pré-bomba	0x11	Binário, dois bytes	Obtém pré-bomba manual, definida por comando 0x10 ou $64 (RS-232z

Fonte: YLPN-1-1x120-50-M Datasheet (2015)

Os comandos são:

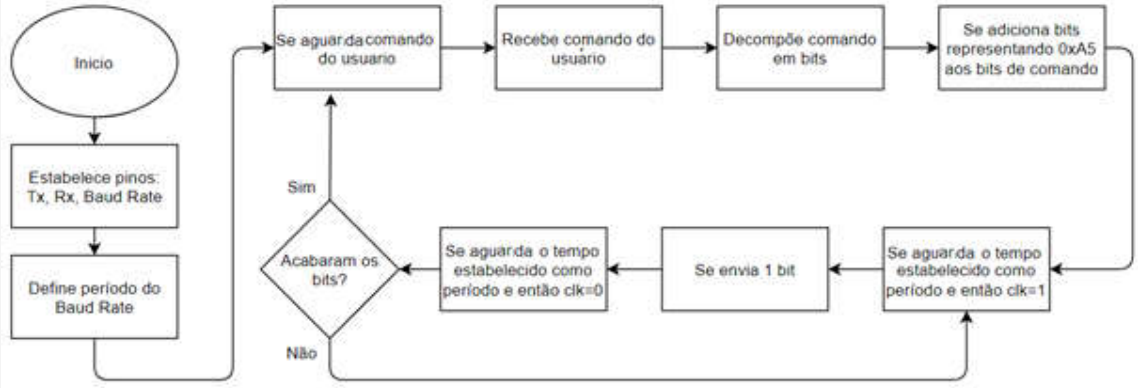
- Hexadecimal 0x05: Mostra ao usuário o número de modos de disparo disponíveis no laser;
- Hexadecimal 0x06: Envia ao laser um novo modo de disparo;
- Hexadecimal 0x07: Mostra ao usuário qual o modo atual de disparo do laser;
- Hexadecimal 0x10: Envia ao laser um novo valor de *prepump*;
- Hexadecimal 0x11: Mostra ao usuário qual o valor atual de *prepump*.

4.2.2 Biblioteca de comunicação serial

A comunicação serial do laser tem como *baud rate* máximo 20 kbits/s (limitado pelo laser), porém devido às configurações do ATMega2560, seu *clock* de 16 MHz, é demonstrado na equação 2 a impossibilidade de se transmitir informações serialmente nesta velocidade utilizando as funções pré-instaladas no microcontrolador (onde DvS é a divisão máxima possível via o comparador interno do sistema e DvR é a divisão via os registradores do Arduino).

Tendo em vista o *baud rate* mínimo que o Arduino Mega 2560 é de 25 kbits/s e a máxima aceita pelo laser é de 20 kbits/s, foi necessário criar uma biblioteca de comunicação serial, o que oferece as vantagens de poder controlar o seu funcionamento e em quais pinos estarão o transmissor (TX) e leitor (RX) de dados. O diagrama de blocos simbolizados na figura 30 representa o funcionamento desta biblioteca.

Figura 30 – Diagrama de blocos da biblioteca de comunicação serial



Fonte: Produção Própria (2019)

Para se definir o período do ciclo de transmissão foi utilizada a equação 3, onde se encontra o período máximo suportado pelo laser, resultando em 50 us.

$$Periodo\ maximo = \frac{1}{Frquencia\ Maxima\ do\ Laser} = \frac{1}{20000} = 50\ us \qquad (3)$$

4.2.3 Configurações iniciais do laser

De acordo com as configurações do laser, foi definido manter os bits de habilitar emissão em 1 (sinalizando que o laser poderá disparar), ativar modulação de emissão em 0 (se mostrou desnecessário, pois modular a emissão reduz a potência emitida pelo laser), ativar o marcador do laser (para sempre ter um referencial durante uso).

4.2.4 Estado do laser

Tendo em vista que o estado atual do laser é representado por 3 bits (pinos 11, 16 e 21), interpretou-se esses dados associando cada combinação binária realizável a um dos possíveis estados do laser, seguindo a interpretação apresentada na tabela 3.

Tabela 3 – Possíveis estados do laser

Os pinos 11, 16 e 21 mostram os seguintes estados do laser			
Estado 2 Pino 11	**Estado 0 Pino 16**	**Estado 1 Pino 21**	**Descrição**
Baixo	Baixo	Baixo	**Alarme de temperatura** / Temperatura do laser está fora da faixa de temperatura operacional
Alto	Baixo	Baixo	**Alarme da fonte de alimentação** / Tensão de alimentação externa está fora da faixa especificada
Baixo	Baixo	Alto	**Operação normal**
Alto	Baixo	Alto	**O laser não está pronto para emissão**
Baixo	Alto	Baixo	**Alarme de reflexão traseira** / O laser é desligado automaticamente devido à alta potência óptica refletida de volta ao laser
Alto	Alto	Baixo	Reservado
Baixo	Alto	Alto	**Alarme do sistema** / Sistema de proteção a laser detecta falha interna
Alto	Alto	Alto	Reservado

Fonte: YLPN-1-1x120-50-M Datasheet (2015)

Por motivos de segurança, caso o estado indique alarme de temperatura, alimentação, reflexão de fundo ou falha no sistema o programa irá automaticamente enviar os dados para baixar a potência para zero, protegendo o laser.

4.2.5 Leitura da potência

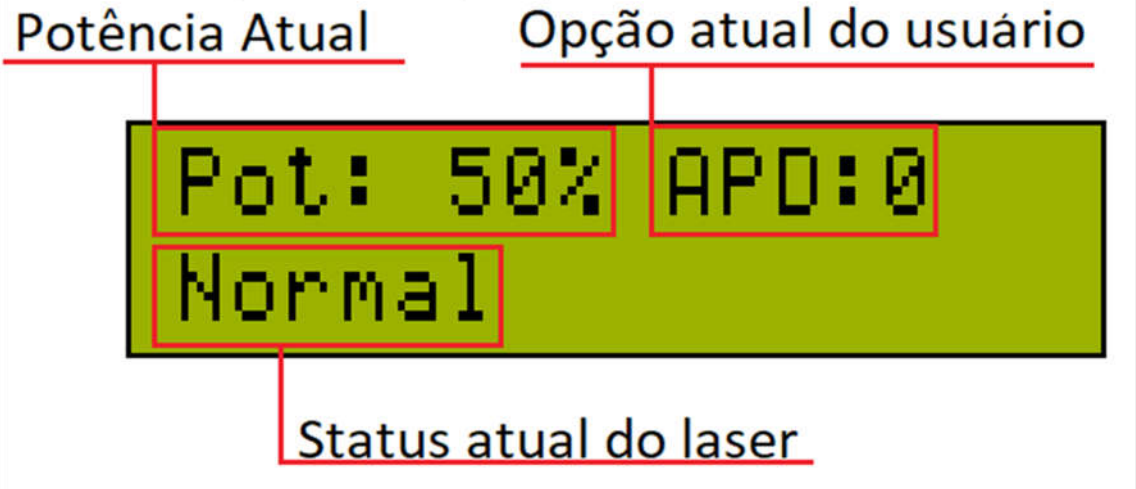
Considerando-se que a potência do laser é informada utilizando comunicação paralela, através de 9 bits sendo um deles o *latch* (r*esponsável* por controlar quando a nova potência é enviada ao sistema), o sistema recebe o valor de potência informado pelo usuário e o traduz em uma palavra binária de 8 bits para enviá-lo ao laser. Para mostrar a potência ao usuário se utilizou um conversor binário/decimal (equação 4), onde cada bit em *logical high* é convertido para seu valor decimal, somado aos outros e então convertido para porcentagens de forma a melhor comunica-lo ao usuário.

$$Potência = \frac{\sum_{n=0}^{7}(Bitn * 2^n) * 100}{2^8} \qquad (4)$$

4.2.6 Implementação

O menu apresentado na figura 31, foi projetado para ser o mais compacto e de fácil utilização. A potência e o estado atual do laser, por motivos de segurança, são sempre presentes no *display,* a direita desses dados está presente a opção atual do usuário.

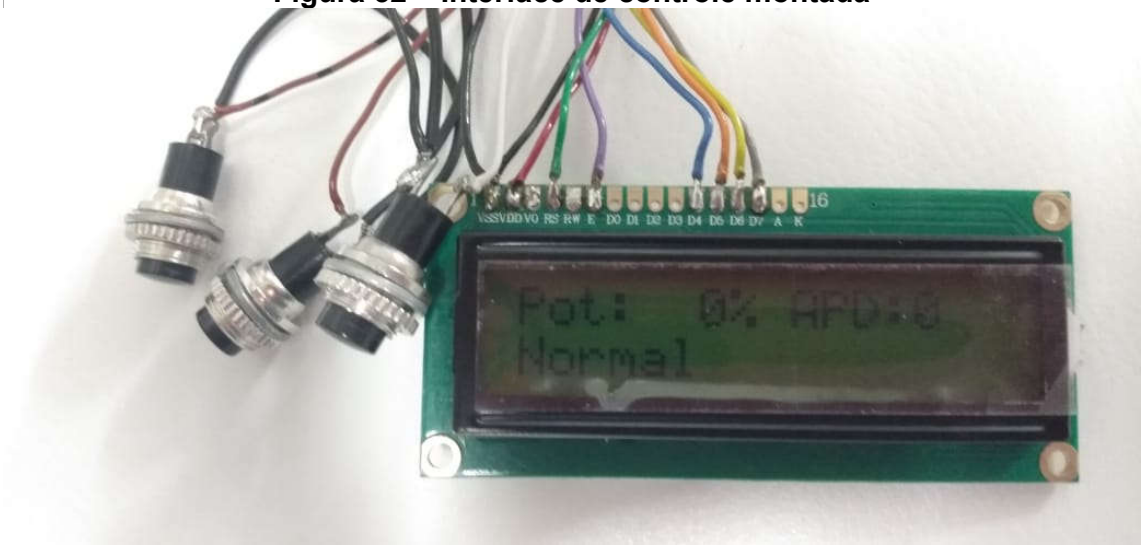
Figura 31 – Menu principal da interface de controle



Fonte: Produção própria (2019).

A montagem da interface consiste do display (LM016L) e três *push buttons* (para interação com o usuário) como mostrado *na* figura 32.

Figura 32 – Interface de controle montada



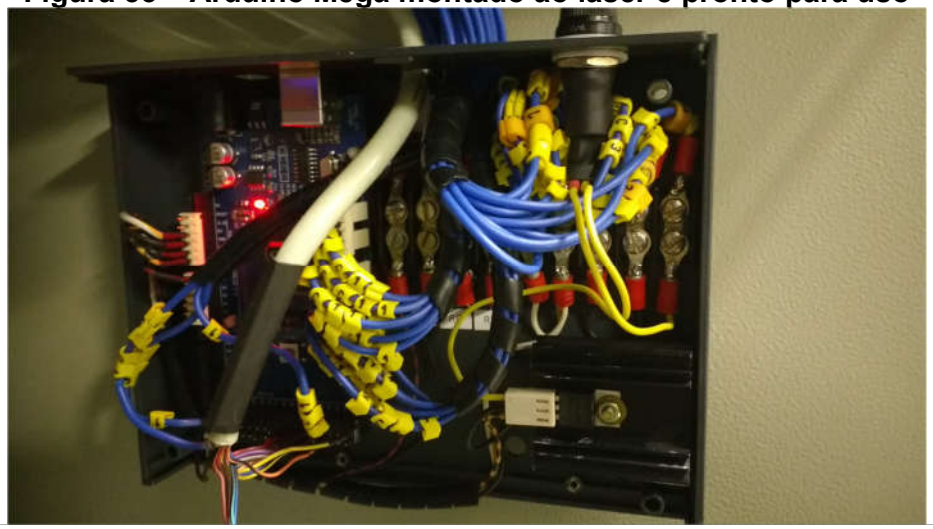
Fonte: Foto retirada no laboratório (2019)

O LM016L funciona a partir de uma tensão de 3,3 V a 5 V, possui ajuste de contraste, selecionador de registro (bit responsável pela verificação se está recebendo um comando ou um dado), *clock enable*, *bit* que define se é escrita ou leitura e 8 pinos para recebimento de dados.

A interface foi planejada de forma a possuir um simples menu para a seleção das ações, um botão para selecionar aquilo que se deseja, um botão para alternar entre as opções que podem ser utilizadas e o último para cancelar a escolha atual.

Por fim o Arduino Mega 2560 foi montado no interior da “torre” (figura 33) para se conectar à interface de ajustes rápidos que futuramente será montada próximo ao enclausuramento do laser.

Figura 33 – Arduino Mega montado ao laser e pronto para uso



Fonte: Foto retirada no laboratório (2019)

O programa divide seu código em funções genéricas sendo controladas pelo *main* (código responsável por unir todas as funções e criar a interface com o usuário). Primeiramente todas as etapas de comunicação paralela (potência e estado do laser), devido à necessidade (por motivos de segurança) de sempre demonstrar esses valores ao usuário, essas etapas do código irão ser ativadas em intervalos fixos de tempo e caso há necessidade de alterar seus valores só ira ocorrer no próximo ciclo de funcionalidade.

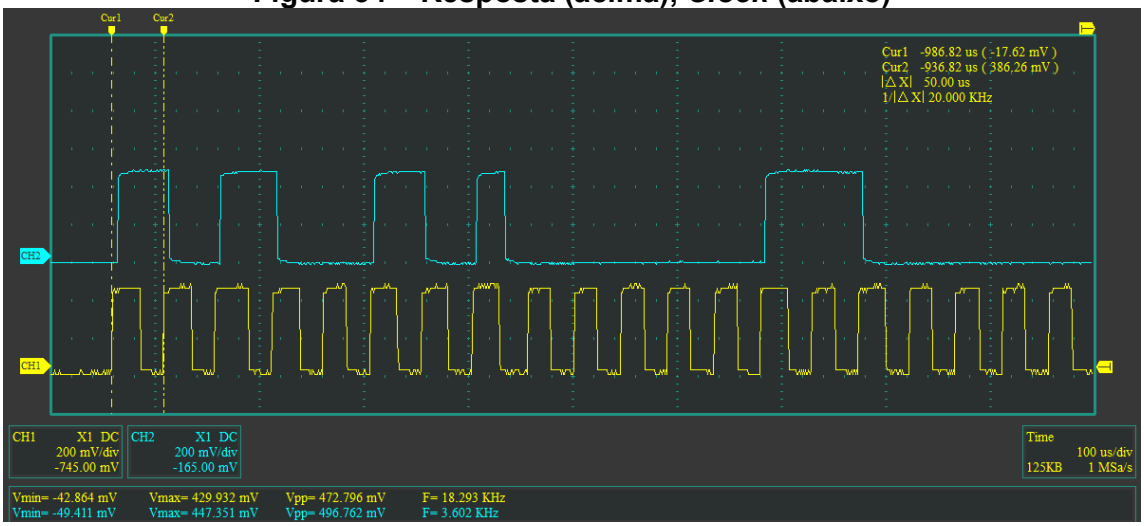
A comunicação dos outros dados do programa ocorre através da biblioteca, criada para este projeto, tornando essa etapa do processo altamente otimizada.

Todo o código feito nesta etapa se encontra no apêndice 1 (código C) e apêndice 2 (biblioteca de comunicação serial em c).

4.2.7 Análise dos comandos

Para se averiguar o funcionamento da plataforma de controle de ajustes rápidos (montada utilizando o ATMeg2560) e que o sinal de *clock* estava como calculado, foi realizado um teste onde se enviou um comando ao laser (alterar o APD para o modo 3) e se ligou um osciloscópio (modelo Instrustar ISDS205X) aos cabos onde estava sendo emitido o *clock* e a resposta do laser a instruções do usuário, assim adquirindo os sinais demonstrados na figura 34.

Figura 34 – Resposta (acima), *Clock* (abaixo)



Fonte: Medição realizada no laboratório (2019).

Analisando-se os sinais, pode ser ver que resposta do laser foi como esperado (tendo como base o exemplo presente em seu datasheet que indica a resposta do laser a este comando em especifico) e que seu *clock* possuiu um período de 50 us exatamente como foi calculado, assim confirmando o funcionamento desta plataforma.

4.3 Interface humano máquina multiplataforma

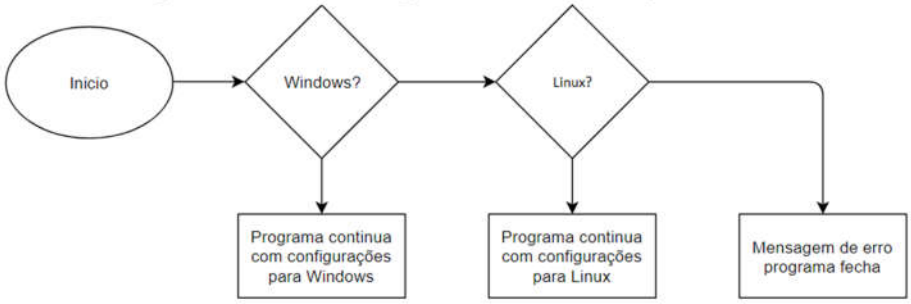
Esta interface tem como objetivo um controle mais detalhado e completo do laser através de uma plataforma criada em *Python* (versão 3.0), escolhido tendo em vista que essa é a linguagem mais comumente utilizada no LMP facilitando o processo de alterar o programa no futuro (adaptar a outros usos ou outros modelos de lasers dentro do LMP) e grande número de bibliotecas e recursos disponíveis. O programa foi desenvolvido utilizando o *software Jupyter Notebook*.

Foi projetado o programa com compatibilidade para Windows e para Linux devido a atualmente o laser estar ligado diretamente a um computador cujo sistema operacional é Windows 7, porém existe a possibilidade de troca-lo por um *Raspberry pi* com Linux devido a seu tamanho mais compacto e facilidade de instalação.

4.3.1 Verificação Windows/Linux

Primeiramente se verifica o sistema operacional sendo utilizado (figura 35) através do comando *platform* da biblioteca *sys,* para que o sistema seja adaptado, dependendo do sistema operacional instalado (Windows ou Linux). Para casos onde nenhum dos dois seja detectado fecha-se o programa e se envia uma mensagem de erro como mostrado na figura abaixo.

Figura 35 – Verificação do sistema operacional



Fonte: Produção própria (2019).

4.3.2 Funções

Para a realização das funções da interface e otimizar seu funcionamento, cada uma das tarefas é colocada em uma função individual, podendo ser chamada pelo programa a qualquer momento. As funções são:

- *serial_ports*: adquire uma lista de todas as portas USB em uso;
- *connect*: conecta a porta escolhida utilizando os parâmetros de *baud rate* = 57600 bits por segundo, desabilitar os bits de paridade e controle de fluxo, e definir os bits de *star / stop* para 1;
- *check_port_open*: verifica se ainda há conexão com a USB;
- *close_actual_port*: desconecta a USB conectada;
- *prepare_commands*: traduz os comandos do usuário, de letras para uma lista de números, utilizado apenas para testar se os comandos enviados estão corretos;
- *send_command*: envia o comando do usuário ao laser;
- *read_feedback*: lê os dados enviados do laser ao usuário;

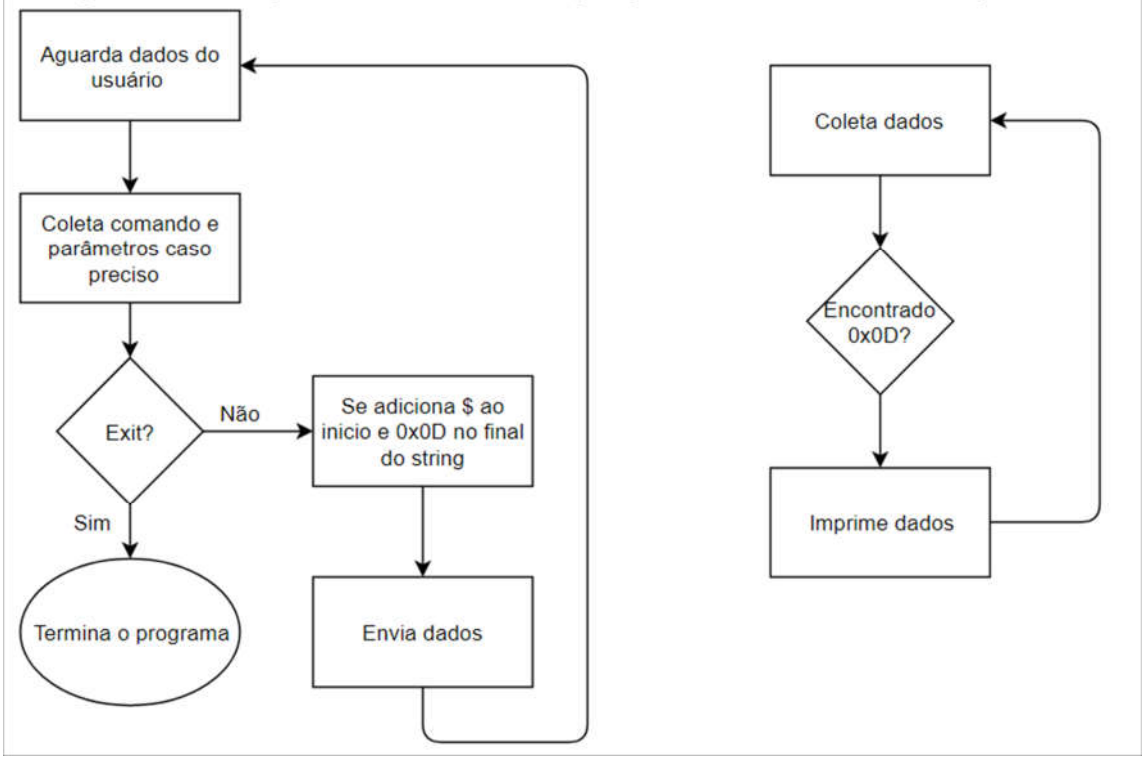
4.3.3 Funcionamento do sistema

Primeiramente é feita a varredura de todos os dispositivos USB ligados à plataforma onde o programa está sendo executado e é pedido ao usuário apontar qual delas é o laser.

Então se inicia a interface gráfica utilizando comandos da biblioteca *tkinter,* para então o processo de envio e recepção de dados poder ser iniciado. Para se assegurar que nenhum dado será perdido a função responsável pela recepção de dados funcionará constantemente junto àquela de envio, e para que as funções ocorram simultaneamente, utilizou-se a biblioteca *Threading*. Quando o usuário terminar seu uso do sistema, será necessário digitar "*exit*" na janela de *input*, isso sinalizará que o programa deve ser fechado, como apresentado na figura 36.

O programa aguarda dados do usuário, verifica se esse é o comando de saída, caso não seja, ele criará uma *string* seguindo os parâmetros definidos pelo laser (inciando com "$", seguido pelos comandos e terminando com "0x0D") e então enviando para o laser. Durante este processo o programa constantemente aguarda por dados do laser para enviá-los ao usuário, como visto na figura 36.

Figura 36 – Diagrama de blocos dos programas funcionando em paralelo



Fonte: Produção própria (2019)

Ao terminar o programa, a porta USB é automaticamente desconectada, para que seu uso fique livre casos outros programas necessitem do laser, e a thread onde a leitura ocorre é fechada.

O código pode ser encontrado no apêndice 3 (código Python) e todos os comandos possíveis no anexo 1.

4.3.4 Interface

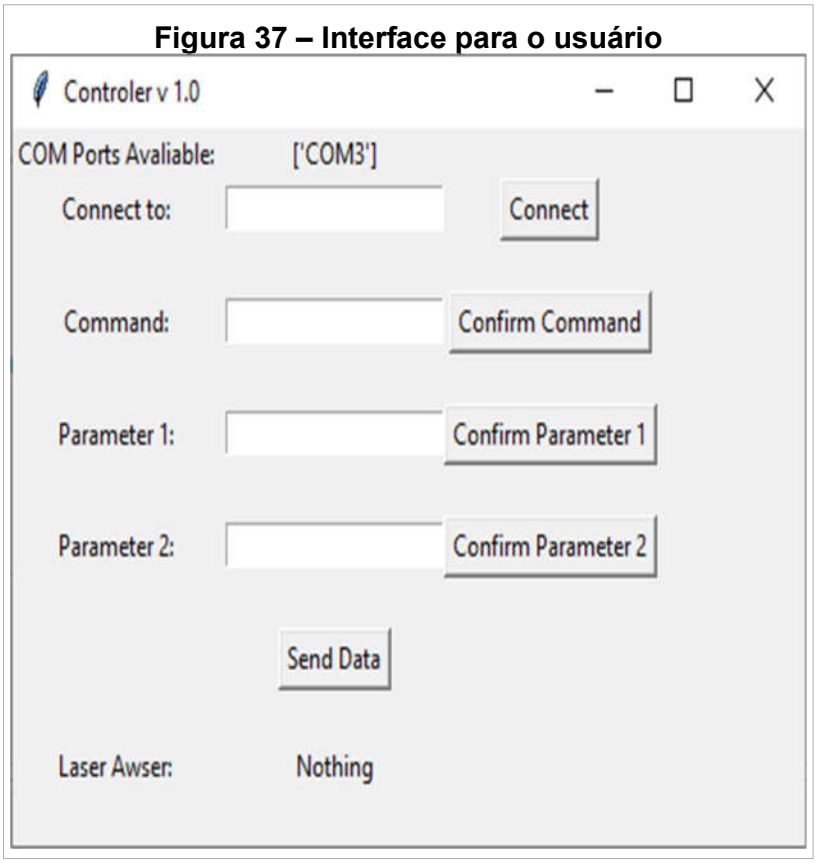
A interface foi montada com o intuito de ser simples e compacta, contendo apenas uma aba para selecionar a entrada onde está conectado o laser, uma área para o usuário selecionar seu comando, uma linha de *input* para, quando necessário, inserir os parâmetros adicionais e um local para o *feedback* enviado pelo laser.

Na interface (figura 37) é apresentado quais portas estão disponíveis para se conectar, dando ao usuário a oportunidade de escolher qual deseja, porém caso seja inserido um valor inválido, o programa irá apenas continuar esperando até uma entrada válida ser fornecida.

Para o envio de dados é requerido inserir os dados (comando e parâmetros adicionais) separadamente e então confirmar se aqueles são os valores desejado, se os dados serão enviados, isso foi feito de forma a proteger o laser de comandos impróprios.

Toda as respostas do laser para o usuário serão apresentadas na parte chamada de "*Laser Awser*" localizada na parte inferior da janela, tornando fácil sua visualização.

Figura 37 – Interface para o usuário



Fonte: Produção própria (2019).

Devido à versatilidade do programa, (por ser em python e montado modularmente), para projetos futuros, essa interface será adaptada para outros lasers e se adicionar a funcionalidade para a leitura de arquivos contendo a planilha de peças onde o laser será utilizado e realizar os ajustes sem a necessidade de interferência humana.

4.4 Implementação e utilização das plataformas de controle

Para averiguar o funcionamento dos sistemas de controle (rápido e completo) foi pego uma placa de aço e desenha sobre ela 3 formas geométricas compostas de 250 linhas de 500 mm de comprimento, 5 mm de grossura e distanciadas entre si 0,2 mm, que ao olho humano vão aparentar serem quadrados.

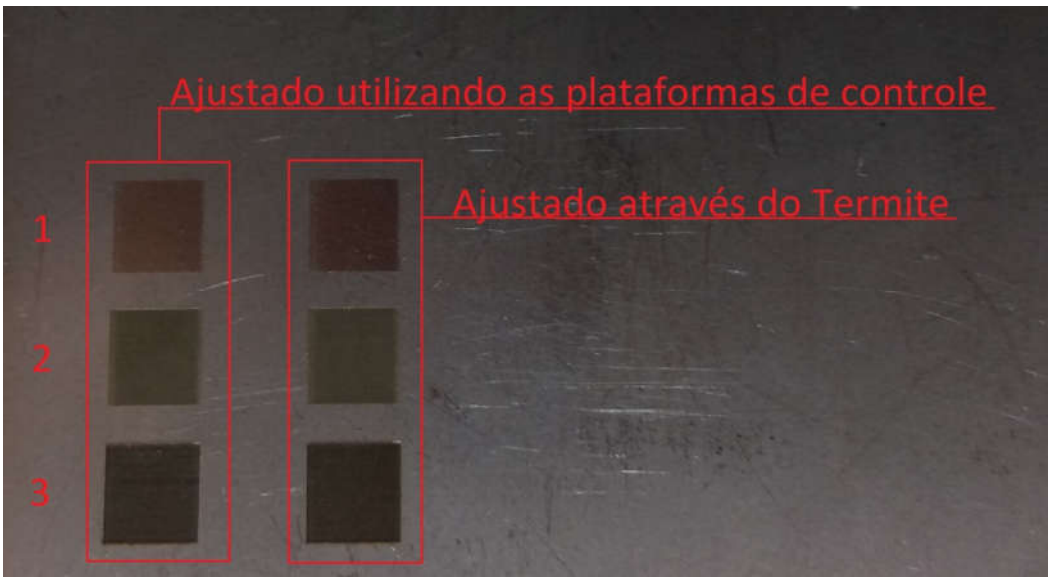
Cada um dos quadrados será feito utilizando diferentes parâmetros ajustados através da interface de ajustes rápidos (quadrado 1), interface de ajustes completos (quadrado 2) e utilizando ambas para ajustar o mesmo (quadrado 3). A seguir serão feitos mais 3 quadrados utilizando a plataforma Termite (método usado antes da aplicação das interfaces de controle) com os mesmos parâmetros para comparação.

Os parâmetros de testes escolhidos foram:

- Quadrado 1: APD = 7 e Potencia = 50%;
- Quadrado 2: APD = 5 e Potencia = 50%;
- Quadrado 3: APD = 3 e Potencia = 50%.

Estes valores foram escolhidos por a variação do APD causa uma visível diferença de cor a área afetada pelo laser (para APD 1 e 2 a cor é muito similar à da placa utilizada, por isso não foram usados) e a potencia foi mantida em 50% durante os testes por, segundo recomendações do doutorando responsável, é a ideal para o material da placa.

Figura 38 – Placa de aço onde foram realizados os testes



Fonte: Foto retirada no laboratório (2019)

Tendo em vista que as cores se alteraram entre os quadrados e as comparando com os testes equivalentes realizados pela plataforma Termite, pode se averiguar que as plataformas de controle estão funcionando perfeitamente e fornecendo ao usuário uma opção mais fácil de utilização do equipamento.

5. CONCLUSÃO

Neste projeto estudou-se o laser YLPN-1-1x120-50-M. Seu funcionamento foi analisado, resultando em grande compreensão de seu funcionamento, de seus protocolos de comunicação, alimentação, métodos de montagem e dos diferentes usos para tal equipamento.

Baseando-se nos conhecimentos em programação C e microcontroladores, foi possível projetar e implementar uma interface compacta e de fácil uso para ajustes do laser *pumping*, ganho do APD e potência total do laser. Esta mesma plataforma é responsável pela comunicação ao usuário do estado atual do laser, além de conter uma medida de segurança (baixa a potência para zero) em casos de mau funcionamento.

A maior dificuldade neste trabalho foi durante o projeto da plataforma de ajustes rápidos utilizando o Arduino Mega 2560 pelo falo do mesmo, devido a sua montagem no interior da torre do laser, não ter muita flexibilidade em quais pinos podem ser utilizados e do mesmo não ser capaz de gerar o *clock* compatível com as especificações do laser. Esses problemas foram resolvidos simultaneamente criando uma biblioteca de comunicação serial customizável capaz de selecionar o exato período desejado e quais pinos de Arduino utilizar.

As dificuldades durante o desenvolvimento da interface humano-máquina (IHM) podem ser atribuídas ao fato do programa em Python necessitar de funções que necessitem funcionarem em paralelo e a montagem de interface gráfica. Porem com a utilização das bibliotecas *tkinter* e *Threading* estas dificuldades foram superadas e a plataforma foi implementada. Os testes realizados desta interface se provaram um sucesso, alterando as configurações do laser de forma eficiente e de fácil compreensão para o usuário.

Ambas as plataformas de controle foram projetadas, aplicadas e testadas assim comprovando seu funcionamento e provando que estão pontas para uso para todos do laboratório LMP.

REFERÊNCIAS

HAMAMATSU, Avalanche Photodiode: A User Guide, 2019.

ATMEL, ATMEGA640-1280-1281-2560-2651 Datasheet, 2007.

BATES, M. P. Programming 8-bit PIC Microcontrollers in C: with Interactive Hardware Simulation. [s.l.] Newnes, 2008.

BARRETT, S. F.; PACK, D. J. Microcontrollers Fundamentals for Engineers and Scientists. [s.l.] Morgan & Claypool Publishers, 2006.

DIELS, J.-C.; ARISSIAN, L. Lasers: The Power and Precision of Light. [s.l.] John Wiley & Sons, 2011.

EICHLER, H. J.; EICHLER, J.; LUX, O. Lasers: Basics, Advances and Applications. [s.l.] Springer International Publishing, 2018.

GEHANI, N. C: An Advanced Introduction : ANSI C Edition. Subsequent edition ed. Summit, NJ: Silicon Pr, 1994.

HUSZARIK, S. YLR-Series User Guide. p. 102, [s.d.].

JONES, D. Arduino: Advanced Strategies to Learn and Execute Arduino Programming. 2012.

KERNIGHAN, B. W.; RITCHIE, D. M. C Programming Language, 2nd Edition. 2 edition ed. Englewood Cliffs, N.J: Prentice Hall, 1988.

LINS, R. D. A. U. et al. Efeitos bioestimulantes do laser de baixa potência no processo de reparo. Anais Brasileiros de Dermatologia, v. 85, n. 6, p. 849–855, dez. 2010.

LINDEN, P. V. DER. Expert C Programming: Deep C Secrets. [s.l.] Prentice Hall Professional, 1994.

LUTZ, M. Programming Python: Powerful Object-Oriented Programming. Fourth edition ed. Beijing Bosotn Farnham Sebastopol Tokyo: O'Reilly Media, 2011.

MA, D. Avalanche photodiodes arrays. 2004

MCKINNEY, W. Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython. [s.l.] Novatec Editora, [s.d.].

PADMANABHAN, T. R. Programming with Python. [s.l.] Springer Singapore, 2016.

POMILIO, J. A. 2. TÉCNICAS DE MODULAÇÃO DE POTÊNCIA. Eletrônica de Potência, p. 26, 2014.

SCHULZE, M.-M. The evaluation of bulbar redness grading scales. 12 jan. 2010.

Site do Laboratório de Mecânica de Precisão, disponível em http://www.lmp.ufsc.br/ acessado em 25 de novembro de 2019

SUMMERFIELD, M. Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming. 1 edition ed. Upper Saddle River, NJ Boston Indianapolis San Francisco New York Toronto Montreal London: Prentice Hall, 2015.

SUSNEA, I.; MITESCU, M. Microcontrollers in Practice. [s.l.] Springer Science & Business Media, 2006.

SVELTO, O. Principles of Lasers. 5. ed. [s.l.] Springer US, 2010.

Termite: a simple RS232 terminal. Disponível em: <https://www.compuphase.com/software_termite.htm>. Acesso em: 2 dez. 2019.

THYAGARAJAN, K.; GHATAK, A. **Lasers: Fundamentals and Applications**. [s.l.] Springer Science & Business Media, 2010.

Tutoriais | Inkscape. Disponível em: <https://inkscape.org/pt-br/aprender/tutoriais/>. Acesso em: 2 dez. 2019.

ZERVAS, M. High power ytterbium-doped fiber lasers - Fundamentals and applications. International Journal of Modern Physics B, v. 28, 30 abr. 2014.

AEROTECH, A3200 Ndrive Hpe Datasheet (2017).

APÊNDICE 1 – CÓDIGO C

```
/* ------------------------------------------------------------------------
 * Project:     Controle do Laser YLPN
 * File:        Laser_2560.c
 * Author:      Cláudio Abilio e Daniel Pereira
 * Created:          25/02/2018
 * Modified:  27/03/2018
 * Version:     1.0
 * Purpose:     Controls an YLPN Laser.
 * ------------------------------------------------------------------------ */

// ----------------------------------------------------------------------------
// System definitions ---------------------------------------------------------
#define F_CPU 16000000UL

// ----------------------------------------------------------------------------
// Header files ---------------------------------------------------------------
#include "lcd4d.h"
#include "SPI_Experimental.h"
#include "Tool_Box.h"

#include <string.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>

// ----------------------------------------------------------------------------
// Project definitions --------------------------------------------------------

// ----------------------------------------------------------------------------
// New data types -------------------------------------------------------------
typedef volatile union systemFlags_t{
      struct{                                                                 //
St210
            uint8_t Temperature_Alarm   :   1;        // 0b000 = 0
            uint8_t Power_Supply_Alarm:       1;            // 0b001 = 1
            uint8_t Normal_Operations:        1;            // 0b010 = 2
            uint8_t Laser_Not_Ready:          1;            // 0b011 = 3
            uint8_t Back_Reflection_Alarm:    1;            // 0b100 = 4
            uint8_t Reserved_1:                     1;            // 0b101 = 5
            uint8_t System_Alarm:                   1;            // 0b110 = 6
            uint8_t Reserved_2:                     1;            // 0b111 = 7
      };
      uint8_t allFlags;
} systemFlags_t;

// ----------------------------------------------------------------------------
// Function declaration -------------------------------------------------------
void  conversor();
void  warning_codes();
void  Apply_APD();

// ----------------------------------------------------------------------------
// Global variables -----------------------------------------------------------
volatile uint16_t newADC;
volatile uint16_t oldADC;
volatile uint16_t percent;
volatile uint8_t  Data_Set;
```

```
volatile uint8_t  APD;
volatile uint8_t  warning;
systemFlags_t   systemFlags;

attachLcd(display);

// ----------------------------------------------------------------------------
// Main function --------------------------------------------------------------
int main(void)
{
	// Variable declaration

	// Variable initialization
	systemFlags.allFlags = 0;
	newADC = 0; oldADC = 0; warning = 0;
	Data_Set = 0xA5; APD = 1;

	// Setting Pins as Output [(Less Significant) PE4 - PE5 - PG5 - PE3 - PH3 - PH4
- PH5 - PH6 (Most Significant) and Latch (PB4)]
	setBit(DDRE, PE4); setBit(DDRE, PE5); setBit(DDRG, PG5);
	setBit(DDRE, PE3); setBit(DDRH, PH3); setBit(DDRH, PH4);
	setBit(DDRH, PH5); setBit(DDRH, PH6); setBit(DDRB, PB4);

	// Setting PullUp PC6(Up) PC7(Select) PC5(Down)
	setBit(PINC, PC6); setBit(PINC, PC7); //setBit(PINA, PA7);

	//Tests
	//setBit(PINA, PA1); setBit(PINA, PA3); setBit(PINA, PA5); setBit(PINA, PA7);
setBit(PINC, PC4);
	setBit(PINL, PL0);

	// ADC Configuration
	Setup_ADC();

	// Timer 0 Configuration
	Setup_Timer0();

	// DISPLAY config
	lcdSetControlPort(&display, &DDRC, &PORTC, PC0, PC1);
	lcdSetDataPort(&display, &DDRC, &PORTC, PC2);
	lcdInit(&display, LCD_16X2, LCD_FONT_5X8);
	lcdStdio(&display);

	lcdClearScreen(&display);
	printf("Laser Controler\nUFSC 2019  ");
	_delay_ms(3000);
	lcdClearScreen(&display);

	// Enable Global Interrupts
	sei();

	Setup_SPI();

	Regulate_Spi();

	while(1){

		if		(newADC >= oldADC+2){conversor(); _delay_us(250);}
		else if (newADC <= oldADC-2){conversor(); _delay_us(250);}
		else if (newADC == 0)		{setBit(PORTB, PB4); _delay_us(250);
clrBit(PORTB, PB4);}
```

```
                lcdClearScreen(&display);

                printf("Pot:%3d%% APD:%d\n", percent, APD-1);

                warning_codes();

                _delay_ms(40);

                if (!isBitSet(PINC, PC6)){ APD = APD + 1; _delay_ms(200);}

                //else if (!isBitSet(PINA, PA1) || !isBitSet(PINA, PA3)
|| !isBitSet(PINA, PA5) || !isBitSet(PINA, PA7) || !isBitSet(PINA, PA6)){ APD = APD -
1; _delay_ms(200);}//  || !isBitSet(PINC, PC4)
                else if (!isBitSet(PINL, PL0)){ APD = APD - 1; _delay_ms(200);}

                switch(APD){
                        case 10: APD = 1; break;
                        case 0 : APD = 9; break;
                        default: break;}

                if (!isBitSet(PINC, PC7)){_delay_ms(200); Apply_APD();}

        }

        return 0;
}

// ------------------------------------------------------------------------
// Interruption handlers ---------------------------------------------------
ISR(ADC_vect)
{
        TIFR0 = (1 << OCF0A);

        newADC = ADC;

        if (newADC == 0){
                percent = 0;
                clrBit(PORTE, PE4); clrBit(PORTE, PE5); clrBit(PORTG, PG5); clrBit(PORTE,
PE3);
                clrBit(PORTH, PH3); clrBit(PORTH, PH4); clrBit(PORTH, PH5); clrBit(PORTH,
PH6);}
        else{newADC = (newADC+1)/4;}
}

// ------------------------------------------------------------------------
// Function definitions ----------------------------------------------------
void conversor(){
        int valor = newADC;

        if (valor >= 128){ valor = valor - 128; setBit(PORTH, PH6);}
        else {clrBit(PORTH, PH6);}

        if (valor >=  64){ valor = valor -  64; setBit(PORTH, PH5);}
        else {clrBit(PORTH, PH5);}

        if (valor >=  32){ valor = valor -  32; setBit(PORTH, PH4);}
        else {clrBit(PORTH, PH4);}

        if (valor >=  16){ valor = valor -  16; setBit(PORTH, PH3);}
        else {clrBit(PORTH, PH3);}

        if (valor >=   8){ valor = valor -   8; setBit(PORTE, PE3);}
```

```
        else {clrBit(PORTE, PE3);}

        if (valor >=   4){ valor = valor -   4; setBit(PORTG, PG5);}
        else {clrBit(PORTG, PG5);}

        if (valor >=   2){ valor = valor -   2; setBit(PORTE, PE5);}
        else {clrBit(PORTE, PE5);}

        if (valor >=   1){ valor = valor -   1; setBit(PORTE, PE4);}
        else {clrBit(PORTE, PE4);}

        setBit(PORTB, PB4);

        _delay_us(250);

        clrBit(PORTB, PB4);

        oldADC = newADC;

        percent = (newADC*100)/256;

}

void warning_codes(){

        if(isBitSet(PINB, PB5)){warning = warning + 1;}        // State 2 - Less
Significant Bit
        if(isBitSet(PINB, PB7)){warning = warning + 2;}        // State 1
        if(isBitSet(PINB, PB6)){warning = warning + 4;}        // State 0 - Most
Significant Bit

        switch(warning){
                case 0:
                if (systemFlags.Temperature_Alarm == 0){        systemFlags.allFlags =
0;      systemFlags.Temperature_Alarm = 1;}
                printf("Temp alarm"); newADC = 0;
                break;
                case 1:
                if (systemFlags.Power_Supply_Alarm == 0){       systemFlags.allFlags =
0;      systemFlags.Power_Supply_Alarm = 1;}
                printf("Pwr Supply Alarm"); newADC = 0;
                break;
                case 2:
                if (systemFlags.Normal_Operations == 0){        systemFlags.allFlags =
0;      systemFlags.Normal_Operations = 1;}
                printf("Normal");
                break;
                case 3:
                if (systemFlags.Laser_Not_Ready == 0){
        systemFlags.allFlags = 0;    systemFlags.Laser_Not_Ready = 1;}
                printf("Laser Not Ready"); newADC = 0;
                break;
                case 4:
                if (systemFlags.Back_Reflection_Alarm == 0){    systemFlags.allFlags =
0;      systemFlags.Back_Reflection_Alarm = 1;}
                printf("Back Reflection"); newADC = 0;
                break;
                case 5:
                if (systemFlags.Reserved_1 == 0){
        systemFlags.allFlags = 0;   systemFlags.Reserved_1 = 1;}
                printf("Reserved 1");
                break;
```

```
                case 6:
                if (systemFlags.System_Alarm == 0){
        systemFlags.allFlags = 0;   systemFlags.System_Alarm = 1;}
                printf("System Alarm"); newADC = 0;
                break;
                case 7:
                if (systemFlags.Reserved_2 == 0){
        systemFlags.allFlags = 0;   systemFlags.Reserved_2 = 1;}
                printf("Reserved 2");
                break;
                default:
                printf("Label Reading Error"); newADC = 0;
                break;
        }

        warning = 0;

}

void Apply_APD(){

        Write_uint8(Data_Set);

        Write_uint8(0x06);

        Write_uint8(APD);

        Regulate_Spi();

}
```

APÊNDICE 2 – BIBLIOTECA DE COMUNICAÇÃO SERIAL EM C

```
#ifndef SPI_EXPERIMENTAL_H_
#define SPI_EXPERIMENTAL_H_

	#define F_CPU 16000000UL

	#include <stdint.h>
	#include <avr/io.h>
	#include <util/delay.h>

	#define CLK_DDR DDRB
	#define CLK_PORT PORTB
	#define CLK_MASK (1 << PB1)

	#define MOSI_DDR DDRB
	#define MOSI_PORT PORTB
	#define MOSI_MASK (1 << PB2)

	#define MISO_DDR DDRB
	#define MISO_PIN PINB
	#define MISO_MASK PB3

	static void Setup_SPI (void)
	{
		CLK_DDR |= CLK_MASK;
		MOSI_DDR |= MOSI_MASK;
		MISO_DDR |= MISO_MASK;
	}

	static inline void CLK_L (void)
	{
		_delay_us(5);
		CLK_PORT &= ~CLK_MASK;
	}

	static inline void CLK_H (void)
	{
		_delay_us(5);
		CLK_PORT |= CLK_MASK;
	}

	static inline void MOSI_L (void)
	{
		MOSI_PORT &= ~MOSI_MASK;
	}

	static inline void MOSI_H (void)
	{
		MOSI_PORT |= MOSI_MASK;
	}

	static inline void Write_Bit (uint8_t x, uint8_t m)
	{
		CLK_L();
		if (x & m) MOSI_H(); else MOSI_L();
		CLK_H();
	}

	static void Write_uint8 (uint8_t x)
	{
```

```
	Write_Bit(x, (1 << 7));
	Write_Bit(x, (1 << 6));
	Write_Bit(x, (1 << 5));
	Write_Bit(x, (1 << 4));
	Write_Bit(x, (1 << 3));
	Write_Bit(x, (1 << 2));
	Write_Bit(x, (1 << 1));
	Write_Bit(x, (1 << 0));
}

static inline void Write_uint16(uint16_t x)
{
	Write_uint8((uint8_t)(x >> 8));
	//_delay_us(7);
	Write_uint8((uint8_t)(x & 0xff));
}

static inline void Read_Bit(uint8_t* x, uint8_t i)
{
	CLK_L();
	CLK_H();

	if (MISO_PIN & MISO_MASK) *x |= 1 << i;
}

static uint8_t Read_uint8(void)
{
	uint8_t x = 0;

	Read_Bit(&x, 7);
	Read_Bit(&x, 6);
	Read_Bit(&x, 5);
	Read_Bit(&x, 4);
	Read_Bit(&x, 3);
	Read_Bit(&x, 2);
	Read_Bit(&x, 1);
	Read_Bit(&x, 0);

	return x;
}

static inline uint16_t Read_uint16(void)
{
	const uint8_t x = Read_uint8();
	return ((uint16_t)x << 8) | (uint16_t)Read_uint8();
}

static void Regulate_Spi(void){
	CLK_L();
	_delay_us(7);
	MOSI_L();
}

#endif /* SPI_EXPERIMENTAL_H_ */
```

APÊNDICE 3 - CÓDIGO PYTHON

```
import sys
import glob
import time
import serial
import msvcrt
import threading
import tkinter as tk

global usb, control1, control2, control3, text_exit


def serial_ports():

    if sys.platform.startswith('win'):
        ports = ['COM%s' % (i + 1) for i in range(256)]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        ports = glob.glob('/dev/tty[A-Za-z]*')
    else:
        raise EnvironmentError('Unsupported Platform')

    result = []

    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass

    return result

def connect(ports):
    return serial.Serial(  port = ports[0],
                           baudrate=57600,
                           parity=serial.PARITY_NONE,
                           stopbits=serial.STOPBITS_ONE,
                           bytesize=serial.EIGHTBITS)

def check_port_open(ser):
    return ser.isOpen()

def close_actual_port(ser):
    ser.close();
    return

def prepare_commands(command):
```

```
    all_numbers = [int(i) for i in str(command)]
    return all_numbers

def send_command(command, port):
    ser = connect(port)
    ser.write(bytearray(command , 'ascii'))
    close_actual_port(ser)
    return

def read_feedback():
    ser = connect(usb)
    if check_port_open(ser):
          while self.portaSerial.inWaiting() > 0:
             caracLido = self.portaSerial.read(1)
             self.text1.insert(tk.END, caracLido);
       self.root.after(1,self.leSerial)

def readSerial():
    ser = connect(usb);
    serial = (ser.readline().strip())
    text_exit = serial.decode('ascii')
    close_actual_port(ser)

def get_port():
    usb = com_select.get()
    if int(usb) in serial_ports():
       root.after(1000, com_select.set("Connected to: "))
    else:
       root.after(1000, com_select.set("Not found: "))

def get_command():
    control1 = command.get()
    if control1 == "Command: ":
       root.after(1000, command.set("Received: "))
    else:
       root.after(1000, command.set("Command: "))

def get_param1():
    control2 = param1.get()
    if control2 == "Parameter 1: ":
       root.after(1000, param1.set("Received: "))
    else:
       root.after(1000, param1.set("Parameter 1: "))

def get_param2():
    control3 = param2.get()
    if control3 == "Parameter 2: ":
       root.after(1000, param2.set("Received: "))
    else:
       root.after(1000, param2.set("Parameter 2: "))
```

```
def setup():
    com_select.set("Connect to: ")
    command.set("Command: ")
    param1.set("Parameter 1: ")
    param2.set("Parameter 2: ")
    Laser_Answer.set("Nothing")
    control1 = " "
    control2 = " "
    control3 = " "

def send_data():
    if control1 == " ":
        root.after(1000, Laser_Answer.set("ERROR"))
    elif control2 == " ":
        command =  ('$' + control1 + '0x0d')
        send_command(command, usb)
    elif control3 == " ":
        command =  ('$' + control1 + ';' + control2 + '0x0d')
        send_command(command, usb)
    else:
        command =  ('$' + control1 + ';' + control2 + control3 + '0x0d')
        send_command(command, usb)

    control1 = " "
    control2 = " "
    control3 = " "

def print_answer(vector):
    root.after(1000, Laser_Answer.set(text_exit))


root = tk.Tk()
root.title("Controler v 1.0")
root.geometry('450x300')

com_select = tk.StringVar()

command = tk.StringVar()

param1 = tk.StringVar()

param2 = tk.StringVar()

Laser_Answer = tk.StringVar()

tk.Label(root, text="COM Ports Avaliable: ").grid(column=0, row=0)
tk.Label(root, text=str(serial_ports())).grid(column=1, row=0)

tk.Label(root, textvariable=com_select).grid(column=0, row=1)
```

```
tk.Entry(textvariable="Select Port: ").grid(column=1, row=1)
tk.Button(root, text="Connect", command=get_port).grid(column=2, row=1)

tk.Label(root, text=" ").grid(column=0, row=2)

tk.Label(root, textvariable=command).grid(column=0, row=3)
tk.Entry(textvariable="Select Command: ").grid(column=1, row=3)
tk.Button(root, text="Confirm Command",
command=get_command).grid(column=2, row=3)

tk.Label(root, text=" ").grid(column=0, row=4)

tk.Label(root, textvariable=param1).grid(column=0, row=5)
tk.Entry(textvariable="Select Parameter 1: ").grid(column=1, row=5)
tk.Button(root, text="Confirm Parameter 1",
command=get_param1).grid(column=2, row=5)

tk.Label(root, text=" ").grid(column=0, row=6)

tk.Label(root, textvariable=param2).grid(column=0, row=7)
tk.Entry(textvariable="Select Parameter 2: ").grid(column=1, row=7)
tk.Button(root, text="Confirm Parameter 2",
command=get_param2).grid(column=2, row=7)

tk.Label(root, text=" ").grid(column=0, row=8)

tk.Button(root, text="Send Data", command=send_data).grid(column=1, row=9)

tk.Label(root, text=" ").grid(column=0, row=10)

tk.Label(root, text="Laser Awser: ").grid(column=0, row=11)
tk.Label(root, textvariable=Laser_Answer).grid(column=1, row=11)

setup()

f1 = threading.Thread(target=readSerial)

f1.start()

root.mainloop()
```

ANEXO 1

Type	Command	Command code	Parameters or return values	Description/Parameters
Read	Device ID	1	string, up to 64 char	Read device identifier written to the laser in the factory
Read	Device SN	2	string, up to 24 char	Read device serial number
Read	FW revision	3	string, up to 255 char	Read device firmware revision
Read	Vendor	99	string, up to 255 char	Read device vendor written to the laser in the factory
Read	Device Status	4	up to 32 bit integer	Read device status, decimal to binary decoding is required
Read	Device temperature	5	float, 1 digit after point	Read module temperature in degree Celsius
Read	Digital interface Status	10	up to 32 bit integer	Reads digital interface status, decimal to binary decoding is required
Read	Extended Status	11	up to 32 bit integer	Read device extended status, decimal to binary decoding is required
Read	BR Counter	12	up to 32 bit integer	Read back reflection counter
Read	Session BR Counter	13	up to 32 bit integer	Read back reflection counter for the current session. The session starts with supplying voltage to the laser module.
Read	Nominal average Power	14	float, 1 digit after point	Read nominal average power of the laser in [W] Return value is float in [W].
Read	Nominal Pulse Duration	15	float, up to 6 digits after point	Read nominal pulse duration of the laser [ns]
Read	Nominal Pulse Energy	16	float, 2 digit after point	Read nominal pulse energy of the laser [mJ]
Read	Nominal Peak Power	17	float, 1 digit after point	Read nominal peak power of the laser in [kW]. Value is calculated from the nominal energy and the nominal pulse duration.
Read	PRR Range	18	see description	Read pulse repetition rates range. Return value is two floats separated by a semicolon, corresponding to minimum and maximum PRR [kHz].

Type	Command	Command code	Parameters or return values	Description/Parameters
Read	Head Temperature	19	float, 1 digit after point	Read remote head temperature in degree Celsius, if the head is installed
Read	Main Supply Voltage	21	float, 1 digit after point	Read main 24V supply voltage in [V]
Read	24V Housekeeping Voltage	22	float, 1 digit after point	Read 24V housekeeping supply voltage in [V]
Read	Operating Mode	23	32 bit integer	Read active control interface operating mode, decimal to binary decoding is required.
Set	Operating Mode	24	32 bit integer	Set active control interface operating mode, binary to decimal encoding is required. The command parameter is validated before the execution. In case some bits are not correct, the command is not executed. All 32 bits (even unused) should have corrects values. To set correctly all bits, the existing operating mode should be read by the command 24, only necessary bits should be updated and then new value sent to the device.
Read	Installed Options	25	32 bit integer	Read list of installed options and operating modes, decimal to binary decoding is required
Set	Start Operating Mode	26	32 bit integer	Set initial control interface operating mode, binary to decimal encoding is required. This mode becomes active after supplying the laser with electrical power. Value is stored permanently in the laser EEPROM.
Read	Start Operating Mode	27	32 bit integer	Read control interface operating mode, which activates after connecting the laser to the supply voltage. The value is stored permanently in the laser EEPROM, decimal to binary decoding is required
Read	Operating Power [W]	33	float, 2 digit after point	Read back operating power in [W] set by command 32 (in RS-232 mode) or via digital interface (in DB-25 mode), but recalculated into Watts using nominal laser parameters.
Read	Operating Power [%]	34	float, 2 digit after point	Read back operating power in [%] set by command 32 (in RS-232 mode) or via digital interface (in DB-25 mode).
Read	Operating Pulse Energy	36	float, 2 digit after point	Read operating pulse energy in [mJ]. Value is calculated using nominal laser parameters and power settings.
Read	PRR monitor	38	float, 1 digit after point	Read back operating PRR in [kHz] set by command 28 (in RS-232 mode) or applied via Sync input of digital interface (in DB-25 mode)

Type	Command	Command code	Parameters or return values	Description/Parameters
Read	Alarm counters	70	16 bit integer	Read alarm counters. The command contains a parameter which specifies the alarm counter: 1 – 24V main supply 2 – 24V housekeeping supply 3 – System 4 –Temperature 5 –Head Temperature
Read	Module Temperature range	58	float, 1 digit after point	Read operating temperature range. Return value is two floats separated by a semicolon, corresponding to minimum and maximum temperatures in degree Celsius.
Read	Nominal frequency	59	float, 1 digit after point	Read back nominal PRR in [kHz]
Read	Critical error counter	95	16 bit integer	Read critical error counter
Read	Critical error code	96	32 bit integer	Read critical error code
Set	Reset critical error alarms	97	32 bit integer	Reset the critical error. The command contains one parameter, which is a one-time use code generated in IPG factory. In case the correct code is sent to the device, the command is executed with answer "Y" and critical error is cleared.

Type	Command	Command code	Parameters or return values	Description/Parameters	Equivalent DB-25 control line
Set	Set PRR	28	float, 1 digit after point	Set operating pulse repetition rate in [kHz]	Sync
Read	Read PRR	29	float, 1 digit after point	Read back operating pulse repetition rate in [kHz] set by command 28	
Set	Laser Emission ON	30		Switch ON laser emission	EM
Set	Laser Emission OFF	31		Switch OFF laser emission.	
Set	Operating Power	32	float, 1 digit after point	Set operating power in [%]. Range 0...100, resolution 255 levels for the full scale	D0-D7 & Latch
Set	Guide Laser ON	40		Switch ON guide laser	RG
Set	Guide Laser OFF	41		Switch OFF guide laser.	
Set	EE ON	42		Switch ON Emission Enable	EE
Set	EE OFF	43		Switch OFF Emission Enable	
Set	Reset Alarms	50		Reset alarms, see alarms description for details	Reset Sequence

Type	Command	Command code	Parameters or return values	Description/Parameters
Read	Read the number of APD modes	55	16 bit integer	Read number of APD modes (N)
Read	Read APD mode description	56	Parameter: M Answer: string, up to 128 char	Read a text description of APD mode with index M. Parameter is M is APD mode index, integer, range 0 to N-1. N is number of APD modes read by command $55
Read	Read APD mode index	68	16 bit integer	Read current APD mode index.
Set	Set APD mode index	69	16 bit integer	Set APD mode index.
Set	Save APD mode index	54		Permanently save the APD mode index to EEPROM. Next start the device will be initialized by saved APD mode index.

Type	Command	Command code	Parameters or return values	Description/Parameters
Read	Maximum Prepump	63	16 bit integer	Return maximum value of the prepump compensation. The value is always 10000.
Set	Prepump	64	16 bit integer	Set the prepump compensation value. Range is 0...10000
Read	Prepump	65	16 bit integer	Read back value of the prepump compensation set by command $64 or through DB-25 serial interface