Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Joaquim Isaac Guimarães Candido

DESENVOLVIMENTO DE *SOFTWARE* PARA UM EQUIPAMENTO CNC

Americana, SP

2021

Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

JOAQUIM ISAAC GUIMARÃES CANDIDO

DESENVOLVIMENTO DE *SOFTWARE* PARA UM EQUIPAMENTO CNC

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas desta Instituição, sob orientação do prof. Ms. Rossano Pablo Pinto.

Área de concentração: Engenharia de *software*.

Americana, SP

2021

Fatec Americana Ministro Ralph Biasi

Centro Paula Souza

SÃO PAULO GOVERNO DO ESTADO

Secretaria de Desenvolvimento Econômico

Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

C223d CANDIDO, Joaquim Isaac Guimarães

Desenvolvimento de software para um equipamento CNC. / Joaquim Isaac Guimarães Candido. – Americana, 2021.

57f.

Monografia (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Rossano Pablo Pinto

1 Engenharia de software 2. Automação I. PINTO, Rossano Pablo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.6

Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

JOAQUIM ISAAC GUIMARÃES CANDIDO

DESENVOLVIMENTO DE *SOFTWARE* PARA UM EQUIPAMENTO CNC

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas desta Instituição, sob orientação do prof. Me. Rossano Pablo Pinto.

Área de concentração: Engenharia de *software*.

Americana, 15 de junho de 2021.

Banca Examinadora:

_____

Rossano Pablo Pinto
Mestrado em Engenharia Elétrica
Faculdade de Tecnologia de Americana, FATEC, Brasil

_____

Kleber de Oliveira Andrade
Doutor em Engenharia Mecânica
Faculdade de Tecnologia de Americana, FATEC, Brasil

_____

Francisco Carlos Mancin
Mestre em Educação
Faculdade de Tecnologia de Americana, FATEC, Brasil

Para além de todo narcisismo, digo que dedico este trabalho a mim mesmo. A esta altura, é um presente – e só eu sei do valor subjacente. Vejo hoje que, de todas as dádivas do universo, a perseverança é a mais gratificante.

Das páginas escritas, destilei sabedoria que transcende a ciência contida nelas; deste ponto em diante, espero calma e tranquilidade, o próximo passo, a proverbial página virada. Contudo, como nem todo o objetivo do trajeto é o destino, fico feliz pelo esforço dos meus, pelo incentivo, por pequenos gestos que significam muito, por palavras duras quando necessário e, não menos importante, por certa garagem conter mais do que somente veículos.

AGRADECIMENTOS

Começando pelo sumário, digo que todos que citarei aqui são meus professores de uma forma ou de outra.

Agradeço a meus pais, seu Nelson e Dona Elsa, por sinceramente proverem mais do que muitas vezes eu acreditei merecer e mais do que jamais tiveram para si mesmos. Em grande medida, se consegui, consegui porque, de ambos, tive todo o suporte do mundo.

Agradeço a Rafael Estevam Baldy dos Reis, professor de programação e aficionado por engenhocas e invenções, por ter me emprestado incontáveis equipamentos e um pedaço da mente para que este projeto fosse possível. Também agradeço seu esforço e paciência dedicados a me fazer compreender cada uma das etapas e preencher as lacunas quando não estavam claras, até mesmo quando deveriam ser óbvias.

Agradeço ao professor Rossano Pablo Pinto, meu orientador, que conseguiu me deixar confortável com a escolha para o projeto e me incentivou a seguir no caminho aqui proposto, sempre ressaltando o valor do aprendizado que eu poderia extrair, além de me dar inúmeras dicas e auxílios ao longo do projeto, sem os quais este não poderia teria sido realizado.

Além destes diretamente relacionados, agradeço a todos os meus outros professores que, mesmo que paralela e indiretamente, ao longo de minha trajetória, possibilitaram que eu pudesse estar neste ponto.

Agradeço também a todas as outras pessoas que participaram desse trajeto. É de imensa felicidade que não cito aqui somente membros da academia, mas, principalmente, amigos próximos e meus entes queridos, que estimo além do que palavras podem descrever.

Por fim, mas não menos importante, agradeço à Academia e a todos os seus membros, em todas as esferas. O valor dessa Instituição é propagado pelo mundo por nós, alunos, mas só é possível pelo amor e perseverança de vocês, não tenho dúvida disso.

"*Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.*"

"Imaginação é mais importante que conhecimento. O conhecimento é limitado. A imaginação envolve o mundo."

– Albert Einstein, em entrevista ao jornal *The Saturday Evening Post*. Flórida, EUA, em outubro de 1929.

RESUMO

Este trabalho descreve tecnicamente o desenvolvimento de uma suíte de *software* controladora para um equipamento eletromecânico que, através de cálculos numéricos, define os movimentos necessários para a produção automatizada, adicionando ou removendo material, de peças físicas, também conhecido como CNC. A proposta inclui todos os estágios de desenvolvimento de *software* para o objetivo proposto: primeiro, descreve o projeto básico contendo requisitos, diagramas e detalhes técnicos exigidos pela engenharia de *software*; de mesmo modo, pretende-se o detalhamento das etapas técnicas de implementação realizadas para que o produto final seja possível, buscando a clareza de cada uma das etapas descritas ao longo da implementação dos requisitos, na medida em que se fizeram necessárias.

Palavras-chave: CNC; engenharia de software*; automação.*

ABSTRACT

This work describes technically the development of a controller software suite for an electromechanical equipment that, through numerical calculations, defines the necessary movements for automated production, by adding or removing material, of physical parts, also known as CNC. The proposal includes all stages of software development for the proposed goal: first, it describes the basic project containing requirements, diagrams and technical details as required by software engineering; also, it intends detailment of the technical implementation steps taken in order for the final product to be possible, pursuing the clarity of each one of the described steps throughout the implementation of the requirements, as they were made necessary.

Keywords: CNC; software engineering; automation.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIAÇÕES

1. **Arduino:** Plataforma de dispositivos eletrônicos de código fonte aberto que tem foco na disponibilização de microcontroladores de fácil programação e uso para projetos de automação [MERRIAM-WEBSTER, 2021].
2. ***bit:*** a menor unidade de informação. Um *bit* representa uma informação binária, ou seja, uns e zeros [STANFORD UNIVERSITY, 2021].
3. ***byte:*** convenção de armazenamento de *bits* de informação geralmente em pacotes de 8 unidades: ou seja, um *byte* armazena *8 bits*. Esta convenção nasce através do padrão ASCII (*American Standard Code for Information Interchange*, ou "Código Padrão [norte-]Americano para Intercâmbio de Informações", em tradução livre) estabelecido em 1963, originalmente utilizando 7 *bits* para armazenar informações, mas adicionando mais um para fins de integridade (paridade) [NORMAN, 2010].
4. **CAD:** Do inglês, *Computer-Aided Development* (Desenvolvimento Auxiliado por Computador, em tradução livre). Trata de desenvolvimento ou desenho de projetos de uma forma auxiliada por ferramentas providas pelo computador.
5. **CAM:** Do inglês, *Computer-aided manufacturing* ("fabricação", ou "manufatura auxiliada por computador"; tradução livre). Trata-se de uma categoria de soluções de *software* que instruem equipamentos CNC, a partir de um modelo concebido, na manufatura de objetos.
6. **CNC:** Do inglês, *Computer Numerical Control* (controle numérico por computador, tradução livre): similar às máquinas NCs, são equipamentos controlados por instrução numérica, contudo, enviadas por computadores [PATEL, 2020].
7. **Comunicação *serial*:** Método de comunicação que se baseia no envio de dados um *bit* por vez em uma forma sequencial. Amplamente utilizado em telecomunicações e afins. Por ser um tipo de comunicação que não exige nenhum outro equipamento além dos dispositivos a se comunicar, é bastante utilizada para a conexão entre computadores e periféricos ou outros dispositivos.
8. ***Endstop*:** "fim de curso", em tradução livre. Tratam-se de dispositivos encontrados nos limites do curso dos eixos dos equipamentos que, quando ativados, permitem que o próprio equipamento saiba que não deve mais se mover pois chegou no limite.

9. ***Firmware*:** *software* de computador de baixo nível (ou seja, manipula o *hardware* diretamente) que funciona por manipular instruções para um *hardware* específico com o objetivo de prover um ambiente padronizado de operação [GANSSLE, 2004]. Em alguns casos, opera como um *software* básico sobre o qual todo o restante operará; em outros, pode operar como um sistema operacional completo [TECHTERMS, 2021].
10. ***Framework* (computação):** Diz-se de plataformas designadas para desenvolvimento de aplicações em *software*. Fornece a base para linguagens de programação que as suportam nativamente com alguns recursos, como comunicação com periféricos, unidades de disco, comunicação com outros *softwares* e plataformas, comunicação entre computadores (rede ou outro meio) etc. Uma linguagem de programação embarcada em uma *framework* permite que o desenvolvedor não precise desenvolver um *software* para cada pequena interação existente, fornecendo algumas ferramentas prontas para operações mais comuns.
11. **GCODE:** Sigla para *Geometric Code* – do inglês, código geométrico (tradução livre), também conhecido como RS-274), é um conjunto de instruções amplamente utilizado como linguagem de programação para equipamentos CNC. Basicamente, um roteiro, ou *script*, GCODE contém comandos posicionais cujos quais o *firmware* pode ler e instruir a máquina para reprogramar a si mesma [REPRAP, 2021].
12. ***Hardware**:* Estrutura física de um equipamento. Peças, estruturas e motores.
13. ***Homing*:** um dos comandos definidos pela linguagem GCODE. Significa "regresso à casa", ou "ao ponto de partida", em tradução livre. Indica que o equipamento deve realizar movimentos para retornar ao ponto determinado como ponto inicial, normalmente X=0, Y=0 e Z=0; porém, outros eixos podem ser afetados por isso e de formas diferentes, dependendo do tipo de maquinário e a forma de operação.
14. **Microcontrolador:** Circuito integrado que contém estruturas de um microprocessador, memória e circuitos necessários para controle de uma ou mais funções de um sistema.
15. **Motor de passo:** Motor que é especializado por prover movimentos *discretos*, ou seja, com início e fim definidos com grande precisão [MATTEDE, 2019].
16. **NC:** Do inglês, *Numerical Control* (controle numérico, tradução livre): a sigla refere-se a máquinas mecânicas e eletromecânicas com capacidade de executar a produção de equipamentos e peças através de comandos "numéricos", a partir de uma fonte de instrução.

As NCs eram codificadas com cartões perfurados, fitas magnéticas, entre outros, e produziam peças que precisavam de velocidade, precisão ou ambos na fabricação.

17. ***Pixel:*** Um pixel é a menor unidade que pode ser representada na exibição de uma imagem digital. É a unidade básica em gráficos digitais bidimensionais e, quando combinados, formam uma imagem completa para fotos, vídeos, texto ou qualquer outra informação visível em um computador [TECNOPEDIA, 2021].
18. ***Software****:* Instruções lógicas para um computador. Programas.
19. **UML (*Unified Modelling Language)*:** Conjunto de notações padronizadas utilizadas para desenvolvimento de diagramas voltados ao desenvolvimento de sistemas [UML DIAGRAMS, 2020].
20. **USB:** *Universal Serial Bus*, ou "Barramento *Serial* Universal", em tradução livre. Trata-se de uma padronização para comunicação *serial* entre equipamentos diversos (ver item 7) [MITCHELL, 2020].

INTRODUÇÃO

Invenções fazem parte do processo humano de transformação do ambiente em que ocupa. A sociedade, a ética, a lógica, a política, a guerra, a roda, o canhão, a corrida espacial, o computador eletrônico – todas, a partir da necessidade de tornar o mundo mais amigável às fragilidades humanas diante do que as regras naturais determinam.

Nesse contexto, é evidente que as invenções têm um papel importante para o desenvolvimento da sociedade, na medida em que permite a realização de tarefas que vão além das possibilidades físicas de uma ou de um conjunto de pessoas, sejam elas materiais ou intelectuais. Além disso, as mudanças trazidas por elas são auspiciosas: trazem segurança, precisão, confiabilidade na repetibilidade e velocidade em processos que demandam muito tempo, esforço físico, custo elevado, perdas eventuais de vidas humanas – ou todas essas consequências em conjunto – para ser realizados manualmente, ou sequer poderiam ser realizados.

Transportes de produtos e pessoas em massa e com velocidade, transformações de materiais para fabricação de objetos industrializados, dentre muitos outros exemplos, deixam clara a razão pela qual o avanço tecnológico tende a gradualmente melhorar seus próprios processos e métodos para cada vez mais conseguir extrair o máximo possível da transformação de materiais em bens socioeconomicamente utilizáveis. Computadores, por exemplo, permitem a estruturação lógica sequencial de tarefas que demandam esforço ou disposição de tempo repetitivo por parte de uma ou mais pessoas, além de executar essas tarefas exponencialmente mais rápida e precisamente.

Um fator muito importante quanto às invenções mais relevantes da história da humanidade é que alteram como a dinâmica econômica funciona. Dentre muitos exemplos, destacam-se as consequências da Revolução Industrial do século XVIII, que, por meio das inovações tecnológicas como as máquinas a vapor como suporte a maquinários de automação industrial, mudaram drasticamente a economia mundial por acelerarem o processo de produção, estabelecendo novos paradigmas e processos que seriam adotados gradualmente no mundo inteiro [KOOJI, 2015].

Além disso, é possível perceber que a mesma máquina a vapor inventada para agilizar os processos de fabricação também foi utilizada para facilitar – automatizar – processos de mineração e para a invenção de locomotivas e das estradas de ferro, estas que encurtaram as proverbiais distâncias e alteraram ainda mais o cenário econômico mundial, além de pavimentar o caminho

para o contexto social e econômico de todo o século XX e que solidificaram o contexto de agora, no início do século XXI.

Ainda nesta linha, observa-se que a precisão de fabricação sob padrões modernos é imprescindível. Engrenagens e componentes complexos não deixam margem para aproximações pois precisam de precisão cirúrgica; a falha em obedecer a este requisito compromete ambos o funcionamento e a qualidade das peças resultantes. Nos casos de fabricação de peças e componentes de modo manual, as chances de que surgisse esse nível de imprecisão foi aumentando gradativamente ao passo que os inventos evoluíram em tamanho e complexidade; com isso, as falhas e variações se tornaram inaceitáveis.

Ao longo do processo de constante evolução das invenções mecânicas, a fabricação de peças para veículos de combustão interna, aviões e foguetes, ou mesmo de *chips* dos computadores modernos – alguns dos pilares da sociedade atual e que, sem dúvida, exigem precisão – demandaram refinamento das técnicas utilizadas para fazê-los; parece, pois, que seriam improváveis e/ou, ao menos, inviáveis, sem máquinas para desenhá-las e manufaturá-las: desse modo, sem as máquinas automatizadas e de precisão sobre-humanas, novas invenções nos padrões sobre-humanos existentes hoje seriam impossíveis, recursivamente graduais.

No contexto de máquinas computadorizadas, esse salto inovativo e de avanço teve seu pico no período da Segunda Grande Guerra. Ora, historicamente é possível observar, por muitas vezes, que o processo de evolução tecnológica durante tempos de guerra avança em proporções muito divergentes à dos tempos de paz. A necessidade de novas invenções que dão a vantagem sobre o inimigo fomentam pesquisas e desenvolvimento tecnológico para otimizar a máquina de guerra, seja para vantagem tática ou para *matar mais eficientemente*. Os processos de fissão atômica para liberação de energia que culminaram nas famigeradas bombas "cruéis" de tão eficientes [FUSTER, 2016] é um grande exemplo de evolução com base na necessidade – ainda que, neste caso, uma muito mais política do que bélica[1]. Nesse sentido, ainda que em outros contextos de invenção, as NCs não são diferentes no tocante à sua evolução.

[1] Os Estados Unidos tinham o objetivo, no contexto da segunda guerra, garantir superioridade tecnológica e política no mundo, de modo a solidificá-la. Com a invenção da bomba atômica *primeiro*, ou seja, antes da Alemanha nazista (apoiada inclusive por Albert Einstein, exilado nos EUA, que temia que a os nazistas conseguissem primeiro, apesar de temer esta invenção), conseguiram garantir a soberania quando, após a derrota da Alemanha, utilizando como pretexto a intenção de "perder menos vidas americanas" em invasões com soldados em terras japonesas, detonaram

Ainda que propostas de automatização industrial tivessem sido concebidos num contexto pré-guerra, a maior parte dos avanços considerados chave foram baseados em necessidades emergidas durante os acontecimentos da Segunda Guerra Mundial. No contexto de automação, historicamente,

> [o]s maiores desenvolvimentos técnicos nos quais [máquinas] NCs são baseadas foram realizados durante a Segunda Guerra Mundial, particularmente nos trabalhos relacionados com servomecanismos[2] de alto desempenho, como os usados em assentamentos automáticos de armas por radares. O foguete V2 alemão e os desenvolvimentos subsequentes norte-americanos fundaram a produção [...] de componentes de muitos dos sistemas NC produzidos atualmente. [FERGUSON, 1978]

As primeiras máquinas, controladas com fitas ou cartões perfurados, tinham sido cogitadas, ao fim da Segunda Guerra, para originalmente serem controladas por circuitos integrados. Contudo, nos anos imediatos do pós-guerra, computadores eram caros; as máquinas NCs também. Juntamente com o custo de manutenção, optou-se pela utilização dos sistemas analógicos: ora, essas máquinas, em seus primórdios, possuíam poucas aplicações que justificassem seu custo inicial, salvo por casos isolados que necessitavam de precisão, como a produção de peças de fuselagem e instrumentos para aviação, por exemplo.

Porém, com o desenvolvimento e avanços de circuitos eletrônicos, foi natural que houvesse a migração dos processos para que esses circuitos substituíssem a mecânica de instruções enviadas para as máquinas NC, posto que, conforme esses componentes iam evoluindo, seus custos foram diminuindo, bem como seu tamanho, além de ajudar na manutenção e na padronização das máquinas. Assim nascem as CNCs, ou seja, máquinas com controle numérico de instruções controladas por *computador* [FERGUSON, 1978].

Existem muitas vantagens em utilizar computadores para essas tarefas. Primeiro, é possível que uma única máquina controle muitos sistemas em paralelo, produzindo máquinas mais

---

duas das novas bombas nucleares contra o império japonês em setembro de 1945, devastando as cidades de Hiroshima e Nagasaki e culminando na imediata rendição incondicional do país à guerra. Com a "demonstração" de poder nuclear, a superioridade bélica norte-americana se estabeleceu, conquistada principalmente através do medo de retaliação nuclear, mesmo após a invenção de bombas nucleares por outras nações após o fim da guerra.

[2] Servomotores são motores cujo posicionamento angular pode ser determinado com precisão.

complexas e, com isso, com apenas um computador é possível controlar toda uma linha de produção.

Adicionalmente, equipamentos eletrônicos costumam possuir mais confiabilidade do que as partes mecânicas para ler e delegar instruções aos equipamentos, além de que quaisquer mudanças necessárias no projeto necessitam apenas de alteração no *software*, enquanto nos casos de instrução mecânica das NCs eram necessárias alteração de partes e mecanismos.

Por fim, é possível fazer cálculos e estimativas de produção de forma automática com base nas variáveis que a máquina pode prever, como tempo de corte, velocidade e precisão personalizadas, autotestes, controle automático de temperatura de componentes, compensações e escala da fabricação de peças a partir de simples configurações, decisões de alta velocidade por parte do equipamento, como detecção de falhas que desencadeiam paradas emergenciais etc.

Além disso, é de se esperar que um computador possa também ser padronizado para que existam protocolos de instrução e manutenção, o que facilita a operação por seguirem um padrão comum (parte dessa padronização, dentre outras, foi trazida à forma que as máquinas recebem as instruções, como é o caso das instruções GCODE [REPRAP, 2021]).

O propósito do trabalho que segue é o de descrever a funcionalidade e o processo de implementação de uma dessas máquinas que dão forma ao mundo industrial atual. A saber, a máquina em questão cabe à categoria de automação dos processos mecânicos de manufatura; mais precisamente, à de CNCs, citada acima.

Tais equipamentos permitem precisão e a repetibilidade idêntica durante a fabricação de partes e peças, além de "redução da fadiga do operador, menos erros causados por falha humana e previsibilidade consistente para cada peça trabalhada" [PATEL, 2020]. Dentro dos critérios industriais modernos, como visto, estas são absolutas necessidades.

É importante salientar que máquinas de fabricação automatizada, desde seus primórdios, existem em inúmeros propósitos e propostas diferentes. Em suma, conforme visto, são máquinas controladas por computadores ou partes mecânicas de controle que produzem fisicamente alguma peça ou componente. Temos, por exemplo, as modernas impressoras 3D – possivelmente o exemplo mais popular delas – que produzem objetos projetados a partir de um modelo digital desenvolvido em *softwares* CAD; contudo, não se reduzem a isso. De forma a melhor visualizar o

contexto, é possível categorizar esses equipamentos em dois grandes grupos, as máquinas *subtrativas* e as máquinas *aditivas* [YADAV; SHARMA; ANAND, 2015].

As máquinas subtrativas criam objetos a partir de uma peça sólida de material. Instruções são enviadas para remover parte do material de forma precisa, criando os objetos desejados através de cortes, perfuração, moagem etc. As aditivas, por sua vez, possuem o propósito de depositar material camada a camada, em formas geométricas precisas. Como o nome sugere, esse tipo de manufatura *adiciona* material para criar um objeto físico. Em ambos os tipos, as instruções são enviadas com o suporte e controle de *softwares* do tipo CAM, e pouca ou nenhuma assistência humana é necessária. Soluções CAM possuem o propósito de interpretar comandos de instruções sequenciais e transferi-los a equipamentos físicos, como é o proposto aqui; em suma, transformam instruções lógicas em movimentos geométricos.

Temos vários exemplos de *software* deste tipo[3], sejam os específicos – criados para comandar maquinário específico, e outros, genéricos, ou seja, podem operar quaisquer equipamentos que estejam dentro das especificações de um protocolo. A solução aqui apresentada, ainda que com conjunto de instruções limitado, tem a intenção de operar no segundo escopo – ainda que, a este ponto, somente contemple a ideia do primeiro, ou seja, é puramente compatível somente com o equipamento aqui desenvolvido.

[3] Primordialmente citamos, para fins de exemplo, o *Repetier-Host*, uma solução gratuita para uso que opera genericamente com equipamentos compatíveis com GCODE, além de possuir inúmeras ferramentas para facilitar o processo de manufatura.

Objetivos

Dado o contexto descrito, seguindo o fluxo proposto pelo curso de Análise e Desenvolvimento de Sistemas, o trabalho que segue tem a intenção de projetar uma suíte de *software* do tipo CAM, para controle de um maquinário CNC, além do *firmware* para envio e controle de instruções para motores da máquina em três eixos diferentes.

A ideia geral do *software* a ser desenvolvido é a da prova de conceito para uma solução funcional de envio de instruções a um equipamento capaz de seguir tais instruções com o objetivo final de criar um objeto ou operação sobre um objeto.

Enfim, em termos mais diretos, o objetivo é o de produzir uma impressora CNC que, dada uma imagem digital processada pelos produtos deste projeto, poderá interpretar a informação e transformá-la em uma impressão física bidimensional.

ESTRUTURA DO TRABALHO

Segue a estrutura de desenvolvimento do trabalho aqui proposto:

O Capítulo 1 apresenta a estrutura básica escolhida para execução do projeto.

O Capítulo 2 apresenta o levantamento de requisitos para a realização do projeto.

O Capítulo 3 esboça o projeto em diagramas visuais do produto esperado.

O Capítulo 4 descreve os componentes, descrevendo a elaboração e implementação.

O Capítulo 5 descreve as tecnologias utilizadas para a realização do projeto.

O Capítulo 6 descreve os testes de funcionamento efetuados durante o projeto.

1. DESENVOLVIMENTO DO SISTEMA

A solução proposta foi realizada em três *software* distintos; eles foram, por sua vez, divididos em passos, ou fases, para que pudessem exibir uma maior clareza e maior eficácia no resultado esperado.

Para cada uma das etapas, foi necessário um desenho estrutural do fluxo de informações e de planejamento das comunicações, tanto internamente quanto com o equipamento físico. As próximas seções detalham os processos necessários em cada uma das etapas.

1.1. Fase de planejamento

O desenvolvimento da solução de *software* aqui intencionada partiu de seu projeto teórico, ou seja, da idealização do que deveria ser realizado, uma vez findado o projeto. Essa aproximação ao problema evoluiu gradativamente e foi bastante clareadora, permitindo que fosse compreendido o escopo do problema antes de termos qualquer avanço no tangente à implementação propriamente dita.

Para realizar o projeto, foram levantados os requisitos técnicos necessários, sendo eles a respeito de quais tecnologias deveriam ser utilizadas e quais técnicas deveriam ser aplicadas, bem como sobre quais equipamentos seriam necessários e os conhecimentos específicos de cada tecnologia, e sua intercomunicação; ainda, foi realizado o levantamento dos requisitos formais funcionais e não-funcionais [SOMMERVILLE, 2007] e diagramas explanatórios quanto ao fluxo da implementação (estas informações serão descritas com mais detalhes mais adiante neste documento), conforme foram se apresentando como necessários ao longo do projeto.

Neste contexto, como motivador inicial, elencou-se os padrões utilizados no mercado atualmente, tanto tecnicamente quanto dos equipamentos e soluções que existem com essa mesma proposta. Ora, existem inúmeros níveis de equipamentos quanto ao porte e seus objetivos, visto que equipamentos CNCs são muito abrangentes; deste modo, a escolha foi a de focar o desenvolvimento em equipamentos de pequeno porte, para fabricação e montagem caseiras, que se utilizam de microcontroladores Arduino [ARDUINO, 2021] e motores de passo de baixo custo e potência. Esse tipo de equipamento, apesar de ter um preço reduzido, é muito confiável,

extremamente versátil e preciso, e são ideais para a realização de projetos muito engenhosos, ainda que não possuam porte industrial.[4]

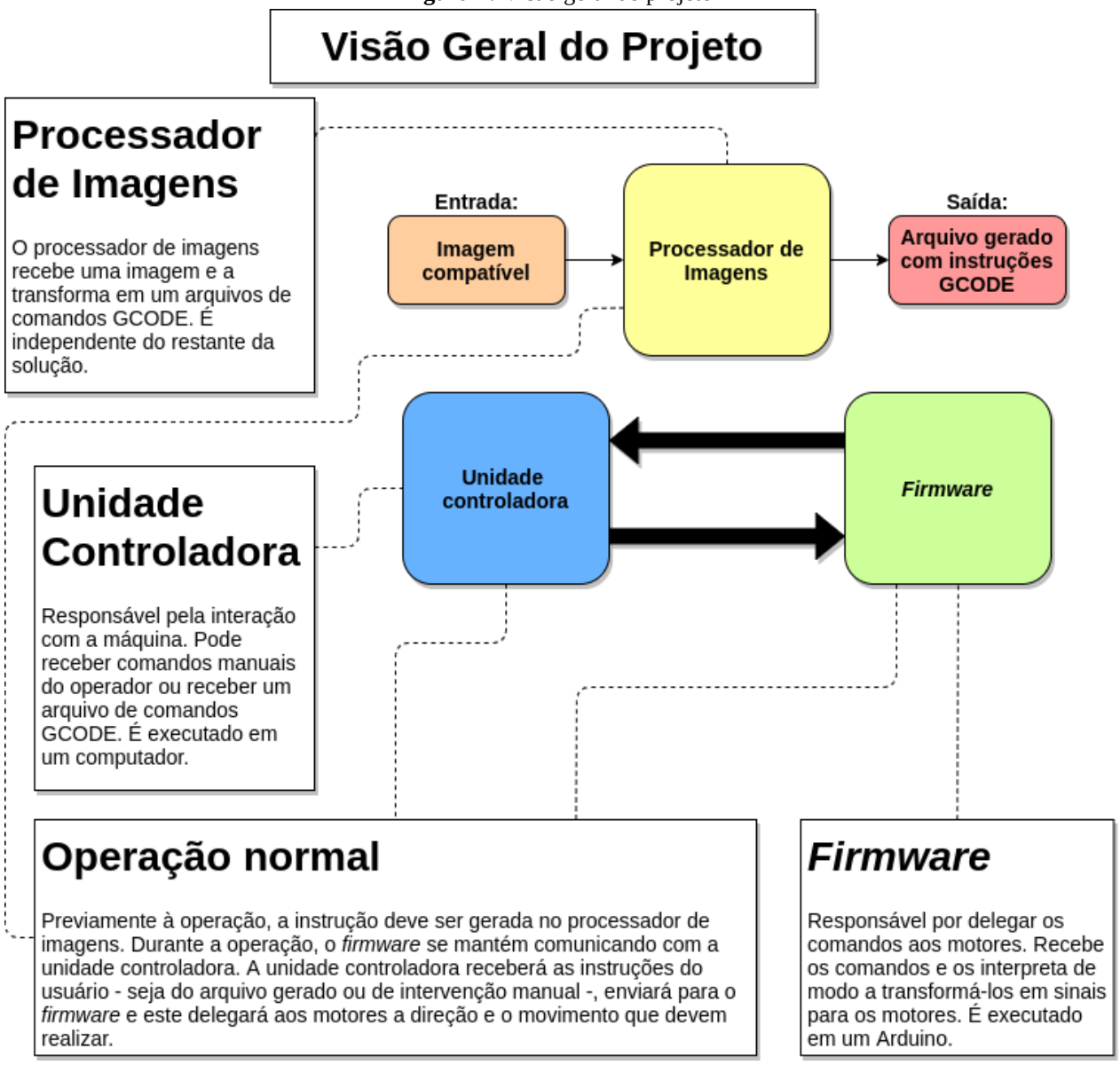
Ainda no contexto de padrões e tecnologias, foram realizados levantamento e estudo de formas de comunicação possíveis entre as partes da solução. É bem comum encontrarmos equipamentos que aceitam carga de instrução – via cartões de memória ou afins acoplados nos próprios equipamentos CNC, dos quais coletam as instruções. Além desses, encontramos equipamentos que aceitam instruções via comunicação imediata, dependendo de outra aplicação que envia em tempo real a instrução ao equipamento durante a sua operação. Em muitos casos, as duas opções estão disponíveis aos operadores. A solução aqui apresentada pretende utilizar a técnica de comunicação em tempo real, enviando comandos através de uma interface *serial* com o equipamento.

1.2. Fase de *design*

Para melhor visualizar a proposta geral, a Figura 1 representa uma visão panorâmica e simplificada das etapas formais e técnicas que deram norte a este projeto:

[4] Parte da escolha do projeto como um todo surge da proposta de ser um projeto de orçamento reduzido; como há a possibilidade de defeitos ou de queima de equipamentos, escolhemos por aqueles de menor valor de peças de reposição. Contudo, o maior motivador foi, de fato, a disponibilidade de um equipamento CNC sem a solução de *firmware* e de controle, e se encontrou à disposição para as empreitadas deste projeto. Em vistas das disciplinas e aos olhos voltados à proposta do curso, isso pareceu permitir um enfoque maior no projeto e desenvolvimento de *software*, ainda que seja voltado a uma solução de *hardware*.

Figura 1: Visão geral do projeto



Fonte: Elaborado pelo autor (2021).

Como citado e representado na Figura 1, a solução foi teorizada para trabalhar com comunicação em tempo real entre dois equipamentos. A saber, de um lado tem-se um microcontrolador Arduino [ARDUINO, 2021] utilizado para o envio de sinais a motores responsáveis pelo movimento do equipamento. O microcontrolador portará um *software* que trabalha como um *firmware* para o maquinário – neste caso, agindo como uma interface entre os motores e a parte lógica das instruções, cujo qual foi desenvolvido especialmente para este projeto.

De outro, foi produzida uma suíte controladora, que interpreta comandos e os traduz em instrução que o *firmware* é capaz de traduzir em pulsos, ou sinais, aos motores. Toda a comunicação entre a controladora e o *firmware* é feita via interface de comunicação *serial* tradicional, em tempo real. A controladora deverá interpretar comandos de modo que o microcontrolador possa instruir a CNC de forma correta; portanto, dado um protocolo definido de comandos, deve instruir movimentos ao equipamento a partir desses comandos.

Por fim, foi produzida uma solução satélite desenvolvida para transformação e interpretação de arquivos de imagens comuns compatíveis para comandos GCODE [REPRAP, 2021]. É importante ressaltar que esta não é essencial para o funcionamento da CNC para comunicação *durante* o processo de impressão, pois, supondo um conjunto de instruções gerados, a exemplo, manualmente, o equipamento também deverá reconhecê-los e operar normalmente, caso estejam corretos de acordo com o protocolo. Contudo, trata-se de um gerador de instruções automatizado. Dada a premissa do projeto, esta solução fez-se necessária por questões de completude e para evitar erros humanos, como é previsto em processos de automação.

1.3. Fase de desenvolvimento

A fase de desenvolvimento apresentou desafios voltados ao conhecimento do *hardware* que foi utilizado para a implementação a respeito do comportamento do microcontrolador, sua linguagem de programação, suas limitações e possibilidades técnicas e os detalhes de operação. Além disso, no que diz respeito à implementação de *software*, muitas técnicas de baixo nível, ou seja, técnicas que envolvem manipulação bruta de dados, como é o caso da comunicação *serial* e operações sobre *bits*, foram necessárias, e em diferentes plataformas.

É importante ressaltar que todo o desenvolvimento teve como premissa a “reinvenção da roda”, ou seja, foi proposta e encorajada a produção de um conjunto de programas que utilizassem um número reduzido de soluções prontas para que a empreitada pudesse fornecer o melhor resultado no tangente ao conhecimento e experimentação do processo de desenvolvimento e implementação.

Nesse contexto, incorporar técnicas voltadas a posicionamento foi essencial, a saber, de transformações matemáticas de instrução em movimento, de manipulação de dados brutos,

comunicação entre sistemas, bem como técnicas para controle fino de envio e recepção de informações, dentre outros, que se pretende esclarecer no decorrer deste trabalho.

1.4. Fase de conclusão

A última etapa do desenvolvimento focou na revisão do projeto como um todo. Nesta etapa houve identificação de um defeito do *firmware*, além de identificar possíveis melhorias de *layout* e apresentação do código e ferramentas utilizadas, baseadas em novas descobertas e estudos sobre as implementações propostas.

Essa revisão foi muito importante para o projeto. Permitiu que fossem corrigidas e/ou mitigadas falhas de implementação e de projeto, que foram identificadas somente após minuciosa inspeção.

Para fortalecer a revisão proposta, foi elaborada uma lista de casos de teste a serem realizados com o equipamento ao final do processo que ajudou a diagnosticar e resolver problemas de operação baseando-se no comportamento esperado do mesmo.

Por fim, após sua conclusão, foi possível identificar a necessidade de uma completa refatoração do *firmware* – que gerou uma segunda versão – resolvendo uma dificuldade de atraso no envio de sinais que ocasionava, por vezes, a dissincronia da comunicação. Além disso, possibilitou elencar possíveis pontos de melhoria na estrutura geral, a ser levada a cabo em futuras iterações.

2. LEVANTAMENTO DE REQUISITOS

Esta Seção trata dos requisitos levantados previamente à elaboração do desenvolvimento proposto. Será dividida em duas tabelas distintas, uma tratando dos requisitos funcionais e outra para os não-funcionais.

Definem-se funcionais aqueles requisitos que "[...] descrevem o que o sistema deve fazer, isto é, definem a funcionalidade desejada do *software*." [SOMMERVILLE, 2007]. Em contrapartida, os requisitos não-funcionais são "[...] aqueles não diretamente relacionados às funções específicas fornecidas pelo sistema." [SOMMERVILLE, 2007].

A Tabela 1 apresenta os requisitos funcionais deste projeto e a Tabela 2 apresenta os requisitos não-funcionais. Denominamos as seguintes siglas para referência nas tabelas:

- **UC**: Unidade Controladora
- **FW**: *Firmware*,
- **PI**: Processador de Imagens.

Tabela 1 – Requisitos Funcionais

Id	Requisito	Categoria	Prioridade
RF001	O PI deve receber um arquivo de imagem compatível	Programação	Essencial
RF002	O PI deve exportar os comandos gerados em um arquivo de extensão .g, em formato de arquivo de texto	Programação	Essencial
RF003	A UC deve se conectar ao FW e monitorar a conexão a todo momento e, em caso de perda de conexão, deve suspender todas as atividades e ativar modo de emergência	Programação	Essencial
RF004	A UC deve ter modos de parada de emergência que suspendem toda a operação	Programação	Essencial
RF005	A UC deve permitir escolha de modo manual ou automático	Programação e *design*	Importante
RF006	No modo manual, a UC deve permitir a entrada de um comando manualmente reportá-lo como válido ou não	Programação	Essencial

Id	Requisito	Categoria	Prioridade
RF007	No modo manual, a UC, em caso de comando válido, deve enviar o comando ao FW e aguardar novo comando	Programação	Essencial
RF008	No modo automático, a UC deve solicitar um arquivo de instruções .g e validá-lo, solicitando um novo arquivo caso seja inválido	Programação	Essencial
RF009	No modo automático, a UC deve enviar sequencialmente os comandos listados no arquivo de entrada	Programação	Essencial
RF010	Antes de cada comando enviado, verificar se o FW reporta algum motivo para parada e ativar modo de emergência ou alerta se o FW solicitar	Programação	Essencial
RF011	O FW deve verificar os *endstops* a cada ciclo e notificar a situação à UC, estejam eles ativados ou não	Programação	Essencial
RF012	O FW deve verificar se a UC enviou novo comando de parada a cada ciclo	Programação	Essencial
RF013	O FW deve interromper o ciclo normal e iniciar o modo de parada se qualquer motivo para parada for acionado localmente ou reportado, seja da parte do FW, seja da parte da UC, e aguardar instruções para continuar	Programação	Essencial
RF014	O FW, caso tudo esteja OK, deve enviar o sinal aos motores para mover a mesa	Programação	Essencial
RF015	A UC deve informar a sua situação e a situação do FW em tela: progresso, posição atual, comandos sendo executados, etc.	Usabilidade	Essencial
RF016	A UC deve exibir os erros em tela de uma forma clara sobre qualquer razão de erro e registrar erros ocorridos em um arquivo de *log*	Confiabilidade	Essencial
RF017	A UC deve reportar a todo momento a situação ao usuário, tais como se a máquina está parada, se movendo, em modo de emergência, etc.	Usabilidade	Essencial

Fonte: elaborado pelo autor (2021).

Tabela 2 – Requisitos Não-Funcionais			
Id	**Requisito**	**Categoria**	**Prioridade**
RNF001	Usar a linguagem de programação C# para desenvolvimento *desktop*	*Hardware* e *Software*	Desejável
RNF002	Os *softwares* para *desktop* devem rodar em *Windows* e *Linux*	Padrões	Essencial
RNF003	O FW deve ser compatível com a plataforma Arduino	Padrões	Essencial
RNF004	O PI deve independer da controladora ou do FW	Padrões	Importante
RNF005	Comandos para o FW devem ser gerados em formato GCODE pelo PI.	Programação	Essencial
RNF006	A interface do PI deve ser em console e exibir texto de ajuda ao usuário, com exemplos de operação	Usabilidade	Desejável
RNF007	A interface do PI deve mostrar as etapas do processo enquanto o realiza a conversão da imagem selecionada	Usabilidade	Importante
RNF008	O PI deve permitir escolher o nome do arquivo .g de saída	Usabilidade	Desejável
RNF009	A interface da UC deve ser em console e informar ajuda completa e lista de comandos ao operador	Usabilidade	Desejável
RNF010	Toda a solução deve ser intuitiva e de fácil operação	Usabilidade	Importante
RNF007	A UC deve informar a sua situação e a situação do FW em tela: progresso, posição atual, comandos sendo executados, etc.	Usabilidade	Essencial
RNF008	A UC deve exibir os erros em tela de uma forma clara sobre qualquer razão de erro e registrar erros ocorridos em um arquivo de *log*	Confiabilidade	Essencial
RNF009	A UC deve reportar a todo momento a situação ao usuário, tais como se a máquina está parada, se movendo, em modo de emergência, etc.	Usabilidade	Essencial

Fonte: elaborado pelo autor (2021).

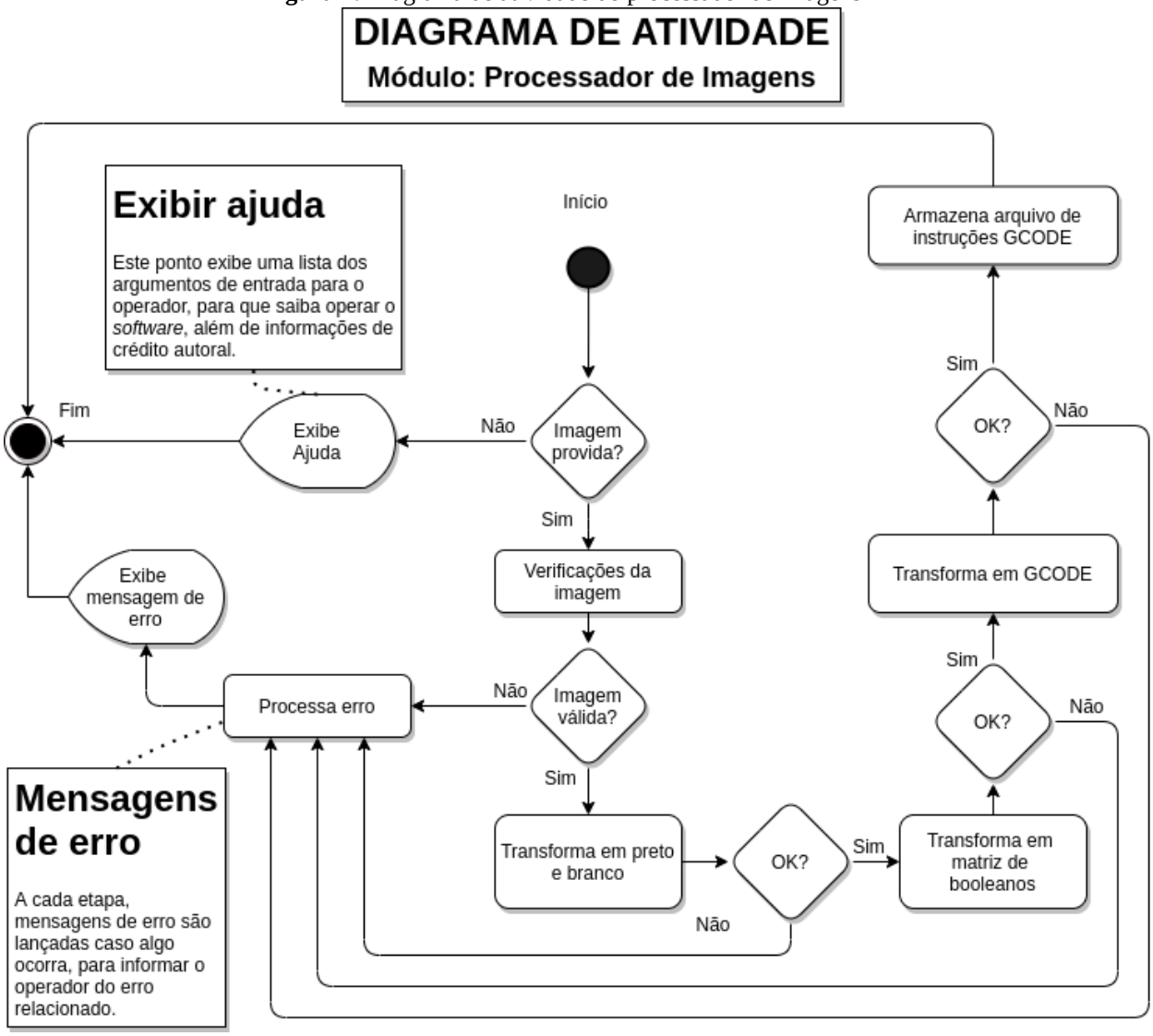
3. DIAGRAMAS

3.1. Diagramas de atividade

Os diagramas das Figuras 2, 3 e 4 têm a intenção de fornecer uma visão mais detalhada dos processos que cada parte do *software* possui internamente. A ideia principal desses diagramas é a familiarização do leitor com a estrutura utilizada para que o objetivo final do projeto fosse alcançado, e explicitar as tomadas de decisão que foram necessárias para atingir tal objetivo de maneira eficiente.

3.1.1. Diagrama de atividades do processador de imagens

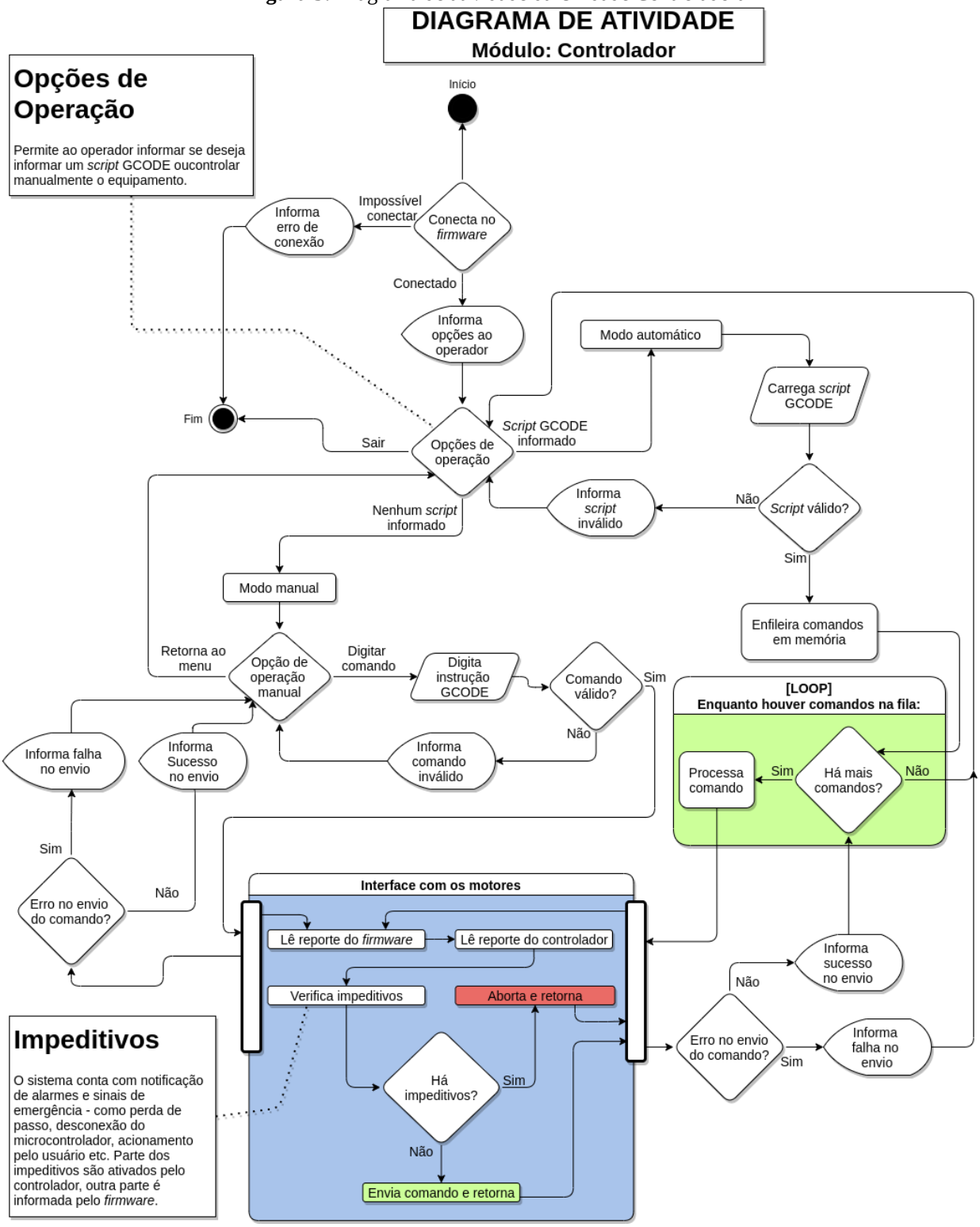
Figura 2: Diagrama de atividade do processador de imagens



Fonte: elaborado pelo autor (2021).

3.1.2. Diagrama de atividades da Unidade Controladora

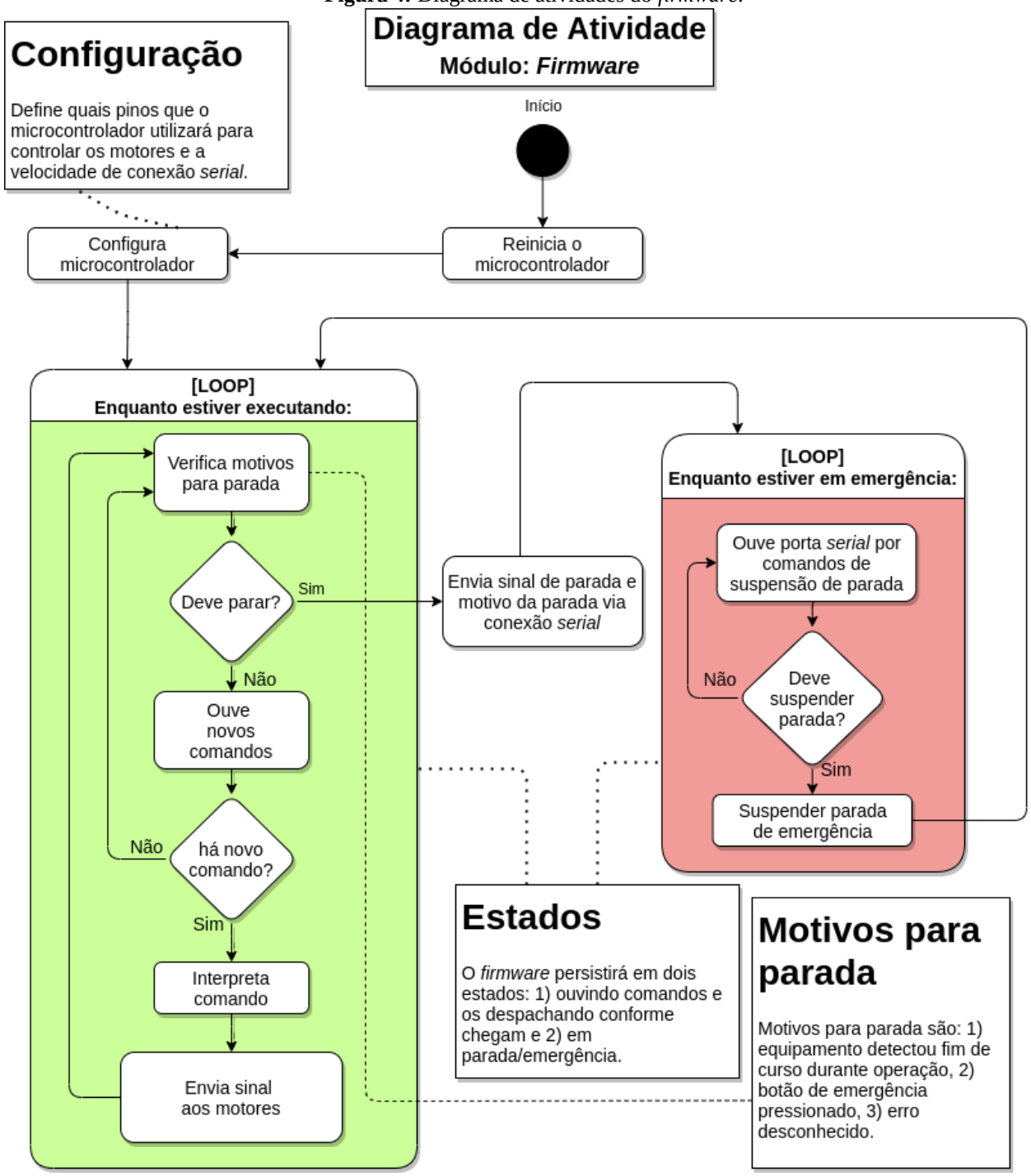
Figura 3: Diagrama de atividade da Unidade Controladora



Fonte: elaborado pelo autor (2021).

3.1.3. Diagrama de atividades do *firmware*

Figura 4: Diagrama de atividades do *firmware*.



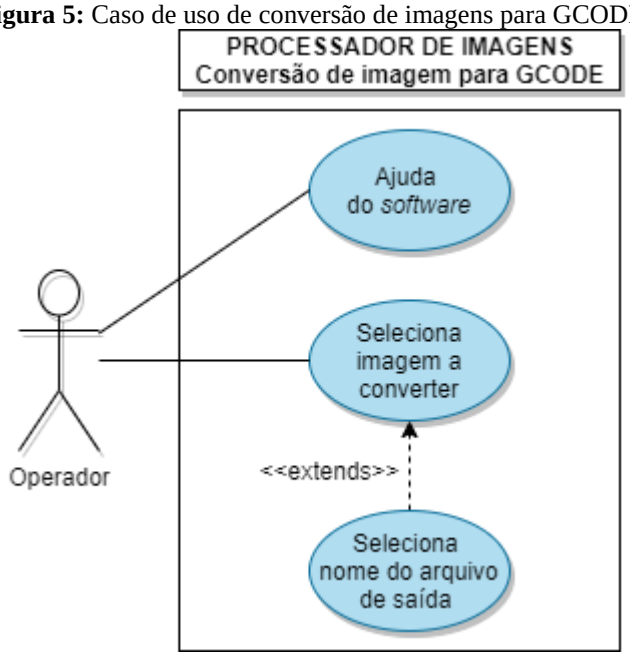
Fonte: elaborado pelo autor (2021).

3.2. Diagramas de caso de uso

3.2.1. Caso de uso: Operação do Conversor de Imagens

A Figura 5 representa os casos de uso [OMG, 2017] para o operador durante o processo de conversão de uma imagem para instruções GCODE.

Figura 5: Caso de uso de conversão de imagens para GCODE

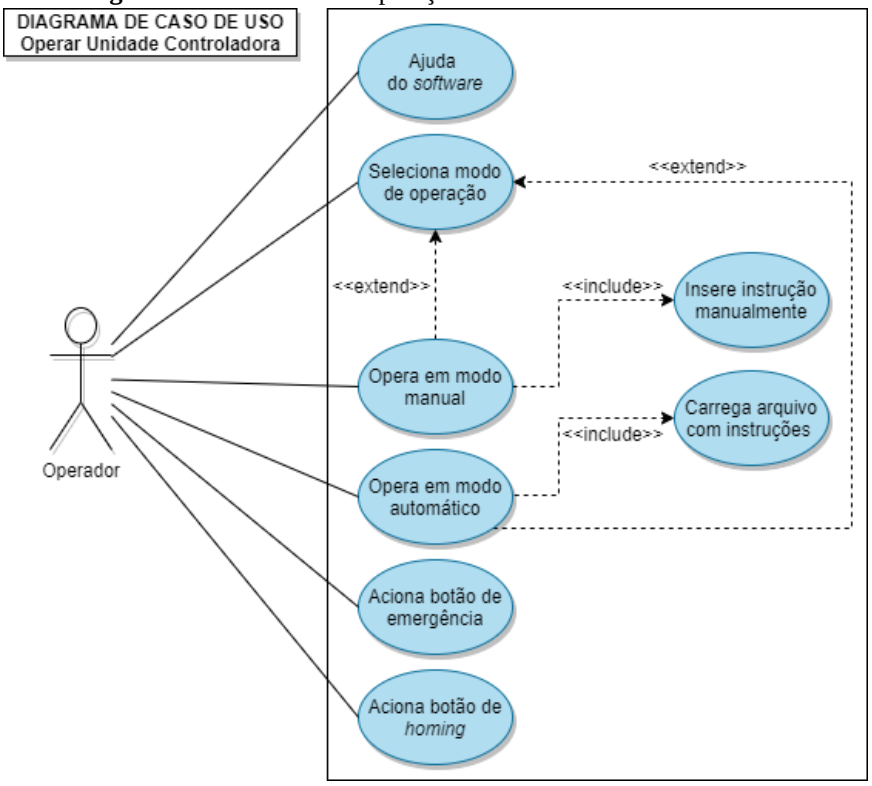


Fonte: elaborado pelo autor (2021).

3.2.2. Caso de uso: Operação Unidade Controladora

A Figura 6 representa os casos de uso para o operador durante o processo de operação da unidade controladora.

Figura 6: Caso de uso de operação da Unidade Controladora



Fonte: elaborado pelo autor (2021).

3.3. Documentação dos casos de uso

As Tabelas 3 e 4 descrevem mais detalhadamente o que é representado pelos casos de uso apresentados nas Figuras 5 e 6, respectivamente.

Tabela 3 – Caso de Uso "Operação do Conversor de Imagens"	
Nome do caso de uso	Operação do Conversor de Imagens
Atores envolvidos	Operador
Objetivo	Este caso de uso tem a intenção de descrever casos de ação do operador ao preparar uma imagem para impressão.
Prioridade de desenvolvimento	Importante
Ações do Ator	**Ações do sistema**
O operador solicita ajuda do sistema.	O sistema informa o manual para o operador.
O operador informa a imagem a converter ao sistema e não especifica um nome de arquivo de saída.	O sistema converte a imagem para GCODE e salva o arquivo com um nome de arquivo padrão.
O operador informa a imagem a converter ao sistema especifica um nome de arquivo de saída.	O sistema converte a imagem para GCODE e salva o arquivo com o nome que o operador escolher.

Fonte: elaborado pelo autor (2021).

Tabela 4 – Caso de Uso "Operação da Unidade Controladora"	
Nome do caso de uso	Operação da Unidade Controladora
Atores envolvidos	Operador
Objetivo	Este caso de uso tem a intenção de descrever casos de ação do operador ao preparar uma imagem para impressão.
Prioridade do desenvolvimento	Essencial
Ações do Ator	**Ações do sistema**
O operador solicita ajuda do sistema.	O sistema informa o manual ao operador.
O operador seleciona o modo de operação manual.	O sistema solicita que o operador informe o comando GCODE a executar.
O operador informa um código GCODE válido.	O sistema envia ao maquinário a instrução para movimento e retorna à interface de entrada de comandos.
O operador informa um código GCODE inválido.	O sistema informa que a instrução é inválida e não envia a instrução ao maquinário. Retorna à interface de entrada de comandos.
O operador seleciona o modo de operação automática.	O sistema solicita que o usuário informe o arquivo contendo as instruções.
O operador informa um arquivo inválido.	O sistema informa que o arquivo é inválido e retorna à interface de seleção de arquivos.
O operador informa um arquivo válido.	O sistema carrega o arquivo de instruções e passa a enviar um comando por vez para o maquinário.
O operador pressiona o botão de parada de emergência.	O sistema aciona o modo emergência no ambiente e envia sinal para o equipamento para interromper quaisquer operações de movimento que estejam executando.
O operador pressiona o botão de *homing*.	O sistema suspende todos processos de impressão e move a ferramenta de impressão para o ponto X=0, Y=0.

Fonte: elaborado pelo autor (2021).

4. DESCRIÇÃO DOS COMPONENTES

Esta Seção tem a missão de clarificar como os componentes foram arquitetados e implementados. Segue, pois, uma descrição técnica e processual destes para definição de protocolos e modos de operação do sistema como um todo. Aqui também se intenciona a unificação e descrição da comunicação entre as partes, tanto no que diz respeito ao *software* quanto a respeito da interação com o *hardware* para qual se destina.

4.1. Conversor de imagens

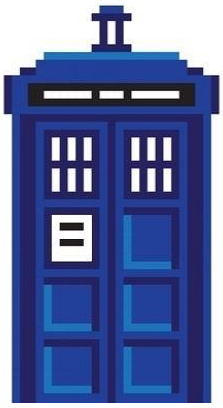
A primeira das soluções a ser apresentada é o Processador de Imagens. O conversor recebe uma imagem compatível, conforme requisitos da aplicação, e o objetivo é o de transformar essa imagem em um conjunto de instruções compreensíveis pela máquina, sendo, no presente caso, um conjunto de instruções GCODE.

Para isso, o *software* utiliza-se da característica inerente às imagens digitais em serem formadas por um conjunto de colunas e linhas de *pixel*, que são informações de cada ponto da imagem. Ou seja, em suma, deverá tratar a imagem como uma *matriz de pontos de n colunas e n linhas*. Como a máquina projetada também é instruída por comandos interpretados em linhas e colunas, ou seja, por uma matriz de pontos, a ideia principal consiste em traduzir a informação que compõe a imagem em um conjunto de comandos compatíveis com o *firmware*.

O primeiro passo consiste em tratar os pontos individualmente. Isso será realizado passando por cada ponto da imagem original para transformá-los em preto e branco. Essa técnica foi utilizada pois permite que a imagem possa ter apenas informações *binárias*, ou seja, existam somente dois estados para representar para a máquina: ou o *pixel* é preto, ou é branco. Essa etapa é importante, pois a máquina CNC também possui apenas dois estados de impressão no que tange seus movimentos, que podem ser resumidos da seguinte forma: ou ela está *0) em movimento sem traçar uma linha*, ou está *1) em movimento traçando uma linha*.

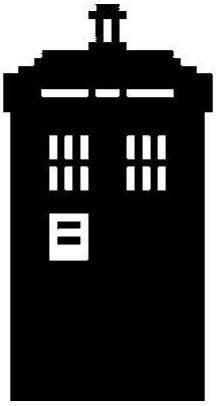
Abaixo, um exemplo de uma conversão real realizada pelo *software*:

Figura 7: Imagem para testes de conversão



Fonte: elaborado pelo autor (2021).

Figura 8: Imagem convertida em preto e branco



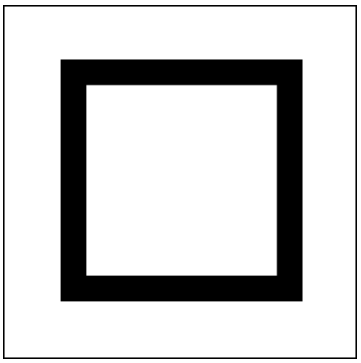
Fonte: elaborado pelo autor (2021).

Nesse sentido, a decisão tomada para o protocolo de conversão foi a de que *pixels* brancos devem representar *0) em movimento sem traçar uma linha* e que os *pixels* pretos deveriam representar *1) em movimento traçando uma linha*.

Uma vez definido o protocolo, será montada uma representação manipulável desses pontos da imagem. Para isso, converte-se essas informações de linhas e colunas binárias da imagem preto e branca resultante do processo acima para uma matriz *booleana*, ou seja, cada *pixel* poderá ser representado ou por 1 ou por 0. Para melhor visualização da ideia aqui apresentada, o que segue é um exemplo prático.

Dado o princípio de que, primitivamente, tanto a impressora projetada quanto as imagens que deverão ser processadas possuem a mesma estrutura matemática, ou seja, derivada de uma estrutura matricial, por serem de mesma natureza, o que será necessário de ser feito é apenas a interpretação de um escopo para outro. Com isso em mente, suponha-se uma imagem de 10 pontos de altura por 10 pontos de largura com um quadrado representado em seu interior, conforme representa a Figura 9:

Figura 9: Representação de um quadrado para conversão



Fonte: elaborado pelo autor (2021).

O objetivo aqui será o de transformar a informação visual da Figura 9 em informações binárias, ou seja, *ligado* ou *desligado*. Para os fins de exemplo de conversão, assumir que a Figura 9 tem o tamanho de 10x10 *pixels*. O *software* passará por cada ponto da imagem, que é uma matriz de *pixels*, verificando a informação que o ponto armazena e, com base nisso, será realizada uma tradução para os propósitos da máquina: será criada uma nova matriz e, se no ponto (0, 0) da imagem original existir a cor branca[5], o ponto (0, 0) será representado na matriz que criamos com o valor 0. Caso o valor seja a cor preta[6], o ponto (0, 0) será representado com o valor 1. Este processo deve se repetir para as posições (0, 1), (0, 2), (0, 3) e assim sucessivamente, até que todos os pontos na imagem original sejam processados e armazenados na nova matriz, de acordo com o algoritmo – ou seja, até chegarmos, neste caso, no ponto (9, 9).

Neste caso, a matriz representa uma imagem quadrada com 100 pontos possíveis, e cada número na imagem representa uma posição (x, y) específica em um plano cartesiano que poderá ser mapeada. Com isso, é possível representar essa matriz por (10, 10), ou seja, 10 colunas por 10 linhas. A imagem representada na Figura 9 gerará, portanto, uma matriz com alguns pontos representados pelo valor 1 e outros representados pelo valor 0, de acordo com as informações encontradas.

---

[5] Nota técnica: a cor branca no esquema RGB [*red-green-blue*, (vernelho-verde-azul)] é comumente representada por um vetor de 3 *bytes*. Como cada *byte* neste vetor que representa um ponto significa a intensidade daquela cor no ponto em questão, variando de 0, menor valor, até 255, maior valor, e a cor branca é o máximo de intensidade em todas as cores, se o ponto encontrado possuir o valor hexadecimal 0xFF, ou 255, nas três posições deste vetor, a intensidade do vermelho (R), verde (G) e o azul (B) estarão em seu máximo – ou seja, o ponto da imagem será representado pela cor branca. O *software* interpreta o *pixel*, portanto, desta forma: verifica se o valor contido representa um ponto preto ou branco – ambos, *numericamente*.

[6] Vetor de 3 *bytes* representando a cor preta HEX = [0x00, 0x00, 0x00] (ver nota 5).

Este exemplo pode ser representado com a matriz bidimensional 10x10 apresentada na Figura 10, contendo a instrução virtual de um traçado conforme tradução efetuada a partir da Figura 9 (o traçado original a representar foi realçado na cor amarela para melhor visualização):

Figura 10: Representação matricial da Figura 9

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fonte: elaborado pelo autor (2021).

O processo descrito oferece um exemplo simplista, porém eficaz, para a demonstração de como a representação *booleana*, ou seja, como um conjunto de instruções *binárias*, é traduzida para uma representação bidimensional de uma imagem. Para fins práticos, é tudo que a máquina precisa, pois essas instruções podem ser utilizadas para orientação sobre um plano cartesiano (x, y), que a máquina poderá processar.

Conforme definido, o equipamento possui dois estados: 0) *em movimento sem traçar uma linha* ou 1) *movimento traçando uma linha*. O equipamento sempre deverá se mover por *todos* os pontos da matriz resultante (no caso de nosso exemplo são os 100, mas variará de acordo com as dimensões da imagem processada), e também que, em alguns pontos, deverá mover-se e ativar a ferramenta que deverá marcar o material; em outros, deverá desativá-la e apenas mover-se por aquele ponto sem marcá-lo; por fim, utilizando-se do protocolo adotado: 1 na matriz resultante será indicador de que a máquina deverá ativar a ferramenta do equipamento e 0 será o indicador de que a máquina deverá retraí-la.

Com o conjunto de regras definido, é possível traduzir as três primeiras linhas da imagem representada na Figura 9 em instruções lógicas proposicionais, da seguinte forma:

1. Vá até o ponto (x0, y0).
2. Desligue a ferramenta.

3. Mova-se até (x9, y0).
4. Desligue a ferramenta.
5. Vá até o ponto (x0, y1).
6. Desligue a ferramenta.
7. Mova-se até (x9, y1).
8. Desligue a ferramenta.
9. Vá até o ponto (x0, y2).
10. Desligue a ferramenta.
11. Mova-se até (x2, y2).
12. Ligue a ferramenta.
13. Mova-se até o ponto (x7, y2).
14. Desligue a ferramenta.
15. Mova-se até o ponto (x9, y2).
16. (...)

Do modo apresentado, o resultado é um conjunto preciso de instruções escritas que definem movimentos que um interlocutor deverá fazer. Porém, a unidade controladora é preparada para receber e interpretar essas instruções através de comandos específicos, a partir do protocolo GCODE, em vistas da padronização que objetivamos no projeto quanto a esta linguagem. Portanto, a próxima etapa consiste na transformação desse conjunto de instruções lógicas proposicionais nos comandos do padrão, ou protocolo, escolhido; conforme acima escolhemos o GCODE, que permite trazermos esse tipo de instrução lógica proposicional para o escopo de instrução de movimento. Assim, seguindo as mesmas três linhas da matriz apresentada, os comandos deverão ficar da seguinte forma:

1. G1 X0 Y0
2. M5
3. G1 X9 YO
4. M5
5. G1 X0 Y0
6. M5
7. G1 X0 Y0
8. M5
9. G1 X0 Y0
10. M5
11. G1 X0 Y0
12. M3
13. G1 X0 Y0
14. M5
15. G1 X0 Y0
16. (...)

O código resultante está de acordo com as especificações do protocolo, ou linguagem de programação, GCODE, seguindo a especificação de máquinas utilizadas para cortes de material a *laser*, ou seja, operando com movimentos até determinado ponto e, para o eixo Z, somente operando com as noções de ligado e desligado. Sendo assim, ao final do mapeamento completo do arquivo de imagem, podemos coletar todas as instruções e as armazenar em um arquivo, para ser lido pela solução que será responsável pelo envio destas instruções para o equipamento, ou seja, a Unidade Controladora, a ser tratada no Seção 5.2. Nesta, será explicitada a mecânica por trás da Unidade Controladora, que deve consumir o arquivo aqui produzido.

4.2. Unidade Controladora

A segunda solução apresentada é a Unidade Controladora. Ela é similar a um *software* CAM, tendo como principal funcionalidade a interpretação de códigos GCODE para que o usuário possa produzir um objeto a partir das instruções inseridas. Ela, após receber um conjunto de instruções, deve traduzi-las a pacotes de *bits* a serem transmitidos para o equipamento. Esta solução foi desenvolvida especialmente para este projeto. Toda a estrutura de transmissão é definida em um protocolo próprio, elaborado em conjunto com o *firmware*, também definido para este projeto, a ser tratado na Seção 5.3.

Primeiramente, o *software* possui duas formas de operação. Ele poderá tanto operar com um conjunto definido de instruções, como é o caso das instruções geradas a partir da solução apresentada na Seção 5.1, ou operação manual. A primeira é automatizada e segue o arquivo gerado como um roteiro, ou *script*; a segunda, permite inserção de instruções manuais de operação individualmente, para fins de reposicionamento ou manutenção no equipamento.

Contudo, ambas utilizam a mesma mecânica; a diferença está somente na forma que a instrução é enviada: manualmente, o *software* pergunta ao usuário operador qual o próximo comando GCODE deve interpretar, e o usuário deverá digitá-lo corretamente; ao enviar o comando, somente o comando digitado será enviado. Quando está no modo automatizado, instruções GCODE são carregadas sequencialmente a partir do arquivo contendo as instruções e serão enviadas até o *software* chegar ao final da lista de comandos que carregou.

O *software* interpreta cada comando válido como uma forma específica de movimento. Como a máquina CNC especificada possui três eixos (X, Y e Z), cada um possuindo um motor de passo, é necessário que cada motor seja instruído individualmente a cada ciclo.

Os motores de passo possuem a característica de conseguirem definir quantidades precisamente a cada movimento que fazem. Isso permite que seja possível prever quantos pulsos são necessários para que o motor se mova uma quantidade definida de milímetros, de acordo com as dimensões dos componentes físicos da máquina, adaptando os cálculos realizados pelo *software* com as novas dimensões. Para fins práticos, o motor *sabe* o quanto se move: ou seja, podemos definir quantos passos são necessários para mover um milímetro, por exemplo.

É importante ressaltar que a definição de movimento com base nas dimensões físicas do equipamento CNC é feita nas configurações do *software*, basicamente permitindo adaptabilidade da solução para diferentes configurações – tanto para caso de troca de peças com especificação diferente (espessura maior dos eixos de movimento, roscagem mais espaçada que necessite mais passos para mover a mesma quantidade de milímetros etc.) quanto para portabilidade para outros equipamentos de especificações diferentes. Para fins desta solução, por se tratar do desenvolvimento do *software* controlador e não do equipamento controlado, ou seja, da montagem física de uma CNC em particular, as especificações realizadas para o maquinário que utilizamos para teste não serão definidas, portanto; o escopo estará limitado ao reforço da possibilidade de configuração do movimento para diferentes equipamentos.

Quanto a seu funcionamento, o *software* controlador inicia seu posicionamento lógico na posição X=0 e Y=0. Para certificar que o equipamento também está na posição X=0 e Y=0, ou seja, para que o posicionamento físico esteja sincronizado de acordo com o posicionamento lógico, o *software* envia um comando de *homing*. Este comando é utilizado para adquirir o absoluto zero do equipamento, ou ponto de partida, enviando movimento negativo, ou seja, que diminui o posicionamento nos eixos X e Y indefinidamente até encontrar o limite mínimo do curso dos eixos. O equipamento faz isso por se utilizar de botões de fim de curso em cada extremo do eixo, chamados de *endstops*: quando o equipamento tocar nos *endstops*, o *software* vai compreender que chegou até o ponto 0 daquele eixo, e armazenará esta informação.

Uma vez que as configurações estejam corretas, ou seja, que o movimento físico esteja de acordo com a configuração lógica do equipamento, este pode agora se orientar com base nos eixos.

Para cada nova instrução, o *software* armazena em memória o local onde a última instrução posicionou os eixos. Portanto, como a primeira instrução enviada é a de posicionamento X=0 e Y=0, temos essas coordenadas armazenadas para futura referência.

A exemplo: supondo que, após o processo de *homing*, ou seja, depois de o equipamento se posicionar em X=0 e Y=0, o *software* controlador receba a instrução "G1 X3 Y10" (leia-se, "mova-se (G1) para a posição 3 no eixo x (X3) e para a posição 10 no eixo y (Y10), de acordo com as especificações GCODE)", este deverá instruir ao equipamento a mover-se 3 unidades no eixo X e 10 no eixo X, e armazenar em memória, após o movimento realizado, que a posição atual da ponta que contém a ferramenta para marcar a mesa física está em (X=3, Y=10). Se o próximo comando de movimento informar a instrução "G1 X2 Y12", o *software* deverá saber que, estando no X=3 e Y=10, deverá caminhar –1 unidade no eixo X e 2 unidades no eixo Y, e armazenar que se encontra na posição X=2 e Y=12 em memória como posição de referência para realizar o próximo movimento.

Assim, toda instrução nova enviada posiciona a ponta do equipamento baseando-se nos pontos X=0 e Y=0 originais. Esse posicionamento serve tanto para orientação sobre a mesa quanto para detecção de movimento que esteja fora da área de operação do equipamento, para fins de segurança. Assim, caso um comando enviado exceda o tamanho da área de operação que force o equipamento a operar além dos limites, por segurança, lança um sinal de emergência e suspende a operação antes mesmo de enviar o comando.

Para cada comando recebido, o *software* valida a entrada e verifica se é um comando válido de acordo com o protocolo. Em seguida, com base na posição atual do eixo impressor, calcula a quantidade de movimento necessário para se deslocar da posição atual para a posição de destino, fazendo a diferença matemática entre eles.

Por fim, a cada ciclo de envio de comandos, a Unidade Controladora mantém-se ouvindo por mensagens enviadas pelo *firmware*. O objetivo desta etapa é a de saber, a cada envio de mensagem, se houve algum comando de parada por algum erro ocorrido, não na transmissão da instrução, mas na execução da tarefa após o recebimento. Isso permite que a controladora informe ao usuário que o equipamento parou por conta de um eixo que acionou o *endstop* de uma maneira não prevista ou qualquer outro erro que faça com que o posicionamento da mesa esteja dessincronizado com o posicionamento armazenado em memória da unidade controladora. Além

disso, permite que o *firmware* envie mensagens customizadas para a unidade controladora, como mensagens de ligamento do equipamento, início e fim de movimento, créditos, dentre outras.

Por fim, todo esse processo de controle de movimento é feito *antes* do envio dos sinais ao *firmware*. Como é possível notar, a unidade controladora detém todo o processamento dinâmico da solução. Assim, uma vez que a decisão do movimento é tomada, a unidade controladora envia o comando de acordo com a decisão via conexão *serial* utilizando o protocolo estabelecido para o projeto, para que o microcontrolador portando o referenciado *firmware* possa distribuir o sinal entre os eixos, cabendo a este apenas a distribuição dos sinais. Os detalhes do protocolo serão discutidos na Seção 5.3.

4.3. Firmware

O *firmware* é a parte da solução que é executada na placa controladora do CNC, nesse caso um Arduino UNO. Este é responsável pela correta manipulação dos motores. A saber, a placa controladora foi escolhida por possuir comunicação *serial* via USB e pinos suficientes para a comunicação com 3 motores, bem como com os sensores de fim de curso (chamados comumente de *endstops*). Mais detalhes serão descritos sobre a especificação do *hardware* utilizado na Seção 5.4.

Em uma visão geral, a Unidade Controladora (conforme descrita na Seção 5.2), após definir qual movimento deverá ser realizado, fica responsável pelo envio de sinais que devem ser traduzidos em movimento. Porém, para fins práticos, tudo que a unidade controladora envia são *bytes* sequenciais através da porta *serial*. Na outra ponta desta comunicação existe um microcontrolador que recebe o sinal e interpreta um *byte* por vez. A única definição faltante é como esses *bytes* são montados para que possamos ensinar ambas as pontas como interpretar a informação transmitida.

Como para qualquer comunicação, é necessário um protocolo para que ela seja efetivamente compreendida por ambas as partes. Portanto, com base no que foi necessário para que as instruções fossem passadas para os motores, estabeleceu-se que, para cada ciclo, um par de instruções deverá ser enviada para cada motor, individualmente, dadas as especificações técnicas do equipamento.

Para melhor domínio das citadas especificações, foram levantados os detalhes de como motores de passo trabalham para que possamos instruí-los corretamente com movimentos: uma vez que temos 3 motores no equipamento, um para o eixo X, um para o eixo Y e outro para o eixo Z, e cada motor, por unidade de movimento – este que chamaremos doravante de *passo*[7] –, deve receber dois sinais, um que indica a *direção* do movimento e outro que indica o *sinal* do movimento propriamente dito, ou seja, o sinal gatilho para mover na direção indicada pelo sinal anterior. O processo descrito deve, então, produzir *um* passo por motor.

Assim, podemos definir que, para uma instrução que consiga, num mesmo ciclo de envio de sinais, mover um passo em cada um dos três motores, é necessário que haja ao menos seis posições, ou três pares de posições, para cada instrução individual que contenha a informação sobre o movimento e a direção intencionados.

Para além do protocolo, pensando ainda em ciclos de envio de sinais, houve a necessidade de compreensão e implementação técnica dos tempos de envio dos sinais para cada motor. Como indicado, esta solução pretende ser genérica; por isso, foi necessário implementar configurações para pequenos atrasos entre comandos. A razão disso é que, segundo especificações técnicas dos motores de passo, existe um tempo que é preciso aguardar entre o envio de qual direção o motor deve se mover e o sinal de movimento propriamente dito. Esse tempo, caso não respeitado, pode fazer com que o equipamento perca algum passo enviado, o que faz com que a sincronia entre o *software* e o *hardware* seja perdida, ocasionando na perda de um trabalho de impressão.

Com isso em vista, para facilitar a comunicação, decidiu-se convencionar o envio de um *byte* por ciclo de movimento. Também se convencionou o envio de *um byte por instrução*, utilizando operações sobre os seis primeiros *bits* dentro do *byte* enviado, partindo do *bit* menos significativo para o mais significativo – ou seja, da direita para a esquerda do vetor contendo 8 *bits*.

[7] Um passo indica uma unidade de movimento para motores de passo. O movimento deste tipo de motor não é constante como num motor tradicional; ao invés disso, cada sinal enviado produz um passo, e o movimento cessa em sequência. Esse movimento é quantificável precisamente, pois a distância percorrida por um passo é precisamente a mesma percorrida pelo passo anterior. Assim, é possível prever a quantidade de passos necessários para um movimento preciso, e essa propriedade é o que determina quantos sinais de movimento devem ser enviados para mover-se com precisão uma certa quantidade de milímetros. Essa característica destes motores é o que o faz essencial para equipamentos CNC ou quaisquer outros que exijam movimentos precisos e repetíveis.

De modo a clarificar este conceito, é considerado este exemplo: supondo que o seguinte comando deva ser enviado:

A. O eixo X deve realizar movimento positivo (ou seja, acrescer uma posição na matriz, por exemplo, mover-se de X=11 para X=12);

B. O eixo Y deve realizar movimento negativo (subtrair uma posição na matriz, por exemplo, mover-se de Y=3 para Y=2);

C. O eixo Z deverá ficar parado (ou seja, deverá ficar na posição que estiver).

Assim, é possível definir que o comando lógico que deverá ser enviado, descrito acima, será o que segue, conforme Tabela 5:

Tabela 5: Estrutura de um comando

Motor	Direção	Mover-se?
Eixo X	(+)	Sim
Eixo Y	(-)	Sim
Eixo Z	(+ ou –)[8]	Não

Fonte: elaborado pelo autor (2021).

Tendo essa coleção de informações, resta somente definir o valor dos pinos e os seus significados: o valor (0) representa sinal desligado ou sinal negativo, e o valor (1) representa sinal ligado ou positivo. Podemos então definir o protocolo da seguinte forma:

- ***bit* posição 0:** eixo X, sinal de direção.
- ***bit* posição 1:** eixo X, sinal de passo.
- ***bit* posição 2:** eixo Y, sinal de direção.
- ***bit* posição 3:** eixo Y, sinal de passo.
- ***bit* posição 4:** eixo Z, sinal de direção.
- ***bit* posição 5:** eixo Z, sinal de passo.
- ***bit* posição 6:** nenhuma instrução.
- ***bit* posição 7:** nenhuma instrução.

[8] É importante ressaltar que, considerando o exemplo proposto, nenhum sinal de *movimento será enviado;* assim, qual instrução para direção para o motor do eixo Z torna-se irrelevante no caso deste exemplo.

Seguindo esta definição, é possível montar um *byte* da seguinte forma:

***byte** instruction = 0b00011011*[9]

Assim, quando o *firmware* recebe um *byte*, ele interpreta cada uma das posições de *bit* dele e sabe o que fazer a cada ciclo de instrução, ou seja, quais pinos deve ligar e quais deve desligar fisicamente no microcontrolador. O equipamento opera, portanto, como um roteador para as instruções: distribui os sinais corretamente para os motores de acordo com a informação que recebe. Do ponto de vista de *hardware*, isso significa enviar sinais de passo e direção para o motor, e o motor deve se mover. Então, como cada sinal que o *firmware* recebe, ou seja, cada pacote de 8 *bits*, ou 1 *byte*, transmitido, o sinal de movimento é realizado – contanto que o *endstop* não esteja ativado.

Essa parte é bastante importante: a cada instrução que recebe, antes de enviá-la para os pinos e realizar o movimento, o *firmware* verifica se os interruptores de fim de curso dos eixos estão ativados. Estarem ativados significa que o eixo está em seu limite físico em uma das extremidades, e se continuar a se mover naquela direção irá romper com o limite do equipamento, ocasionando quebras ou até mesmo perigo ao operador, uma vez que esse equipamento é projetado também para portar uma ponta com uma broca em alta velocidade, para operação em madeira ou placas eletrônicas ou outro tipo de desenho.

Nesse sentido, quando o *endstop* está ativado, o equipamento entra no modo de *emergência* automaticamente, reporta à Unidade Controladora a parada e a razão dela. Com isso, passa a descartar comandos de movimento que recebe e passa a aguardar a controladora liberar novos movimentos; até lá, o equipamento não se move.

Para que isso fosse possível, foram utilizados os dois *bits* restantes que não foram utilizados no pacote de movimento enviado a cada ciclo, mas o utilizados de uma maneira ligeiramente diferente.

---

[9] Notações iniciadas pelo prefixo “0b” são interpretadas por algumas linguagens de programação (como C, C# e outras) como sendo uma manipulação de *bits*. A saber, seja uma variável declarada como *byte valor = 00000011*, a configuração binária deste *byte* será [00 00 10 11], pois representará o número decimal 11. Caso seja declarada a mesma variável com a notação referenciada, ou seja, *byte valor = 0b00000011*, isso significa que foram declarados exatamente quais *bits* devem conter 1 em *binário*, e a configuração fica exatamente como declarado, [00 00 00 11], neste caso representando o número decimal 3. A diferença está aqui justamente na manipulação desejada, ou seja, operações sobre *bits*, e não no número decimal que tais *bits* representam [WIKICHIP, 2016].

Enquanto apenas são considerados os *bits* que estão em 1 ou 0 no envio de comandos por ciclo, para o envio de comandos tanto de parada de emergência pelo operador quanto para suspendê-la – com é necessário caso a máquina pare pela questão de fim de curso após ativação do *endstop* –, foi adicionada uma verificação adicional: se o sétimo *bit* do *byte* enviado estiver ativado, o *firmware* entende que o comando recebido não é de movimento para os motores, cujos quais descartaria por estar parado em um modo de emergência, e entende que deve considerar um comando a respeito do estado de emergência da máquina.

Sendo assim, recebendo um *byte* com a sétima posição ativada, pode-se usar a primeira posição do *byte*, que era utilizada para definir a direção do motor do eixo X, para notificar ao *firmware* que deve suspender o modo de emergência. O exemplo é o que segue, para ficar clara a diferença entre um comando de movimento e um de suspensão de modo emergencial, conforme definições do protocolo:

- 0b00000001: comando de direção (+) e nenhum movimento.
- 0b01000001: suspensão de emergência.

Uma vez suspenso o modo de emergência, os eixos, mesmo tocando em algum dos *endstops*, podem voltar a se mover, ou seja, o *firmware* pode voltar a receber comandos e os enviará para os eixos. Por isso, o envio de comando para liberação de movimento neste caso deve vir acompanhado de um comando *homing*; nesse sentido, esta é a única função que pode realizar a suspensão da emergência, por questões de segurança.

Como justificativa para a decisão aqui tomada sobre o formato do protocolo: vale ressaltar que tudo em um computador resume-se a operações sobre números. Com essa premissa, infere-se que cada comando enviado pode ser traduzido em um número; assim, também se compreende que *bytes* e a notação binária são somente versões representativas de números com propósitos diferentes, mas possuindo a mesma capacidade de representá-los. Isso quer dizer que é possível traduzir um comando binário qualquer enviado aos motores – por exemplo, 0b00111011 para “X+, Y– e Z+”, de acordo com o protocolo estabelecido – para uma notação decimal, ou seja, para números amigáveis a seres humanos. No caso desse exemplo do binário 0b00111011, a sua versão no sistema decimal seria o número 59. Para fins práticos, portanto, seria possível criar uma tabela com 128 posições (que é o máximo de posições que um *byte* pode armazenar) com cada número associado a um conjunto de movimentos, ou seja, cada número representando um comando.

Contudo, foi escolhida a notação binária, e há uma razão forte para isso. Não só elimina qualquer possibilidade de tornar obscuro o significado de cada um desses números, mas fornece melhor informação visual, dado um comando enviado, a respeito de quais pinos de quais motores estão sendo ativados – o que agiliza imensamente o desenvolvimento e a correção de possíveis erros, baseando-se nas posições acionadas pelos *bits*, conforme mencionado. A opção quanto aos números, em contrapartida, nos daria uma tabela imensa, contendo conjunto de comandos e seu representativo numérico sem muito significado atrelado, ou seja, que não indica *exatamente* o que a instrução significa, fornecendo um controle menos óbvio. Nesse sentido, é muito mais fácil compreender quais pinos o comando "0b00111011" está fazendo do que identificar o que o número 59 faz.

4.4. *Hardware*

O *hardware* utilizado para o desenvolvimento é composto de muitas partes, algumas adquiridas e algumas fabricadas durante a montagem. O equipamento CNC, contudo, não foi montado especificamente para o projeto; é um equipamento que está em posse para desenvolvimento cujo qual não possui nenhuma parte de *software* implementada, ou seja, contempla apenas o *hardware.*

Seguem as partes relevantes para o projeto, listadas na Tabela 6.

Tabela 6 – Lista de componentes físicos

Descrição	Quantidade	Comentários
Microcontrolador	1	Arduino UNO rev. 3 conforme página do dispositivo na fabricante [ARDUINO, 2021], portando o microcontrolador ATmega16U2 [ALLDATASHEET, 2009].
CNC *shield*	1	Modelo V2.
Motor de passo	3	Motor NEMA Modelo 17HS4401 conforme listagem de configurações [OMC STEPPER ONLINE, 2021].
Barra roscada	2	Barra para movimentação dos eixos Z e X. Possui 6.35mm entre os flancos e diâmetro de 8mm, medidas utilizando método de medição de flancos e um paquímetro [CALIBRATOOLS, 2021].
Botões de fim de curso (*endstops)*	2	1 *endstop* para o eixo X e outro para o eixo Y.
Computador	1	Máquina rodando *Microsoft Windows* utilizada para desenvolvimento e execução das aplicações.
Cabo USB	1	Conector entre o Arduino e o computador para comunicação de comandos.
Botão de emergência	1	Botão de emergência do tipo “cogumelo” que corta a energia fisicamente dos motores.
Periféricos como presilhas, suportes, estruturas plásticas e outros.	-	Peças variadas que definiram a estrutura do equipamento CNC para os propósitos.

Fonte: Elaborado pelo autor (2021).

A Figura 11 contém uma imagem do equipamento físico utilizado para o projeto e desenvolvimento deste projeto:

Figura 11: Imagem do equipamento CNC utilizado para desenvolvimento.

Fonte: Elaborado pelo autor (2021).

5. DESENVOLVIMENTO E TECNOLOGIAS UTILIZADAS

Durante a análise inicial, foi realizado levantamento das tecnologias de *software* que poderiam ser utilizadas. Para o microcontrolador, foi necessária a compreensão da sintaxe da linguagem utilizada nos Arduinos, pois é bastante específica. Porém, para as soluções de *desktop*, a maior parte das linguagens de programação poderiam, de uma forma ou outra, prover o necessário para o desenvolvimento; portanto, escolheu-se o ambiente de maior afinidade.

Como o equipamento de *hardware* não foi montado especialmente para o projeto, houve a necessidade de estudo das tecnologias envolvidas para que fosse possível comandá-las através do *software* proposto. Nas seções a seguir, as tecnologias que envolveram o projeto serão detalhadas de acordo.

Ademais, cito aqui ferramentas de suporte necessárias para a elaboração deste trabalho como parte das tecnologias utilizadas.

5.1. *Softwares* de desenvolvimento*:*

5.1.1. Linguagem de programação *desktop*: C# e .NET 5.0

C# é uma linguagem de programação orientada a objetos desenvolvida pela empresa *Microsoft Corporation* [MICROSOFT, 2021]. Nasceu em 2002 como parte da iniciativa de sua *framework* [TECHTERMS, 2021] .NET e é uma das linguagens que foram desenvolvidas como parte do padrão CLI (*Common Language Infrastructure)* [TECHTARGET, 2007], uma solução para que aplicações, mesmo em linguagens de programação diferentes como Visual Basic e F# possam ser executadas na mesma plataforma através da execução de código em um ambiente virtual comum de instrução. Foi originalmente desenvolvida apenas para o sistema operacional Microsoft *Windows* desde sua versão 1.0, lançada em 2002 juntamente com a primeira versão da linguagem C#.

A *framework* foi dependente da plataforma *Windows* até sua versão 4.8, suportando até a versão 7.3 da linguagem C#. Contudo, em paralelo ao projeto original, a empresa lançou em 2016 o .NET Core, uma versão multiplataforma de código-fonte aberto da *framework*, sendo desenvolvida para substituir a antiga versão dependente do ambiente *Microsoft Windows*, e implementando a versão 8.0 do C#. A partir da versão 9.0 da linguagem, passa a se chamar apenas .NET, agora em sua versão 5.0, solidificando e padronizando o ambiente como uma poderosa *framework* multiplataforma e de código-fonte aberto e gratuito para uso.

5.1.2. *Microsoft Visual Studio 2019 Community Edition*

O *Microsoft Visual Studio 2019 Community Edition* é uma IDE (Ambiente de Desenvolvimento Integrado, em tradução livre) com recursos avançados para desenvolvimento de *software* disponível para a plataforma *Microsoft Windows* e *Apple macOS* [MICROSOFT, 2021]. Conta com ferramentas de verificação e otimização de código-fonte, ferramentas de depuração e *debugging*, além de diversos recursos voltados à produtividade do desenvolvedor e várias linguagens de programação. É uma IDE desenvolvida pela *Microsoft Corporation* e é oferecida gratuitamente pela empresa desde sua primeira versão, lançada em 1997, apesar de possuir versões pagas que oferecem recursos avançados ao desenvolvedor, voltados principalmente para uso profissional.

5.1.3. Linguagem de programação: Arduino C++

A linguagem C++ é uma linguagem orientada a objetos derivada da linguagem C, utilizando-a como base. Foi desenvolvida entre 1979 e 1983 por Bjarne Stroustrup, e é uma linguagem poderosa, utilizada até os dias de hoje [CPLUSPLUS.COM, 2021].

O equipamento Arduino faz uso da versatilidade da linguagem C++ como base para programação e esta é disponibilizada juntamente com uma *framework* contendo muitos recursos específicos da plataforma, como leitura e escrita em pinos, interrupções de *hardware* para manipulação física da execução do processador, biblioteca de recursos matemáticos otimizada para o equipamento, dentre outros.

5.1.4. Aplicativo *Diagrams*

O aplicativo *Diagrams* é uma plataforma gratuita de desenvolvimento de diagramas baseados na UML e está disponível tanto em sua versão *web* quanto sua versão para *desktop* [DIAGRAMS.NET, 2021]. Permite a criação de inúmeros modelos de diagramas para diversos propósitos. Os diagramas deste trabalho foram desenvolvidos nesta plataforma em sua versão *web*.

5.2. *Hardware* utilizado

5.2.1. Motores de passo

O motor de passo é um servomotor elétrico de movimento discreto, ou seja, permite instruções de movimento exatas, e suas medidas são conhecidas. Sua rotação não é livre, como é o

caso de motores de corrente alternada ou um motor à combustão; seu estado padrão é parado, ou estacionado. Funciona por caminhar em passos definidos e, justamente por conta da precisão de cada um dos passos, é possível conhecer a quantidade de movimento realizada. Seu funcionamento mais comum conta com um conjunto de ímãs elétricos que mantém o eixo travado em determinado ponto e, quando um passo é executado, o motor energiza os eletroímãs de modo a executar o movimento, um passo por vez.

O equipamento CNC utilizado conta com três motores de passo, um para cada eixo, de 1.8 graus por passo. Isso quer dizer que, para completar uma volta, o motor necessita executar 200 passos (360 graus divididos 1.8 graus por passo). Utiliza-se dessa configuração conhecida para estimar a quantidade de movimento efetuada a cada instrução.

5.2.2. Arduino UNO

Arduino é uma plataforma de *hardware* focada em automação [ARDUINO, 2021]. Permite programação de instruções e conta com pinos, utilizados para interpretação de dados e estímulos externos, como sensores de luz, temperatura, acionamento de interruptores etc. Utiliza de instruções escritas em linguagem de programação C++ equipada com uma *framework* voltada para o controle do equipamento (ver tópico 5.1.3).

O Arduino UNO, modelo em pauta, é uma placa microcontroladora derivada do projeto Arduino cujo foi selecionado por conter pinos suficientes para todas as instruções do *hardware* CNC e comunicação via interface *serial*. Além disso, conta com 32kB de memória para instruções e um processador de 16 MHz – mais do que suficiente para os propósitos do projeto.

5.2.3. *Shield CNC V2*

O *shield* CNC é uma placa de expansão compatível com o Arduino UNO voltada para automação de equipamentos CNC. É desenhada com o objetivo de ser encaixada sobre o microcontrolador, garantindo o mapeamento de pinos específicos para o *shield,* além de oferecer entradas para alimentação elétrica aos motores. O *shield* é preparado para suportar até 3 eixos de operação e permite adição de componentes, como *drivers* para motores, e permite configurações físicas diversas através de seletores manuais. Foi utilizada a versão 2 da placa em questão para o projeto.

5.2.4. Driver para motores de passo A4988

Os *drivers* são circuitos que orientam a comunicação lógica entre o motor de passo e o microcontrolador [POLOLU ROBOTICS AND ELECTRONICS, 2021]. Seu objetivo principal é o de mandar potência para o motor, recebendo os sinais de direção e movimento e aplicando a corrente elétrica necessária para que o motor opere. É também este componente que define a forma que o sinal é enviado ao motor, ou seja, quaisquer comandos enviados passam necessariamente pela tradução que o *driver* faz, de modo a converter instruções para sinal nativo para os motores.

O modelo de *driver* utilizado é o A4988, fabricado pela *Allegro Microsystems*, empresa multinacional de *microchips*. É um dos *drivers* para motores de passo de pequeno porte mais utilizados para CNCs, sendo bastante comum para projetos de tamanho e potência reduzidos. O projeto atual utiliza três desses *drivers*, um para cada motor de passo em operação.

6. CASOS DE TESTE

Ao final do desenvolvimento do *software* em sua versão inicial, foram elaborados cenários de testes no formato *Gherkin* [FONSECA, 2019] para garantir o bom funcionamento de acordo com o esperado pelo projeto de abertura. Os casos de teste e seus resultados estão relacionados na Tabela 7.

Tabela 7: Casos de Teste do Equipamento

Id	Cenário	Resultado
CT001	**Dado:** uma imagem válida. **E:** a imagem é compatível. **Quando:** carregar a imagem no Processador de Imagens. **Então:** o Processador de imagens gera o arquivo com instruções.	PASSOU
CT002	**Dado:** uma imagem válida. **E:** a imagem é incompatível. **Quando:** carregar a imagem no Processador de Imagens. **Então:** o Processador de imagens deve descartar a imagem. **E:** informar que a imagem é incompatível.	PASSOU
CT003	**Dado:** uma imagem inválida. **Quando:** carregar a imagem no Processador de Imagens. **Então:** o Processador de imagens deve descartar a imagem. **E:** informar que a imagem é inválida.	PASSOU
CT004	**Dado:** um comando válido. **E:** o modo manual de operação está selecionado. **E:** não há modo de emergência ativado. **E:** não há *endstops* ativados. **Quando:** o comando é enviado. **Então:** o equipamento deve se movimentar.	PASSOU
CT005	**Dado:** um comando válido. **E:** o modo manual de operação está selecionado. **E:** existe modo de emergência ativado. **Quando:** o comando é enviado. **Então:** o equipamento deve descartar o comando e informar da emergência.	PASSOU

Id	Cenário	Resultado
CT006	**Dado:** um comando válido. **E:** o modo manual de operação está selecionado. **E:** não existe modo de emergência ativado. **E:** um dos *endstops* está ativado. **Quando:** o comando é enviado. **Então:** descartar o comando e informar da emergência. **E:** ativar o modo de emergência.	PASSOU
CT007	**Dado:** um comando inválido. **E:** o modo manual de operação está selecionado. **Quando:** o comando é enviado. **Então:** a controladora descarta o comando e informa “comando inválido”.	PASSOU
CT008	**Dado:** uma lista de comandos válidos. **E:** o modo automático de operação está selecionado. **E:** não há modo de emergência ativado. **E:** não há *endstops* ativados. **Quando:** um conjunto de instruções é carregado. **Então:** o equipamento deve se movimentar uma vez por comando até finalizar todas as instruções.	PASSOU
CT009	**Dado:** uma lista de comandos válidos. **E:** o modo automático de operação está selecionado. **E:** existe modo de emergência ativado. **Quando:** um conjunto de instruções é carregado. **Então:** o equipamento deve descartar a lista de comandos e informar da emergência.	PASSOU
CT010	**Dado:** uma lista de comandos válidos. **E:** o modo automático de operação está selecionado. **E:** não existe modo de emergência ativado. **E:** um dos *endstops* está ativado. **Quando:** um conjunto de instruções é carregado. **Então:** o equipamento deve descartar a lista de comandos e informar da emergência. **E:** ativar o modo de emergência.	PASSOU
CT011	**Dado:** uma lista de comandos inválidos. **E:** o modo automático de operação está selecionado. **Quando:** um conjunto de instruções é carregado. **Então:** a controladora descarta a lista de comandos e informa “arquivo de instruções inválido”. **E:** oferecer possibilidade de nova carga de instruções	PASSOU

Id	Cenário	Resultado
CT012	**Dado:** uma operação de movimento. **Quando:** o botão de emergência na controladora for pressionado. **Então:** a controladora deve enviar sinal de emergência ao equipamento. **E:** encerrar o envio de comandos. **E:** informar o usuário da parada por intervenção do usuário.	PASSOU
CT013	**Dado:** uma operação de movimento. **Quando:** o botão de emergência no equipamento for pressionado. **Então:** o fornecimento de energia dos motores deve ser interrompido.	PASSOU
CT014	**Dado:** uma operação de movimento. **Quando:** o botão de *homing* na controladora for pressionado. **Então:** a controladora deve enviar instruções para retornar a ponta impressora ao ponto inicial. **E:** encerrar o envio de comandos. **E:** informar o usuário do comando de *homing*.	PASSOU
CT015	**Dado:** um comando para se mover até X=10, Y=10. **E:** a ferramenta impressora se localiza em X=0, Y=0. **Quando:** o comando equivalente é enviado. **Então:** a ferramenta deve ser movimentada de X=0, Y=0 até o ponto que representa X=10, Y=10.	PASSOU
CT016	**Dado:** um comando para se mover para uma posição fora dos limites da mesa de operação definida. **Quando:** o comando equivalente é enviado. **Então:** a unidade controladora deve descartar o comando. **E:** ativar o modo de emergência e notificar a emergência.	PASSOU
CT017	**Dado:** o equipamento em operação normal. **Quando:** o equipamento for fisicamente desconectado da porta USB. **Então:** a unidade controladora deve suspender o envio de comandos. **E:** ativar o modo de emergência e notificar a emergência.	PASSOU

Fonte: elaborado pelo autor (2021).

CONSIDERAÇÕES FINAIS

O desenvolvimento deste projeto voltado para controles de máquinas CNC proveu um contato extenso com muitas áreas e conceitos da automação, como era o objetivo inicial. De um lado, houve uma grande quantidade de diferentes tecnologias com as quais o contato foi inevitável, além da interoperabilidade das partes envolvidas, que ofereceu muito, mas demandou em igual proporção. De outro, houve uma necessidade de estudo de casos e incorporação de novas noções de *software* e *hardware* das quais foi possível extrair um conhecimento poderoso e duradouro, além de abrir portas para pesquisas e desenvolvimento futuros e da expansão do horizonte dessa área de interesse para outros trabalhos.

Um ponto interessante de ressaltar é que, mesmo com todo o processo de engenharia envolto no desenvolvimento de *software*, o fato é que existe uma margem de erro prevista. Metodologias trazem em seu escopo ciclos de teste para detecção de erros e mesmo ciclos dedicados exclusivamente para refatoração e correção de *bugs*.

Com isso, o desenvolvedor que trata somente com o *software*, cercado de rotinas de *backup* que permitem retorno a um estado anterior seguro em caso de falha catastrófica, passa a não se preocupar muito com eles para além do fluxo de trabalho; torna-se acostumado com comportamentos não esperados e os enumera para correção já como parte do seu dia-a-dia, dado que a premissa é a de que *sempre haverá bugs*.

Ainda que o foco seja sempre em um sistema sem falhas e altamente apoiado em métodos e ideias que diminuem os erros implementados, aumentando a produtividade através de automação de testes e outras soluções afins, o desenvolvimento puramente de *software* costuma ser *permissivo*. Contudo, não é este o caso quando se trata de equipamentos físicos. O *hardware* não costuma tolerar muitas falhas e, dependendo da intensidade ou qualidade, pode custar desde um motor ou uma placa eletrônica, um projeto inteiro, por vezes de valor elevado, ou até mesmo uma vida humana.

Trabalhar com equipamentos físicos reforça exponencialmente o poder dos processos de engenharia e a importância do rigor durante a implementação. Ainda, exige que sejam criados métodos mais seguros para o processo de testes, e que o desenvolvedor opere sendo mais específico

e consciente com o ambiente, pois é necessário que se conheça o equipamento e seus limites para a prevenção de possíveis problemas ou danos, por vezes, irreversíveis.

Além disso, ficou claro que a interação entre as duas pontas do projeto, ou seja, entre o *hardware* e o *software*, é muito mais complexa do que apenas a conversa estruturada entre dois equipamentos. Em uma ocasião isso ficou bastante claro, como esboçada a seguir.

Durante o projeto, ocorreu um *bug* por cerca de uma semana a respeito do controle de passos do motor do eixo X. Nessa ocasião, foram realizados ajustes no *software* tentando encontrar a causa do problema. Depois de muitos ajustes supérfluos nos cálculos efetuados pela unidade controladora, o motor passou a se mover de acordo com o compensando matematicamente pelas "correções"; contudo, passou a operar com menos potência em relação aos outros.

Além disso, todos os outros eixos funcionam corretamente com a configuração original, e a solução parecia incorreta por ser paliativa, principalmente por ser *aplicada para corrigir um problema desconhecido*. Dado que isso ocorria em apenas um dos motores, a conclusão lógica seguinte foi a suspeita de falha do *hardware*. Iniciou-se trocando a fonte alimentadora dos motores por outras duas similares, sem sucesso. Ao efetuar testes invertendo os motores entre os eixos, ou seja, alternando os motores do eixo X com os motores do eixo Y, o eixo X ainda permaneceu se comportando do mesmo modo. Concluiu-se que não era uma dificuldade no motor, mas no envio de sinais para aquele eixo X, portanto.

Após alguns dias efetuando testes, a parte de fiação do equipamento foi revisada e refeita, e pelo menos dois motores diferentes foram utilizados para testar. Restando somente a placa controladora, o equipamento foi desmontado por completo; somente neste momento que a análise visual revelou que uma configuração física do equipamento, desconhecida até aquele momento, estava habilitando *micro passos* para o eixo X.

Ao encontrar a documentação dos *drivers* sobre este recurso, verificou-se do que se tratavam, os micro passos. A leitura revelou que esta configuração permite que o *driver* do motor envie sinais para que seja realizado meio passo em um pulso, aumentando a precisão do motor, e oferecendo um movimento mais suave. Ao invés de um passo inteiro, ou seja, em vez de um pulso significar um passo, um sinal de movimento é traduzido em meio passo, além de outras subdivisões de um passo completo. No caso do motor utilizado, contendo 200 passos inteiros por volta (360

graus), ou simplesmente 200 passos por volta, cada passo representa precisamente 1.8 graus. Sendo assim, com a configuração de meio passo por sinal, o motor que estava sendo controlado no eixo X não mais se movia em 1.8, mas em 0.9 graus por passo, o que trouxe a quantidade de passos necessários para dar uma volta para 400. Essa configuração, ainda que proporcione algumas possibilidades, sacrifica a força total que o motor pode exercer. A perda de força de um passo inteiro para meio passo é de cerca de 30%, o que, na ocasião, era suficiente para que o motor não fosse capaz de operar de acordo com o esperado.

Removido o seletor que aciona esta função na placa controladora, os micro passos foram desativados para o *driver* e todo o sistema voltou a funcionar corretamente. Contudo, todo o processo entre a identificação do problema e sua solução levou cerca de uma semana e contaram com algumas refatorações desnecessárias em implementações que estavam funcionando corretamente. Além disso, ocasionou na troca de conexões no equipamento, o que aumentou o orçamento do projeto sem a mais remota necessidade.

É curioso ressaltar que este foi um dos primeiros problemas encontrados fisicamente na aplicação. Contudo, essa experiência desencadeou uma cautela maior com o *hardware* desde o começo do projeto, que se provou sendo muito valiosa durante a análise do restante da solução e seus detalhes, evitando testes que não levassem em consideração uma análise detalhada de todo o contexto.

Por fim, acredita-se que, findado o projeto, este alcançou seu propósito maior – qual seja, o aprendizado na prática com a implementação de um sistema que possui aplicações reais, além da interação com equipamentos periféricos que dependem da implementação bem elaborada e de forma rigorosa para alcançar um equipamento funcional e confiável.

Trabalhos futuros

Como projeção para trabalhos futuros, a intenção deste projeto é continuar se expandindo. A ideia principal e mais necessária é melhorar a comunicação com o *firmware*, adicionando mais funcionalidades e verificações de erros, bem como uma interação mais robusta entre este e a controladora. Isso deve exigir uma refatoração grande no *firmware*, bem como na própria controladora.

Adicionalmente, é intencionada a melhoria do *feedback* visual que a controladora exibe ao operador. Atualmente, para cada comando processado, a unidade controladora apenas exibe a posição em X e Y na tela, além do comando processado. A intenção é a de criar um posicionamento virtual, ou seja, uma representação da mesa física do equipamento e a posição da ferramenta impressora em tela, bem como outras informações, tais como se está em estado de emergência, se existe algum alarme ativado, a porcentagem de conclusão do trabalho, tempo estimado etc., com representação visual e interação.

Numa nota menos essencial, mas que aumentaria a produtividade, é a capacidade de se utilizar de manches, ou controles analógicos, para o modo de operação manual. Isso melhoraria muito a facilidade de posicionamento, pois, como visto, atualmente a interação manual exige a entrada de códigos GCODE, que não é uma forma muito prática.

Pretende-se criar também funcionalidades que permitam pausar a impressão, por exemplo, para manutenção manual da ferramenta impressora, sem perder a impressão corrente. Além disso, pretende-se adicionar funcionalidades de aceleração e desaceleração do trabalho de impressão, prezando a velocidade ou a precisão do trabalho a ser feito, dependendo da escolha do usuário.

Por fim, outro recurso a ser adicionado é o de definição automática de altura da mesa. Atualmente, a altura da mesa é definida manualmente para que a ferramenta toque o objeto. Por meio de um botão que é acionado quando o eixo Z tocar a mesa, é possível detectar qual a altura do material sobre o qual a máquina deve operar, para que possa se adequar aos mais diferentes cenários sem necessidade de operação ou intervenção manual.

REFERÊNCIAS

WHAT is a Stepper Motor? **Adafruit**, 2014. Disponível em <https://learn.adafruit.com/all-about-stepper-motors>. Acesso em: 01 de junho de 2021.

WHAT *is an interrupt?* **Adafruit**, 2014.. Disponível em <https://learn.adafruit.com/multi-tasking-the-arduino-part-2/what-is-an-interrupt>. Acesso em: 12 de junho de 2021.

ATMEGA16U2 Datasheet (PDF) – ATMEL Corporation. ***AllDataSheet***, 2009. Disponível em <https://pdf1.alldatasheet.com/datasheet-pdf/view/313554/ATMEL/ATmega16U2.html>. Acesso em: 3 de maio de 2021.

ARDUINO UNO rev3. **Arduino,** 2021. Disponível em <https://store.arduino.cc/usa/arduino-uno-rev3. Acesso em: 01 de junho de 2021.

LANGUAGE Reference. **Arduino,** 2021. Disponível em <https://www.arduino.cc/reference/en>. Acesso em: 7 de junho de 2021.

WHAT is Arduino? **Arduino,** 2021. Disponível em <https://www.arduino.cc/en/guide/introduction>. Acesso em: 14 de maio de 2021.

MEDIÇÃO de roscas. **CalibraTools,** 2021. Disponível em <https://www.calibratools.com.br/pdf/medicao-roscas.pdf>. Acesso em: 12 de junho de 2021.

HISTORY of C++. **CPlusPlus.com**, 2021. Disponível em <https://www.cplusplus.com/info/history>. Acesso em: 7 de junho de 2021.

CONTROLE de Motor de Passo Nema com *Driver* A4988. **Curto Circuito**, 2019. Disponível em <https://www.curtocircuito.com.br/blog/Categoria%20Arduino/controle-de-motor-de-passo-nema-driver-a4988>. Acesso em: 3 de maio de 2021.

DIAGRAMS. ***Diagrams.net,*** 2021. Disponível em <https://app.diagrams.net>. Acesso em: 7 de junho de 2021.

FERGUSON, N.C. *A history of numerically controlled machine tools.* **Academia.edu,** 1978. Disponível em

<https://www.academia.edu/670021/A_history_of_numerically_controlled_machine_tools>. Acesso em: 03 de maio de 2021.

FONSECA, Helonise. Gherkin: introduzindo seus conceitos e benefícios. ***One Day Testing***, 2019. Disponível em <https://blog.onedaytesting.com.br/gherkin/>. Acesso em: 14 de junho de 2021.

FUSTER, Jay. *Atomic Bomb Genetics.* ***Stanford University***, 2016. Disponível em <<http://large.stanford.edu/courses/2016/ph241/fuster2/>. Acesso em: 13 de maio de 2021.

GANSSLE, Jack. ***The Firmware Handbook. The definitive guide to embedded firmware design and Applications.*** Editora: Elsevier. Oxford, Inglaterra, 2004.

KOOJI, B.J.G. van der. ***The Invention of the Steam Engine.*** Versão 1.1. Editora: *University of Technology*, Delft. Holanda, 2015.

MATTEDE, Henrique. Tipos de motores elétricos, quais são? **Mundo da Elétrica,** 2019. Disponível em <https://www.mundodaeletrica.com.br/tipos-de-motores-eletricos-quais-sao/>. Acesso em: 11 de junho de 2021.

MICROCONTROLLER. **Merriam-Webster,** 2021. Disponível em <https://www.merriam-webster.com/dictionary/microcontroller>. Acesso em: 14 de maio de 2021.

THE history of C#. ***Microsoft Corporation,*** 2020. Disponível em <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history>. Acesso em: 7 de junho de 2021.

WELCOME to the Visual Studio IDE. ***Microsoft Corporation,*** 2021. Disponível em <https://docs.microsoft.com/pt-br/visualstudio/get-started/visual-studio-ide?view=vs-2019>. Acesso em: 7 de junho de 2021.

MITCHELL, Bradley. *What is a USB port? Your USB cable plugs in here.* ***Life Wire,*** 2020. Disponível em <https://www.lifewire.com/what-is-a-usb-port-818166>. Acesso em: 30 de maio de 2021.

NORMAN, Jeremy. *ASCII (American Standard Code for Information Interchange) is Promulgated*. ***History of Information,*** 2010. Disponível em <https://www.historyofinformation.com/detail.php?id=803>. Acesso em: 23 de maio de 2021.

NEMA 17 17HS15-1504S Full Datasheet. ***OMC Stepper Online,*** 2021. Disponível em <https://www.omc-stepperonline.com/download/17HS15-1504S.pdf>. Acesso em: 03 de junho de 2021.

OMG® Unified Modeling Language. **OMG,** 2017. Disponível em <https://www.omg.org/spec/UML/2.5.1/PDF>. Acesso em: 10 de junho de 2021.

PATEL, Nairutya. *Study on computer numerical control (CNC) technology.* ***International Research Journal of Engineering and Technology (IRJET)****, Volume* 7, *Issue* 3. Tamilnadu, Índia, 2020.

DMOS Microstepping Driver with Translator And Overcurrent Protection. ***Pololu Robotics and Electronics,*** 2014. Disponível em <https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf>. Acesso em: 03 de maio de 2021.

G-CODE. RepRap. **REPRAP,** 2021. Disponível em: <https://reprap.org/wiki/G-code>. Acesso em: 11 de fevereiro de 2021.

SOMMERVILLE, I. **Engenharia de *Software*.** 8ª Edição. Editora: Pearson Addison-Wesley. São Paulo, 2007.

BITS and Bytes. ***Stanford University,*** 2021. Disponível em <https://web.stanford.edu/class/cs101/bits-bytes.html>. Acesso em: 13 de maio de 2021.

COMMON Language Infrastructure (CLI). ***TechTarget,*** 2007. Disponível em <https://searchapparchitecture.techtarget.com/definition/Common-Language-Infrastructure-CLI>. Acesso em: 7 de junho de 2021.

FIRMWARE. ***TechTerms,*** 2021. Disponível em <https://techterms.com/definition/firmware>. Acesso em: 15 de fevereiro de 2021.

Framework. ***TechTerms,*** 2021. Disponível em <https://techterms.com/definition/framework>. Acesso em: 7 de junho de 2021.

PIXEL. ***Technopedia,*** 2021. Disponível em <https://www.techopedia.com/definition/24012/pixel>. Acesso em: 13 de maio de 2021.

*ACTIVITY Diagrams.**Uml Diagrams,*** 2020. Disponível em <https://www.uml-diagrams.org/activity-diagrams.html>. Acesso em: 10 de junho de 2021.

BINARY Literal. ***Wikichip,*** 2016. Disponível em: <https://en.wikichip.org/wiki/binary_literal>. Acesso em: 30 de maio de 2021.

YADAV, Karuna, SHARMA, Kamal Raj, ANAND, Dimple. *Additive and Subtractive Manufacturing: A Review. In: Journal of Material Science and Mechanical Engineering (JMSME)*, Volume 2, Number 2. Nova Delhi, Índia, 2015.

APÊNDICE 1: INTERRUPÇÕES DE *HARDWARE*

O projeto que deu origem ao *firmware* propõe originalmente dois estados, conforme observado nos diagramas de estado e de atividade. O estado principal, ou estado normal, aguarda comandos de movimento ou de outras instruções, e o faz indefinidamente. A cada ciclo, verifica se há nova instrução e processa os comandos conforme são recebidos. O segundo estado trata de um outro laço indefinido, cujo qual ocorre caso exista alguma razão para parar de se movimentar. Este estado força o equipamento a ignorar comandos de movimento e ouve apenas comandos de suspensão deste estado, conforme definido no protocolo.

A verificação de situações para parada, sob a visão do projeto original, deve ser feita dentro do laço principal de execução. Isso não proporciona o melhor desempenho possível, pois faz a verificação a cada iteração do laço principal; em vistas de que não é a todo momento que a verificação por sinais de emergência vai ser acionada, esta solução, apesar de funcionar, não é a mais eficiente, pois ocupa processamento.

Mais especificamente, na implementação corrente, existem três etapas. A primeira verifica se há alguma razão para entrar em modo de emergência, a segunda verifica se há algum comando sendo recebido, e a terceira tenta processar o valor recebido, cada uma consumindo parte do processamento disponível.

Contudo, existe uma forma mais eficiente, tanto a nível de simplicidade quanto a nível de desempenho, para que isso seja realizado. A forma em questão utiliza-se de um método denominado *interrupção* [ADAFRUIT, 2021], suportado por alguns microcontroladores. Ela consiste em interromper a execução de uma instrução no processador a nível de *hardware*, direcionando a execução exclusivamente para outra função ou comportamento, única e exclusivamente se caso um pino específico no microcontrolador seja ativado.

Sendo assim, não é mais de responsabilidade do laço principal ativar uma verificação de emergência e, sim, do próprio *hardware* alterar o fluxo de execução quando, a exemplo, um *endstop* seja ativado ou o botão de emergência seja acionado pelo operador. Nesse sentido, não haverá processamento dedicado dentro do laço apenas para as verificações sobre pinos: o pino acionado *interrompe* a execução principal através de um sinal fornecido pelo botão pressionado no equipamento e força o processador a executar outras instruções, como as instruções dedicadas à

verificação de casos de emergência. Por não ser um processo ativo como na primeira implementação, e, sim, receptivo, a velocidade de execução é otimizada por diminuir a carga no processador, além de garantir que cada tarefa seja executada precisa e exclusivamente quando necessário.

Esta é uma funcionalidade a ser incluída em futuras iterações do projeto visando otimização máxima, e já faz parte dos complementos previstos.