PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E ARTES PUC GOIÁS
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

MANUTENÇÃO E MONTAGEM DE IMPRESSORA 3D DE UMA ADAPTAÇÃO DE UMA FRESADORA CASEIRA

FREDERICO CÉSAR PERILLO DA VEIGA JARDIM

GOIÂNIA
2023

FREDERICO CÉSAR PERILLO DA VEIGA JARDIM

MANUTENÇÃO E MONTAGEM DE IMPRESSORA 3D DE CONSTRUÇÃO DE UMA FRESADORA CASEIRA

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador:
Prof. M.E.E. Marcelo Antonio Adad de Araújo

GOIÂNIA
2023

FREDERICO CÉSAR PERILLO DA VEIGA JARDIM

MANUTENÇÃO E MONTAGEM DE IMPRESSORA 3D DE CONSTRUÇÃO DE UMA FRESADORA CASEIRA

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica e Artes, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Engenharia de Computação, em _____/_____/_____.

_____
Prof. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

_____
Orientador: Professor M.E.E. Marcelo Antonio Adad de Araújo

_____
Prof. M.Sc. Mirian Sandra Rosa Gusmão

_____
Prof. Dr. Antônio Marcos Melo Medeiros

GOIÂNIA
2023

A aqueles que acreditaram no impossível de me formar.

Ao professor Marcelo Adad por não ter desistido de mim e ao Professor Antônio Marcos pelo auxílio em momentos de desespero.

E a aqueles que me incentivaram a continuar e não desistir para fazer curso técnico

RESUMO

O presente trabalho apresenta a utilização de uma CNC (*Computer Numeric Control*) utilizada para adaptação de uma impressora 3D artesanal, para utilizar na criação de esculturas em filamento, usando algumas partes reutilizadas de um computador, e o hardware de um microcontrolador, bem como a utilização de uma CNC *Shield* V3 juntamente com um Arduino Uno. Focando principalmente na manutenção e a montagem da mesma. Atualmente, as esculturas são limitadas por tamanho e proporção aos limites da área de trabalho da impressora e com a tecnologia até a presente data. Mas não foi possível fazer a impressão de uma peça, apenas o derretimento do filamento pela extrusora. A realização desse trabalho é a manutenção e montagem de uma impressora 3D com base em uma adaptação de máquina ferramenta com controle numérico computadorizado controlado via software que se movimenta em três eixos (X, Y e Z).

***Palavras-Chave**: Arduino Uno. Manutenção. Impressora 3D. Montagem. CNC. CNC Shield V3. G-code. GRBL. PVC*

ABSTRACT

The Present Study sought the use of a CNC (Computer Numeric Control) used for adapting an home made 3D printer, to make filament sculptures, using reuseful computers parts, with a microcontroller hardware, just using a CNC *SHIELD* V3 together with a Arduino Uno plate. Principal focus on setting and maintence of it. Actually, the sculptures are limited by size and printer work area proportion limits and its technology until the current due date. But was not possible to made a sculture, just make extrucsure melt the filamento. This possibility to do this work, by this, is proposed: the setting and maintence of a 3D printer adapting with a machine tool base with a computer numeric control, controlled by a software in three axes. (X, Y and Z),

Keywords: *Arduino Uno. Maintence. 3D Printer. Setting. CNC. CNC Shield V3. G-code. GRBL. PVC.*

LISTA DE FIGURAS

LISTA DE SIGLAS

3D	3 Dimensões
AGC	*Apollo Guidance Computer*
APT	*Automatically Programed Tools*
AutoPromp	*Automatic Programming of Machine Tools*
BIOS	*Basic Input/Output System*
CAD	*Computer-Aided Design*
CAM	*Computer-Aided Manufacturing*
CD	*Compact Disc*
CI	Circuito Integrado
CNC	*Computer Numeric Control*
CMYK	*Cyan, magenta, yellow, key*(referência a preto *BLACK* para não confundir)
CPS	*Character Per Second*
DPI	*Dots Per Inch*
E/S	Entrada/Saída
EIA	*Eletronics Industry Association*
IBM	*International Business Machines*
IDE	*Integrated Development Environment*
LED	*Light-Emitting Diode*
LPM	*Line Per Second*
MIT	Instituto de Tecnologia de Massachusetts
NC	*Numerical Control*
PCB	*Printed Circuit Board*
PPS	*Page Per Second*
PVA	Poliacetato de vinila
ROM	*Read Only Memory*
RUR	*Rossumovi Univerzální Roboti*
SMPS	*Switched Mode Power Supply*
SPI	*Serial Peripheral Interface*
SVG	*Scalable Vetor Graphics*
TCC	Trabalho de Conclusão de Curso
UGS	*Universal G Code Sender*
ULA	Unidade Lógico Aritmética
USB	*Universal Serial Bus*
UV	Ultravioleta
VCC	Tensão Corrente Continua

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Levando em consideração o cenário do mercado de computação gráfico-industrial, a tecnologia avança esporadicamente, e vai de encontro à redução dos custos de forma a deixar mais acessível à população. Empresas de tecnologia vem apresentando novas ideias revolucionárias todos os dias, têm confiado mais nas inovações, a cada dia mudando a trajetória da humanidade. (SAMPAIO, 2021).

Muitas das atividades têm se transformado por necessidade do avanço tecnológico: troca de informações, acesso que transpõe a dificuldade com a agilidade, possibilitando assim a construção de pequenas máquinas em casa, até mesmo dispensando a necessidade de grandes máquinas (MALHEIROS & VARGAS, 2019).

A história da impressora se inicia em 1938, por meio do americano Chester Carlson. Fazendo isso por meio de fotocópias de imagens e textos, em 1953 implementou-se a primeira impressora em super velocidade que também era utilizada pela universidade Univac. As impressoras foram evoluindo e de acordo com a evolução da computação seja em nível de processamento quanto em cálculo de ponto flutuante, já em 1983 as impressoras atuais como a laser foram criadas. (HISTÓRIA DE TUDO, 2020).

Após a capacidade de desenvolver uma projeção gráfica tridimensional, também é possível abstrair uma modelagem em três dimensões. Assim no final dos anos 80 se iniciou a produção em prototipagem rápida (SRP), ou *fabber,* onde produziam pequenos objetos a partir do CAD (*Computer Aided Design*), onde se pode projetar e visualizar um objeto (FRANCISCO, 2012).

Devido ao emprego de todo tipo de maquinários industriais, viabilizou-se o aumento na produção mecânica, ocorrendo então, inovações tecnológicas que não existiam implementadas na época, por necessidade de competir diretamente. (NEVES; SOUSA, 2019).

Sucedeu naquele período, a substituição da força muscular humana ou de animais em forças hidráulicas, a vapor e posteriormente com o emprego de atuadores elétricos. Também se substituiu a precisão e talento humano na produção em massa, usando uma estrutura padronizada. Contemporaneamente o avanço tecnológico e aprimoramentos viraram uma necessidade (NEVES; SOUSA, 2019).

Ainda em 1948, as máquinas industriais passaram a ser controladas numericamente. E posteriormente computadorizadas, usando a tecnologia CNC (SMARTEC, 2018).

Com um código específico, máquinas controladas por um computador podem ser definidas como “controle numérico computadorizado”, Código G. Atualmente no meio

industrial, é bastante comum encontrar algum equipamento que utiliza esta tecnologia. Para máquinas CNC, pode-se destacar: plotter, fresadoras mecânicas, tornos, impressoras 3D, máquinas de cortes e uma infinidade de dispositivos que necessitem de um controle numérico adaptado. Essas máquinas controladas possuem uma quantidade específica de graus de liberdade, que no caso serão 3 graus lineares, tanto para movimentos de translação quanto de rotação, que define o alcance de aplicação do dispositivo de forma a gerar geometrias adaptáveis da melhor forma possível ao trabalho a ser realizado pela máquina ferramenta (FORD, 2016).

Justifica-se a necessidade do presente trabalho, na manutenção de um equipamento didático, utilizado nas dependências da PUC Goiás, utilizado para os mais diversos fins.

Em primeira instância, neste capítulo apresenta os principais objetivos e métodos para realização deste trabalho, elucidando os resultados esperados, lista de atividades, problemas e testes.

No capítulo dois, e em suas seções, será apresentado o referencial teórico e assuntos importantes para melhor entendimento do trabalho.

O capítulo três apresenta a solução, mostrando os componentes de hardware e software a utilizar na criação de uma impressora 3D artesanal a partir de uma fresadora mecânica.

O capítulo 4 é o plano de montagem e implementação da impressora.

Capítulo 5 de considerações finais

Capítulo 6 é os resultados obtidos.

1.2 Objetivos

Esta seção descreve o objetivo geral e objetivos específicos do trabalho. A proposta e a ideia a ser atingida.

1.2.1 Objetivo Geral

O presente trabalho tem por objetivo adaptar um impressora 3D artesanal a oartir de uma fresadora caseira que se utiliza de um Arduino Uno e uma CNC *Shield* V3, sistemas semelhantes a um controle numérico computadorizado.

1.2.2 Objetivos Específicos

O primeiro objetivo específico está diretamente relacionado a se utilizar os conhecimentos adquiridos no curso de Engenharia de Computação, fazer a manutenção da ferramenta-máquina.

O segundo objetivo específico refere-se à necessidade de várias ferramentas para produzir uma atividade específica e a proposta é produzir uma ferramenta multi-tarefa apenas modificando uma parte para produzir uma impressora com maior praticidade E usando uma solução computacional.

Um terceiro objetivo é adquirir experiência nas ferramentas de softwares e hardwares a serem utilizadas neste trabalho.

1.3 Justificativa

O trabalho a ser apresentado tem como justificativa primordial a criação de peças 3D de forma caseira a partir de outra máquina-ferramenta, de forma adaptativa. Uma vez que as máquinas ferramentas no mercado serem extremamente caras, e fazer uma máquina intercambiável tornando o trabalho algo mais útil. Que irá proporcionar uma visão de "*make it yourself*", ou seja: "faça você mesmo" com a ideia de ser fácil fazer a troca do cabeçote e apenas a modificação do programa.

1.4 Métodos

O presente trabalho segundo sua natureza é uma apresentação de uma aplicação da automação para área de impressão 3D, baseado em teorias já construídas com uma implicação prática em sua realização. Para tal, em sua fundamentação teórica, será utilizado em pesquisas bibliográficas e experimentais.

Inicialmente será realizado um estudo sobre a fundamentação teórica, de assuntos relacionados à área de projetos tridimensionais de impressão e automação, com um microcontrolador integrado a uma CNC, confeccionada a partir de sucatas de computadores pessoais e uma impressora industrial, fornecendo conceitos e definições necessários para a compreensão do tema, utilizando fontes confiáveis de artigos, livros, apostilas e periódicos.

No desenvolvimento de produção de um controlador numérico computadorizado (CNC), foi adaptado com base numa fresadora 3D caseira para uma impressora 3D caseira. Com a utilização de um código específico, o código G, é possível o controle simultâneo dos eixos X, Y e Z utilizando o Marlin, Ultimaker Cura, Pronterface e o Universal Gcode Sender no controlador para criação de esculturas tridimensionais impressas. (CCV INDUSTRIAL, 2019).

1.5 Resultados esperados

Com o presente trabalho, relacionado aos objetivos gerais e específicos que sejam alcançados, um funcionamento da impressora 3D e extrusão completa do filamento de PVA a manutenção e montagem feitas com êxito. A expectativa que a impressora caseira, produzido *in home,* pelo desenvolvedor acadêmico, é abranger os horizontes de novas ideias e oportunidades para a adaptabilidades de projetos para mercado de máquinas multitarefas.

Espera-se que os resultados do TCC, possam auxiliar acadêmicos, universidades ou empresas na construção de máquinas mais adaptativas ou com ideias adaptativas para este mesmo projeto que possam servir em trabalhos específicos de manufatura de design ou projetos tridimensionais.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo abrange o referencial teórico relevante e necessário para a composição desse trabalho, através de conceitos e definições utilizadas para melhor compreensão dessa aplicação.

2.1 O que é Impressora?

Em 1938, Charles Chester Carlson, um estadunidense, inventou o processo de reprodução de imagens e textos por meio de máquinas fotocopiadoras. Em 1953 criaram a primeira impressora em alta velocidade. A mesma foi utilizada no *UNIVAC,* o primeiro computador comercial fabricado nos Estados Unidos da América. Na Figura 1- Charles Carlson inventor da impressora.

Figura 1- Charles Carlson

fonte:www.netland.net

As impressoras são distinguidas pelo tipo de tecnologia usada na impressão, sejam as mesmas em CPS (caracteres por segundo), LPM (linhas por minuto), até as *laserjet* que são classificadas em PPM (páginas por minuto).

A qualidade de impressão é definida por quantidade de pontos por polegada quadrada (DPI). As impressoras de agulhas ou matricial utilizam micro agulhas para detalhar a pontuação de caracteres especiais modulando cada caractere pelo seu tamanho na matriz, replicando algo semelhante a uma máquina de escrever. Já as impressoras de jatos de tinta, utilizam vários tipos de jatos coloridos, usando a cadeia de cores CMYK (ciano, magenta, amarelo e preto). Essa tinta é ionizada no papel por meio de pratos ionizados da impressora. Porventura, a impressão à *laser* utiliza laser sobre uma transparência em acetato especial, assim a qualidade da impressão sendo de boa qualidade. Como é precisa pelo efeito fotoelétrico do

laser ela tem bastante contraste e mistura bem o exemplo de texto e imagem. (INFOPEDIA, 2022). Figura 2 e 3 mostram a impressora matricial de alta velocidade da *UNIVAC* e a impressora a laser de 1981

Figura 2- impressora matricial de alta velocidade da Univac

Fonte:www.timetoast.com

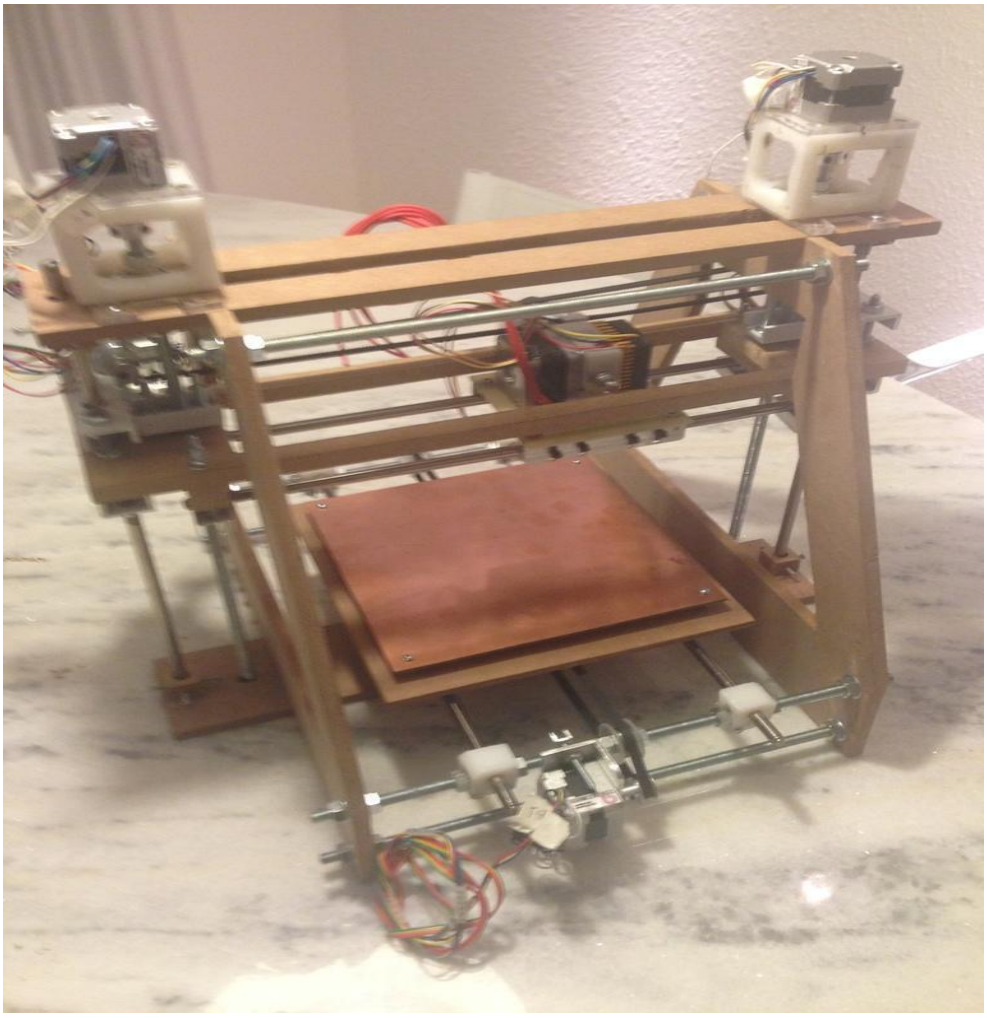
Figura 3- Impressora a Laser 1983

Fonte: www.timetoast.com

2.2 Impressora 3D

Na década de 40 foram criados os protótipos de impressão tridimensional. Logo após Charles Carlson criar a impressora comum, as ideias vieram com impressão de imagens tridimensionais. Dessa forma, Carvalho (1999) entendia que as máquinas que faziam a subtração, ou seja, eles esculpiam em um material fixo retirando o excesso da imagem presente dentro. Ela gera objetos através de líquidos, metais ou papéis. Um modelo 3D é dividido em camadas transversais assim ele seria gerado por cada camada. A Figura 4 exemplifica uma Impressora 3D

Figura 4- Impressora 3D caseira

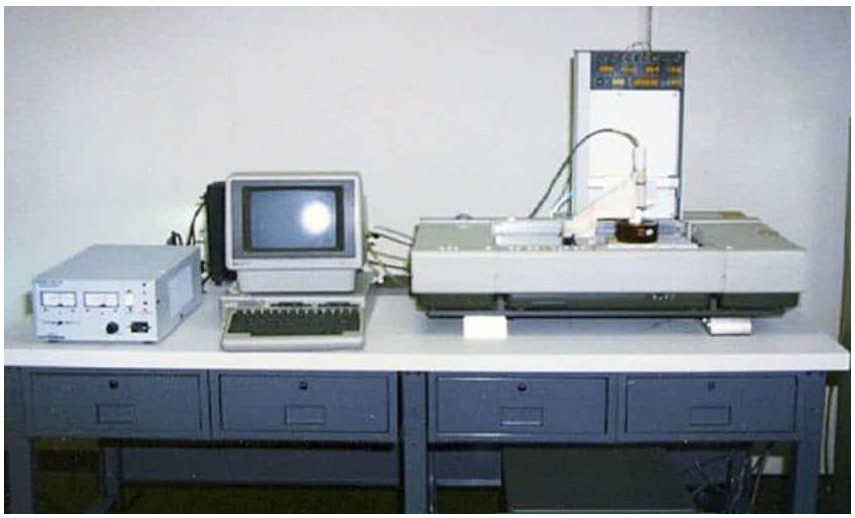


Fonte: Mesa, 2013

No cotidiano o que se referência como impressora 3D surgiu em 1984, Charles "Chuck" Hull, que trabalhava numa empresa de lâmpadas UV, sugeriu à empresa usá-las para

outra finalidade: curar resina fotossensível para criar produtos. Conforme mostrado na Figura 5 (GONÇALVES,2020)

Figura 5- Impressora 3D 1984



Fonte: Gonçalves, 2020

Baião (2012) diz que os desenhos gerados pela ferramenta CAD são monocromáticos e se aproximam do modelo sólido por pequenas facetas e triângulos. Quanto menores os triângulos, melhor será a definição da imagem, uma vez que, pode-se detalhar e aumentar o número de camadas de impressão. Porém, isto irá aumentar consideravelmente o tempo de processamento.

> "O processo de impressão funciona de modo que o filamento sai do cartucho e entra pela extrusora até chegar ao ponto onde é aquecido a 120°C. "(Baião, 2012)

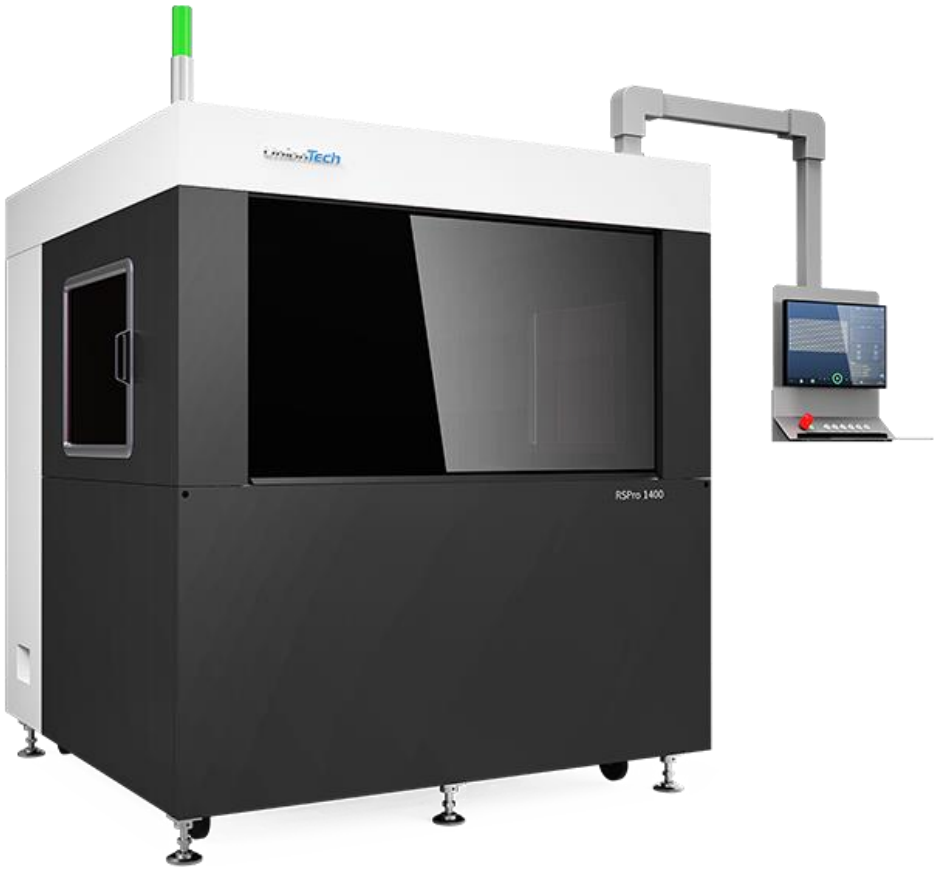
A impressora tem duas cabeças que colocam o suporte e o molde em camadas finíssimas, a espessura não pode ser vista a olho nú. As partes móveis são feitas já modulares no molde para serem liberadas e destacadas após a impressão. O molde sai ainda quente da impressora e constrói a imagem por camadas como se estivesse imprimindo, sem *botton up* da forma que foi orientado. (Baião, 2012)

Com a evolução da impressora se fez novos métodos de impressão, a primeira, e mais comum, é a citada anteriormente de impressão em lâminas, que são as camadas de impressão. O segundo método consiste em aplicar jatos do material em pó por meio de um cartucho de impressão, que são unidos de forma seletiva por outro cartucho com conteúdo adesivo. É a atual mais rápida que existe de acordo com Pankiewicz, esse método também permite a aplicação colorida na impressão não dependendo do filamento. (PANKIEWICZ, 2009).

Uma outra variação é a de fotopolímeros em estado líquido, que são injetados e tratados em camadas por meio de uma lâmpada UV (Ultravioleta). A combinação de preto e branco cria tons de cinza, que é utilizado em eletro-eletrônicos.(PANKIEWICZ, 2009).

Ilustra-se, na Figura 6, o ultimo modelo mais complexo de impressoras 3D da UnionTECH para o mercado de impressão de peças automobilísticas 2022. (UnionTech, 2022).

Figura 6 - Impressora RSPRO 1400 da union tech de dimenssões de 1400 x 700 x 500 mm



Fonte:UnionTech,2022

2.3 Tecnologia CNC desde a origem

Conforme afirma Sousa (1998), as máquinas-ferramenta atuais são usadas desde a antiguidade. O torno mecânico, por exemplo, é desde a antiguidade, a máquina-ferramenta mais utilizada. A invenção da primeira fresadora primitiva é creditada a um artesão italiano chamado Torriano, por volta do ano de 1540. Torriano construiu uma espécie de relógio

planetário para o rei Carlos V da Espanha. Este relógio continha cerca de 1800 engrenagens e levou 3 anos e meio para ser construído.

Em 1949 o resultado das pesquisas foi surgido em Massachusetts, no laboratório do Instituto de Tecnologia de Massachusetts (MIT), com a união da Força Aérea Norte-americana (U.S. Air Force) e a empresa Parsons Corporation of Traverse City, Michigan, onde foi adotada uma fresadora de três eixos, a Hydrotel se tornou a cobaia das experiências na Cincinnati Milling Machine Company. A primeira aparição do comando numérico, substituindo os controles e comandos convencionais, composto por uma leitora de fita de papel perfurado, unidade de processamento de dados e servomecanismo nos eixos. Após anos de testes, a máquina veio a funcionar ao público em março de 1952, e em 1953 o relatório final foi publicado em maio daquele ano. A U.S. Air Force começou a saltar em desenvolvimento de forma estupenda, pois as peças complexas e de grande precisão, empregadas na fabricação das aeronaves, principalmente os aviões a jato de uso militar, a partir desse momento são desenvolvidas de forma rápida, precisa e barata, reduzindo-se os prazos de entrega do produto desde o projeto, até o acabamento final. A indústria aeronáutica sempre foi precursora no desenvolvimento e avanço no CN, atualmente mais voltada para o comando de desenvolvimento IA autônomo.

Revoluções efémeras aconteceram na década de 50. A partir de 1957, houve nos Estados Unidos da América, uma grande corrida na fabricação de máquinas com CN, pois agora o CN era transformado em máquinas convencionais. Este novo processo foi cada vez mais utilizado na manufatura. Assim surgindo novos fabricantes e novas multinacionais usando até seus próprios comandos e utilização para o CN.

Devido ao grande número de fabricantes e multinacionais, começaram a surgir problemas, sendo a multilinguagem e a falta de padronização. A falta de padronização era bastante sentida em empresas que tinham mais de uma máquina de comandos, sendo necessário programar cada máquina de forma diferente da outra, então diferentes técnicos e diferentes preços, e diferentes técnicas, o que eleva bastante os custos de fabricação. Em 1958, por intermédio da EIA (Eletronic Industries Association) organizou uma associação para padronização do. Houve então a padronização de entrada conforme padrão RS-244 que depois passou a EIA244A ou ASC II. Atualmente o meio mais usado de entrada de dados para o CNC é via computador, mas já foram usados fita perfurada, comandos manuais, posicionamento manual, entre outros de menor destaque. A linguagem destinada a programação de máquinas era a APT (Automatically Programed Tools), desenvolvida pelo MIT, daí para frente foram desenvolvidas outras linguagens para a geração contínua de contornos como AutoPrompt

(Automatic Programming of Machine Tools), Action e outros que até hoje surgem para novas aplicações. O circuito integrado, fez com que houvesse grande redução no tamanho de comandos, embora sua capacidade de armazenamento tenha aumentado. Em 1967 surgiram no Brasil as primeiras máquinas CN, vindas do exterior. No início da década de 70, surgem as primeiras máquinas CNC (Controle Numérico Computadorizado), e no Brasil a Gurgel inicia a fabricação CN nacional. A partir deste período, a evolução das máquinas ocorre juntamente com a evolução computacional, fazendo com que os comandos (CNC) sejam empregados cada vez amis capacidade de processamento, velocidade e precisão. Com isso, a confiabilidade do sistema está cada vez mais precisa e a padronização perfeita.

2.4 Motor de passo

Com a necessidade da modernização das máquinas CNC e com o intuito de atender o mercado, a necessidade de controlar os motores por coordenadas surgiu, o francês Marius Lavet, desenvolveu em 1936 o motor de passo, ou em inglês, *stepper motor*. Os motores de passo são dispositivos que convertem sinais elétricos em energia mecânica deslocando o movimento por uma frequência em passos. Os motores de passo são divididos em três tipos: relutância variável, magneto permanente e híbridos. (RODRIGUES; DELMONDES, 2016). Neste projeto focaremos no motor de passo Híbrido, que é o utilizado.

O motor de passo é um equipamento que trabalha a frequencia de alimentação em passos de forma proporcional a sua velocidade de rotação, o que faz dele um dispositivo Síncrono. Ele é capaz de girar em um número específico de graus, 7,5° ou 15° a cada pulso. O motor recebe os dados da *CNCSHIELD* por meios de sinais condicionados, aonde são transformados nos movimentos dos eixos X,Y,Z (RODRIGUES; DELMONDES, 2016).

Um dos motores utilizados para o projeto é apresentado na Figura 7, este sendo um motor Hibrido.

Figura 7 - Motor de passo Híbrido utilizado no trabalho

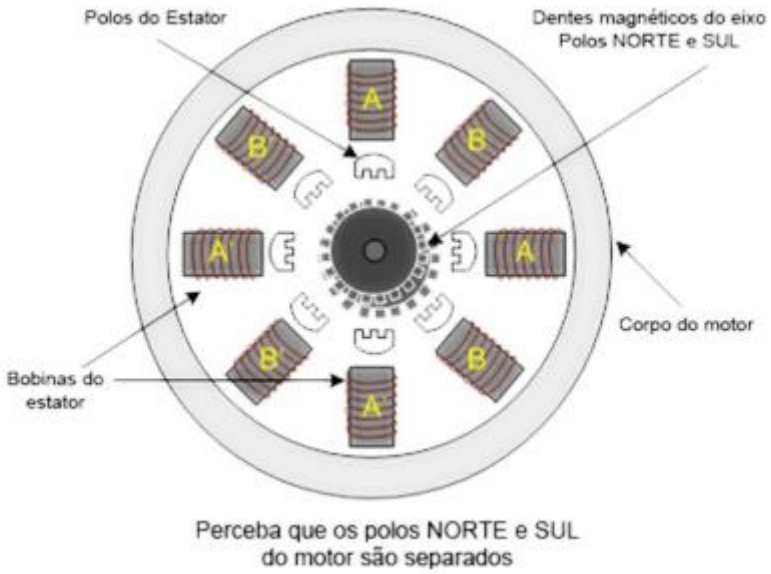
(Fonte: Rodrigues ; Delmondes, 2016)

2.4.1 Motor de passo Híbrido

Os motores híbridos podem ser de 3,4 ou 5 fases, sendo o de quatro fases o mais comum no mercado. O mesmo tem as melhores características possíveis dos motores de passo de imã permanente e relutância variável por este ser hibrido.

A estrutura do motor é de um eixo de polos Norte/Sul alternados, então o eixo do rotor tem dois grupos de dentes no Motor Hibrido. A Figura 8 mostra o motor com os imãs dentro.

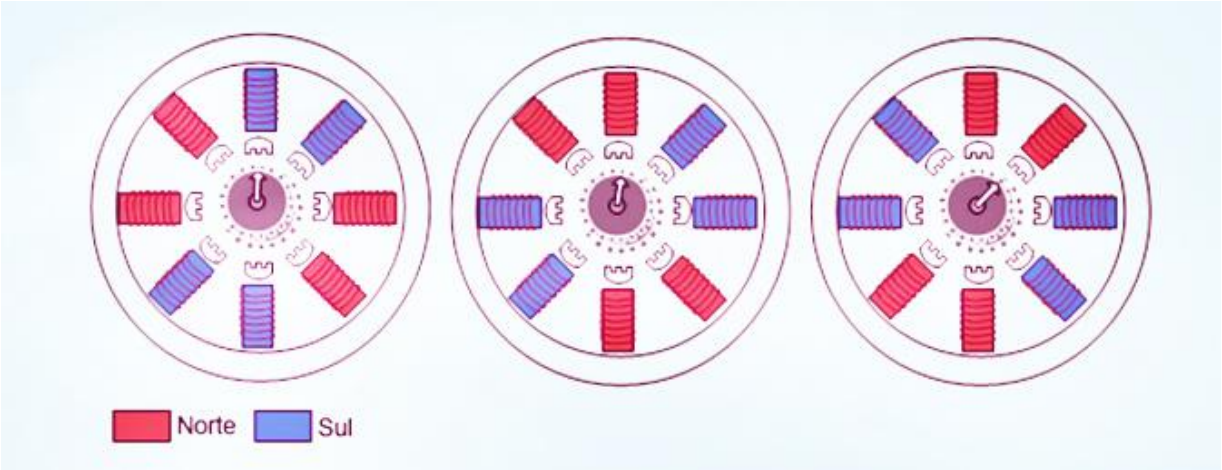
Figura 8 - Motor de Passo Hibrido



Fonte: Rodrigues ; Delmondes, 2016

A figura 9 por sua vez exemplifica como os imãs dentados tem que estar sempre alternados durante a passagem de rotação e passos

Figura 9 - Funcionamento do Motor de passo Híbrido



(Fonte: adaptado Rodrigues; Delmondes, 2016)

3 SOLUÇÃO DO PROBLEMA

Este capítulo representa a solução adotada no referido trabalho de conclusão de curso.

O projeto concluído é em base da manutenção e montagem então a primeira proposta é a montagem da impressora. A impressora uma vez montada será verificado o funcionamento dela. Após isso será feito a manutenção trocas e acertamentos do projeto com base nos estudos e pesquisa desenvolvida. Detalhada pelos pontos explicados.

Posteriormente serão tratados os componentes de software para a aplicação da Execução da impressão.

3.1 Componentes de Hardware

Esta seção introduz os componentes físicos do hardware para serem implicados na formação do sistema computacional da impressora.

3.1.2 Fontes de alimentação

As fontes chaveadas, comutadas ou do inglês SMPS *(Switched Mode Power Supply)* são fontes que controladoras de tensão em carga que abre e fecha um circuito transformador, transistorizado para manter o tempo de abertura e fechamento deste circuito com a tensão desejada.

Estas fontes chaveadas são caracterizadas pelo seu alto rendimento, tendo seu próprio dissipador de calor com um cooler próprio e chegando a fornecer toda energia que os circuitos das cargas precisam para o funcionamento normal. A mesma é Bivolt (110V-220V com switch de tensão), com amperagem de 20A e VCC (Tensão Corrente Continua) de 12 Volts para saída (Porém na prática a tensão elétr4ica da fonte chaveada chega aos 10,6 Volts e 19,4 Amperes.), (Adaptado Rodrigues; Delmondes, 2016).

A figura 10 é a demonstração e uma fonte chaveada comercial.

Figura 10 - Fonte chaveada ilustrativa da fonte utilizada na Impressora

Fonte: Banco de imagens google

3.1.3 Arduino Uno

Um microcontrolador consiste em um pequeno computador onde são concentradas todas as funções principais em um único chip de circuito integrado (CI).(GARCIA,2018). O microprocessador é um sistema dependente de sua CPU e um conjunto de periféricos necessários para o seu funcionamento. Pode-se citar memória de dados, de programa e o circuito de *clock*. Diferente de um sistema tradicional, independe de periféricos separados pois já os tem dentro da pastilha. (Kerschbaumer,2018)

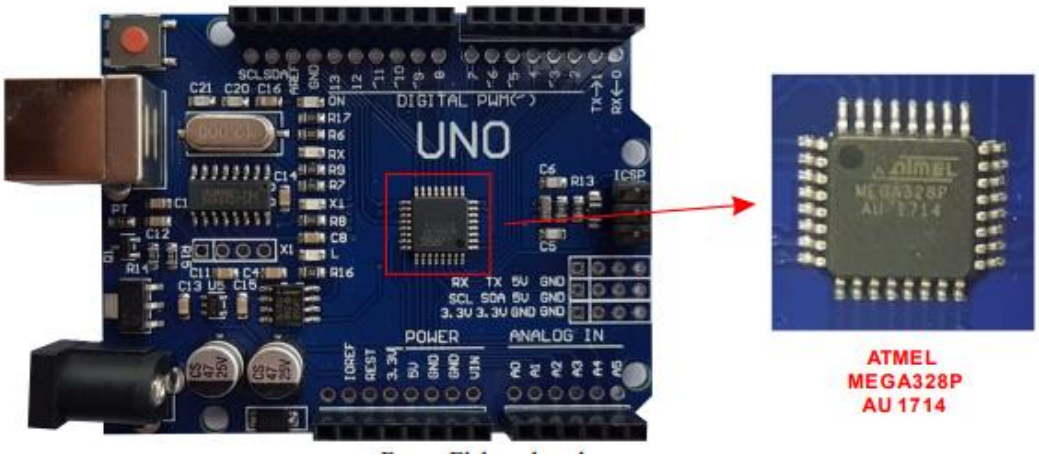
Extremamente versáteis, os microcontroladores são utilizados pelo seu comportamento que depende principalmente da matriz de controle ampliada presente nele. Assim, o mesmo microcontrolador pode ser utilizado para uma ou várias funções diferentes apenas mudando o código de programação. (Kerschbaumer,2018).

O uso do Arduino oferece diversas vantagens em relação a outros microcontroladores. Seu software é multiplataforma, funcionando em Linux, Windows e Mac, além disso, por ser *open source,* é possível expandi-lo com o uso de bibliotecas C++. Já seu hardware é barato e acessível, e por também ser open source é possível expandi-lo com módulos personalizados (ABOUT, 2021).

O Arduino Uno é uma placa de microcontrolador que possui código aberto, sendo assim, software livre e um circuito físico programável. Ele é baseado no microcontrolador Microchip ATmega e foi desenvolvido pela Arduino e o Uno é um tipo de placas dessa família Arduino. O microcontrolador citado possui um conjunto de pinos de entrada e saída que são programáveis com o software Arduino IDE usando a linguagem de programação C/C++ com pequenas modificações. Sua entrada é um cabo USB do tipo B e pode ser alimentado tanto pelo cabo USB quanto por uma bateria externa com tensão elétrica entre 7 e 20V. O Arduino Uno é uma das placas mais usada da família dessa série, e usada muitas vezes em institutos educacionais por sua fácil manutenção e integração com diversos projetos eletrônicos, incluindo as máquinas CNC, por isso o uso dele será tão importante nesse trabalho (ARDUINO,2015).

O Arduino a ser usado no trabalho, possui um microcontrolador r3 MEGA328P da Atmel acoplado nele. Este é o pilar central do chip, sendo o sistema nervoso do mesmo, por receber todas as informações atribuídas a ele e interpretar de diversas formas. Podem-se utilizar dispositivos de entrada e saída como motores, LED's, botão entre outros e possibilitar a ele instruções de interação com o projeto do usuário (BOXALL, 2013). A Figura 11 apresenta a placa de Arduino a ser usado e o referido microcontrolador.

Figura 11 - Placa arduino Uno e microcontrolador ATMEL MEGA328P



Fonte: ARDUINO, 2015

A placa de microcontrolador também possui 7 conectores de alimentação, sendo eles: IOREF, RESET, 3,3V, 5V, dois GND e VIN. A entrada IOREF serve como uma referência de tensão para que o determinado circuito selecione a fonte de alimentação 32 adequada. O RESET serve para resetar o microcontrolador externamente, as entradas de 3,3V e 5V fornecem a tensão para a placa, os GNDs são pinos de terra e o VIN é a tensão de entrada da

placa quando se usa uma bateria externa. A Figura 12 mostra os conectores de alimentação da placa (SOUZA, 2013).

Figura 12 - Conectores de Alimentação da placa Arduino-Uno

Fonte:Desenvolvida pelo autor

3.1.4 Entradas

O Arduino Uno possui duas entradas para alimentação: a entrada USB (Universal Serial Bus) é a Fonte Externa. A Fonte Externa possui conector tipo Jack e sua tensão de entrada tem de estar entre 7V e 20V. Se a tensão da placa estiver abaixo de 7V, criasse uma instabilidade para o regulador de tensão, por esse motivo, a fonte de alimentação adequada para a fonte externa de acordo com a entrada. (SOUZA, 2013).

A entrada USB de tipo B serve tanto para alimentar a placa, quanto para enviar e receber dados, pois é através desta entrada que o presente trabalho irá enviar as instruções para a placa (BOXALL, 2013). A Figura 13 e 14 apresenta os dois tipos de alimentação citados acima.

Figura 13 - Entradas de energia e USB (ângulo superior)

Fonte:Elaborada pelo autor

Figura 14 - Entradas de energia e USB (ândulo inclinado)

Fonte:Elaborada pelo autor

O Arduino possui seis pinos para entradas analógicas e quatorze pinos para entradas ou saídas digitais dos quais seis podem ser também usados para entradas de valores analógicos, são eles os pinos 3, 5, 6, 9, 10 e 11 que asseguram uma saída PWM de 8 bits, variação da intensidade de um dispositivo usando-se a função analogWrite(), exemplo a intensidade de um motor girar (BANZI, 2011).

Os pinos digitais podem ser tanto para entrada quanto para saída, tendo somente importante informar na programação se será de entrada ou saída, usando as funções pinMode(), digitalWrite() ou digitalRead() (PULCINELLI, 2014).

Os pinos 0 e 1 também podem ser utilizados para comunicação porta serial (serial port) e tem a função de receber (RX) e transmitir (TX) dados para outros dispositivos e para o computador através da comunicação via USB (BOXALL, 2013).

Os pinos 10, 11, 12 e 13 podem também ser destinados à comunicação SPI (comunicação entre dois Arduino). A figura 15 apresenta os pinos analógicos e digitais.

Figura 15 - Entradas e saídas digitais(destaque vermelho) e entradas analógicas(entrada amarela)

Fonte:Elaborado pelo Autor

A placa também possui quatro LEDs, (*Light-Emitting Diodes*, em português, Diodos Emissores de Luz). Um, denominado ON, indica se há ou não corrente na placa, informando assim, se está ou não funcionando. Os LEDs TX e RX acendem quando há carregamento de dados, tanto transmitindo quanto recebendo pela porta serial USB ou dispositivos. Por último, o LED L é para uso próprio e está ligado ao pino 13, localizado nas entradas/saídas digitais (BOXALL, 2013). A Figura 16 destaca os LEDs presentes na placa Arduino Uno.

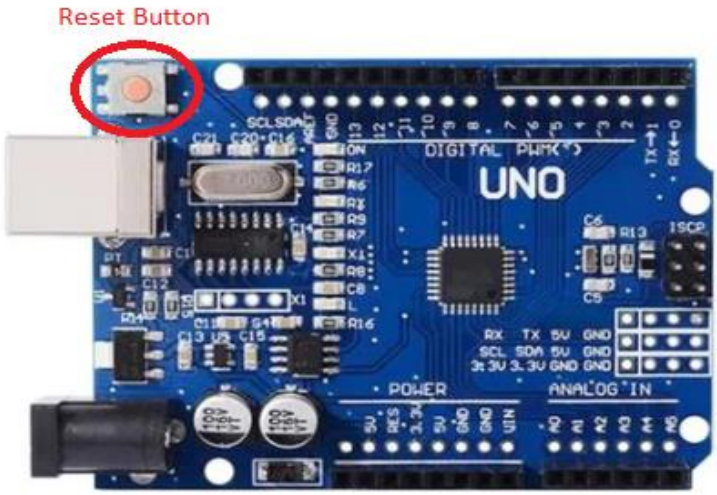
Figura 16 - LED's de terminações ligadas

Fonte: SAMPAIO,2021

O botão reset do Arduino serve para resetar o sistema, em especial quando o sistema estiver com problemas ou necessitar de uma reinicialização, ele será de suma importância, basta resetar a placa que será feito o restart dela (BOXALL, 2013). A Figura 17 apresenta o botão de reset da placa de microcontrolador.

c

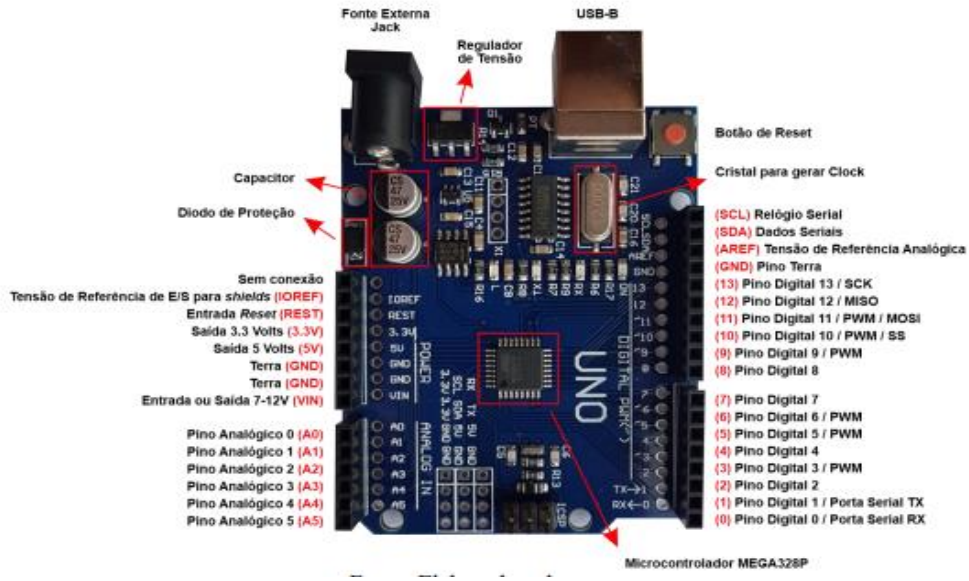
Figura 17 - Botão Reset



Fonte:Elaborada pelo autor

A Figura 18 apresenta um resumo das entradas gerais de forma específica e os componentes apresentados na placa ARDUINO UNO. Pode-se observar como geral a placa não serve apenas como um chip conector, mas um complexo controlador.

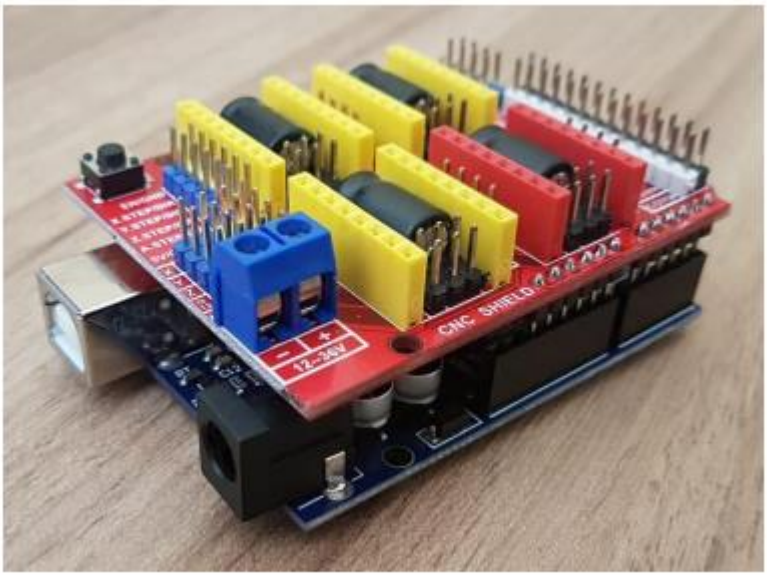
Figura 18 - Placa Uno com todos os componentes detalhados



Fonte: ARDUINO, 2015

A placa de microcontrolador Arduino Uno também permite a expansibilidade com *Shields*, adicionando assim, mais funções de hardware. A *Shield* deverá ser conectada nas pinagens do Arduino como apresentado na Figura 19. A CNC-*Shield* permite ao Arduino Uno aumentar a função para controle numérico computadorizado, o qual só foi capaz com a expansão dela e o presente trabalho se utilizará dessas duas.

Figura 19 - Placa Uno acoplada no CNC-SHIELD



Fonte:SAMPAIO,2021

3.2 CNC *SHIELD*

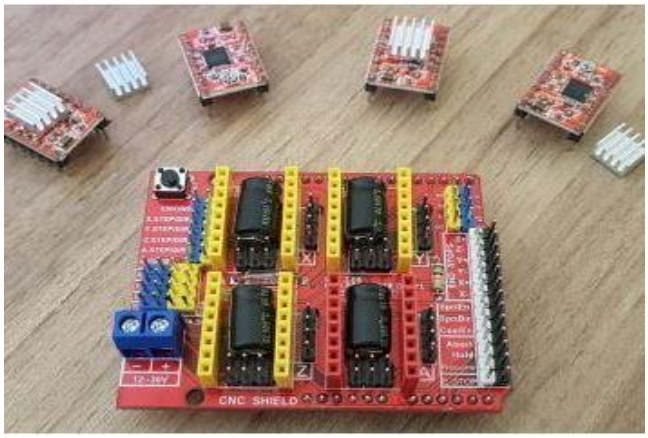
A expansão do Arduino CNC-*Shield* ampara a placa Arduino como apoio para energizar e acionar os motores de passo para realizar todas as funções necessárias na operação da CNC. É possível usar controladores ao invés de escudo, porém estes são mais baratos, portanto, mais acessíveis aos projetos menores (HANDSOME technology, 2019). Existem diferentes placas CNC existentes e neste projeto será usado o tipo open-source V3.51 que possui uma fácil manuseabilidade e compreensão o tornando mais favorável para o uso desta placa para vários projetos de Controles Numéricos Computadorizados. (DEVAN, 2020).

Para criação da mini CNC artesanal, a *Shield* fará integração com o Arduino Uno como tipo de escudo, requisitando uma fonte de alimentação entre 12 e 36V para acionamento de seus quatro motores de passo plug-in A4988. A *Shield* executa o firmware GRBL do Arduino na CNC pelo fato de o GRBL ser facilmente programado para o Arduino e esta placa

possui vários periféricos de entrada e saída para a ligação desejada e exata do equipamento (DEVAN, 2020).

O CNC-*Shield* V3 vem equipado com quatro motores de passo e quatro dissipadores de calor. Por ser um projeto de adaptação, não será preciso o uso dos dissipadores ou de um cooler para resfriamento de todo o processo. Os únicos coolers disponíveis estão relacionados diretamente à placa para dissipar o calor do processador e da Extrusora que precisa aquecer para derreter o filamento . Com suporte para quatro eixos, o CNC-*Shield* V3 também possui o eixo A para duplicar os eixos X, Y e Z caso seja necessário, porém, para este trabalho será usado somente três motores A4988 para serem plugados nos slots dos eixos X, Y e Z, três dimensões. Movimentos precisos são realizados na máquina que possui acionamentos simultâneos dos eixos X, Y e Z, os quais são configurados inicialmente através do código G carregado na placa. (HARDWARE, 2018). A figura 19 apresenta a CNC-*Shield* a ser usada no trabalho e os quatro drivers A4988.

Figura 20 - Placa CNC-SHIELD com 4 drivers A4988



Fonte: Sampaio, 2021

3.3 Componentes de Software

Esta seção descreve os componentes de software na criação do trabalho.

Enquanto o hardware computacional é a parte física do computador, o software é a parte lógica e não palpável da máquina. O Software tem por função explicitar os comandos para a matriz de controle do microcontrolador, através de um conjunto de instruções lógicas que acompanha o conjunto de hardware e é montada por um montador de software em linguagem de máquina, *assembly*.

Existem dois tipos de software, o software de sistema e o software de aplicativo. O software de sistema permite a interação do computador com o usuário por meio de interfaces

gráficas ou cadeia de comandos, eles fazem a comunicação com o computador que entendem, somente, linguagem de máquina através do seu firmware, mistura de hardware e software mais conhecido como BIOS (Basic Input/Output System, em português, Sistema Básico de Entrada e Saída) do computador, que possui as rotinas para inicialização, checagem, configuração e funcionamento básico da máquina. Os sistemas operacionais como Windows, Linux e MacOs são exemplos de tipos de softwares de sistema. Os softwares de aplicativos são programas construídos a partir do uso de sistemas compilados ou interpretados no sistema operacional. São os programas criados por programadores para executar tarefas diversas como editores de texto, plataformas de jogos, vídeos, músicas, aplicativos de redes sociais e muitos outros usados cotidianamente (RONALDO GOGONI, 2019b).

3.3.1 Arduino IDE

Arduino *Integrated Development Environment* (IDE) é um editor de Código computacional online ou um software de computador escrito na linguagem C, C++, python e Java que permite configurar comandos e classes no microcontrolador Arduino através de códigos, também conhecido como algoritmos embarcados, compilados nesse editor e geralmente salvos em formato ".ino" (ARDUINO, 2021).

Neste trabalho o micro não trabalhará com a extensão ".ino" e sim com a extensão grbl para ser possível a tradução do código G e para isso será instalado o firmware GRBL na plataforma IDE.

A IDE é uma plataforma simplificada precisando apenas conectar o microcontrolador desejado por um cabo USB no computador, selecionar o controlador no qual está usando e em seguida deve compilar o programa que automaticamente será enviado para a placa. Caso seja usado uma placa ultrapassada ou que não esteja incluída no histórico de placas do Arduino IDE, é possível baixar o modelo da placa para que o programa a reconheça na sua base de dados (ARDUINO, 2021).

3.3.2 Código G

Conforme Sousa (1998), a linguagem de programação mais usada nas máquinas-ferramenta é conhecida como o código G de programação. Este código foi desenvolvido para traduzir as informações numéricas de usinagem da peça em comandos que pudessem ser lidos e processados automaticamente pelas máquinas ferramenta. Não há um padrão rígido fechado

seguido por todos os fabricantes para os diversos comandos do código G. Existe um padrão para o código G de 54 programação de máquinas-ferramenta estabelecido pelo Instituto Alemão de Padronização desde o final da década de 1960, conhecido como DIN 66025. Este contém somente recomendações que procuram padronizar não apenas o significado dos diversos comandos como também a estrutura que se deve adotar nos programas em código G. Entretanto, os comandos de sistemas CNC de fabricantes diferentes, apesar de serem baseados no padrão DIN 66025, não seguem o mesmo padrão, aproveitando-se principalmente do fato deste permitir uma certa flexibilidade na programação, uma vez que há códigos que não receberam significados específicos.

Desta forma cada fabricante procura criar comandos que facilitem a programação em seu sistema específico. São definidas funções mais ou menos elaboradas, dependendo da sofisticação de cada sistema de controle particular. Com o objetivo de facilitar a programação, e eventuais manutenções da ferramenta, uma linguagem simbólica de programação deve ser criada com o intuito de representar da forma mais compacta possível as informações de usinagem por meio de símbolos, seguindo-se regras fixas estabelecidas, e ao mesmo tempo deve possibilitar a leitura e interpretação sem maiores dificuldades por um operador humano (SOUSA, 1998).

Quando criada, a intenção dessa linguagem era que não fosse necessária a contratação de especialistas para o trabalho de programação. Ela deveria ser facilmente aprendida e se manter simples o suficiente para que o engenheiro de planejamento, com sua preocupação voltada principalmente para o processo a ser realizado na máquina pudesse aprendê-la sem maiores dificuldades.

Atualmente, um programa em código G é gerado automaticamente por muitos programas CAD/CAM. Um programa em código G é formado por linhas de comando contendo informações de usinagem na forma de frases CNC. Essas frases (ou comandos) são compostas de palavras CNC, as quais por sua vez são compostas em geral por um código de endereçamento alfanumérico. Esse endereço indica ao computador o tipo de informação que se segue. Cada letra de endereçamento (X, Y, Z. G, M, etc) informa ao controle o tipo de palavra que se está programando, e o número informa o valor desta palavra. Por exemplo, a instrução X -0.5 sinaliza ao controle que -0.5 é um valor de coordenada que se refere ao eixo X (SOUSA, 1998). 55 As instruções são lidas, interpretadas e executadas na ordem em que aparecem na programação. Porém, mesmo executando o programa comando a comando, o sistema de controle da impressora está constantemente lendo o comando seguinte programado, a fim de posicionar corretamente a ferramenta para a execução do comando a seguir. Essa

tarefa é executada pelo look-ahead buffer (ou memória de trabalho) do controle. Essa particularidade do sistema de controle evita que haja uma parada completa da máquina a cada novo comando, assegurando uma movimentação suave da ferramenta na transição de um comando para outro. Essa parada seria extremamente prejudicial à qualidade do acabamento da peça e influencia diretamente no tempo de execução do trabalho (SOUSA, 1998).

Há ainda comandos como a compensação do raio da ferramenta nos quais esta leitura do comando seguinte é fundamental para que o controle possa calcular o ponto de finalização correto do comando atual. Quando a compensação do raio da ferramenta é utilizada em centros de torneamento, por exemplo, deve-se sempre posicionar o raio da ferramenta perpendicularmente tanto à superfície atual quanto à seguinte. Na compensação automática, o controle se encarrega de determinar o ponto correto para a finalização do comando atual a partir desta leitura do próximo comando. Assim, o ponto final do comando atual será calculado de forma diferente quando o próximo comando for um movimento circular tangente à superfície, sendo usinada e quando for um movimento linear que intercepta a direção atual de movimento, por exemplo. Uma vez dado o comando de compensação, a ferramenta será deslocada do valor correspondente no próximo movimento programado. Da mesma forma, quando a compensação é cancelada, a posição da ferramenta volta à sua posição sem compensação apenas no movimento seguinte programado.

Existem alguns comandos do código G com características especiais, eles permanecem ativos uma vez que são chamados. Comandos com essa característica são denominados modais. Desta forma, se o movimento atual da máquina for do mesmo tipo do movimento anterior (uma interpolação linear ou uma mesma velocidade de cone da ferramenta, por exemplo), não é necessário que o código correspondente seja repetido. Um comando modal só é desativado se for programado ou um comando de desativação específico ou um comando conflitante (uma interpolação linear, por exemplo, é cancelada se for programado um avanço rápido da ferramenta, para posicionamento).

Outros comandos somente permanecem ativos na 56 linha de comando em que aparecem e são, portanto, conhecidos como comandos não modais. Se houver dúvida da necessidade de uma programação com comando redundante, não há problema algum em incluí-lo na programação. Com essa atitude pode-se garantir com uma maior segurança que a máquina-ferramenta irá se comportar conforme o esperado. Caso se trate realmente de um comando redundante, o controle irá simplesmente ignorá-lo (SOUSA, 1998).

Segundo Sousa (1998), as funções miscelâneas são caracterizadas pela letra de endereçamento M e pode-se dizer que funcionam como interruptores programáveis. Elas

controlam funções auxiliares como o liga/desliga do fluido de corte, o liga/desliga da rotação da ferramenta, a troca de ferramentas, paradas opcionais, entre outras. A quadro 1 abaixo indica os principais códigos de endereçamento que são utilizados nos sistemas da impressora e seus respectivos significados. Um mesmo endereço pode ter significados diferentes, dependendo da função preparatória utilizada.

Quadro 1 - Principais comandos Código G para controle da CNC

Função	**Endereço**	**Significado**
Número do programa	N	Número do programa
Número de sequência	O	Numeração das linhas do programa
Função preparatória	G	Especifica um tipo de movimento (linear, arco, etc.)
Palavra referente à dimensão	X, Y, Z, U, V, W, A, B, C	Comando para o movimento segundo o eixo coordenado especificado
	I,J,K	Coordenada do centro do arco
	R	Raio do arco
Função velocidade de corte	F	Taxa de velocidade por minuto Taxa de velocidade por revolução
Função velocidade de rotação	S	Velocidade de rotação
Função para escolha da ferramenta	T	Número da ferramenta
Função auxiliar	M	Controle liga/desliga na máquina-ferramenta
Designação do número do programa	P	Número do subprograma
Número de repetições	P	Número de repetições do subprograma

O GCodepossui outras funções além dessas, como mover para frente ou para trás, de forma rápida ou lenta ou então para frente ou em forma de ângulo (REPLICATORG, 2011).

- G0 - Movimento rápido; 36
- G1 - Movimento coordenado;
- G2 - Arco - no sentido horário;
- G3 - Arco - no sentido anti-horário;
- G4 – Permanecer;

• G10 - Criar um sistema de coordenadas a partir do desvio absoluto;
• G17 - Selecionar plano XY (padrão);
• G18 - Selecionar plano XZ (não implementado no ReplicatorG);
• G19 - Selecionar plano YX (não implementado no ReplicatorG);
• G20 - Centímetros como unidades;
• G21 - Milímetros como unidades;
• G30 - Vai para o ponto intermediário (não implementado no ReplicatorG);
• G31 – Sonda simples (não implementado no ReplicatorG);
• G32 – Área de sonda (não implementado no ReplicatorG);
• G54-G59 – Usa o sistema de coordenadas do G10 P0-5;
• G90 - Posicionamento absoluto;
• G91 - Posicionamento relativo;
• G92 - Definir posição atual dos eixos;
• G94 - Modo de taxa de alimentação (não implementado no ReplicatorG); 37
• G97 - Taxa de velocidade de eixo;
• G161 - Negativo preciso;
• G162 - Positivo preciso;

Sousa (1998) afirma que a sintaxe de programas CNC especifica o formato que esses programas devem seguir para que os comandos existentes sejam corretamente interpretados pelo sistema de controle. A sintaxe a ser seguida por programas CNC varia entre os diferentes fabricantes dos equipamentos. Considerando que cada linha de comando é um bloco de informações, um programa para CNC consiste em um grupo de blocos que determinam uma sequência de usinagem. Cada programa é especificado por um número de programa, geralmente composto por 4 dígitos decimais, permitindo uma numeração de 0001 a 9999.

Os blocos de informações se iniciam com um número de sequência e são finalizados por um código de final de bloco.

O *Universal G-Code Sender* (UCS) é uma plataforma de código livre baseada em Java, compatível com GRBL, responsável por emitir o Gcode para o microcontrolador a fim de testar as instalações. Dados de movimentação axial são inseridos manualmente no programa e os motores movimentam conforme o desejado. Posteriormente, quando os motores estiverem testados e corretamente fixados aos eixos, o programa "*GRBL Controller*" tomará o lugar do "*Universal G-Code Sender*", pois, ele receberá os dados de coordenadas do programa CAD

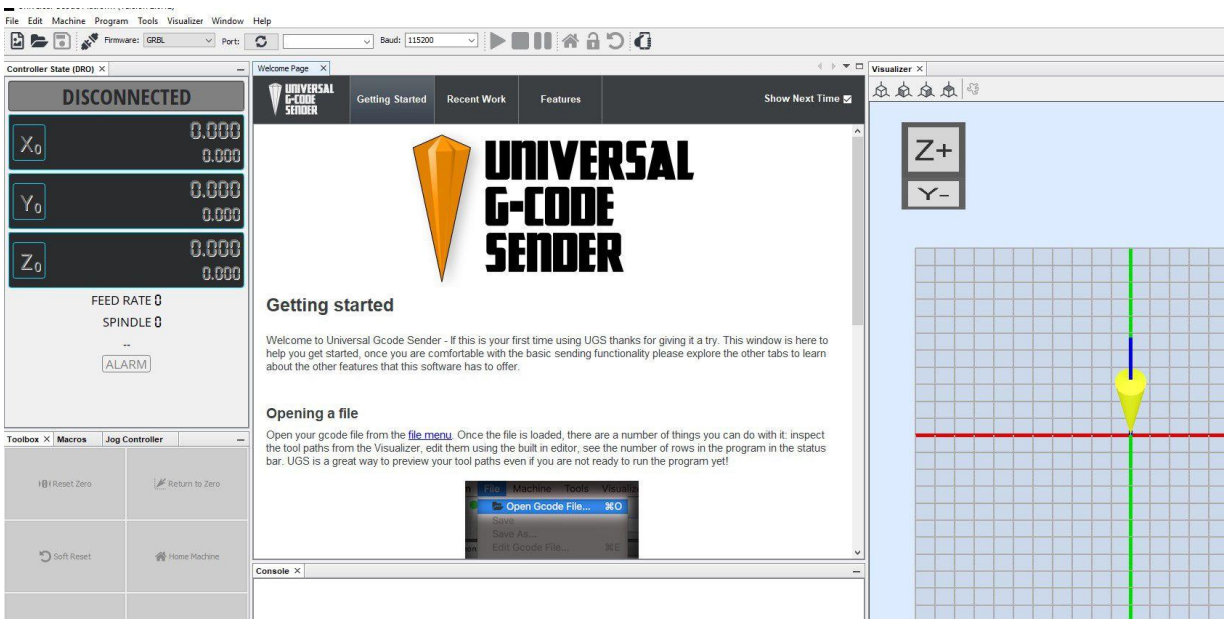
utilizado para desenvolver o layout das placas de circuito impresso e automaticamente enviará eles corretamente interpretados para o GRBL que está instalado no microcontrolador.

3.3.3 Universal G-Code Sender

O Universal Gcode Sender (UCS) é uma plataforma de código livre baseada em Java, compatível com GRBL, responsável por emitir o Gcode para o microcontrolador a fim de testar as instalações.

Dados de movimentação axial são inseridos manualmente no programa e os motores movimentam conforme o desejado. Posteriormente, quando os motores estiverem testados e corretamente fixados aos eixos, o programa "*GRBL Controller*" tomará o lugar do "*Universal G-Code Sender*", pois, ele receberá os dados de coordenadas do programa CAD utilizado para desenvolver o layout das placas de circuito impresso (EAGLE) e automaticamente enviará eles corretamente interpretados para o GRBL que está instalado no microcontrolador. A figura 21 é a interface do *Universal Code Sender*

Figura 21 - Universal code Sender



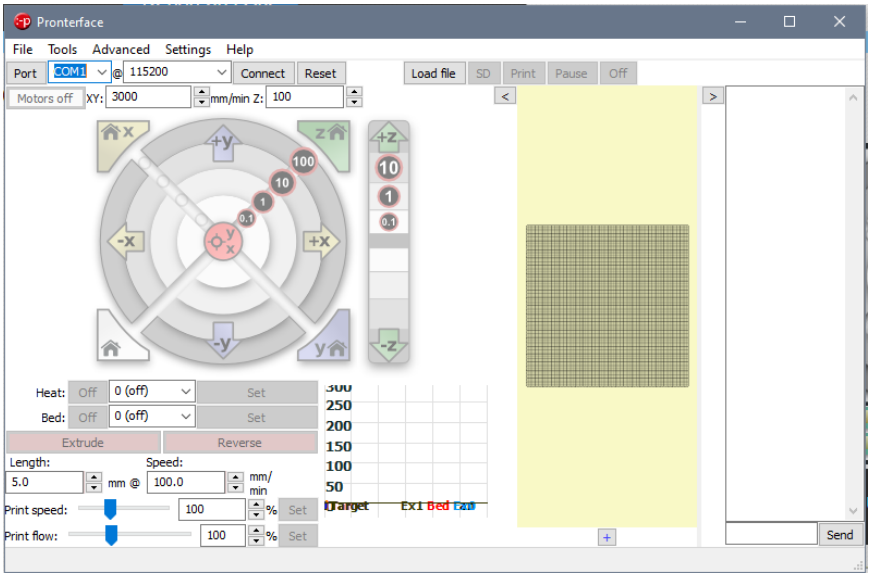
(Fonte: https://winder.github.io/ugs_website/download/)

3.3.4 PronterFace

Após isso é necessário a instalação do marlin que irá gerar o código fonte em GCODE e assim o *PrintRun* com a interface *Pronterface* que será minha interface de configuração manual da máquina.

Ele é a IDE mais popular para interfaces de uso para impressoras 3D e CNC. O mesmo utiliza *Pronsole* para comandos interativos e *Printcode* que faz a tradução do código para GCODE. Representativo da interface no *Pronterface* na Figura 22.

Figura 22 - interface Pronterface



Fonte: Print do autor, 2023

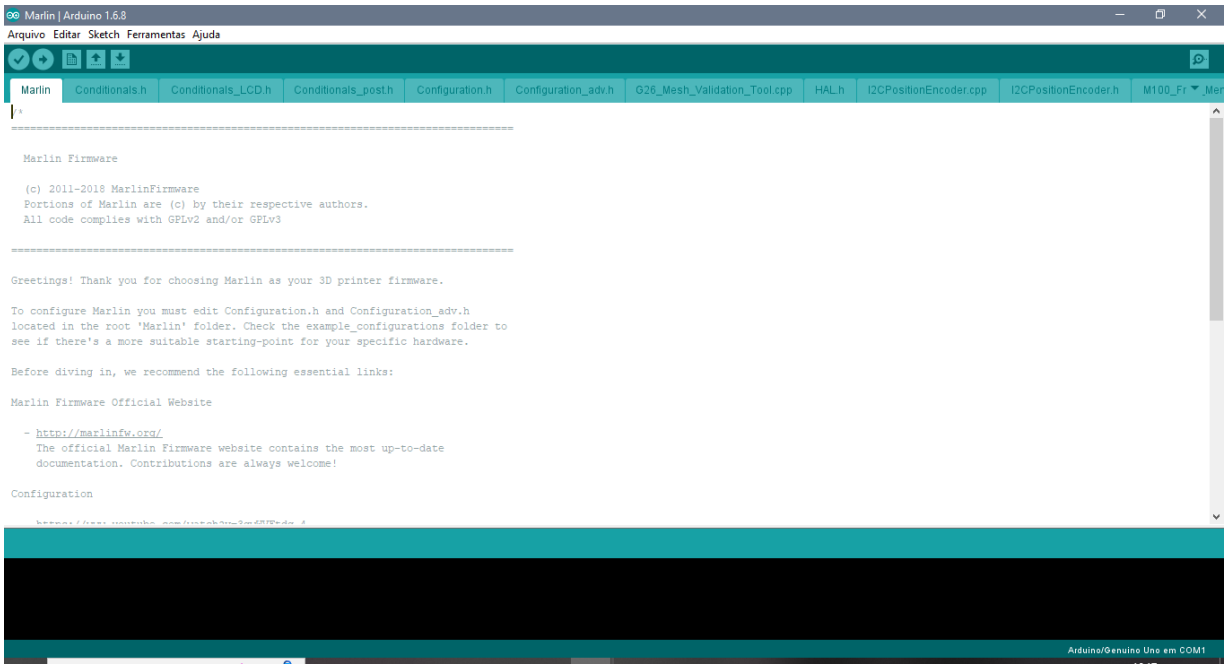
3.3.5 Marlin software

O Marlin software é uma interface Arduino gráfica de produção criada em 2011 pela RepRap e a Ultimaker hoje em dia ele é o *driver* mais presente em todas as impressoras 3D do mundo. A particularidade do Marlin é que além de poder se programar as instruções em C++ e Gcode, o Marlin tem a opção de integração com os modeladores 3d como CURA,Slic3r, etc... (Marlin, 2023).

O marlin pode ser controlado por um aplicativo *standalone* numa *Opensouce* como o *Pronterface* já descrito anteriormente. (Marlin, 2023).

A figura 23 demonstra o *firmware* Marlin em execução no Arduino.

Figura 23: Marlin em execução no Arduino



(Fonte: Marlin, 2023)

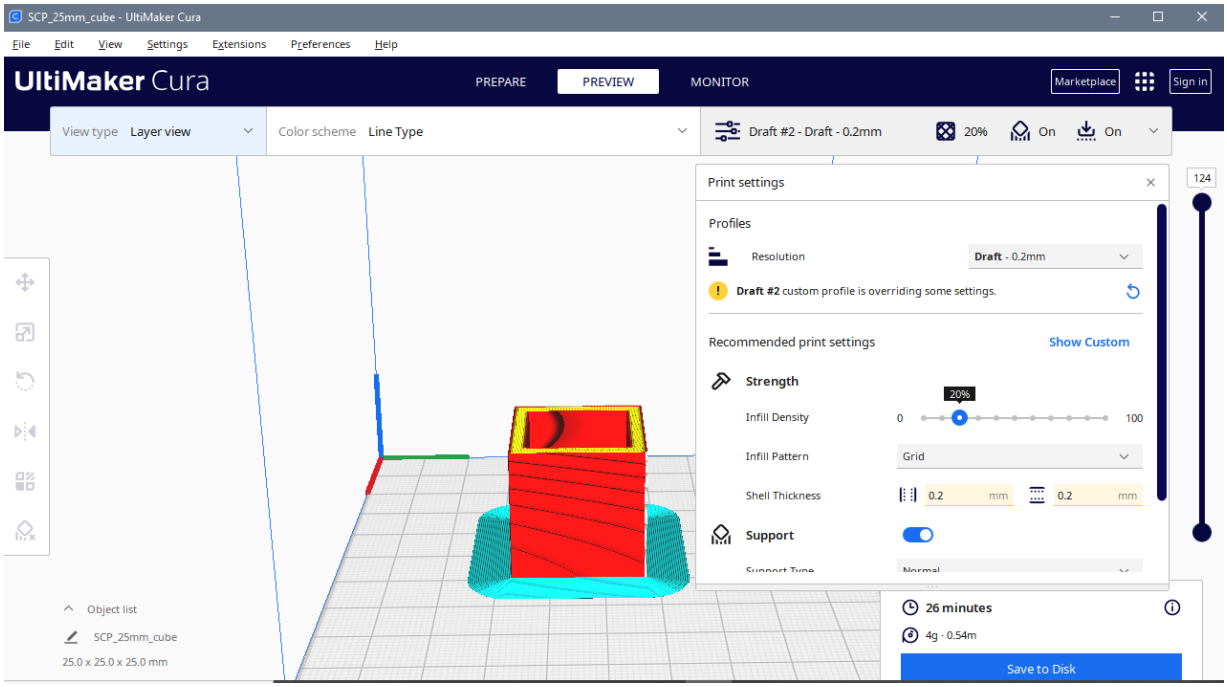
3.3.6 UltimakerCura

UltimakerCura é um aplicativo fatiador de código para modelagem para impressão 3D. Cura em si é o aplicativo fatiador, criado por David Braam, que posteriormente foi contratado pela Ultimaker, uma empresa Holandesa(ou neerlandesa de acordo com a descrição atual da ONU para países Baixos.) responsável por manufatura 3D.(Ultimaker, 2023)

O aplicativo Cura é um dos melhores, se não o melhor, fatiador que pode trabalhar com vários códigos e linguagens de programação sendo c++, Python, SML. Ele é compatível com o Marlin e o *Pronterface* para a especificação da Impressora 3D caseira.

A figura 24 refere-se ao modelo de cubo oco proposto e feito pelo graduando no Cura.

Figura 24:UltimakerCura



(Fonte: Autoria própria, 2023)

4 Descrição da impressora

Esta seção se refere à montagem da impressora passo-a-passo realizado na confecção e funcionamento dos componentes em específico da impressora 3D em geral.

A impressora segue um modelo de fresadora com Perfils de alumino de 4x370 mm de largura e Perfis de alumínio 2x600mm formando uma Mesa base com fundo emborrachado para a fresa. Com duas chapas de metal furadas corridas para fixar a *heatbed*, Fuso X, Fuso Z e Dois fusos Y, varias chapas de metal para acoplamento dos mancais e cabeçote com extrusora acoplada e parafusada para impressão, switches de estado final para parar os eixos X, Y, e Z de acordo com os limites da Fresadora.

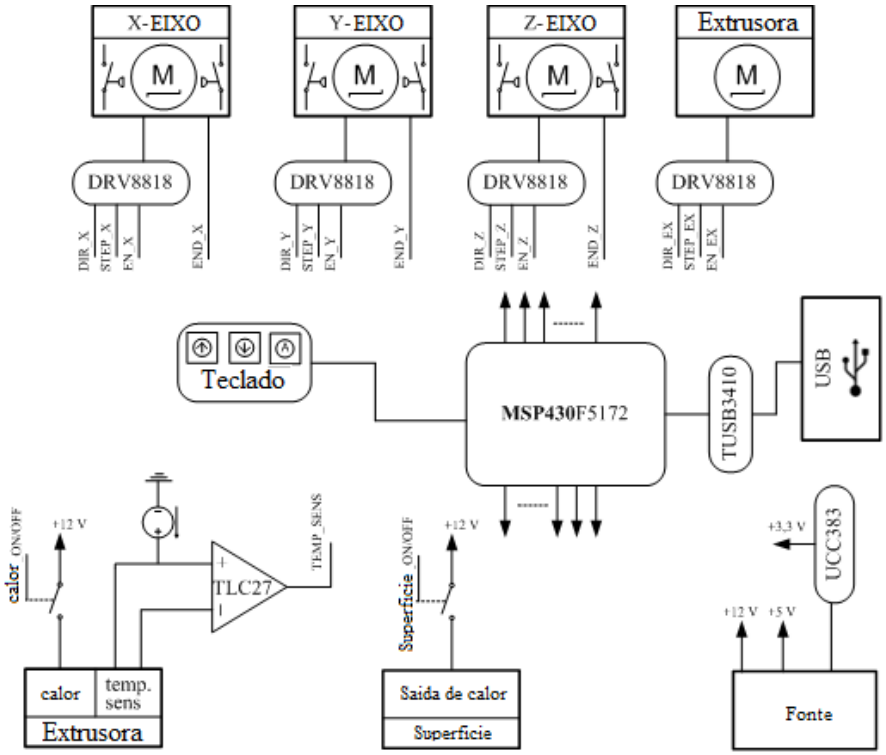
4.1 Modelagem

Ao iniciar o processo é necessário definir o método de impressão, se será modelagem em sólidos ou modelagem de superfícies, podendo ser bidimensional ou tridimensional.

Na modelagem de objetos sólidos, se preenche por dentro do objeto assim gerando menos erros na impressão, é assim que funciona na maioria dos CAD profissionais. Por sua vez, a modelagem por superfícies compõe as paredes da impressão, deixando o meio oco. A maioria dos CAD livres ou abertos usa essa metodologia.

A Figura 25 se trata de um diagrama de blocos de uma impressora 3D de forma geral como um esqueleto a ser seguido do projeto.

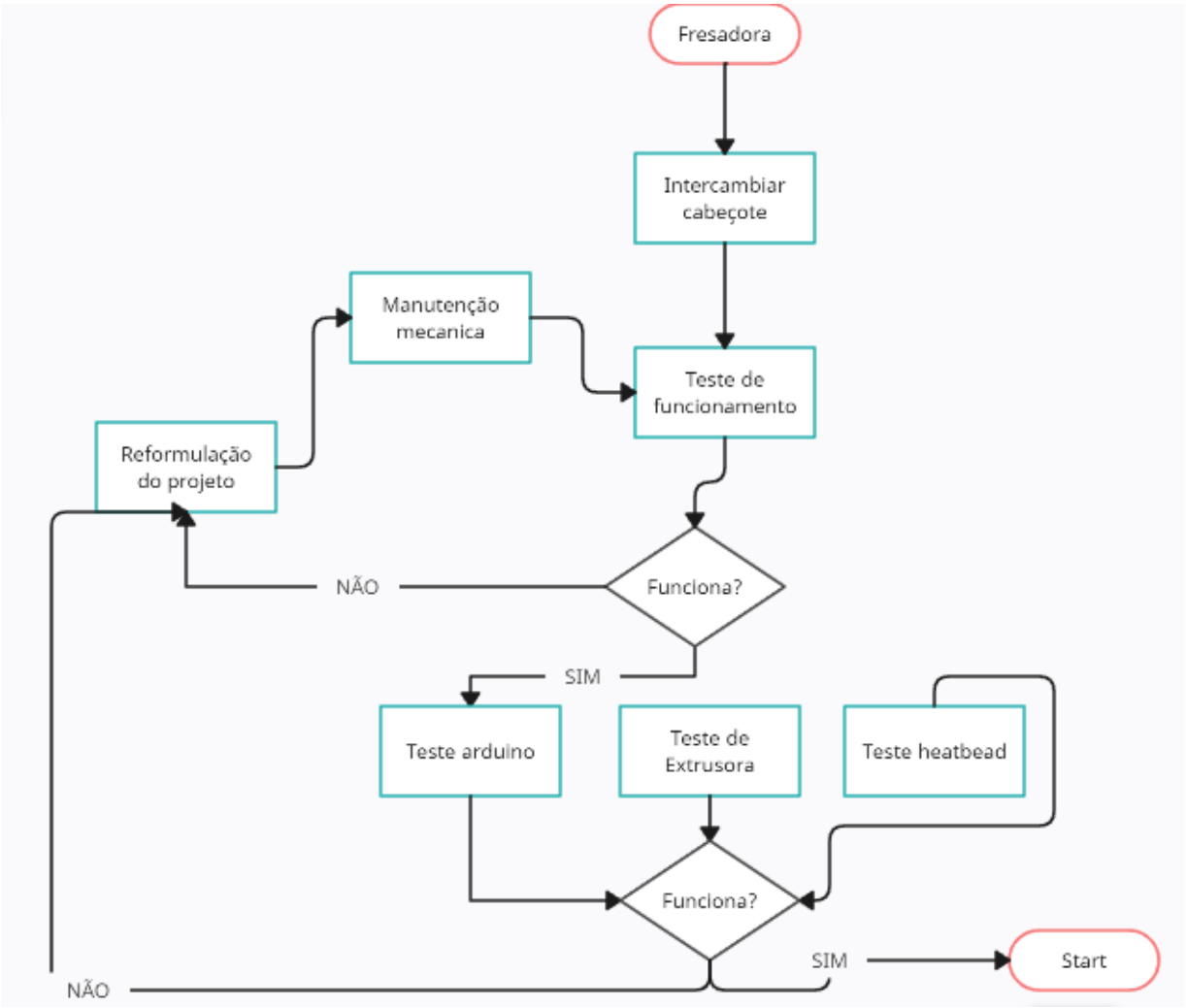
Figura 25: Diagrama de bloco da arquitetura da impressora.



Fonte: CHAKRAVORTY, 2020

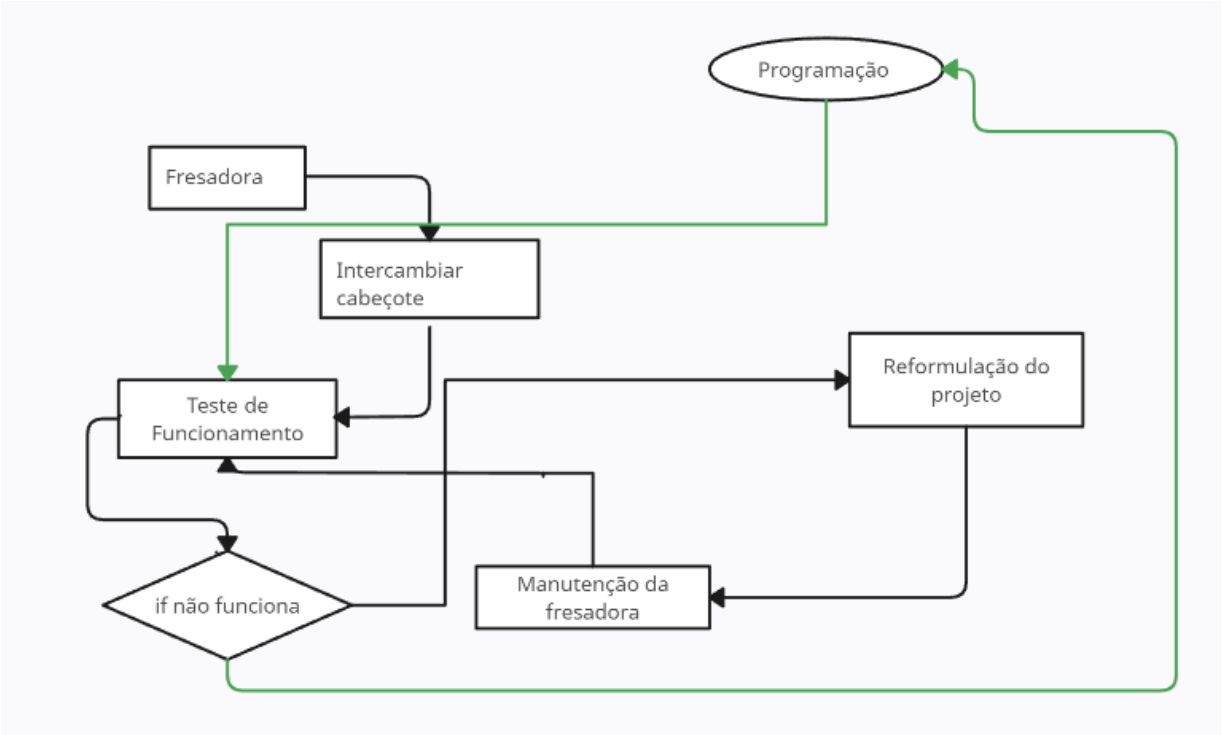
Com base na modelagem o seguimento do projeto foi feito num Fluxograma de montagem ilustrado como na Figura 26. No ideal e o objetivo era o fluxograma, porém o projeto seguiu uma forma diferente representado no Diagrama de blocos da Figura 27.

Figura 26 – Fluxograma de montagem



(Autoria Própria, 2023)

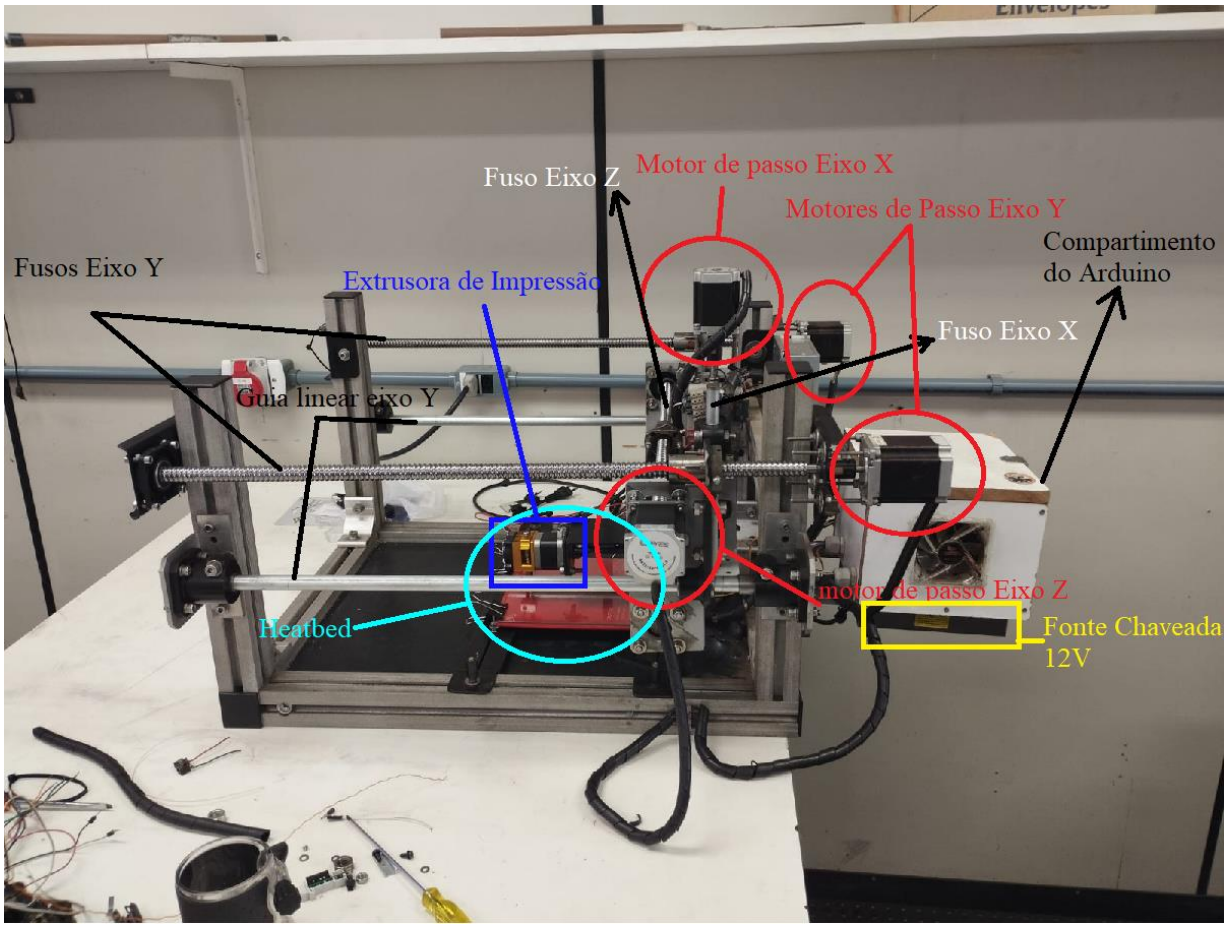
Figura 27 – Digrama de blocos de montagem e manutenção real



(Autoria Própria, 2023)

Por ventura, após a montagem da fresadora com a extrusora acoplada é apresentada na Figura 28 de forma descritiva.

Figura 28 – Figura Descritiva da Impressora

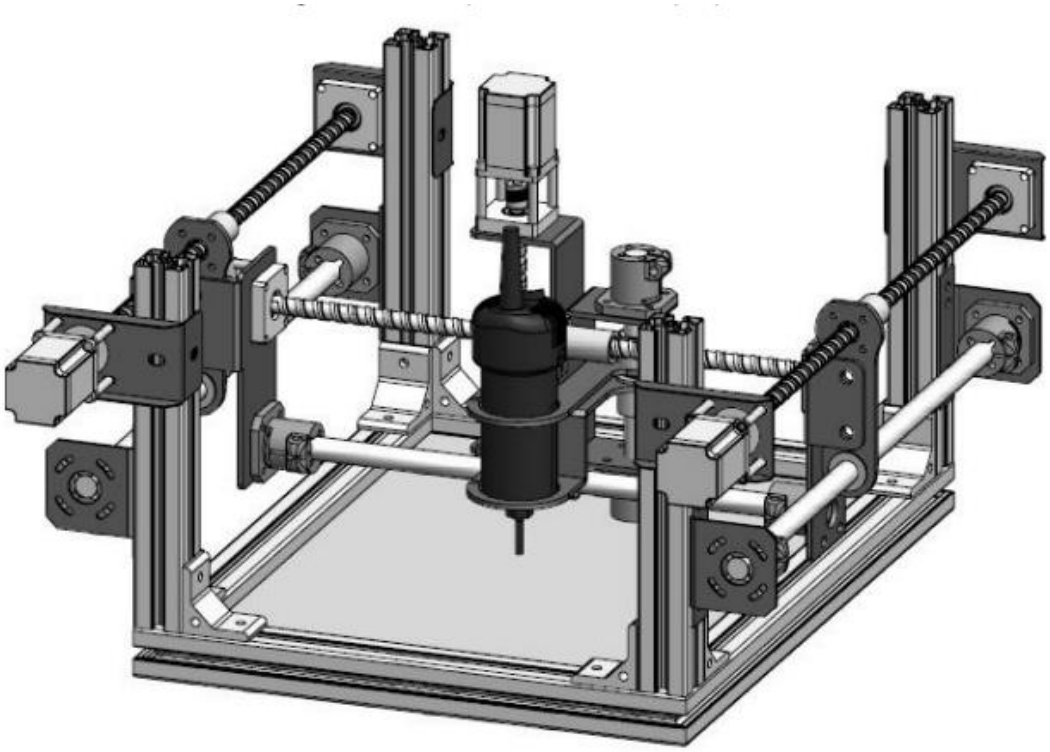


(Autoria própria, 2023)

4.2 Montagem

A montagem da Impressora 3D é a partir de uma CNC intercambiável já montada anteriormente por Rodrigues; Delmondes, 2016. A ideia é intercambiar o modelo representado na figura 29 pelo cabeçote do Motor *Spindle* para um cabeçote de extrusora de impressora 3D.

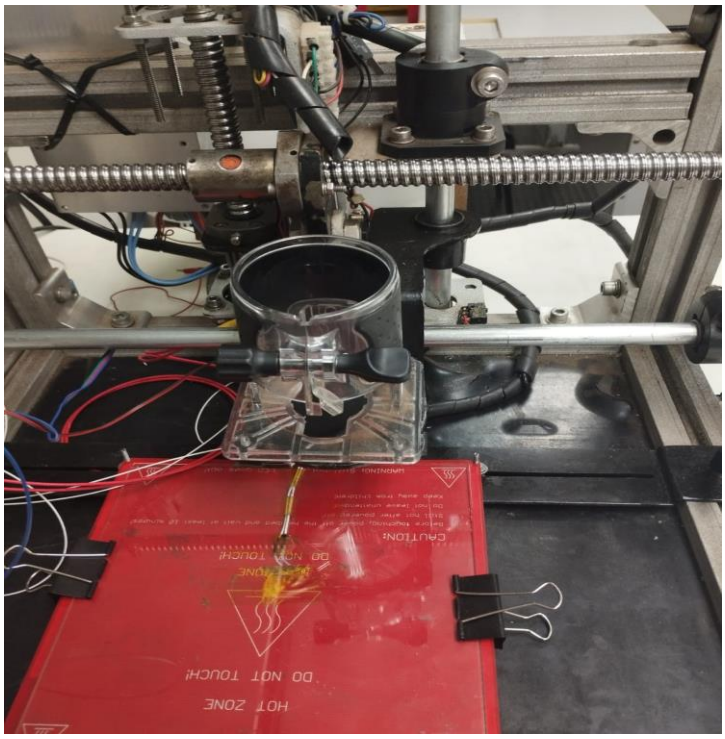
Figura 29: Máquina- ferramenta CNC intercambiável



(Fonte: Rodrigues; Delmondes, 2016)

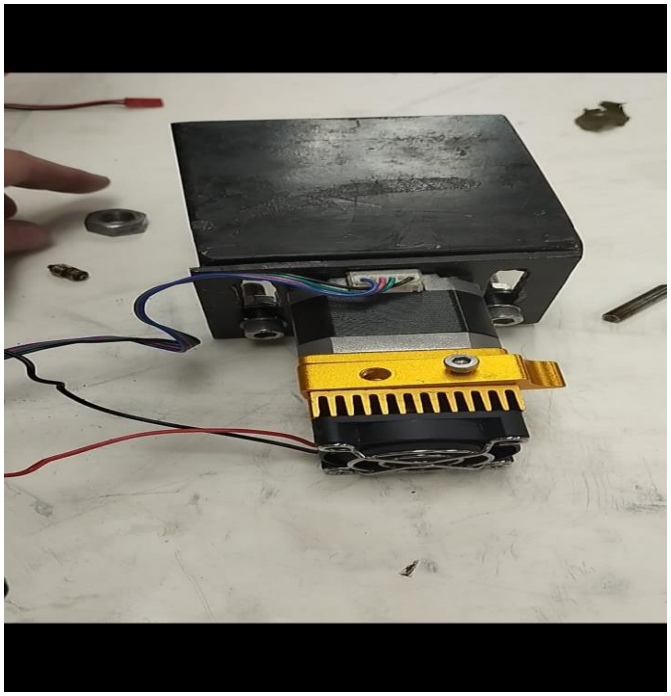
As aplicações se devem diretamente a usar os eixos já implementados dos eixos X, Y e Z utilizando os motores de passo com as guias de fuso, porém agora adaptando com o cabeçote da extrusora retirando o cabeçote da *Spindle*. A figura 30 mostra o cabeçote parafusado do Motor *Spindle*, Já a figura 31 é o cabeçote da extrusora já parafusado pronto para ser adaptado com o auxílio de imãs magnéticos superfortes.

Figura 30 - Cabeçote do Motor Spindle



(Fonte: Autoria Própria,2023)

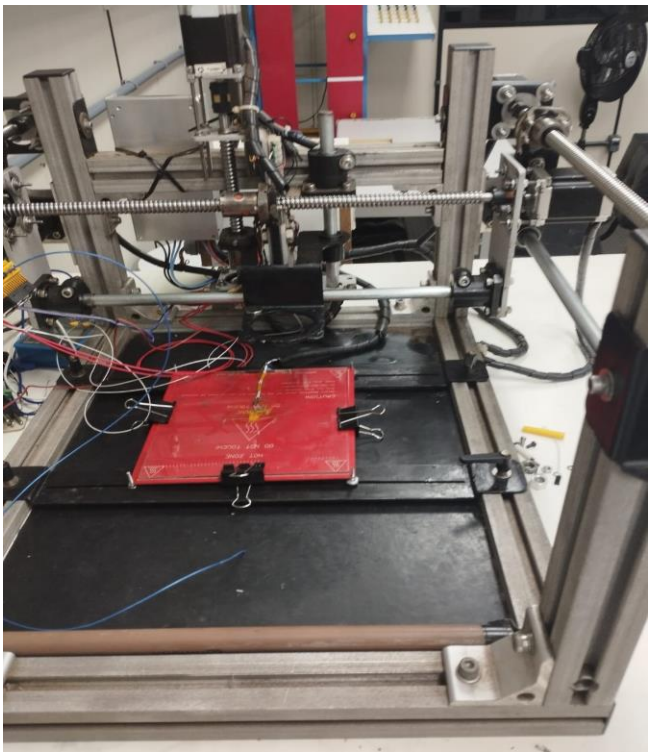
Figura 31 - Cabeçote da extrusora parafusado.



(Fonte: Autoria própria,2023)

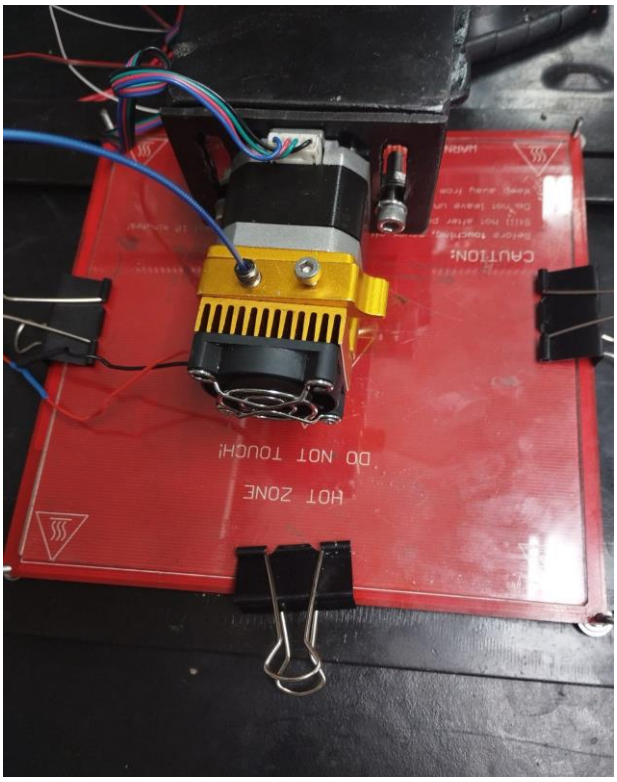
A CNC sem o cabeçote ficara da forma a seguir na figura 32 e quando aplicado teremos a figura 33.

Figura 32 - CNC sem o cabeçote do motor Spindle



(Fonte: Autoria Própria,2023)

Figura 33 - CNC com o cabeçote de impressão acoplado



(Fonte: Autoria Própria,2023)

4.2 HeatBed

A *heatbed* (cama de aquecimento de forma literal), é uma placa PCB com alta capacidade de resistência ao calor. Podendo variar a capacidade de calor de 90º a 110ºC. O valor de aquecimento dela é programado no Marlin pelo Gcode.

A *Heatbed* tem vários avisos de componente extremamente quente como pode-se perceber na Figura 31, além dos avisos de conexão e voltagem máxima de 12 volts.

A *heatbed* segue um padrão desde janeiro de 2015 com dimensões de 200x200mm, ou 400cm² 2mm de espessura. Normalmente para não ter um aquecimento direto na impressão temos uma placa de vidro de 3mm logo acima da PCB, assim com o derretimento do filamento de PVA será aplicado diretamente no vidro aquecido, após o resfriamento a placa não permite o PVA colar inteiramente no vidro, mesmo após solidificar. Então com uma ajuda de cola bastão, é possível fazer a retirada da escultura de impressão.

A figura 34 é a *heatbead* com vidro prensado já acoplado na impressora 3d

Figura 34 - Heatbed com vidro prensado

(Fonte: Autoria Própria, 2023)

5. Considerações Finais.

A partir do estudo realizado para o presente trabalho é possível concluir que o reparo pode ser realizado com sucesso, usando a base inicialmente no anterior trabalho de Delmontes, uma vez que o trabalho original era uma Fresadora 3D.

O desafio do emprego dos conhecimentos adquiridos no curso de graduação de Engenharia da Computação, é de grande importância nessa aplicação dos conceitos e práticas, além de agregar conhecimentos primordiais na área de fabricação de produtos, relacionados a áreas Engenharia: Elétrica, da Computação e Controle de automação.

Ao longo do desenvolvimento do projeto os desafios a serem desenvolvidos foram abrangentes como as soldas feitas, manutenção direta da fresadora, programação do código G e a preocupação com o funcionamento do projeto. Uma vez desenvolvida a adaptação, a troca de peças, testes com multitestes e fontes de teste. Sempre analisando tanto o funcionamento da *heatbead*, do Arduino com a CNC *Shield*, os motores de passo, alinhamento dos eixos e direcionamento correto da montagem.

Baseado na construção montagem e manutenção neste projeto, o código implantado faz a orientação especifica do direcionamento da impressão com uma imagem teste. Já o âmbito da construção é limitado em 3 dimensões nos eixos X, Y e Z com os sensores de ponto final que são *switchs* de toque posicionados nos limites do eixo principal onde está acoplado o cabeçote da extrusora. Basicamente programados com um comando *if* condicionado a toque "PARE" e *else* para "Continuar" até outro toque.

Já a extrusora tem uma nova programação depois de ser trocada apenas com o reconhecimento pelo Marlin. Mas os comandos continuam os mesmos para imprimir de acordo com as coordenadas do Código G, mas a mesma aquece e derrete o filamento, e assim com as orientações vá despejando pelo bico de extrusão para formar a peça 3D. No atual projeto não foi impresso uma imagem, apenas o teste de impressão com um pedaço de filamento saindo assim, o filamento derretido.

A *heatbead* segue a mesma ideia da extrusora, programada para esquentar enquanto a extrusora tem filamento ou a peça está sendo formada. No Marlin com a programação do Arduino Uno, pode se controlar se esquenta ou não e a temperatura da *heatbead* variando de 90 a 110ºC.

Foram realizadas a troca do cabeçote, compra de nova extrusora completa, novas instalações elétricas, troca do motor de passo, troca de porcas, parafusos, arruelas e cantoneiras. Troca de mancal e rolamento do mancal difuso. A manutenção com querosene e óleo

lubrificante. As diversas soldas das conexões elétricas baseado no circuito e planejamento elétrico.

5.1 Projetos e recomendações futuras.

Para projetos futuros uma capacidade maior ou troca do Arduino para um mais moderno e com maior capacidade de processamento não só para adicionar uma interface homem-máquina como também a abrangência para o *Slicer* ter maior liberdade de impressão.

Outra ideia que poderá ser implementada é uma terceira opção ao cabeçote intercambiável, um cabeçote de impressão para impressora 3D, cabeçotes como uma caneta para desenho numa folha fixa. Ou, porventura adaptar para uma extrusora dupla.

REFERÊNCIAS

A SIMPLE, OPEN SOURCE 3D PRINTING PROGRAM.Disponível em: <https://www.softpedia.com/get/System/System-Miscellaneous/ReplicatorG.shtml>Acesso em : 31 mar de 2023

ABOUT Arduino. **Arduino**, 2021. Disponível em: < https://www.arduino.cc/en/about >. Acesso em: 31 de mar. de 2022.

ARDUINO. **Arduino Software (IDE)**. 2015. Traduzido e disponível em: https://www.arduino.cc/en/guide/Environment#toc4. Acesso em: 09 out. 2022. .

ARDUINO. **O que é Arduino?** Traduzido e disponível em:

https://www.arduino.cc/en/guide/introduction. Acesso em: 18 out. 2022, 14:31h.

BANZI, Massimo. **Getting Started with Arduino**. 2. ed. Sebastopol: O'reilly Media, 2011.

BAIÃO, Francisco José. **Funcionalidades e tecnologias da impressão 3D.** Trabalho de conclusão de curso (TCC) de total autoria da instituição de ensino UNIVERSIDADE SÃO FRANCISCO. Itatiba, 2012. Disponível em: <http://lyceumonline.usf.edu.br/salavirtual/documentos/2347.pdf> Acesso em 31 de agosto das 2022 às 16:04

BEN-ARI, Mordechai; MONDADA, Francesco. **Elements of Robotics**. Cham: Springer Open, 2018. 308 p. Disponível em: https://link.springer.com/content/pdf/10.1007%2F978-3319-62533-1.pdf. Acesso em: 22:04 abr. 2021, 00:02h.

BOXALL, John. **ARDUINO WORKSHOP ARDUINO WORKSHOP**: a hands-on introduction with 6 5 projects. San Francisco: No Starch Press, 2013.

BUGLIA, Fernando. **Estudando os Sistemas Produtivos**. 2019. Disponível em: https://infoenem.com.br/estudando-os-sistemas-produtivos/. Acesso em: 29 abr. 2021.

CAMOTICS (org.). **CAMotics**: simulação de código aberto e usinagem auxiliada por computador. Simulação de código aberto e usinagem auxiliada por computador. Disponível em: https://camotics.org/. Acesso em: 22 jan. 2021, 11:00h.

CARVALHO, Jonas de. **Prototipagem Rápida**. Disponível em:<www.numa.org.br/conhecimentos/conhecimentos_port/pag_conhec/prototipagem.html> . Acesso em: 10 set. 2022.

CCV INDUSTRIAL (org.). **O que é CNC?** 2019. Disponível em: https://ccvindustrial.com.br/o-que-e-cnc/. Acesso em: 18 jan. 2021.

CHAKRAVORTY, Dibya. **3D Printer G-code Commands List & Tutorial**. 2020. Disponível em: https://all3dp.com/g-code-tutorial-3d-printer-gcode-commands/. Acesso em: 14 abr. 2021, 01:40h.

COFFLAND, Joseph. **Você pode construir um CNC, mas pode explicá-lo?** 2017. Disponível em: https://buildbotics.com/blog/you-can-build-a-cnc-but-can-you-explain-it/. Acesso em: 20 mar. 2021, 22:31h.

DEANS, Marti. **G-Code for CNC Programming (2020 Update)**. 2018. Disponível em: https://www.autodesk.com/products/fusion-360/blog/cnc-programming-fundamentals-gcode/. Acesso em: 14 abr. 2021, 11:31h.

EMBARCADOS (org.). **Sistema Embarcado – O que é? Qual a sua importância?** 2013. Disponível em: https://www.embarcados.com.br/sistema-embarcado/. Acesso em: 24 fev. 2021, 13:31h.

FITZPATRICK, M. **Introdução aos processos de usinagem.** Porto Alegre: AMGH, 2013. Disponível em: https://statics-submarino.b2w.io/sherlock/books/firstChapter/115145429.pdf. Acesso em 25 abr. 2021, 23:42h.

FORD, Edward. **Make: Getting Started with CNC**. San Fracisco: Maker Media, 2016. 167 p.

GARCIA, Fernando Deluno. I**NTRODUÇÃO A SISTEMAS EMBARCADOS** . 2018. Disponível em: <https://embarcados.com.br/sistemas-embarcados-e-microcontroladores/>. Acesso em: 25 out. 2022 02:20h.

GONÇALVES, Raquel. **IMPRESSORAS. 2020.** Disponível em :< https://www.timetoast.com/timelines/impressoras-76521303-94da-4a9b-a320-78d020a82bd8>; Acesso em 14 set de 2022

HARDWARE, Maker (org). **CNC *Shield* Guide**. Perth: Maker Group Global, 2018, 12 p. Disponível em: https://www.makerstore.com.au/download/publications/CNC-Shield-Guidev1.0.pdf. Acesso em: 12 Agosto 2022, 11:23h.

HISTÓRIA DE TUDO. **História da Impressora**. Copyright © História de Tudo, 2020. Disponível em: <https://www.historiadetudo.com/impressora> Acesso em: 31 de agosto de 2022, 15:46h.

WIKIPEDIA HEATBED. Disponível em : <https://reprap.org/wiki/PCB_Heatbed> Acesso em: 25 de maio de 2023, 21:11h.

INKSCAPE (org.). **Visão geral do Inkscape**. 2020. Disponível em: https://inkscape.org/ptbr/sobre/. Acesso em: 22 agost. 2022, 13:30h.

KERSCHBAUMER, Ricardo. **Engenharia de controle e automação - Microcontroladores.** Disciplina de Microcontroladores ministrada pelo Prof, Ricardo Kerschbaumer do Instituto Federal de Educação, Ciência e Tecnologia Catarinense Capus Luzerna, do ano de 2018. Todo e total uso do Instituto Federal de Educação, Ciência e Tecnologia Catarinense Capus Luzerna. Disponível em: https://professor.luzerna.ifc.edu.br/ricardo-kerschbaumer/wp-content/uploads/sites/43/2018/02/Apostila-Microcontroladores.pdf . acesso em: 16 out de 2022.

LEE, Jan. **Pioneiros da Computação**: john t. parsons. John T. Parsons. 2021. Disponível em: https://history.computer.org/pioneers/parsons.html. Acesso em: 20 mar. 2021, 22:40h.

LIMA, Thiago. **AGC – O primeiro grande Sistema Embarcado**. 2014. Disponível em: https://www.embarcados.com.br/agc-primeiro-grande-sistema-embarcado/. Acesso em: 25 set. 2022.

MALHEIROS, Marcos; VARGAS, Fernanda. **This is Book Three**. São Paulo: Minds, 2019. 108 p.

MARCICANO, João Paulo P. **Introdução ao Controle Numérico**. 2020. Disponível em: http://sites.poli.usp.br/d/pmr2202/arquivos/aulas/cnc.pdf. Acesso em: 08 abr. 2021, 01:10h.

MECÂNICA INDUSTRIAL (org.). **O que é máquina de controle numérico. 2017.** Disponível em:https://www.mecanicaindustrial.com.br/o-que-e-maquina-de-controlenumerico/. Acesso em 14 set de 2022.

METZ, Devan. **Arduino CNC *Shield*: Guia do comprador**: escudos arduino cnc. ESCUDOS

MESA, Claudio. **Impressora 3D caseira. 2013.** Disponível em: https://labdegaragem.com/forum/topics/impressora-3d-caseira. Acesso em 14 set 2022.

ARDUINO CNC. 2020. Disponível em: https://all3dp.com/2/arduino-cnc-shield-buyersguide/. Acesso em: 19 jan. 2021, 13:58h.

PANKIEWICZ, Igor. **Como Funciona a Impressora 3D?** . 29 jul. 2009. Disponível em :https://www.tecmundo.com.br/impressora/2501-como-funciona-a-impressora-3d-.htm . Acesso em: 7 set. 2022.

Porto Editora – **impressora (informática) na Infopédia.** Porto: Porto Editora. [consult. 2022-09-14]. Disponível em: https://www.infopedia.pt/$impressora-(informatica).

POZZEBOM, Rafaela. **O que são sistemas embarcados?** 2014. Disponível em: https://www.oficinadanet.com.br/post/13538-o-que-sao-sistemas-embarcados. Acesso em: 24 set. 2022, 13:40.

PULCINELLI, Márcio. **Arduino pra que te quero**. 2014. Disponível em: https://www.tiespecialistas.com.br/arduino-pra-que-te-quero/. Acesso em: 07 maio 2021, 20:42h.

REGISTER, Thomas (org.). **How CAD/CAM Programs Work**. 2021. Disponível em: https://www.thomasnet.com/articles/custom-manufacturing-fabricating/cad-cam-softwareexplanation/. Acesso em: 15 abr. 2021, 11:32h.

RODRIGUES, Lucas; DELMONDES, Rômulo. **CNC INTERCAMBIÁVEL APLICADA A CONFECÇÃO DE PLACAS DE CIRCUITO IMPRESSO**. 2016. 129 f. TCC (Graduação) - Curso de Engenharia Elétrica, Pontifícia Universidade Católica de Goiás, Goiânia, 2016.

SAMPAIO, Giovana de Souza. **CONSTRUÇÃO DE UMA MINI PLOTTER ARTESANAL PARA DESENHO.** Trabalho de Conclusão de Curso com total direitos da Escola Politécnica da Pontifícia Universidade Católica de Goiás. 2021. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/bitstream/123456789/2850/1/TCC%202%20-%20GIOVANNA%20DE%20SOUSA%20SAMPAIO%20-%20CONSTRU%c3%87%c3%83O%20DE%20UMA%20MINI%20PLOTTER%20ARTESANAL%20PARA%20DESENHO.pdf> Acesso em 5 out. de 2022

SMARTEC (org.). **Uma breve história da automação industrial**. 2018. Disponível em: https://www.smartec-automacao.com.br/blog/15532-2/. Acesso em: 05 abr. 2021, 21:05.

SMID, Peter. **CNC Programming Handbook**: a comprehensive guide to practical cnc programming. 2. ed. New York: Industrial Press, 2003. 529 p. Disponível em: https://3lib.net/book/1074203/fa98be?dsource=recommend. Acesso em: 13 maio 2021, 17:14h.

SOUZA, Fábio. **Arduino UNO**. 2013. Disponível em:

https://www.embarcados.com.br/arduino-uno/. Acesso em: 03 maio 2021.

SOUSA, Flávia Maria Guerra de. **Controle de Fresadora para a Prototipagem de Circuitos Impressos**. Campinas – SP: Universidade Estadual de Campinas – Departamento de Engenharia de Computação e Automação Industrial, 1998.

TECNOLOGIA, Cnc (org.). **Tecnologia CNC**: o que e cnc?. O que e CNC?. 2014. Disponível em: http://cnctecnologia.no.comunidades.net/index.php. Acesso em: 15 abr. 2021, 01:15h.

TECHNOLOGY, Handson (org**). 3-Axis CNC/Stepper Motor *Shield* for Arduino.** Colatina: UNESC, 2019,12 p. Disponível em: https://www.passeidireto.com/arquivo/63396081/cnc-shield-v-3. Acesso em: 13 maio 2021, 00:36h.

UnionTech. **Industrial 3D printer-Product.** 2022. Disponível em: https://www.uniontech3d.com/product/detail/2877. Acesso em 28 de set de 2022.

WARFIELD, Bob. **Programação CNC com código G: tutorial gratuito definitivo [2020]**. 2020. Disponível em: https://www.cnccookbook.com/cnc-programming-g-code/#chapter1. Acesso em: 15 abr. 2021, 17:30h.

WOLF, Marilyn. **Computers as Components**: principles of embedded computing system design. 3. ed. Waltham: Elsevier, 2012. 508 p.

Anexo 1-Termo de Publicação

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR
Av. Universitária, 1069 • Setor Universitário
Caixa Postal 86 • CEP 74605-010
Goiânia • Goiás • Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br • reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante Frederico César Perillo da Veiga Jardim do Curso de Engenharia de Computação, matrícula 2013200330149-8, telefone: (62)982468386 e-mail fredpvj@hotmail.com, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do Autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado MANUTENÇÃO E MONTAGEM DE IMPRESSORA 3D DE CONSTRUÇÃO DE UMA FRESADORA, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto(PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 28 de fevereiro de 2023.

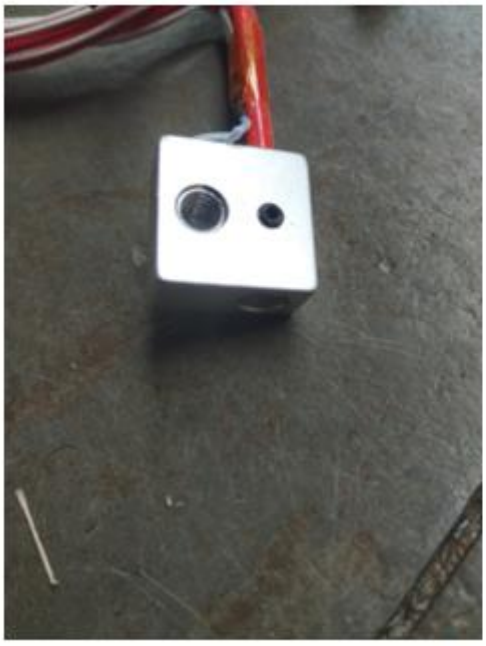
Assinatura do autor: _____

Nome completo do autor: Frederico César Perillo da Veiga Jardim

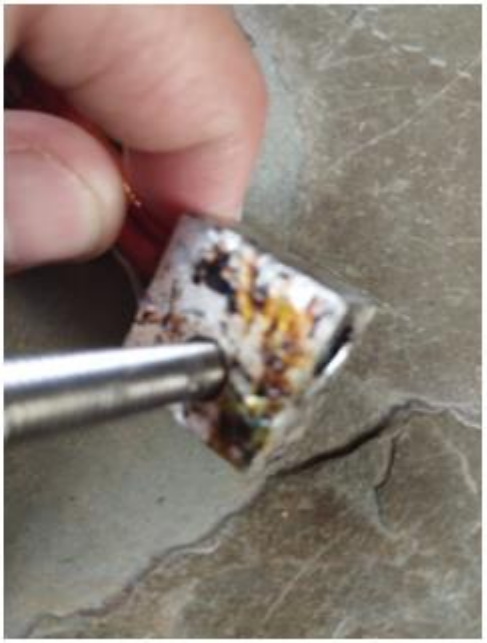
Assinatura do professor–orientador: _____

Nome completo do professor-orientador: Marcelo Antonio Gabriel de Carvalho

Anexo 2- Equipamentos comprados, trocados, soldas feitas, substituições e equipamentos com defeito.



Aquecedor de filamento novo.



Aquecedor de filamento queimado

Bico de evasão do filamento NOVO

Bico de filamento completamente entupido

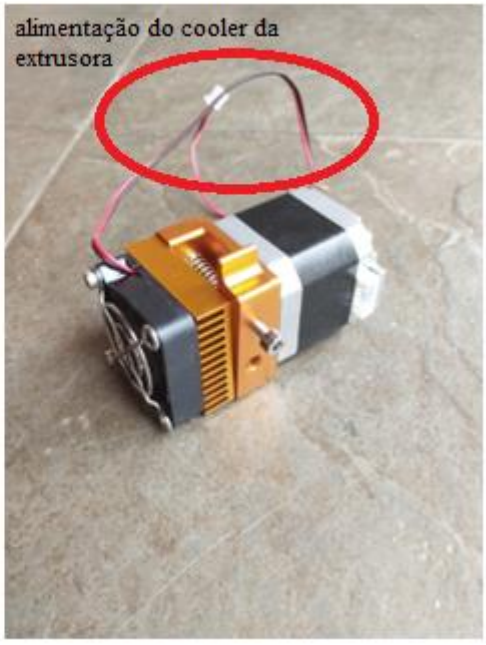
Tubo guia de filamento novo com porca

Tubo de filamento completamente destruído

Novo equipamento com os fios novos

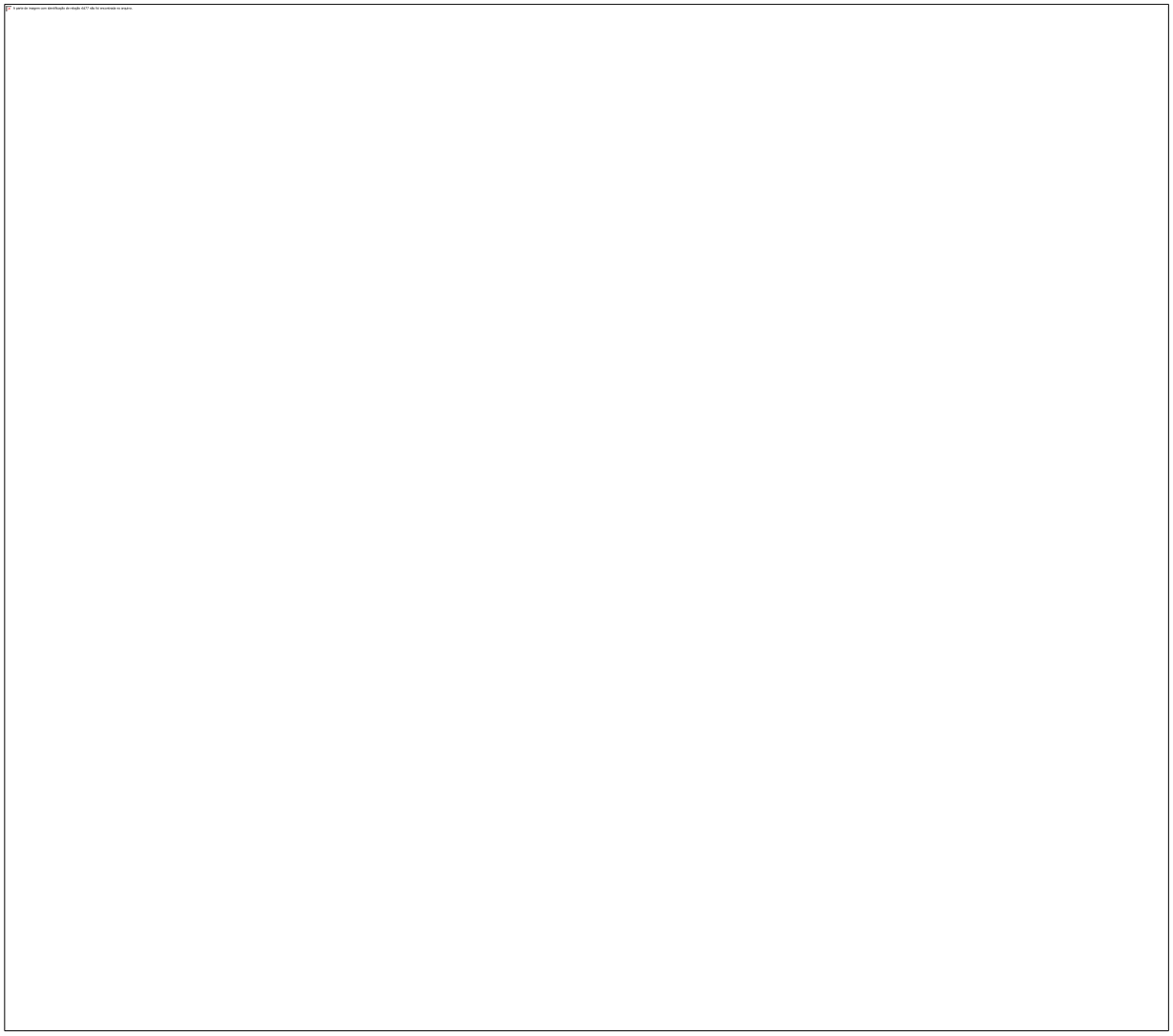
Fios de alimentação completamente corroídos

Cabos de conexão da extrusora



Extrusora nova e cabos de alimentação do cooler

Porcas e arruelas



Soldas feitas com espaguete isolante de solda termo retrátil

Anexo 3- Código do Marlin.

```
#include "MarlinConfig.h"

#if ENABLED(G26_MESH_VALIDATION)

  #include "Marlin.h"
  #include "planner.h"
  #include "stepper.h"
  #include "temperature.h"
  #include "ultralcd.h"
  #include "parser.h"
  #include "serial.h"
  #include "bitmap_flags.h"

  #if ENABLED(MESH_BED_LEVELING)
    #include "mesh_bed_leveling.h"
  #elif ENABLED(AUTO_BED_LEVELING_UBL)
    #include "ubl.h"
  #endif

  #define EXTRUSION_MULTIPLIER 1.0
  #define RETRACTION_MULTIPLIER 1.0
  #define PRIME_LENGTH 10.0
  #define OOZE_AMOUNT 0.3

  #define INTERSECTION_CIRCLE_RADIUS 5
  #define CROSSHAIRS_SIZE 3

  #if CROSSHAIRS_SIZE >= INTERSECTION_CIRCLE_RADIUS
    #error "CROSSHAIRS_SIZE must be less than INTERSECTION_CIRCLE_RADIUS."
  #endif

  #define G26_OK false
  #define G26_ERR true

  /**
   *  G26 Mesh Validation Tool
   *
```

```
 * G26 is a Mesh Validation Tool intended to provide support for the Marlin Unified Bed Leveling System.
 * In order to fully utilize and benefit from the Marlin Unified Bed Leveling System an accurate Mesh must
 * be defined. G29 is designed to allow the user to quickly validate the correctness of her Mesh. It will
 * first heat the bed and nozzle. It will then print lines and circles along the Mesh Cell boundaries and
 * the intersections of those lines (respectively).
 *
 * This action allows the user to immediately see where the Mesh is properly defined and where it needs to
 * be edited. The command will generate the Mesh lines closest to the nozzle's starting position. Alternatively
 * the user can specify the X and Y position of interest with command parameters. This allows the user to
 * focus on a particular area of the Mesh where attention is needed.
 *
 * B #  Bed        Set the Bed Temperature. If not specified, a default of 60 C. will be assumed.
 *
 * C    Current    When searching for Mesh Intersection points to draw, use the current nozzle location
 *                 as the base for any distance comparison.
 *
 * D    Disable    Disable the Unified Bed Leveling System. In the normal case the user is invoking this
 *                 command to see how well a Mesh as been adjusted to match a print surface. In order to do
 *                 this the Unified Bed Leveling System is turned on by the G26 command. The D parameter
 *                 alters the command's normal behaviour and disables the Unified Bed Leveling System even if
 *                 it is on.
 *
 * H #  Hotend     Set the Nozzle Temperature. If not specified, a default of 205 C. will be assumed.
 *
 * F #  Filament   Used to specify the diameter of the filament being used. If not specified
 *                 1.75mm filament is assumed. If you are not getting acceptable results by using the
 *                 'correct' numbers, you can scale this number up or down a little bit to change the amount
```

```
 *                of filament that is being extruded during the printing of the various lines on the bed.
 *
 *  K    Keep-On     Keep the heaters turned on at the end of the command.
 *
 *  L #  Layer       Layer height. (Height of nozzle above bed)  If not specified .20mm will be used.
 *
 *  O #  Ooooze      How much your nozzle will Ooooze filament while getting in position to print.
This
 *                is over kill, but using this parameter will let you get the very first 'circle' perfect
 *                so you have a trophy to peel off of the bed and hang up to show how perfectly you have
your
 *                Mesh calibrated. If not specified, a filament length of .3mm is assumed.
 *
 *  P #  Prime       Prime the nozzle with specified length of filament. If this parameter is not
 *                given, no prime action will take place. If the parameter specifies an amount, that much
 *                will be purged before continuing. If no amount is specified the command will start
 *                purging filament until the user provides an LCD Click and then it will continue with
 *                printing the Mesh. You can carefully remove the spent filament with a needle nose
 *                pliers while holding the LCD Click wheel in a depressed state. If you do not have
 *                an LCD, you must specify a value if you use P.
 *
 *  Q #  Multiplier  Retraction Multiplier. Normally not needed. Retraction defaults to 1.0mm and
 *                un-retraction is at 1.2mm   These numbers will be scaled by the specified amount
 *
 *  R #  Repeat      Prints the number of patterns given as a parameter, starting at the current location.
 *                If a parameter isn't given, every point will be printed unless G26 is interrupted.
 *                This works the same way that the UBL G29 P4 R parameter works.
 *
 *                NOTE:  If you do not have an LCD, you -must- specify R. This is to ensure that you
are
 *                aware that there's some risk associated with printing without the ability to abort in
 *                cases where mesh point Z value may be inaccurate. As above, if you do not include a
 *                parameter, every point will be printed.
 *
 *  S #  Nozzle      Used to control the size of nozzle diameter. If not specified, a .4mm nozzle is
assumed.
 *
 *  U #  Random      Randomize the order that the circles are drawn on the bed. The search for the
closest
 *                undrawn cicle is still done. But the distance to the location for each circle has a
```

```
 *                random number of the size specified added to it. Specifying S50 will give an interesting
 *                deviation from the normal behaviour on a 10 x 10 Mesh.
 *
 *   X #  X Coord.    Specify the starting location of the drawing activity.
 *
 *   Y #  Y Coord.    Specify the starting location of the drawing activity.
 */

// External references

extern Planner planner;

// Private functions

static uint16_t circle_flags[16], horizontal_mesh_line_flags[16], vertical_mesh_line_flags[16];
float g26_e_axis_feedrate = 0.025,
      random_deviation = 0.0;

static bool g26_retracted = false; // Track the retracted state of the nozzle so mismatched
                                   // retracts/recovers won't result in a bad state.

static float g26_extrusion_multiplier,
             g26_retraction_multiplier,
             g26_layer_height,
             g26_prime_length,
             g26_x_pos, g26_y_pos;

static int16_t g26_bed_temp,
               g26_hotend_temp;

static int8_t g26_prime_flag;

#if ENABLED(ULTIPANEL)

  /**
   * If the LCD is clicked, cancel, wait for release, return true
   */
  bool user_canceled() {
    if (!is_lcd_clicked()) return false; // Return if the button isn't pressed
    lcd_setstatusPGM(PSTR("Mesh Validation Stopped."), 99);
```

```
    #if ENABLED(ULTIPANEL)
      lcd_quick_feedback(true);
    #endif
    wait_for_release();
    return true;
   }

   bool exit_from_g26() {
    lcd_setstatusPGM(PSTR("Leaving G26"), -1);
    wait_for_release();
    return G26_ERR;
   }

  #endif

  void G26_line_to_destination(const float &feed_rate) {
   const float save_feedrate = feedrate_mm_s;
   feedrate_mm_s = feed_rate;
   prepare_move_to_destination();  // will ultimately call ubl.line_to_destination_cartesian or
ubl.prepare_linear_move_to for UBL_SEGMENTED
   feedrate_mm_s = save_feedrate;
  }

  void move_to(const float &rx, const float &ry, const float &z, const float &e_delta) {
   float feed_value;
   static float last_z = -999.99;

   bool has_xy_component = (rx != current_position[X_AXIS] || ry != current_position[Y_AXIS]); //
Check if X or Y is involved in the movement.

   if (z != last_z) {
    last_z = z;
    feed_value = planner.max_feedrate_mm_s[Z_AXIS]/(3.0);  // Base the feed rate off of the
configured Z_AXIS feed rate

    destination[X_AXIS] = current_position[X_AXIS];
    destination[Y_AXIS] = current_position[Y_AXIS];
    destination[Z_AXIS] = z;                    // We know the last_z==z or we wouldn't be in this block
of code.
    destination[E_CART] = current_position[E_CART];
```

```
    G26_line_to_destination(feed_value);
    set_destination_from_current();
   }

   // Check if X or Y is involved in the movement.
   // Yes: a 'normal' movement. No: a retract() or recover()
   feed_value = has_xy_component ? PLANNER_XY_FEEDRATE() / 10.0 :
planner.max_feedrate_mm_s[E_AXIS] / 1.5;

   if (g26_debug_flag) SERIAL_ECHOLNPAIR("in move_to() feed_value for XY:", feed_value);

   destination[X_AXIS] = rx;
   destination[Y_AXIS] = ry;
   destination[E_CART] += e_delta;

   G26_line_to_destination(feed_value);
   set_destination_from_current();
  }

  FORCE_INLINE void move_to(const float where[XYZE], const float &de) {
move_to(where[X_AXIS], where[Y_AXIS], where[Z_AXIS], de); }

  void retract_filament(const float where[XYZE]) {
   if (!g26_retracted) { // Only retract if we are not already retracted!
    g26_retracted = true;
    move_to(where, -1.0 * g26_retraction_multiplier);
   }
  }

  void recover_filament(const float where[XYZE]) {
   if (g26_retracted) { // Only un-retract if we are retracted.
    move_to(where, 1.2 * g26_retraction_multiplier);
    g26_retracted = false;
   }
  }

  /**
   * Prime the nozzle if needed. Return true on error.
   */
```

```
inline bool prime_nozzle() {

 #if ENABLED(ULTIPANEL)
  float Total_Prime = 0.0;

  if (g26_prime_flag == -1) {  // The user wants to control how much filament gets purged

    lcd_external_control = true;
    lcd_setstatusPGM(PSTR("User-Controlled Prime"), 99);
    lcd_chirp();

    set_destination_from_current();

    recover_filament(destination); // Make sure G26 doesn't think the filament is retracted().

    while (!is_lcd_clicked()) {
      lcd_chirp();
      destination[E_CART] += 0.25;
      #ifdef PREVENT_LENGTHY_EXTRUDE
        Total_Prime += 0.25;
        if (Total_Prime >= EXTRUDE_MAXLENGTH) return G26_ERR;
      #endif
      G26_line_to_destination(planner.max_feedrate_mm_s[E_AXIS] / 15.0);
      set_destination_from_current();
      planner.synchronize();    // Without this synchronize, the purge is more consistent,
                                // but because the planner has a buffer, we won't be able
                                // to stop as quickly. So we put up with the less smooth
                                // action to give the user a more responsive 'Stop'.

      SERIAL_FLUSH(); // Prevent host M105 buffer overrun.
    }

    wait_for_release();

    lcd_setstatusPGM(PSTR("Done Priming"), 99);
    lcd_quick_feedback(true);
    lcd_external_control = false;
  }
  else
 #endif
```

```
 {
  #if ENABLED(ULTRA_LCD)
   lcd_setstatusPGM(PSTR("Fixed Length Prime."), 99);
   lcd_quick_feedback(true);
  #endif
  set_destination_from_current();
  destination[E_CART] += g26_prime_length;
  G26_line_to_destination(planner.max_feedrate_mm_s[E_AXIS] / 15.0);
  set_destination_from_current();
  retract_filament(destination);
 }

 return G26_OK;
}

mesh_index_pair find_closest_circle_to_print(const float &X, const float &Y) {
 float closest = 99999.99;
 mesh_index_pair return_val;

 return_val.x_index = return_val.y_index = -1;

 for (uint8_t i = 0; i < GRID_MAX_POINTS_X; i++) {
  for (uint8_t j = 0; j < GRID_MAX_POINTS_Y; j++) {
   if (!is_bitmap_set(circle_flags, i, j)) {
    const float mx = _GET_MESH_X(i),  // We found a circle that needs to be printed
                my = _GET_MESH_Y(j);

    // Get the distance to this intersection
    float f = HYPOT(X - mx, Y - my);

    // It is possible that we are being called with the values
    // to let us find the closest circle to the start position.
    // But if this is not the case, add a small weighting to the
    // distance calculation to help it choose a better place to continue.
    f += HYPOT(g26_x_pos - mx, g26_y_pos - my) / 15.0;

    // Add in the specified amount of Random Noise to our search
    if (random_deviation > 1.0)
     f += random(0.0, random_deviation);
```

```
        if (f < closest) {
          closest = f;              // We found a closer location that is still
          return_val.x_index = i;   // un-printed  --- save the data for it
          return_val.y_index = j;
          return_val.distance = closest;
        }
      }
    }
  }
  bitmap_set(circle_flags, return_val.x_index, return_val.y_index);   // Mark this location as done.
  return return_val;
}

/**
 * print_line_from_here_to_there() takes two cartesian coordinates and draws a line from one
 * to the other. But there are really three sets of coordinates involved. The first coordinate
 * is the present location of the nozzle. We don't necessarily want to print from this location.
 * We first need to move the nozzle to the start of line segment where we want to print. Once
 * there, we can use the two coordinates supplied to draw the line.
 *
 * Note:  Although we assume the first set of coordinates is the start of the line and the second
 * set of coordinates is the end of the line, it does not always work out that way. This function
 * optimizes the movement to minimize the travel distance before it can start printing. This saves
 * a lot of time and eliminates a lot of nonsensical movement of the nozzle. However, it does
 * cause a lot of very little short retracement of th nozzle when it draws the very first line
 * segment of a 'circle'. The time this requires is very short and is easily saved by the other
 * cases where the optimization comes into play.
 */
void print_line_from_here_to_there(const float &sx, const float &sy, const float &sz, const float &ex,
const float &ey, const float &ez) {
  const float dx_s = current_position[X_AXIS] - sx,   // find our distance from the start of the actual
line segment
        dy_s = current_position[Y_AXIS] - sy,
        dist_start = HYPOT2(dx_s, dy_s),        // We don't need to do a sqrt(), we can compare the
distance^2
                                                // to save computation time
        dx_e = current_position[X_AXIS] - ex,   // find our distance from the end of the actual line
segment
        dy_e = current_position[Y_AXIS] - ey,
        dist_end = HYPOT2(dx_e, dy_e),
```

```
            line_length = HYPOT(ex - sx, ey - sy);

          // If the end point of the line is closer to the nozzle, flip the direction,
          // moving from the end to the start. On very small lines the optimization isn't worth it.
          if (dist_end < dist_start && (INTERSECTION_CIRCLE_RADIUS) < ABS(line_length))
            return print_line_from_here_to_there(ex, ey, ez, sx, sy, sz);

          // Decide whether to retract & bump

          if (dist_start > 2.0) {
            retract_filament(destination);
            //todo:  parameterize the bump height with a define
            move_to(current_position[X_AXIS], current_position[Y_AXIS], current_position[Z_AXIS] +
0.500, 0.0);  // Z bump to minimize scraping
            move_to(sx, sy, sz + 0.500, 0.0); // Get to the starting point with no extrusion while bumped
          }

          move_to(sx, sy, sz, 0.0); // Get to the starting point with no extrusion / un-Z bump

          const float e_pos_delta = line_length * g26_e_axis_feedrate * g26_extrusion_multiplier;

          recover_filament(destination);
          move_to(ex, ey, ez, e_pos_delta);  // Get to the ending point with an appropriate amount of extrusion
        }

        inline bool look_for_lines_to_connect() {
          float sx, sy, ex, ey;

          for (uint8_t i = 0; i < GRID_MAX_POINTS_X; i++) {
            for (uint8_t j = 0; j < GRID_MAX_POINTS_Y; j++) {

              #if ENABLED(ULTIPANEL)
                if (user_canceled()) return true;     // Check if the user wants to stop the Mesh Validation
              #endif

              if (i < GRID_MAX_POINTS_X) { // We can't connect to anything to the right than
GRID_MAX_POINTS_X.
                                           // This is already a half circle because we are at the edge of the bed.
```

```
                if (is_bitmap_set(circle_flags, i, j) && is_bitmap_set(circle_flags, i + 1, j)) { // check if we can
do a line to the left
                 if (!is_bitmap_set(horizontal_mesh_line_flags, i, j)) {

                  //
                  // We found two circles that need a horizontal line to connect them
                  // Print it!
                  //
                  sx = _GET_MESH_X(  i  ) + (INTERSECTION_CIRCLE_RADIUS -
(CROSSHAIRS_SIZE)); // right edge
                  ex = _GET_MESH_X(i + 1) - (INTERSECTION_CIRCLE_RADIUS -
(CROSSHAIRS_SIZE)); // left edge

                  sx = constrain(sx, X_MIN_POS + 1, X_MAX_POS - 1);
                  sy = ey = constrain(_GET_MESH_Y(j), Y_MIN_POS + 1, Y_MAX_POS - 1);
                  ex = constrain(ex, X_MIN_POS + 1, X_MAX_POS - 1);

                  if (position_is_reachable(sx, sy) && position_is_reachable(ex, ey)) {

                    if (g26_debug_flag) {
                      SERIAL_ECHOPAIR(" Connecting with horizontal line (sx=", sx);
                      SERIAL_ECHOPAIR(", sy=", sy);
                      SERIAL_ECHOPAIR(") -> (ex=", ex);
                      SERIAL_ECHOPAIR(", ey=", ey);
                      SERIAL_CHAR(')');
                      SERIAL_EOL();
                      //debug_current_and_destination(PSTR("Connecting horizontal line."));
                    }
                    print_line_from_here_to_there(sx, sy, g26_layer_height, ex, ey, g26_layer_height);
                  }
                  bitmap_set(horizontal_mesh_line_flags, i, j);   // Mark it as done so we don't do it again, even
if we skipped it
                 }
                }

                if (j < GRID_MAX_POINTS_Y) { // We can't connect to anything further back than
GRID_MAX_POINTS_Y.
                                             // This is already a half circle because we are at the edge  of the bed.
```

```
                    if (is_bitmap_set(circle_flags, i, j) && is_bitmap_set(circle_flags, i, j + 1)) { // check if we can
do a line straight down
                      if (!is_bitmap_set( vertical_mesh_line_flags, i, j)) {
                        //
                        // We found two circles that need a vertical line to connect them
                        // Print it!
                        //
                        sy = _GET_MESH_Y(  j  ) + (INTERSECTION_CIRCLE_RADIUS -
(CROSSHAIRS_SIZE)); // top edge
                        ey = _GET_MESH_Y(j + 1) - (INTERSECTION_CIRCLE_RADIUS -
(CROSSHAIRS_SIZE)); // bottom edge

                        sx = ex = constrain(_GET_MESH_X(i), X_MIN_POS + 1, X_MAX_POS - 1);
                        sy = constrain(sy, Y_MIN_POS + 1, Y_MAX_POS - 1);
                        ey = constrain(ey, Y_MIN_POS + 1, Y_MAX_POS - 1);

                        if (position_is_reachable(sx, sy) && position_is_reachable(ex, ey)) {

                          if (g26_debug_flag) {
                            SERIAL_ECHOPAIR(" Connecting with vertical line (sx=", sx);
                            SERIAL_ECHOPAIR(", sy=", sy);
                            SERIAL_ECHOPAIR(") -> (ex=", ex);
                            SERIAL_ECHOPAIR(", ey=", ey);
                            SERIAL_CHAR(')');
                            SERIAL_EOL();

                            #if ENABLED(AUTO_BED_LEVELING_UBL)
                              debug_current_and_destination(PSTR("Connecting vertical line."));
                            #endif
                          }
                          print_line_from_here_to_there(sx, sy, g26_layer_height, ex, ey, g26_layer_height);
                        }
                        bitmap_set(vertical_mesh_line_flags, i, j);   // Mark it as done so we don't do it again, even if
skipped
                      }
                    }
                  }
                }
              }
            }
```

```
  return false;
}

/**
 * Turn on the bed and nozzle heat and
 * wait for them to get up to temperature.
 */
inline bool turn_on_heaters() {
  millis_t next = millis() + 5000UL;
  #if HAS_HEATED_BED
    #if ENABLED(ULTRA_LCD)
      if (g26_bed_temp > 25) {
        lcd_setstatusPGM(PSTR("G26 Heating Bed."), 99);
        lcd_quick_feedback(true);
        #if ENABLED(ULTIPANEL)
          lcd_external_control = true;
        #endif
    #endif
        thermalManager.setTargetBed(g26_bed_temp);
        while (ABS(thermalManager.degBed() - g26_bed_temp) > 3) {

          #if ENABLED(ULTIPANEL)
            if (is_lcd_clicked()) return exit_from_g26();
          #endif

          if (ELAPSED(millis(), next)) {
            next = millis() + 5000UL;
            thermalManager.print_heaterstates();
            SERIAL_EOL();
          }
          idle();
          SERIAL_FLUSH(); // Prevent host M105 buffer overrun.
        }
    #if ENABLED(ULTRA_LCD)
      }
      lcd_setstatusPGM(PSTR("G26 Heating Nozzle."), 99);
      lcd_quick_feedback(true);
    #endif
  #endif
```

```
  // Start heating the nozzle and wait for it to reach temperature.
  thermalManager.setTargetHotend(g26_hotend_temp, 0);
  while (ABS(thermalManager.degHotend(0) - g26_hotend_temp) > 3) {

    #if ENABLED(ULTIPANEL)
      if (is_lcd_clicked()) return exit_from_g26();
    #endif

    if (ELAPSED(millis(), next)) {
      next = millis() + 5000UL;
      thermalManager.print_heaterstates();
      SERIAL_EOL();
    }
    idle();

    SERIAL_FLUSH(); // Prevent host M105 buffer overrun.
  }
  #if ENABLED(ULTRA_LCD)
    lcd_reset_status();
    lcd_quick_feedback(true);
  #endif

  return G26_OK;
}

float valid_trig_angle(float d) {
  while (d > 360.0) d -= 360.0;
  while (d < 0.0) d += 360.0;
  return d;
}

/**
 * G26: Mesh Validation Pattern generation.
 *
 * Used to interactively edit the mesh by placing the
 * nozzle in a problem area and doing a G29 P4 R command.
 *
 * Parameters:
 *
 *  B  Bed Temperature
```

```
 *  C  Continue from the Closest mesh point
 *  D  Disable leveling before starting
 *  F  Filament diameter
 *  H  Hotend Temperature
 *  K  Keep heaters on when completed
 *  L  Layer Height
 *  O  Ooze extrusion length
 *  P  Prime length
 *  Q  Retraction multiplier
 *  R  Repetitions (number of grid points)
 *  S  Nozzle Size (diameter) in mm
 *  U  Random deviation (50 if no value given)
 *  X  X position
 *  Y  Y position
 */
void gcode_G26() {
  SERIAL_ECHOLNPGM("G26 command started. Waiting for heater(s).");

  // Don't allow Mesh Validation without homing first,
  // or if the parameter parsing did not go OK, abort
  if (axis_unhomed_error()) return;

  g26_extrusion_multiplier    = EXTRUSION_MULTIPLIER;
  g26_retraction_multiplier   = RETRACTION_MULTIPLIER;
  g26_layer_height            = MESH_TEST_LAYER_HEIGHT;
  g26_prime_length            = PRIME_LENGTH;
  g26_bed_temp                = MESH_TEST_BED_TEMP;
  g26_hotend_temp             = MESH_TEST_HOTEND_TEMP;
  g26_prime_flag              = 0;

  float g26_nozzle            = MESH_TEST_NOZZLE_SIZE,
        g26_filament_diameter = DEFAULT_NOMINAL_FILAMENT_DIA,
        g26_ooze_amount       = parser.linearval('O', OOZE_AMOUNT);

  bool g26_continue_with_closest = parser.boolval('C'),
       g26_keep_heaters_on       = parser.boolval('K');

  if (parser.seenval('B')) {
    g26_bed_temp = parser.value_celsius();
    if (g26_bed_temp && !WITHIN(g26_bed_temp, 40, 140)) {
```

```
    SERIAL_PROTOCOLLNPGM("?Specified bed temperature not plausible (40-140C).");
    return;
  }
}

if (parser.seenval('L')) {
  g26_layer_height = parser.value_linear_units();
  if (!WITHIN(g26_layer_height, 0.0, 2.0)) {
    SERIAL_PROTOCOLLNPGM("?Specified layer height not plausible.");
    return;
  }
}

if (parser.seen('Q')) {
  if (parser.has_value()) {
    g26_retraction_multiplier = parser.value_float();
    if (!WITHIN(g26_retraction_multiplier, 0.05, 15.0)) {
      SERIAL_PROTOCOLLNPGM("?Specified Retraction Multiplier not plausible.");
      return;
    }
  }
  else {
    SERIAL_PROTOCOLLNPGM("?Retraction Multiplier must be specified.");
    return;
  }
}

if (parser.seenval('S')) {
  g26_nozzle = parser.value_float();
  if (!WITHIN(g26_nozzle, 0.1, 1.0)) {
    SERIAL_PROTOCOLLNPGM("?Specified nozzle size not plausible.");
    return;
  }
}

if (parser.seen('P')) {
  if (!parser.has_value()) {
    #if ENABLED(ULTIPANEL)
      g26_prime_flag = -1;
    #else
```

```
        SERIAL_PROTOCOLLNPGM("?Prime length must be specified when not using an LCD.");
        return;
      #endif
    }
    else {
      g26_prime_flag++;
      g26_prime_length = parser.value_linear_units();
      if (!WITHIN(g26_prime_length, 0.0, 25.0)) {
        SERIAL_PROTOCOLLNPGM("?Specified prime length not plausible.");
        return;
      }
    }
  }

  if (parser.seenval('F')) {
    g26_filament_diameter = parser.value_linear_units();
    if (!WITHIN(g26_filament_diameter, 1.0, 4.0)) {
      SERIAL_PROTOCOLLNPGM("?Specified filament size not plausible.");
      return;
    }
  }
  g26_extrusion_multiplier *= sq(1.75) / sq(g26_filament_diameter); // If we aren't using 1.75mm filament, we need to
                                                                    // scale up or down the length needed to get the
                                                                    // same volume of filament

  g26_extrusion_multiplier *= g26_filament_diameter * sq(g26_nozzle) / sq(0.3); // Scale up by nozzle size

  if (parser.seenval('H')) {
    g26_hotend_temp = parser.value_celsius();
    if (!WITHIN(g26_hotend_temp, 165, 280)) {
      SERIAL_PROTOCOLLNPGM("?Specified nozzle temperature not plausible.");
      return;
    }
  }

  if (parser.seen('U')) {
    randomSeed(millis());
    // This setting will persist for the next G26
```

```
  random_deviation = parser.has_value() ? parser.value_float() : 50.0;
 }

 int16_t g26_repeats;
 #if ENABLED(ULTIPANEL)
  g26_repeats = parser.intval('R', GRID_MAX_POINTS + 1);
 #else
  if (!parser.seen('R')) {
   SERIAL_PROTOCOLLNPGM("?(R)epeat must be specified when not using an LCD.");
   return;
  }
  else
   g26_repeats = parser.has_value() ? parser.value_int() : GRID_MAX_POINTS + 1;
 #endif
 if (g26_repeats < 1) {
  SERIAL_PROTOCOLLNPGM("?(R)epeat value not plausible; must be at least 1.");
  return;
 }

 g26_x_pos = parser.seenval('X') ? RAW_X_POSITION(parser.value_linear_units()) :
current_position[X_AXIS];
 g26_y_pos = parser.seenval('Y') ? RAW_Y_POSITION(parser.value_linear_units()) :
current_position[Y_AXIS];
 if (!position_is_reachable(g26_x_pos, g26_y_pos)) {
  SERIAL_PROTOCOLLNPGM("?Specified X,Y coordinate out of bounds.");
  return;
 }

 /**
  * Wait until all parameters are verified before altering the state!
  */
 set_bed_leveling_enabled(!parser.seen('D'));

 if (current_position[Z_AXIS] < Z_CLEARANCE_BETWEEN_PROBES) {
  do_blocking_move_to_z(Z_CLEARANCE_BETWEEN_PROBES);
  set_current_from_destination();
 }

 if (turn_on_heaters() != G26_OK) goto LEAVE;
```

```
current_position[E_CART] = 0.0;
sync_plan_position_e();

if (g26_prime_flag && prime_nozzle() != G26_OK) goto LEAVE;

/**
 *  Bed is preheated
 *
 *  Nozzle is at temperature
 *
 *  Filament is primed!
 *
 *  It's  "Show Time" !!!
 */

ZERO(circle_flags);
ZERO(horizontal_mesh_line_flags);
ZERO(vertical_mesh_line_flags);

// Move nozzle to the specified height for the first layer
set_destination_from_current();
destination[Z_AXIS] = g26_layer_height;
move_to(destination, 0.0);
move_to(destination, g26_ooze_amount);

#if ENABLED(ULTIPANEL)
  lcd_external_control = true;
#endif

//debug_current_and_destination(PSTR("Starting G26 Mesh Validation Pattern."));

#if DISABLED(ARC_SUPPORT)

  /**
   * Pre-generate radius offset values at 30 degree intervals to reduce CPU load.
   */
  #define A_INT 30
  #define _ANGS (360 / A_INT)
  #define A_CNT (_ANGS / 2)
  #define _IND(A) ((A + _ANGS * 8) % _ANGS)
```

```
              #define _COS(A) (trig_table[_IND(A) % A_CNT] * (_IND(A) >= A_CNT ? -1 : 1))
              #define _SIN(A) (-_COS((A + A_CNT / 2) % _ANGS))
              #if A_CNT & 1
                #error "A_CNT must be a positive value. Please change A_INT."
              #endif
              float trig_table[A_CNT];
              for (uint8_t i = 0; i < A_CNT; i++)
                trig_table[i] = INTERSECTION_CIRCLE_RADIUS * cos(RADIANS(i * A_INT));

            #endif // !ARC_SUPPORT

            mesh_index_pair location;
            do {
              location = g26_continue_with_closest
                ? find_closest_circle_to_print(current_position[X_AXIS], current_position[Y_AXIS])
                : find_closest_circle_to_print(g26_x_pos, g26_y_pos); // Find the closest Mesh Intersection to
where we are now.

              if (location.x_index >= 0 && location.y_index >= 0) {
                const float circle_x = _GET_MESH_X(location.x_index),
                            circle_y = _GET_MESH_Y(location.y_index);

                // If this mesh location is outside the printable_radius, skip it.
                if (!position_is_reachable(circle_x, circle_y)) continue;

                // Determine where to start and end the circle,
                // which is always drawn counter-clockwise.
                const uint8_t xi = location.x_index, yi = location.y_index;
                const bool f = yi == 0, r = xi >= GRID_MAX_POINTS_X - 1, b = yi >=
GRID_MAX_POINTS_Y - 1;

                #if ENABLED(ARC_SUPPORT)

                  #define ARC_LENGTH(quarters)  (INTERSECTION_CIRCLE_RADIUS * M_PI * (quarters) /
2)
                  float sx = circle_x + INTERSECTION_CIRCLE_RADIUS,   // default to full circle
                        ex = circle_x + INTERSECTION_CIRCLE_RADIUS,
                        sy = circle_y, ey = circle_y,
                        arc_length = ARC_LENGTH(4);
```

```
// Figure out where to start and end the arc - we always print counterclockwise
if (xi == 0) {                          // left edge
  sx = f ? circle_x + INTERSECTION_CIRCLE_RADIUS : circle_x;
  ex = b ? circle_x + INTERSECTION_CIRCLE_RADIUS : circle_x;
  sy = f ? circle_y : circle_y - INTERSECTION_CIRCLE_RADIUS;
  ey = b ? circle_y : circle_y + INTERSECTION_CIRCLE_RADIUS;
  arc_length = (f || b) ? ARC_LENGTH(1) : ARC_LENGTH(2);
}
else if (r) {                           // right edge
  sx = b ? circle_x - INTERSECTION_CIRCLE_RADIUS : circle_x;
  ex = f ? circle_x - INTERSECTION_CIRCLE_RADIUS : circle_x;
  sy = b ? circle_y : circle_y + INTERSECTION_CIRCLE_RADIUS;
  ey = f ? circle_y : circle_y - INTERSECTION_CIRCLE_RADIUS;
  arc_length = (f || b) ? ARC_LENGTH(1) : ARC_LENGTH(2);
}
else if (f) {
  sx = circle_x + INTERSECTION_CIRCLE_RADIUS;
  ex = circle_x - INTERSECTION_CIRCLE_RADIUS;
  sy = ey = circle_y;
  arc_length = ARC_LENGTH(2);
}
else if (b) {
  sx = circle_x - INTERSECTION_CIRCLE_RADIUS;
  ex = circle_x + INTERSECTION_CIRCLE_RADIUS;
  sy = ey = circle_y;
  arc_length = ARC_LENGTH(2);
}
const float arc_offset[2] = {
  circle_x - sx,
  circle_y - sy
};

const float dx_s = current_position[X_AXIS] - sx,   // find our distance from the start of the
actual circle
        dy_s = current_position[Y_AXIS] - sy,
        dist_start = HYPOT2(dx_s, dy_s);
const float endpoint[XYZE] = {
  ex, ey,
  g26_layer_height,
  current_position[E_CART] + (arc_length * g26_e_axis_feedrate * g26_extrusion_multiplier)
```

```
                };

                if (dist_start > 2.0) {
                  retract_filament(destination);
                  //todo:  parameterize the bump height with a define
                  move_to(current_position[X_AXIS], current_position[Y_AXIS], current_position[Z_AXIS] +
0.500, 0.0);  // Z bump to minimize scraping
                  move_to(sx, sy, g26_layer_height + 0.500, 0.0); // Get to the starting point with no extrusion
while bumped
                }

                move_to(sx, sy, g26_layer_height, 0.0); // Get to the starting point with no extrusion / un-Z
bump

                recover_filament(destination);
                const float save_feedrate = feedrate_mm_s;
                feedrate_mm_s = PLANNER_XY_FEEDRATE() / 10.0;
                plan_arc(endpoint, arc_offset, false);  // Draw a counter-clockwise arc
                feedrate_mm_s = save_feedrate;
                set_destination_from_current();
                #if ENABLED(ULTIPANEL)
                  if (user_canceled()) goto LEAVE; // Check if the user wants to stop the Mesh Validation
                #endif

              #else // !ARC_SUPPORT

                int8_t start_ind = -2, end_ind = 9; // Assume a full circle (from 5:00 to 5:00)
                if (xi == 0) {                  // Left edge? Just right half.
                  start_ind = f ? 0 : -3;         //  03:00 to 12:00 for front-left
                  end_ind = b ? 0 : 2;            //  06:00 to 03:00 for back-left
                }
                else if (r) {                   // Right edge? Just left half.
                  start_ind = b ? 6 : 3;          //  12:00 to 09:00 for front-right
                  end_ind = f ? 5 : 8;            //  09:00 to 06:00 for back-right
                }
                else if (f) {                   // Front edge? Just back half.
                  start_ind = 0;                  //  03:00
                  end_ind = 5;                    //  09:00
                }
                else if (b) {                   // Back edge? Just front half.
```

```
      start_ind = 6;                //  09:00
      end_ind = 11;                 //  03:00
    }

    for (int8_t ind = start_ind; ind <= end_ind; ind++) {

      #if ENABLED(ULTIPANEL)
        if (user_canceled()) goto LEAVE;         // Check if the user wants to stop the Mesh Validation
      #endif

      float rx = circle_x + _COS(ind),           // For speed, these are now a lookup table entry
            ry = circle_y + _SIN(ind),
            xe = circle_x + _COS(ind + 1),
            ye = circle_y + _SIN(ind + 1);

      #if IS_KINEMATIC
        // Check to make sure this segment is entirely on the bed, skip if not.
        if (!position_is_reachable(rx, ry) || !position_is_reachable(xe, ye)) continue;
      #else                                      // not, we need to skip
        rx = constrain(rx, X_MIN_POS + 1, X_MAX_POS - 1); // This keeps us from bumping the
endstops
        ry = constrain(ry, Y_MIN_POS + 1, Y_MAX_POS - 1);
        xe = constrain(xe, X_MIN_POS + 1, X_MAX_POS - 1);
        ye = constrain(ye, Y_MIN_POS + 1, Y_MAX_POS - 1);
      #endif

      print_line_from_here_to_there(rx, ry, g26_layer_height, xe, ye, g26_layer_height);
      SERIAL_FLUSH();  // Prevent host M105 buffer overrun.
    }

  #endif // !ARC_SUPPORT

  if (look_for_lines_to_connect()) goto LEAVE;
 }

 SERIAL_FLUSH(); // Prevent host M105 buffer overrun.

} while (--g26_repeats && location.x_index >= 0 && location.y_index >= 0);

LEAVE:
```

```
    lcd_setstatusPGM(PSTR("Leaving G26"), -1);

    retract_filament(destination);
    destination[Z_AXIS] = Z_CLEARANCE_BETWEEN_PROBES;

    //debug_current_and_destination(PSTR("ready to do Z-Raise."));
    move_to(destination, 0); // Raise the nozzle
    //debug_current_and_destination(PSTR("done doing Z-Raise."));

    destination[X_AXIS] = g26_x_pos;                        // Move back to the starting position
    destination[Y_AXIS] = g26_y_pos;
    //destination[Z_AXIS] = Z_CLEARANCE_BETWEEN_PROBES;          // Keep the nozzle where
it is

    move_to(destination, 0); // Move back to the starting position
    //debug_current_and_destination(PSTR("done doing X/Y move."));

    #if ENABLED(ULTIPANEL)
      lcd_external_control = false;     // Give back control of the LCD Panel!
    #endif

    if (!g26_keep_heaters_on) {
      #if HAS_HEATED_BED
        thermalManager.setTargetBed(0);
      #endif
      thermalManager.setTargetHotend(0, 0);
    }
  }

#endif // G26_MESH_VALIDATION
```