# Introducere

Pentru lucrarea de licență mi-am propus să fac un router CNC[[1]](#footnote-1) capabil să deseneze sau să graveze în lemn diferite imagini sau text prin intermediul unui limbaj de programare folosit de CNC-uri sau imprimante 3D numit GCODE.

Aplicația este realizată pentru platforma hardware Arduino Uno. Aceasta primește comenzi GCODE prin intermediul portului USB. Comenzile primite vor fi analizate și executate rezultând astfel diferite tipuri de mișcări cu ajutorul unor motoare pas cu pas.

Exista câteva *firmware*-uri similare. Am venit cu ceva diferit prin faptul că am ales să scriu aplicația într-o manieră OOP[[2]](#footnote-2), cu un cod abstractizat foarte mult de partea hardware, ușor de înțeles și configurat.

Pentru că m-am axat mai mult pe partea de software, pentru cadrul CNC-ului am folosit o platformă deja existentă pe care am adaptat-o la cerințele mele. Componentele acestui cadru le-am printat cu ajutorul unei imprimante 3D.

În primul capitol voi vorbi despre resursele software, hardware și cele mecanice folosite în acest proiect pentru realizarea si utilizarea CNC-ului.

În cel de-al doilea capitol voi discuta despre asamblarea componentelor hardware. Voi insista mai mult pe legătura dintre partea hardware cu cea software.

În capitolul 3 voi discută despre limbajul GCODE, mai specific despre sintaxa și semantica ale instrucțiunilor limbajului. Voi aduce în discuție doar comenzile pe care le voi implementa în software-ul CNC-ului.

În capitolul 4 vor fi detaliate formulele și algoritmii folosiți pentru calcularea distanței, simularea vitezei și pentru deplasarea în spațiul cartezian tridimensional.

Având discutate toate aceste fundamente necesare, în capitolul 5 vom trece la implementarea software-ului.

În capitolul 6 vom face un scurt rezumat utilizării CNC-ului împreună cu *tool-*urile menționate pentru conversia imaginilor în fișiere GCODE, precum și a utilizării scriptului pentru trimiterea fișierelor prin USB către CNC.

Iar în ultimul capitol vom trage câteva concluzii.

Contents

[Introducere 1](#_Toc536804461)

[1 Resurse utilizate 4](#_Toc536804462)

[1.1 Software 4](#_Toc536804463)

[1.1.1 Aplicația CNC în Arduino 4](#_Toc536804464)

[1.1.2 Aplicația de comunicație cu Arduino 4](#_Toc536804465)

[1.1.3 *Tool*-uri pentru conversia imaginilor 4](#_Toc536804466)

[1.2 Hardware 5](#_Toc536804467)

[1.2.1 Microcontroler 5](#_Toc536804468)

[*1.2.2* CNC *shield* 6](#_Toc536804469)

[1.2.3 Motoare pas cu pas 7](#_Toc536804470)

[1.2.4 Drivere pentru motoarele pas cu pas 8](#_Toc536804471)

[1.2.5 *Endstop*-uri pentru limitarea axelor 9](#_Toc536804472)

[1.2.6 Sursă de alimentare 10](#_Toc536804473)

[1.3 Mecanice 10](#_Toc536804474)

[1.3.1 Componente printate din plastic 10](#_Toc536804475)

[1.3.2 Tije cromate 10](#_Toc536804476)

[1.3.3 Rulmenți 11](#_Toc536804477)

[1.3.4 Șuruburi trapezoidale 12](#_Toc536804478)

[1.3.5 Hub de cuplaj flexibil pentru motor 12](#_Toc536804479)

[1.3.6 Tije filetate, și piulițe 12](#_Toc536804480)

[1.3.7 Pat suport pentru desenare 12](#_Toc536804481)

[1.3.8 Unealta pentru gravat 12](#_Toc536804482)

[2 Asamblare 13](#_Toc536804483)

[2.1 Asamblarea componentelor electrice 13](#_Toc536804484)

[2.2 Asamblarea componentelor mecanice 13](#_Toc536804485)

[3 Limbajul GCODE 14](#_Toc536804486)

[3.1 Sintaxa limbajului GCODE 14](#_Toc536804487)

[3.2 Comenzi GCODE modale 14](#_Toc536804488)

[3.2.1 Grupul comenzilor de mișcare 15](#_Toc536804489)

[3.2.2 Grupul comenzilor pentru selectarea planului 17](#_Toc536804490)

[3.2.3 Grupul comenzilor pentru poziționare 18](#_Toc536804491)

[3.2.4 Grupul comenzilor pentru setarea unității de măsură 18](#_Toc536804492)

[3.2.5 Comenzi personale rezervate 18](#_Toc536804493)

[3.3 Comenzi GCODE non modale 18](#_Toc536804494)

[3.4 Comenzi GCODE auxiliare 19](#_Toc536804495)

[4 Algoritmi 20](#_Toc536804496)

[4.1 Formula pentru calculul distanței fizice de deplasare 20](#_Toc536804497)

[4.2 Simularea vitezei de deplasare 21](#_Toc536804498)

[4.3 Algoritmul pentru mișcare liniară simplă 23](#_Toc536804499)

[4.4 Algoritmul pentru interpolare liniară 23](#_Toc536804500)

[4.5 Algoritmul pentru interpolare circulară 24](#_Toc536804501)

[5 Implementare 28](#_Toc536804502)

[5.1 Arhitectura generală a aplicației 28](#_Toc536804503)

[5.1.1 Clasa Machine 30](#_Toc536804504)

[5.1.2 Clasa Motor 32](#_Toc536804505)

[5.2 Fișierele de configurație 34](#_Toc536804506)

[5.2.1 Configurația Hardware 34](#_Toc536804507)

[5.2.2 Configurația CNC-ului 34](#_Toc536804508)

[5.3 Implementarea interpretorului de comenzi 34](#_Toc536804509)

[5.4 Problema preciziei a numerelor fracționare 34](#_Toc536804510)

[6 Utilizare 36](#_Toc536804511)

[7 Concluzii 37](#_Toc536804512)

[8 Bibliografie 38](#_Toc536804513)

# Resurse utilizate

În acest capitol voi discuta despre resursele software, hardware, cât și cele mecanice folosite pentru realizarea proiectului.

## Software

În această secțiune sunt menționate resursele software folosite pentru realizarea CNC-ului în Arduino, cât și cele folosite pentru realizarea a unui program ce comunică prin portul USB cu aplicația CNC.

Pe lângă asta mai sunt menționate și câteva *tool*-uri pentru a face conversia imaginilor în fișiere GCODE.

### Aplicația CNC în Arduino

Deoarece am ales să folosesc platforma hardware Arduino, am folosit ca și limbaj de programare, limbajul C++, aceasta fiind singura opțiune disponibilă, însă după părerea mea si una dintre cele mai bune alegeri pentru programarea sistemelor embedded[[3]](#footnote-3). Oferă toate facilitățile *low-level[[4]](#footnote-4)* ale limbajului C, cat si posibilitatea de a programa la un nivel cat mai *high-level[[5]](#footnote-5)* folosind concepte de programare orientata pe obiect.

Compilatorul folosit pentru a transforma limbajul C++ în limbaj mașină pentru microcontrolerulAtmega328p de pe Arduino este avr-g++. Acesta suporta aproape toate *featurile* limbajului C++, însă fără posibilitatea utilizării conceptului de Excepții.

Petru operațiile de input/output pe pini cât si pentru partea de comunicație cu portul serial, am folosit biblioteca oferită de Arduino și biblioteca AVR a microcontrolerului.

### Aplicația de comunicație cu Arduino

Pentru dezvoltarea aplicației de comunicație prin USB cu CNC-ul am ales sa folosesc limbajul *Python* cu ajutorul pachetului *PySerial*.

### *Tool*-uri pentru conversia imaginilor

Pentru convertirea imaginilor în limbajul GCODE, am folosit aplicația *desktop* Inkscape, împreună cu alte tool-uri online de conversie a fișierelor cu extensia SVG în GCODE.

Cu aplicația Inkscape am transformat imaginea în formatul SVG, iar cu ajutorul unui tool online (jscut) am convertit imaginea SVG într-un fișier ce conține instrucțiuni GCODE.

**<TODO: ref inkscape si jscut>**

## Hardware

În această secțiune voi enumera toate resursele hardware împreună cu specificațiile lor folosite în proiect. Voi începe cu cea mai importantă componentă, microcontrolerul.

### Microcontroler

Microcontrolerul este cea mai importantă componentă hardware din acest proiect. Aici va fi *uploadat* codul sursă compilat al CNC-ului.

Așa cum am menționat și în introducere, placa de dezvoltare folosită pentru realizarea CNC-ului este Arduino Uno. Această placă folosește microcontrolerul Atmega328p.

|  |  |
| --- | --- |
| Figura – Arduino UNO | Specificații tehnice generale:  - frecventa microcontroler: **16Mhz**  - memorie flash: **32KB**  - memorie SRAM: **2KB**  - memorie EEPROM: **1KB**  - pini digitali I/O: **14**  - pini analogici: **6**  - tensiune alimentare placa: **7-20V**  - tensiune operare microcontroler: **5V** |

Așadar vom avea la dispoziție **32KB** memorie flash pentru scrierea codului și **2KB** de memorie RAM pentru variabilele utilizate în execuția codului.

Am ales această placă deoarece este o platformă foarte populară. Din acest motiv există o comunitate destul de mare, iar documentația se găsește ușor.

Unul dintre cele mai importante motive pentru care am ales această placă este faptul că se poate monta peste ea un *CNC shield,* componentă hardware despre care voi vorbi in subcapitolul 1.2.2. Acest *shield*simplifica foarte mult conexiunile fizică dintre motoarele pas cu pas si *driver*-ele pentru motoare, pe care voi le discuta în subcapitolul 1.2.4.

Un alt avantaj este dat de faptul ca interfațarea fizica dintre laptop si Arduino este făcută prin intermediul unui cablu USB de tip A către tip B. Un alt motiv pentru care am ales această platformă este faptul ca *upload-*areasoftware-ului pe microcontroler se face foarte ușor prin intermediul acestui USB fără a mai fi nevoie de un programator special pentru a face acest lucru.

Un dezavantaj în domeniul programării *embedded* este faptul că se poate face *debug* doar cu hardware specializat, iar in ajutorul acestei probleme, platforma Arduino, vine cu trimiterea mesajelor de *debug* prin intermediului portului USB. Lucru ce rezolva parțial această problemă fără prea multe costuri în plus.

Toate aceste avantaje menționate mai sus legate de simplitatea folosirii plăcii Arduino prin intermediul USB-ului, sunt datorită faptului că această platformă are un *chip* de convertire a semnalului USB către interfața serial a microcontrolerului.

Iar nu in ultimul rând, legat de costuri, aceasta platforma împreuna cu celelalte componente hardware, *shield-ul* CNC si driverele pentru motoare, sunt o soluție relativ ieftina din punct de vedere financiar, iar cerințele hardware pentru implementarea CNC-ului sunt suficiente.

### CNC *shield*

In figura următoare am atașat o imagine cu *shield*-ulpentru CNC folosit în acest proiect. Voi face referire la aceasta imagine în capitolul următor de asamblare la partea de hardware.

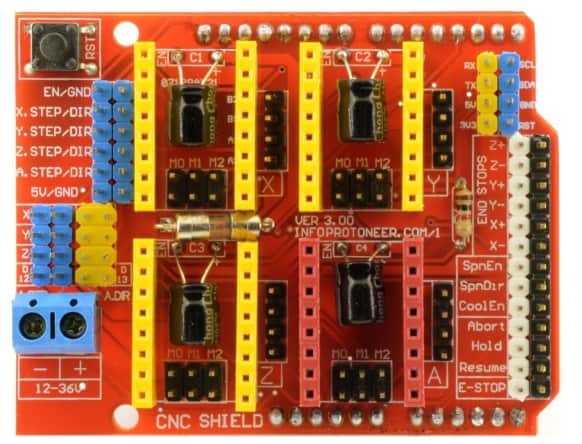


Figura – CNC shield

După cum am menționat și în subcapitolul precedent unul dintre cele mai importante motive pentru care am ales platforma Arduino este faptul că acest *shield* se poate monta peste Arduino.

Pe acest *shield* se pot monta 3 drivere pentru motoare pas cu pas pentru fiecare dintre cele 3 axe X, Y si Z, respectiv 3 motoare pas cu pas. În plus se mai poate monta încă un driver plus încă un motor pentru a ajuta una dintre axe.

Pentru a detecta când axele s-au deplasat la limitele lor fizice, pe *shield* se pot monta 6 *endstop*-uri, câte 2 pe fiecare axă pentru a semnala sfârșitul axei, poziția zero a unei axe sau celălalt capăt.

Tensiunea de alimentare pentru acest *shield* este între 12 si 36 de volți. Acest curent electric de intrare este folosit doar pentru a alimenta motoarele pas cu pas. Placa Arduino este alimentată prin USB, nu prin curentul de alimentare al *shield-*ului. Acest lucru facilitează separarea curentului de intensitate și amperaj mai mare folosit de motoarele pas cu pas de curentul folosit de microcontroler (5V tensiunea maxima și 200mA).

Voi intra în mai multe detalii despre modul cum funcționează acest CNC *shield* împreună cu Arduino și driverele pentru motoarele pas cu pas în capitolul 2 de asamblare.

### Motoare pas cu pas

Motoarele pas cu pas sunt niște motoare electrice în care o rotație completă de 360 de grade se împarte la un număr egal de pași, de obicei 200 de pași.

Astfel, aceste motoare sunt de foarte mare precizie și sunt foarte des utilizate în proiecte precum CNC-uri, imprimante 3D, sau în robotică.

Pentru acest proiect am avut nevoie de 3 motoare, fiecare motor pune în mișcare câte o axa, X, Y, respectiv axa Z. Modelul pe care l-am ales este: Motor pas cu pas **17HS8401S** Nema17.

|  |  |
| --- | --- |
| Figura – Motor Pas cu Pas Nema17 | Specificații tehnice:  - pas: **1.8grade, adică 200 pași in 360 grade**  - cuplu: **0.59Nm**  - curent: **1.7 A**  - dimensiuni: **specifice standardului Nema17** |

Mișcarea acestor motoare va fi controlată de către *driverele* pentru motoarele pas cu pas despre care voi discuta în subcapitolul următor.

### Drivere pentru motoarele pas cu pas

Driverele pentru motoarele pas cu pas sunt componente special construite pentru a controla rotația precisă a unui motor. Cu ajutorul lor se poate ajusta limita maximă de curent trimisă către motor, direcția in care motorul se deplasează, precum si executarea pașilor.

În plus de asta, aceste drivere vin cu opțiunea de a face *microstepping*. Un motor pas cu pas face într-o rotație completă de 360 grade exact 200 de pași egali. Cu ajutorul *microstepping-ului* controlează fazele curentului către motor astfel încât o rotație completă poate avea mai mult de 200 de pași. Astfel putem activa aceste opțiuni ajungând pana la 6400 de micro pași pe o rotație completă.

Cu alte cuvinte aceste drivere simplifică foarte mult interfațarea dintre motoare și microcontroler. Folosind astfel doar 2 pini de pe microcontroler, putem controla pașii și direcția unui astfel de motor pas cu pas.

Pentru alegerea *drivere*-lor existau mai multe posibilități, însă cele mai accesibile opțiuni au fost următoarele 2 *drivere*.

|  |  |
| --- | --- |
| Figura – Driver A4988 | Figura – Driver DRV8825 |
| Specificații tehnice:  Driver chip: **A4988**  Tensiune motor: **8V - 35V**  Tensiune chip: **3V - 5.5V**  Curent maxim: **2A**  Moduri *microstepping*: **1/2, 1/4, 1/8, 1/16** | Specificații tehnice:  Driver chip: **DRV8825**  Tensiune motor: **8V - 45V**  Tensiune chip: **3V - 5.5V**  Curent maxim: **2.5A**  Moduri *microstepping*: **1/2, 1/4, 1/8, 1/16, 1/32** |

Pentru proiect am folosit 2 *drivere* DRV8825 pentru motoarele de pe axele X si Y, și un *driver* A4988 pentru motorul de pe axa Z.

Nu este o diferență foarte mare între aceste 2 alegeri, una dintre diferențe ar fi că DRV8825 suportă *microstepping* mai mare. Cu un DRV8825 având opțiunea maxima de *microstepping* **1/32** activată atunci o rotație completă a unui motor pas cu pas ajunge la 6400 de micro pași, pe când la A4988 cu opțiunea maximă de *microstepping* **1/16** activată vom ajunge la 3200 de pași pe o rotație completă.

Din punct de vedere electric, ambele *drivere* suportă fără probleme curentul si voltajul necesar funcționării optime ale motoarelor pas cu pas alese.

### *Endstop*-uri pentru limitarea axelor

|  |  |
| --- | --- |
| *Switch*-urile endstop au rolul de a detecta când unealta CNC-ului s-a deplasat la capătul unei axe.  Acestea vor ajuta și la efectuarea procedurii de *homing*[[6]](#footnote-6) a uneltei.  Am folosit trei astfel de *switch*-uri, cate unul pentru fiecare axă. | C:\Users\uid41395\Desktop\endstop_switch.jpg  Figura – Endstop switch |

### Sursă de alimentare

|  |  |
| --- | --- |
| *Shield*-ul CNC primește ca input o tensiune între 12 și 36 de volți Acest curent de intrare va alimenta motoarele. Un asemenea motor pas cu pas consuma maxim 1.7A, deci avem nevoie de o sursă care sa fie capabilă sa furnizeze un curent de minim aproximativ 6A la tensiunea de 12V, adică de minim 70wati. Pentru acest proiect am ales o sursă ATX de calculator cu următoarele specificații: | C:\Users\uid41395\Desktop\sursa.jpg  Figura – Sursă aliminetare |

Curentul de 15A furnizat la 12v de către această sursa fiind mai mult decât suficient pentru nevoile motoarelor, deci nu vom avea nici un fel de probleme în acest sens.

## Mecanice

În acest ultim subcapitol sunt prezentate toate componentele mecanice folosite la realizarea proiectului, rolul lor, precum si specificațiile unor componente mai importante.

### Componente printate din plastic

Componentele printate din plastic se află în linkul din referință.

**<TODO add: adaptor pentru axa Z, mecanismul de prindere al uneltei, și al pixului >**

### Tije cromate

Rolul acestor tije este de a liniariza mișcarea uneltei pe cele trei axe împreună cu ajutorul rulmenților despre care voi discuta în subcapitolul următor.

|  |  |
| --- | --- |
| * pentru axa X am folosit 3 tije cromate (10mm diametru) de 40cm lungime * pentru axa Y am folosit 3 tije cromate (10mm diametru) de 40cm lungime * pentru axa Z am folosit 2 tije cromate (8mm diametru) de 20cm lungime | C:\Users\uid41395\Desktop\tije cromate.jpg  Figura – Tije cromate de 10, respective 8 mm diametru |

### Rulmenți

Mișcarea unei axe trebuie să se producă liniar, fără erori de deplasare în alte direcții și cu o rezistența cat mai mică la mișcare. Rulmenții liniari folosiți se vor deplasa pe tijele cromate.

În proiect am folosit și rulmenți 608RS contragreutate, pentru a ghida rotirea șuruburilor trapezoidale ce vor fi menționate în subcapitolul următor. Acest tip de rulmenți se găsesc în popularele jucăriile figget spinner.

* am utilizat 5 rulmenți liniari LM10UU pentru axa X.
* am utilizat 4 rulmenți liniari LM10UU pentru axa Y.
* am utilizat 4 rulmenți liniari LM8UU pentru axa Z.
* am utilizat 3 rulmenți 608RS pentru fiecare axă.

|  |  |
| --- | --- |
| C:\Users\uid41395\Desktop\rulemnt_liniar.jpg  Figura – Rulment liniar | C:\Users\uid41395\Desktop\rulment_contragreutate.png  Figura – Rulment contragreutate 608RS |

### Șuruburi trapezoidale

|  |  |
| --- | --- |
| Pentru mișcarea celor trei axe am folosit trei șuruburi trapezoidale (8 mm diametru, 30cm lungime). O rotație a completă de 360 de grade a acestui șurub mută piulița exact **8mm**. V-om folosi această specificație în formula pentru calcularea distanței de deplasare din subcapitolul 4.1 | C:\Users\uid41395\Desktop\surub-trapezoidal-m8-pas-8mm.jpg  Figura – Șurub trapezoidal |

### Hub de cuplaj flexibil pentru motor

|  |  |
| --- | --- |
| Aceasta piesă face legătura dintre motor și șurubul trapezoidal. Am avut nevoie 3 astfel de piese. | C:\Users\uid41395\Desktop\hub_de_cuplaj.jpg  Figura - Hub de cuplaj flexibil |

### Tije filetate, și piulițe

Tijele filetate si piulițele au avut rolul de prinde a cadrului. Pentru proiect am avut nevoie de următoarele materiale:

* 9x 220mm tije filetate (8mm)
* 1x 370mm tija filetata (8mm)
* 2x 170mm tija filetate (8mm)
* 2x 420mm tije filetate (10mm)
* 56x piulițe (8mm)
* 12x piulițe (10mm)

### Pat suport pentru desenare

Aceasta a fost printre ultimele componente montate la CNC. Am ales o suprafață dintr-un plastic transparent de dimensiunile: 30 cm lungime si 25 cm lățime. Cu toate acestea doar 20 cm de centimetri sunt disponibili pentru spațiul efectiv de lucru.

### Unealta pentru gravat

Pentru a tăia materialul prin deplasările CNC-ului am ales o unealtă funcțională pentru gravat.

# Asamblare

În acest capitol vom discuta despre asamblarea componentelor hardware și vom explica cum este făcută legătura dintre ele și implementarea abstractizării hardware detaliată în subcapitolul 5.2.1.

## Asamblarea componentelor electrice

**<TODO>**

## Asamblarea componentelor mecanice

Pentru componentele mecanice printate am folosit o platformă de CNC disponibilă pe site-ul dedicat *Thingiverse* (1). Linkul acestui proiect se va regăsi în referințe. Nu am insistat pe descrierea părții de asamblare a componentelor mecanice deoarece în linkul din referință se află si un video cu asamblarea scheletului de CNC.

Voi menționa în schimb doar componentele 3D create de mine.

**<TODO: img cu CNC>**

# Limbajul GCODE

Limbajul GCODE(Geometric CODE) este un limbaj de programare foarte popular folosit de imprimantele 3D si CNC-uri. Instrucțiunile limbajului GCODE indică unde și cum o astfel de mașina trebuie să se deplaseze pentru a desena, decupa, sau grava anumite forme în diferite materiale.

În următoarele subcapitole voi discuta cele mai importante instrucțiuni GCODE pe care am ales sa le implementez în *software*-ul de pe Arduino. astfel încât CNC-ul să fie funcțional și să poată executa fișiere GCODE trimis linie cu linie prin intermediul USB-ului.

## Sintaxa limbajului GCODE

O instrucțiune generică în GCODE are următoarea formă, unde **nn** reprezintăun număr întreg sau fracționar, iar parantezele pătrate sugerează faptul că anumite simboluri pot lipsi:

**[Gnn] [Xnn] [Ynn] [Znn] [Fnn] [Mnn]**

* **G -** simbol folosit pentru a selecta o comandă GCODE de mișcare
* **X -** semnifică o poziție de deplasare pe axa X, în planul orizontal
* **Y -** semnifică o poziție de deplasare pe axa Y, în planul vertical
* **Z -** semnifică o poziție de deplasare pe axa Z, in adâncime
* **F -** setează viteza cu care unealta CNC-ului se deplasează, reprezentată în **mm/min**
* **M -** alte funcții diverse

Exemplu de comandă GCODE: **G01 X50.25 Y20.1** **F1000**

Această comanda executa o mișcare de interpolare liniara (**G01**) de către un nou punct având coordonata **X: 50.25** și coordonata **Y: 20.1**, cu o viteza de **1000** de milimetrii pe minut.

Pe lângă acestea, mai sunt și câteva simboluri specifice pentru comenzile **G02** și **G03** despre care voi vorbi în unul dintre următoarele capitole.

Limbajul GCODE nu este unul standard, pot exista anumite variații. De exemplu pentru abordarea mea am ales să folosesc simbolul **#** pentru a afișa diferite mesaje de diagnosticare, de a modifica anumiți parametrii interni, sau de a executa diferite rutine.

## Comenzi GCODE modale

Un aspect foarte important de menționat este faptul că anumite comenzi GCODE sunt modale, în sensul că unele comenzi le pot dezactiva pe cele precedente. Daca o comanda este dintr-un anumit grup, execuția unei comenzi din același grup dezactivează comanda precedenta.

In acest subcapitol voi menționa cele mai importante dintre aceste grupuri de comenzi modale.

### Grupul comenzilor de mișcare

În acest grup se află următoarele comenzi: G00, G01, G02 si G03.

Acesta este cel mai important grup de comenzi. În acest grup fac parte 4 tipuri de mișcări pe care mașina le poate face. Aceste comenzi sunt primitivele de mișcare ale CNC-ului. Cu ajutorul lor se poate efectua orice tip de desen, oricât ce complex ar fi.

* **G00** reprezintă o mișcare liniară a axelor într-un mod sincron. De exemplu dacă suntem la poziția P0(0, 0) și următoarea comandă urmează să fie executată: **G00 X40 Y20**

Atunci se executa mișcarea ca în figura de mai jos. Mai întâi se execută mișcarea primei axe până când noua poziție, în cazul de față **X: 40**, este atinsă. Apoi se executa mișcarea pe cea de-a doua axă până când poziția **Y: 20** a fost atinsă.



Figura – Comanda G00, mișcare liniară simplă

* **G01** reprezintă o mișcare liniară de interpolare. De exemplu dacă suntem la poziția P0(0, 0) și următoarea comandă urmează să fie executată: **G00 X40 Y20**

Atunci se execută o mișcare liniară pe dreapta formată dintre cele doua puncte, punctul P0 și punctul destinație P1(40, 20). Mișcarea obținută va fi echivalentă cu cea din imaginea următoare.



Figura – Comanda G01, interpolare liniară

* **G02** si **G03** reprezintă mișcări de interpolare circulară. Cu ajutorul acestor comenzi se pot crea cercuri sau arcuri de diferite dimensiuni.

Singura diferența dintre G02 si G03 este că comanda G02 produce cercuri **în sensul acelor de ceasornic**, iar comanda G03 produce cercuri **invers acelor de ceasornic**.

Pentru aceste două tipuri de comenzi exista încă alte 3 simboluri esențiale:

* + **I** - distanța incrementală pe coordonata X relativă la centrul cercului
  + **J** - distanța incrementală pe coordonata Y relativă la centrul cercului
  + **R** - raza cercului

Pentru a desena cercul cerut, CNC-ul are nevoie să-și calculeze centru cercului. Exista doua metode prin care putem desena un cerc. Prima metodă este sa trimitem parametrii **I** si **J**, iar cea de-a doua metoda este sa trimitem doar raza cercului prin parametrul **R**, atunci ceilalți doi parametrii pot sa lipsească.

Pentru a desena un arc trebuie să trimitem prin parametrii **X** si **Y** coordonatele celuilalt capăt al arcului. Atunci CNC-ul va trasa un arc începând de la punctul curent pana la punctul destinație. Punctul destinație trebuie neapărat să se situeze pe cerc. In caz contrar CNC-ul va răspunde cu un mesaj de eroare, sau va executa un arc complet de 360 grade.

Presupunem că avem următoarele două cazuri și ne aflăm în punctul **P0(20, 10)**:

1. Executăm comanda: **G02 X30 Y20 I10 J0**
2. Executăm comanda: **G03 X30 Y20 I10 J0**

Pentru primul caz se va desena arcul verde din următoarea imagine, iar pentru cel de-al doilea caz se va desena cercul marcat cu roșu.

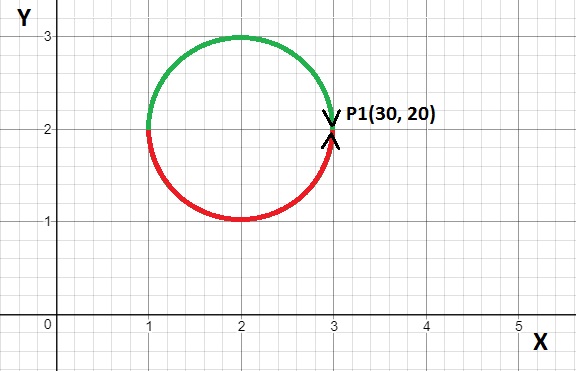


Figura – Interpolare circulară, comenzile G02 și G03

### Grupul comenzilor pentru selectarea planului

Aceste comenzi au impact doar asupra comenzilor de interpolare circulara **G02** si **G03**. Cu ajutorul lor putem alege planul în care dorim să executăm o comandă de interpolare circulară. Exista 3 comenzi pentru selectarea planului:

|  |  |
| --- | --- |
| * G17 – selectează planul **XY**, modul ales implicit * G18 – selectează planul **XZ** * G19 – selectează planul **YZ** | C:\Users\ciprian\AppData\Local\Microsoft\Windows\INetCache\Content.Word\G17_G18_G19.jpg  Figura – Selectarea planului |

### Grupul comenzilor pentru poziționare

Acest grup de comenzi conține două comenzi pentru selectarea modului de poziționare, referitor la cei trei parametrii, X, Y si Z.

Comanda **G90** reprezintă poziționarea absolută a parametrilor X, Y si Z relativ la poziția zero a axelor. De exemplu dacă suntem la poziția P0(10, 10, 10) și avem de executat următoarea comanda: **G01 X20 Y5 Z10** atunci CNC-ul se va poziționa la punctul P1(20, 5, 10).

Comanda **G91** reprezintă poziționarea incrementala a parametrilor X, Y si Z relativ la poziția curenta a axelor. De exemplu dacă suntem la poziția P0(10, 10, 10) și avem de executat următoarea comanda: **G01 X20 Y5 Z10** atunci CNC-ul se va poziționa la punctul P1(40, 15, 20).

### Grupul comenzilor pentru setarea unității de măsură

Grupul acesta este unul dintre cele mai simple, pentru a selecta unitatea de măsură în milimetrii, se trimite comanda **G21,** iar pentru a se opta pentru unitatea de măsură în inch se trimite comanda **G20**. Opțiunea implicit aleasă la inițializarea CNC-ului este cea în milimetrii.

### Comenzi personale rezervate

După cum am menționat și în introducere limbajul GCODE nu este unul standard, pot exista anumite variații. Cu scopul de diagnosticare și testare, am ales să rezerv următoarele comenzi: **G250, G251, G252, G253, G254** si **G255**.

Aceste comenzi le-am tratat ca făcând parte din grupul de comenzi de mișcare, așadar le dezactivează pe cele menționate în primul grup din subcapitolul 3.2.1.

De exemplu am ales să folosesc comanda **G255** ca o versiune mai optimizată a comenzii **G01**, ce lucrează intern cu numere întregi în loc de numere fracționare pentru a eficientiza calculele făcute în algoritmul de interpolare liniară.

## Comenzi GCODE non modale

În subcapitolul precedent am vorbit despre comenzile modale, acestea fac parte dintr-un anumit grup, executarea unei comenzi diferite din același grup dezactivează comanda precedenta.

In acest subcapitol voi discuta despre comenzile non modale.

* Comanda **G04** este o instrucțiune de așteptare, echivalentul unei funcții de *delay(ms)* din programare. Această comandă poate veni însoțită de simbolul **P** urmat de un număr ce specifica timpul in milisecunde de așteptare. Opțional, timpul specificat de așteptare poate fi trecut în parametrul **X**, semnificând același lucru.
* Comanda **G28** este o procedură, in care CNC-ul își deplasează toate axele în poziția zero. Această poziție este detectată cu ajutorul *endstop*-urilor menționate în capitolul de resurse hardware 1.2.5.
* Comanda **G92** este o instrucțiune ce setează poziția curentă a axelor ca fiind poziția zero a axelor. Această comandă este utilă în cazul în care dorim să desenăm ceva în mijlocul axelor patului de desenat.

Exemplu pentru desenarea in mijlocul patului:

1. Executam comanda **G28** pentru a deplasa toate axele la poziția zero.
2. Mutam axele undeva în centrul patului cu comanda: **G01 X100 Y100 Z0**
3. Trimitem comanda G92 pentru a seta poziția curenta ca fiind poziția zero a celor trei axe. CNC-ul va desena relativ la aceasta poziție din mijloc, următoarele mișcări.

## Comenzi GCODE auxiliare

Comenzile auxiliare încep cu simbolul **M**. Câteva dintre cele mai importante comenzi pe care am ales sa le implementez ar fi:

* Comanda **M02** înseamnăsfârșitul scriptului GCODE trimis. Pentru această comandă poate implementa diferite proceduri de distanțare a axelor fața de poziția curentă pentru a putea să avem acces mai ușor la patul de desenare.
* Comanda **M17** dezactivează toate cele 3 motoare pentru a le putea deplasa manual daca este cazul. Daca motoarele sunt dezactivate, atunci toate instrucțiunile de mișcare executate de CNC nu vor avea nici un efect asupra motoarelor, ele vor modifica doar poziția logica interna în care se află CNC-ul.
* Comanda **M18** activează toate cele 3 motoare.

# Algoritmi

În secțiunea curentă voi detalia cei mai importanți algoritmi folosiți în implementarea CNC-ului. Voi încerca să aleg notații cât mai sugestibile pentru a avea o trasabilitate cât mai buna între algoritmi și implementare.

Așadar în primul subcapitol voi discuta cum este calculată distanța reală de la un punct la altul, în cel de-al doilea subcapitol voi vorbi despre cum este simulată viteza motoarelor, iar ultimele subcapitole voi descrie algoritmii pentru cele patru tipuri de mișcări posibile reprezentate în limbajul GCODE de G00, G01, G02 si G03 discutate in subcapitolul 3.2.1.

**Obs:** Deplasarea se execută pas cu pas având o anumită rezoluție fizică între pași, calculată de formula definită în primul subcapitol, vom folosi aceleiași notații peste tot parcursul acestei secțiuni. În algoritmii ce urmează a fi prezentați în ultimele trei subcapitole vom utiliza și următoarele două interfețe pentru deplasarea motoarelor. Vom face abstractizare de implementarea lor. Acestea vor fi explicate în capitolul implementare.

1. **Motor.step(direction)** – deplasează fizic motorul cu un pas, în direcția dată
2. **Motor.getPos()** – returnează poziția curentă a motorului

## Formula pentru calculul distanței fizice de deplasare

CNC-ul primește prin intermediul simbolurilor **X**, **Y**, si **Z** numere fracționare ce vor reprezenta mișcări de deplasare ale celor 3 axe, având unitatea de măsură în milimetri. Pentru a executa o mișcare pe o distanță precisă, avem nevoie sa deducem câteva formule.

În primul rând pentru a deduce cum calculam aceste distanțe, avem nevoie să știm câți milimetrii sunt parcurși pe o axă atunci când CNC-ul deplasează axa exact cu un singur pas al motorului.

Din specificațiile șurubului trapezoidal, din subcapitolul 1.3.4, știm că într-o rotație completă de 360 grade, piulița trapezoidală se deplasează 8 milimetrii.

Din specificațiile motoarelor pas cu pas folosite în acest proiect, din subcapitolul 1.2.3, știm că un motor execută exact 200 de pași egali pentru a efectua o rotație completă, iar din specificațiile *drivere*-lor utilizate, din subcapitolul 1.2.4, știm că avem posibilitatea de a activa diferite opțiuni de *microstepping* mărind astfel numărul de pași făcuți de motor într-o rotație de 360 grade.

Luând în considerare cazul în care nu avem nici o opțiune de *microstepping* activată, atunci putem deduce că pe un pas executat al motorului parcurge distanța de **8mm / 200**, adică **0.04mm**.

Vom nota cu **STEP\_REZOLUTION**, distanța în mm parcursă pe o axa într-un pas al motorului, notăm cu **TRAVEL\_DISTANCE\_360** specificația șurubului trapezoidal, in cazul de fața este 8mm, iar cu **NUMBER\_OF\_STEPS\_360** numărul de pași necesari al unui motor într-o rotație de 360 grade. Generalizând obținem următoarea formulă.

**STEP\_REZOLUTION = TRAVEL\_DISTANCE\_360 / NUMBER\_OF\_STEPS\_360**

Pe baza acestei formule am generat tabelul următor luând în calcul și opțiunile de *microstepping*.

|  |  |  |
| --- | --- | --- |
| Opțiune *microstepping* | NUMBER\_OF\_STEPS\_360 | STEP\_REZOLUTION |
| Fără microstepping | 200 | 0.04 mm |
| 1/2 | 400 | 0.02 mm |
| 1/4 | 800 | 0.01 mm |
| 1/8 | 1600 | 0.005 mm |
| 1/16 | 3200 | 0.0025 mm |
| 1/32 (doar pe DRV8825) | 6400 | 0.00125 mm |

## Simularea vitezei de deplasare

Limbajul GCODE specifică viteza cu care unealta trebuie să se deplaseze atunci când un anumit tip de mișcare este executat. Este important ca unele operațiuni de mișcare ale CNC-ului să se execute la o viteza mai mică. Spre exemplu, dorim să gravăm în metal la o viteza mică deoarece materialul este foarte dur și nu permite prelucrarea lui la viteze mari. În caz contrar dorim să efectuăm și operațiuni de mișcare la viteze mari, spre exemplu atunci când unealta se deplasează fără să atingă materialul, pentru a economisi timp de lucru.

Pentru a putea simula viteza de deplasare avem nevoie să definim mai întâi câteva noțiuni. Viteza de deplasare trimisă ca ***input*** către CNC va fi reprezentă în milimetrii pe minut (mm/min). Vom putea simula viteza motoarelor controlând un timp de așteptare intre pași. Deci ***output***-ul acestui algoritm va fi acest timp de așteptare dintre pași. Cu cât timpul de așteptare este mai mic, viteza de deplasare crește, iar cu cat timpul de așteptare este mai mare viteza de deplasare o să scadă.

Pentru formula vitezei vom avea nevoie și de notațiile definite în subcapitolul precedent, ci anume de **NUMBER\_OF\_STEPS\_360** și de **TRAVEL\_DISTANCE\_360**.

Vom nota cu simbolul **RPM** numărul de rotații ale unui motor pe minut, input-ul problemei îl vom nota cu **SPEED**, iar *output*-ul îl vom nota cu **DELAY**.

Astfel putem calcula, însă avem nevoie sa convertim **RPM** în **RPS**, adică rotații pe secunda:

**RPM = SPEED / TRAVEL\_DISTANCE\_360**

**RPS = RPM / 60**

Avem numărul de rotații pe secunda, acum trebuie să aflăm cu câți pași pe secundă trebuie să ne deplasam astfel încât să atingem viteza dată ca input. Acest rezultat îl obținem în felul următor:

**STEPS\_PER\_SECOND = RPS \* NUMBER\_OF\_STEPS\_360**

Având toate aceste rezultate la dispoziție putem calcula ușor acel timp de așteptare dintre pași pentru a simula viteza dorită. În funcție de unitatea de timp în care dorim să obținem *output*-ul acestui algoritm, aplicăm una dintre formulele de mai jos:

**DELAY\_s = 1 / STEPS\_PER\_SECOND**

**DELAY\_ms = 1.000 / STEPS\_PER\_SECOND**

**DELAY\_us = 1.000.000 / STEPS\_PER\_SECOND**

Așadar în *software*-ul CNC-ului va trebui să implementam o astfel de funcție, care sa calculeze *delay*-ul pentru a simula viteza dorită.

În tabelul următor am trecut câteva valori orientative pentru timpul de așteptare dintre pașii a motoarelor pentru a simula viteza dată ca input. Am introdus în tabel valorile pentru pragul inferior al vitezei minime, 8m/min, o viteză medie, cât ș pragul superior al vitezei. In tabel vor fi calculate *delay*-urile pentru toate opțiunile posibile de microstepping ale driverelor.

|  |  |  |  |
| --- | --- | --- | --- |
| Viteza (mm/min) | RPM | NUMBER\_OF\_STEPS\_360 | DELAY\_us |
| 8 | 1 | 200 | 300 000us = 0.3s |
| 8 | 1 | 400 | 150 000us = 0.15s |
| 8 | 1 | 800 | 75 000us = 75ms |
| 8 | 1 | 1600 | 37500us |
| 8 | 1 | 3200 | 18750us |
| 8 | 1 | 6400 | 9375us |
| 1000 | 125 | 200 | 2040us |
| 1000 | 125 | 400 | 1020us |
| 1000 | 125 | 800 | 510us |
| 1000 | 125 | 1600 | 255us |
| 1000 | 125 | 3200 | 127us |
| 1000 | 125 | 6400 | 63us |
| 2040 | 255 | 200 | 1176us |
| 2040 | 255 | 400 | 588us |
| 2040 | 255 | 800 | 294us |
| 2040 | 255 | 1600 | 147us |
| 2040 | 255 | 3200 | 73us |
| 2040 | 255 | 6400 | 36us |

La viteza minimă simulata, motorul se învârte cu 1 RPM, iar la viteza maximă motorul ajunge undeva aproximativ la 255 RPM. Am ales aceste limite de minim si maxim pentru a proteja driverele si motoarele de supraîncălzire.

## Algoritmul pentru mișcare liniară simplă

Algoritmul pentru mișcarea liniară simplă este cel mai intuitiv si simplu de înțeles din aceste ultime trei subcapitole. Algoritmul va fi utilizat în implementarea comenzii **G00** discutată în subcapitolul 3.2.1.

Avem ca input **p0** punctul curent de pornire si **p1** fiind punctul destinație, iar ca output o deplasare liniară simplă, mai întâi pe axa X până ce p1.x este atins, după care se execută o deplasare pe axa Y până ce p1.y este atins, apoi se deplasează ultima axă până când se ajunge la p1.z.

**INPUT: p0, p1**

**OUTPUT: simple linear movement until p1 is reached**

REVERSE = 0

FORWARD = 1

moveAxis(motor\_x, p1.x)

moveAxis(motor\_y, p1.y)

moveAxis(motor\_z, p1.z)

**moveAxis**(motor, newPos):

**if**(motor.getPos() < newPos):

direction = FORWARD

**else**:

direction = REVERSE

**while**(motor.getPos() != newPos):

motor.step(direction)

## Algoritmul pentru interpolare liniară

Algoritmul din secțiunea curentă va implementa comanda **G01**. Acest algoritm va trebui să mute unealta CNC-ului în spațiul 3D de la poziția curentă la noua poziție trimisă ca input. Astfel rezultatul obținut va fi o mișcare liniară pe segmentul format din cele doua puncte.

Avem ca input **p0** punctul curent de pornire, și **p1** fiind punctul destinație.Output-ul este mișcare de interpolare liniara, de la p0 spre p1 mergând pe ecuația dreptei celor doua puncte.

**INPUT: p0, p1**

**OUTPUT: linear interpolation movement between p0 and p1**

REVERSE = 0

FORWARD = 1

directionVector = (p1.x–p0.x, p1.y–p0.y, p1.z–p0.z)

**if** (directionVector.x < 0): direction\_x = REVERSE

**else**: direction\_x = FORWARD

**if** (directionVector.y < 0): direction\_y = REVERSE

**else**: direction\_y = FORWARD

**if** (directionVector.z < 0): direction\_z = REVERSE

**else**: direction\_z = FORWARD

euclidianDistance = **sqrt**(directionVector.x2 + directionVector.y2 + directionVector.z2)

incrementResolution\_x = (**STEP\_RESOLUTION** / euclidianDistance) \* **abs**(directionVector.x)

incrementResolution\_y = (**STEP\_RESOLUTION** / euclidianDistance) \* **abs**(directionVector.x)

incrementResolution\_z = (**STEP\_RESOLUTION** / euclidianDistance) \* **abs**(directionVector.x)

step\_x = 0, step\_y = 0, step\_z = 0

**while**((motor\_x.getPos()!=p1.x) **OR** (motor\_y.getPos()!=p1.y) **OR** (motor\_z.getPos()!=p1.z) ):

step\_x = step\_x + incrementResolution\_x

step\_y = step\_y + incrementResolution\_y

step\_z = step\_z + incrementResolution\_z

**if** (step\_x >= **STEP\_RESOLUTION**):

motor\_x.step(direction\_x)

step\_x = step\_x - STEP\_RESOLUTION

**if** (step\_y >= **STEP\_RESOLUTION**):

motor\_y.step(direction\_y)

step\_y = step\_y - STEP\_RESOLUTION

**if** (step\_z >= **STEP\_RESOLUTION**):

motor\_z.step(direction\_z)

step\_z = step\_z - STEP\_RESOLUTION

Ideea algoritmului este de a găsi cu cât trebuie să incrementam individual pe fiecare axă astfel încât să ne deplasăm pe linia dintre cele două puncte.

Pentru că nu putem să ne deplasăm mai puțin decât rezoluția fizică a motoarelor, vom incrementa cu rezoluția calculată individual pe fiecare axă, până când putem face un pas fizic, adică depășim valoarea constantei STEP\_RESOLUTION. Din acest motiv, o astfel de linie va fi desenată în ca în imaginea din figura următoare.

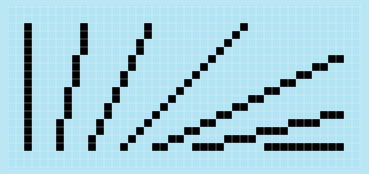


Figura – Aproximare interpolare liniară

Însă prin faptul că rezoluția dintre pași, STEP\_RESOLUTION, are o valoare foarte mică acest lucru va fi insesizabil.

## Algoritmul pentru interpolare circulară

In acest ultim subcapitol vom detalia algoritmul de interpolare circulară pentru desenarea cercurilor. Acest algoritm va implementa comenzile **G02** si **G03**. Vom desena cercul

Vom avea nevoie de ecuația cercului: **(x − center.x)2 + (y − center.y)2 = R2**. Rescriem ecuația în următoarea forma: **FXY = (x − center.x)2 + (y − center.y)2 - R2.**

FXYne va da informația dacă punctul curent în care ne aflăm este pe ecuația cercului sau nu. Daca FXY< 0, atunci punctul curent este în interiorul cercului, v-a trebui sa ne deplasam spre exterior, iar daca FXY > 0, atunci punctul curent este în interior și va trebui să ne deplasam spre interior.

**Obs:** Așadar acest algoritm de fapt aproximează interpolarea circulară, cercul desenat de CNC nu va fi unul ideal în care toate punctele prin care trecem sa fie pe ecuația cercului, însă aceste aproximări vor fi insesizabile.

În continuare avem nevoie să știm unde ne situăm pe cerc relativ la centru. In variabila **A** vom reprezenta cu 0 dacă suntem în stânga cercului, iar cu 1 dacă suntem în dreapta. În variabila **B** vom reprezenta cu 0 dacă suntem jos, respectiv 1 daca suntem sus.

Combinând cazurile de mai sus FXY cu A si B, vom avea 8 cazuri posibile de tratat în pentru luarea unei decizii de deplasare pe axele X si Y. Luând și posibilitatea în care avem de trasat cu o anumită orientare (in sensul acelor de ceasornic sau invers), atunci avem în total 16 cazuri posibile de luat în considerare.

Având toate aceste lucruri în vedere putem trece la algoritm. Vom avea ca input centrul cercului, raza acestuia, precum și direcția în care trasăm cercul (in sensul acelor de ceasornic sau invers). Output-ul este o mișcare circulară, aproximată la ecuația cercului.

**INPUT: p0, R, center, DIRECTION (0 for clockwise, 1 for counter clockwise)**

**OUTPUT: circular movement**

REVERSE = 0

FORWARD = 1

step\_x = 0;

step\_y = 0;

p = p0

**do:**

// calculam daca punctual curent se afla in interiorul sau exteriorul cercului

FXY = (p.x + center.x)2 + (p.y+center.y)2 – R2

**if**(FXY < 0): F = 0

**else:** F = 1

// calculam cadranul in care se afla punctul current

**if**(p.x < center.x): A = 0

**else:** A = 1

**if**(p.y < center.y): B = 0

**else:** B = 1

decide\_steps(DIRECTION, F, A, B)

if(step\_x == -1): motor\_x.step(REVERSE)

if(step\_x == 1): motor\_x.step(FORWARD)

if(step\_y == -1): motor\_y.step(REVERSE)

if(step\_y == 1): motor\_y.step(FORWARD)

**while**((motor\_x.getPos()!=p1.x) **OR** (motor\_y.getPos()!=p1.y) **OR** (motor\_z.getPos()!=p1.z) )

**decide\_steps**(DIR, F, A, B):

// compute if we step on x direction or y direction

decision\_binary = ((DIR<<3) | (F<<2) | (A<<1)) + B;

**switch**(decision\_binary)

{

// clockwise directions

case 0: step\_x = -1; break; // (clockwise, interior, left, lower)

case 1: step\_y = 1; break; // (clockwise, interior, left, upper)

case 2: step\_y = -1; break; // (clockwise, interior, right, lower)

case 3: step\_x = 1; break; // (clockwise, interior, right, upper)

case 4: step\_y = 1; break; // (clockwise, exterior, left, lower)

case 5: step\_x = 1; break; // (clockwise, exterior, left, upper)

case 6: step\_x = -1; break; // (clockwise, exterior, right, lower)

case 7: step\_y = -1; break; // (clockwise, exterior, right, upper)

// counter clockwise directions

case 8: step\_y = -1; break; // (!clockwise, interior, left, lower)

case 9: step\_x = -1; break; // (!clockwise, interior, left, upper)

case 10: step\_x = 1; break; // (!clockwise, interior, right, lower)

case 11: step\_y = 1; break; // (!clockwise, interior, right, upper)

case 12: step\_x = 1; break; // (!clockwise, exterior, left, lower)

case 13: step\_y = -1; break; // (!clockwise, exterior, left, upper)

case 14: step\_y = 1; break; // (!clockwise, exterior, right, lower)

case 15: step\_x = -1; break; // (!clockwise, exterior, right, upper)

}

Funcția ***decide\_steps*** ia în calcul toate cele 16 cazuri discutate mai sus și va decide în care dintre cele doua axe, X sau Y, trebuie să se execute un pas și în ce direcție.

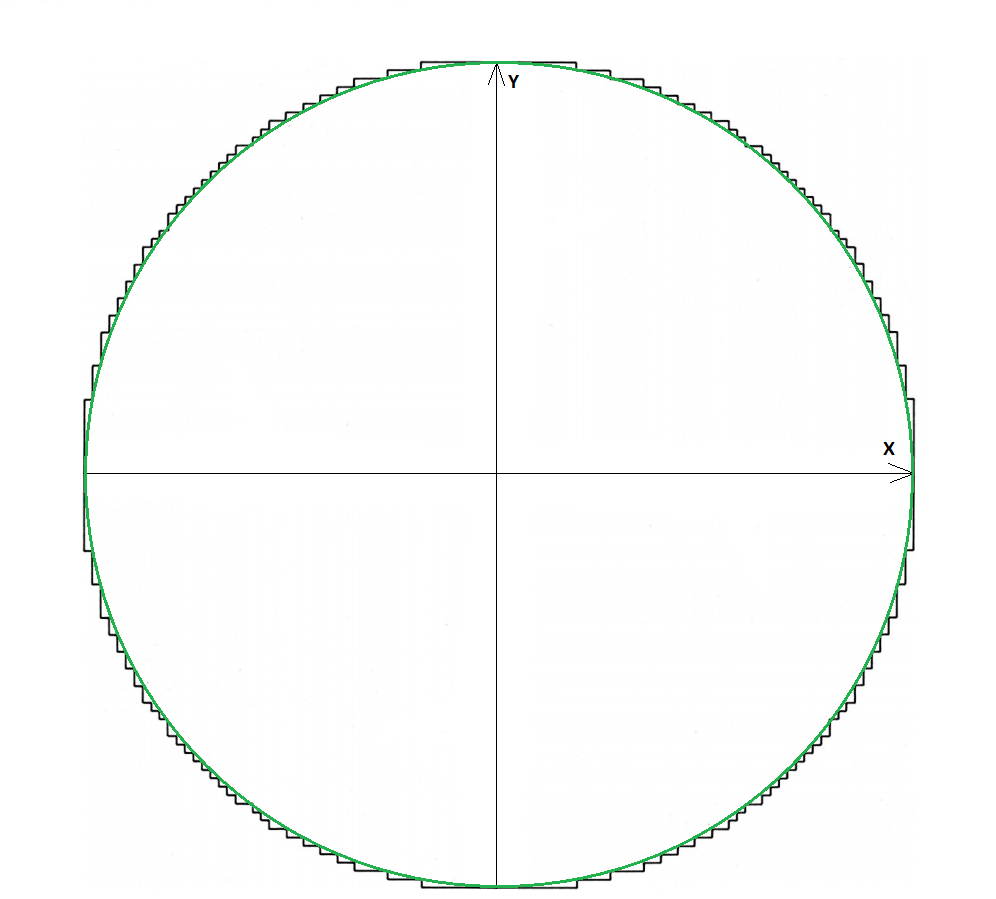


Figura – Aproximare a mișcarii circulare pentru desenarea unui cerc

Astfel algoritmul se va deplasa ca în imaginea din figura anterioară. Cu verde am reprezentat un cerc ideal, iar cu negru vedem cum se va comporta algoritmul în desenarea cercului. Ca și în cazul liniilor, acest aspect va fi insesizabil.

# Implementare

Având toate aceste noțiuni detaliate în capitolele precedente, putem trece la partea de implementare.

Dorim ca CNC-ul să primească comenzi prin intermediul portului USB. Putem trimite comenzi prin USB folosind diverse aplicații de tip *Serial Terminal*, gen PuTTY, însă am ales să implementez un script in Python pentru a face acest lucru. Cu ajutorul acestui script voi avea posibilitatea să trimit și fișiere întregi către CNC.

Comenzile vor fi trimise linie cu linie. Dacă comanda trimisă este validă atunci aceasta va fi analizată și executată.

In primul subcapitol vom discuta despre arhitectura aplicației

## Arhitectura generală a aplicației

Următoarea diagramă reprezintă descompunerea funcțională minimalistă a aplicației.

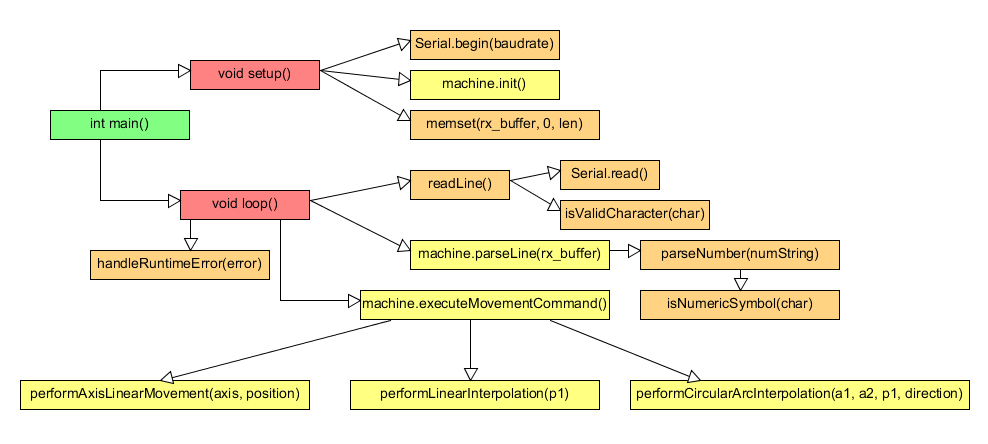


Figura – Diagrama de descompunere funcțională

În punctul de intrare al aplicației, funcția *main*, se apelează cele doua funcții specifice platformei Arduino, *setup* și *loop*.

În funcția *setup* se inițializează obiectul Serial folosit pentru comunicația prin USB. În comunicația serial trebuie stabilită viteza cu care se desfășoară comunicația în ambele *endpoint*-uri. Viteza se specifică prin baud rate, adică numărul de biți trimiși pe secundă. Vom avea în vedere ca același baud rate folosit în *software*-ul CNC-ului, să fie folosit și pe scriptul făcut în *Python*.

In funcția de *setup* se mai inițializează și o instanță a clasei *Machine*. Vom vorbi despre această clasă în următorul subcapitol.

De asemenea se inițializează și *buffer-*ul de recepție a comunicației, aici vor fi stocate temporar datele primite prin USB.

Funcția *loop* verifică dacă conexiunea serial este activă. În caz contrar se microcontrolerul se resetează. Dacă conexiunea este activă se așteaptă primirea unei linii complete în buffer. Primirea unei linii este semnalată printr-un *flag*. Când *flagul* este setat se apelează funcția de validare și analizare, *parseLine*.

Funcția *parseLine* extrage instrucțiunile GCODE din linia primită si creează o noua instanță a unei comenzi. Dacă este semnalată o instrucțiune de schimbare a poziției în care se afla unealta CNC-ului, atunci începe o procedură de deplasare conform tipului de mișcare pe care trebuie să-l executăm. După ce procedura de deplasare se termina, CNC-ul va trimite răspuns înapoi dacă acțiunea s-a terminat cu succes sau dacă a intervenit o eroare.

Înainte de a intra în alte detalii voi prezenta în următoarea imagine diagrama de compoziție a claselor:

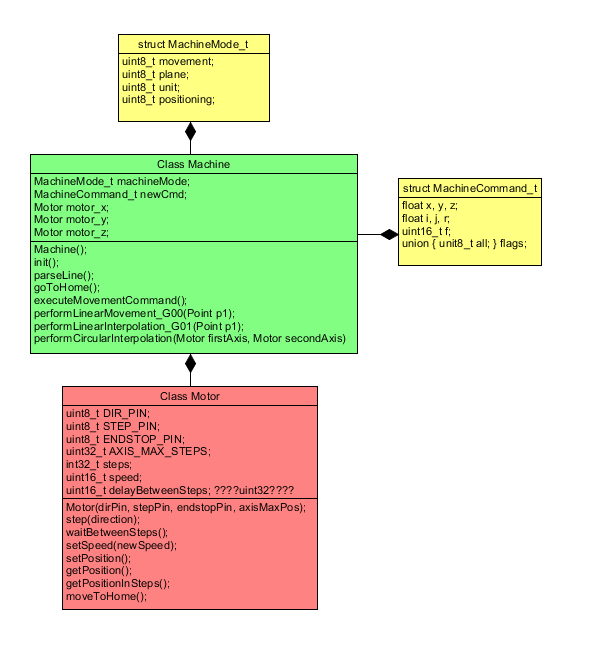


Figura – Diagrama claselor

### Clasa Machine

Clasa **Machine** este clasa nucleu a aplicației. Această clasă va fi de tipul *singleton*, deci va exista o singură instanță ce va fi inițializată la pornirea aplicației. Aici vor fi ținute stările interne ale mașinii, grupurile de comenzi activate, comanda curentă de procesat, precum și cele trei instanțe pentru fiecare motor.

Am discutat despre comenzile modale ale limbajului GCODE în subcapitolul 3.2. Pentru a memora în ce moduri de operare se află CNC-ul am creat următoarea structura:

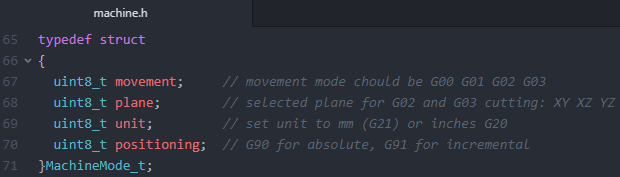


Figura – Structura pentru starea comenzilor modale

Atunci când primim comenzi de deplasare a uneltei CNC-ului, acestea vor fi executate în modurile selectate din această structură.

De exemplu fie următoarea secvență de instrucțiuni trimise CNC-ului:

1. **G01** – se alege modul de mișcare în interpolare liniară
2. **G90** – se alege modul de poziționare absolută a coordonatelor
3. **G21** – se alege unitatea de măsura în milimetrii
4. **X50.10 Y25.90** – mutam unealta la punctul (50.10mm, 25.90mm) folosind poziționarea absolută

În clasa **Machine** avem și comanda curentă de deplasare ce urmează a fi executată. O comandă de deplasare este implementă intern ca fiind o instanță a următoarei structuri de date:

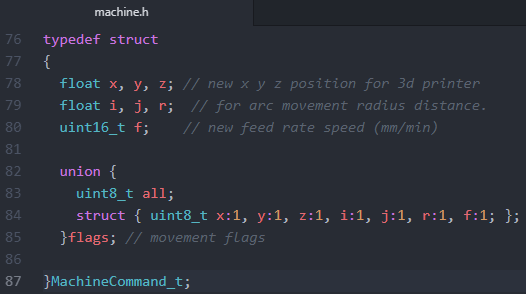


Figura – Structura interna a unei comenzi de deplasare

Aici avem valori pentru toate simbolurile GCODE ce indică coordonate în spațiu. Respectiv **x, y, z**, pentru deplasările în cele trei planuri, și simbolurile **i, j, r,** necesare pentru mișcările circulare G02 si G03. Pe lângă mai avem câmpul **f** pentru a indica viteza de deplasare.

Pentru toate acestea avem o listă de *flag*-uri care indică ce simboluri am primit în comanda de deplasare curentă. Cu ajutorul lor vom lua decizia daca ne deplasăm într-o singură dimensiune, în 2D, sau în 3D. Pentru cercuri vom lua decizia dacă avem de trasat un cerc complet sau doar un arc. Toate aceste 7 *flag*-uri ocupă un singur byte.

Ultimele trei câmpuri din clasa **Machine** sunt cele trei motoare de pe fiecare axă. Poziția curentă a uneltei CNC-ului este stocată intern în aceste trei instanțe.

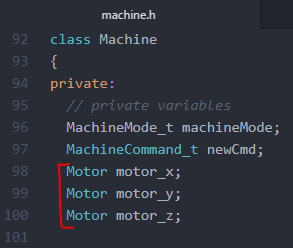


Figura – Câmpurile din clasa Machine

### Clasa Motor

Clasa **Motor** vafi folosită pentru a reprezenta câte o instanță a unui motor de pe fiecare axă. În această clasă este implementată abstractizarea hardware a motorului.

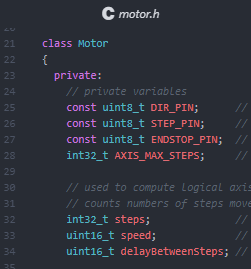


Figura – Câmpurile din clasa Motor

În imaginea din figura de mai sus avem variabilele interne folosite.

* Variabila **DIR\_PIN** este pinul conectat la inputul DIR de pe driverul de motoare. Dacă setăm pe valoarea HIGH[[7]](#footnote-7), atunci motorul face un pas intr-o direcție, iar pe LOW[[8]](#footnote-8) merge în direcția opusă.
* Variabila **STEP\_PIN** reprezintă pinul cu ajutorul căruia putem da comandă motoarelor prin intermediul driverului, să execute un pas. Făcând o tranziție de pe HIGH pe LOW, sau invers, motorul execută un pas. Adăugând și un delay între aceste tranziții, putem simula efectul de viteză al motoarelor.
* Variabila **ENDSTOP\_PIN** reprezintă pin-ul pentru *switch*-ul *endstop*. Dacă valoarea citită de pe acest pin este LOW, atunci înseamnă că unealta a ajuns la poziția de sfărșit inferior al axei.
* Variabila **AXIS\_MAX\_STEPS** reprezintă numărul maxim de pași ce pot fi făcuți pe o axă pentru a ajunge din poziția 0 pâna în celalalt capăt.
* Variabila **steps** reprezintă poziția în pași relativ la poziția zero a axei. Inițial am ales să reprezint poziția motorului ca fiind distanța față de poziția zero a axei prin reprezentată intern prin intermediul unei variabile de tip *float*. M-am lovit de o problemă pe care am detaliat-o în subcapitolul 5.4, așa că am implementat poziția motorului ca fiind distanța în pași făcuți față de poziția zero a axei reprezentată intern prin intermediul unei variabile de tip int pe 32 de biți.
* Variabila **speed** va reprezenta viteza de deplasare a motorului, reprezentată în **mm/min**
* Variabila **delayBetweenSteps** este *delay*-ul calculat cu ajutorul formulei din subcapitolul 4.2, va fi utilizată pentru simularea vitezei de deplasare a uneltei.

Pentru a efectua pași cu un motor avem expusă interfața publică a motorului

## Fișierele de configurație

**<TODO>**

### Configurația Hardware

**<TODO>**

### Configurația CNC-ului

**<TODO>**

## Implementarea interpretorului de comenzi

**<TODO>**

## Problema preciziei a numerelor fracționare

Implementând poziționarea uneltei CNC-ului, am ales să reprezint poziția curentă prin tipul de date *float*. Am implementat poziția curentă într-o anumită axă ca fiind distanța față de poziția zero a axei.

Când poziția într-o axă se modifica cu un pas incrementam sau decrementam poziția curentă cu valoarea constantei **STEP\_RESOLUTION** în funcție de direcția în care mergeam.

Luăm doar cazul în care trebuia să mă deplasez la o poziție pozitivă mai mare decât poziția curentă. Deci ca să ajungem la noua poziție trebuia sa incrementez poziția curentă cu valoarea constantei **STEP\_RESOLUTION** pană când ajungem la destinație.

Pentru distanțe relativ mici nu am sesizat nici un fel de probleme. Însă deplasând unealta pe distanțe mai mari am constatat că uneori executa mai mulți sau mai puțini pași decât mă așteptam să execute.

Problema se intensifica atunci când am activat opțiunile de *microstepping*. Activând microstepping rezoluția dintre pași devine mai mică, deci ca sa parcurg o anumită distanță numărul pașilor creste. Făcând mai mulți pași, implicit asta înseamnă că trebuia să incrementez de mai multe ori cu constanta **STEP\_RESOLUTION** de fiecare dată când făceam un pas.

Problema apare de la precizia numerelor *float*. În tabelul următor sunt prezentate care sunt de fapt valorile exacte stocate în acest tip de date pentru toate cazurile de microstepping.

|  |  |  |
| --- | --- | --- |
| STEPPS | STEP\_RESOLUTION | Reprezentarea exactă în float |
| 200 | 0.04 | 0.039999999106 |
| 400 | 0.02 | 0.019999999553 |
| 800 | 0.01 | 0.009999999776 |
| 1600 | 0.005 | 0.004999999888 |
| 3200 | 0.0025 | 0.002499999944 |
| 6400 | 0.00125 | 0.001249999972 |

Așadar, pentru a scăpa de această problemă am ales reprezint poziția curentă pe o anumită axă ca fiind distanța în pași relativ la poziția zero a axei. Am stocat această aceasta poziție într-o variabila de timp int32.

De data aceasta de fiecare dată când o sa efectuam un pas, pentru a actualiza și poziția în loc să incrementăm sau sa decrementam cu valoarea rezoluției unui pas, o să incrementam cu 1 pentru un pas înainte, respectiv cu -1 pentru un pas înapoi.

Pentru a putea afla ce distanță maximă putem reprezenta în aceasta abordare am împărțit valoarea maxima posibilă ce încape într-un int32 la cazul de microstepping cel mai mare posibil, adică 6400 de pași pe rotație.

Astfel am obținut 335544. Acest număr reprezintă numărul maxim de rotații ce poate fi contorizat în această abordare. Stim că o rotație completă inseamnă 8mm, deci facând o înmultimre obtinem 2.684.352mm, adică 2684metri.

fiind suficient pentru a putea reprezenta distanțe fizice de până la

# Utilizare

Pentru utilizarea CNC-ului avem nevoie să convertim imaginea dorită în fișier GCODE.

**<TODO>**

# Concluzii

Memoria utilizată

**<TODO>**

[Figura 1 – Arduino UNO 5](#_Toc536787532)

[Figura 2 – CNC shield 6](#_Toc536787533)

[Figura 3 – Motor Pas cu Pas Nema17 7](#_Toc536787534)

[Figura 4 – Driver A4988 8](#_Toc536787535)

[Figura 5 – Driver DRV8825 8](#_Toc536787536)

[Figura 6 – Endstop switch 9](#_Toc536787537)

[Figura 7 – Sursă aliminetare 10](#_Toc536787538)

[Figura 8 – Tije cromate de 10, respective 8 mm diametru 11](#_Toc536787539)

[Figura 9 – Rulment liniar 11](#_Toc536787540)

[Figura 10 – Rulment contragreutate 608RS 11](#_Toc536787541)

[Figura 11 – Șurub trapezoidal 12](#_Toc536787542)

[Figura 12 - Hub de cuplaj flexibil 12](#_Toc536787543)

[Figura 13 – Comanda G00, mișcare liniară simplă 16](#_Toc536787544)

[Figura 14 – Comanda G01, interpolare liniară 17](#_Toc536787545)

[Figura 15 – Interpolare circulară, comenzile G02 și G03 18](#_Toc536787546)

[Figura 16 – Selectarea planului 18](#_Toc536787547)

[Figura 17 – Aproximare interpolare liniară 25](#_Toc536787548)

[Figura 18 – Diagrama de descompunere funcțională 28](#_Toc536787549)

[Figura 19 – Diagrama claselor 30](#_Toc536787550)

[Figura 20 – Structura pentru starea comenzilor modale 31](#_Toc536787551)

[Figura 21 – Structura interna a unei comenzi de deplasare 31](#_Toc536787552)

[Figura 22 – Câmpurile din clasa Machine 32](#_Toc536787553)

[Figura 23 – Câmpurile din clasa Motor 33](#_Toc536787554)

# Bibliografie

1. **AUTODESK.** Getting started with G-Code. [Online] https://www.autodesk.com/industry/manufacturing/resources/manufacturing-engineer/g-code.

2. **MICROCHIP.** ATmega328p Datasheet. [Online] https://www.microchip.com/wwwproducts/en/ATmega328p.

3. **ARDUINO.** Arduino Reference. [Online] https://www.arduino.cc/reference/en/.

4. **Ken Goldberg, Melvin Goldberg.** XY Interpolation Algorithms. *Robotics Age.* 1983.

5. **Javier, Arnedo.** Homemade CNC. *Thingiverse.* [Online] 2018. https://www.thingiverse.com/thing:2794509.

1. CNC acronimul vine de la expresia în engleză: Computer Numerical Control [↑](#footnote-ref-1)
2. OOP este o paradigma de programare orientată pe obiecte [↑](#footnote-ref-2)
3. Sistem *embedded* este o combinație dintre hardware si software ce îndeplinește o funcție în timp real [↑](#footnote-ref-3)
4. *Low-level* se referă la programarea cât mai aproape de detaliile hardware ale platformei [↑](#footnote-ref-4)
5. *High-level* se referă la programarea la un nivel cât mai abstractizat de detaliile hardware [↑](#footnote-ref-5)
6. Homing este termenul folosit pentru deplasarea uneltei în pozițiile 0 ale celor trei axe. [↑](#footnote-ref-6)
7. HIGH este notația în implementare pentru valoarea digitală 1 [↑](#footnote-ref-7)
8. LOW este notația în implementare pentru valoarea digitală 0 [↑](#footnote-ref-8)