

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Router CNC Arduino

propusă de

Ciprian Ștefănel Andrei

Sesiunea: *Februarie, 2019*

Coordonator științific

Lect. Dr. Cosmin Vârlan

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

Router CNC Arduino

Ciprian Ștefănel Andrei

Sesiunea: *Februarie, 2019*

Coordonator științific

Lect. Dr. Cosmin Vârlan

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP,
absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de
..... specializarea, promoția
....., declar pe propria răspundere, cunoscând consecințele falsului în
declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr.
1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul:

_____elaborată sub îndrumarea dl. / d-na
_____, pe care urmează să o susțină în fața
comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin
orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la
introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări
științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei
lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie

răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Router CNC Arduino*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, 07.02.2019

Absolvent *Ciprian Ștefanel Andrei*

Introducere

Pentru lucrarea de licență mi-am propus să fac un *router* CNC¹ 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.

Există și alte aplicații software similare, însă am venit cu ceva diferit prin faptul că am ales să scriu aplicația într-o manieră OOP², 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 și utilizarea CNC-ului. Voi discuta ce rol are fiecare componentă în parte și voi prezenta specificațiile tehnice ale unor componente mai importante.

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 discuta despre limbajul GCODE, mai specific despre sintaxa și semantica instrucțiunilor acestuia. Voi aduce în discuție doar comenzile pe care le voi implementa în aplicația software.

În prima parte a capitolului 4 vor fi explicate formulele folosite pentru calcularea distanței și simularea vitezei, iar în cea de-a doua parte vor fi detaliați algoritmi 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 despre modul de utilizare al CNC-ului, iar în ultimul capitol vom trage câteva concluzii.

¹ CNC acronimul ce vine în engleză de la: Computer Numerical Control

² OOP este o paradigmă de programare orientată pe obiecte

Cuprins

Introducere	1
1 Resurse utilizate	4
1.1 Software	4
1.1.1 Aplicația CNC în Arduino	4
1.1.2 Aplicația de comunicație cu Arduino	4
1.1.3 Unelte pentru conversia imaginilor.....	4
1.2 Hardware	5
1.2.1 Microcontroler	5
1.2.2 CNC <i>shield</i>	6
1.2.3 Motoare pas cu pas.....	7
1.2.4 Drivere pentru motoarele pas cu pas.....	7
1.2.5 <i>Endstop</i> -uri pentru limitarea axelor	9
1.2.6 Sursă de alimentare	9
1.3 Mecanice	9
1.3.1 Componente printate din plastic	10
1.3.2 Tijă cromată	10
1.3.3 Rulmenți.....	10
1.3.4 Șuruburi trapezoidale	11
1.3.5 <i>Hub</i> de cuplaj flexibil pentru motor.....	11
1.3.6 Tijă filetată și piulițe	11
1.3.7 Pat suport pentru desenare	11
1.3.8 Unealtă CNC.....	11
2 Asamblare	12
2.1 Asamblarea componentelor electrice	12
2.2 Asamblarea componentelor mecanice.....	13
2.3 Starea finală platformei mecanice	14
3 Limbajul GCODE	15
3.1 Sintaxa limbajului GCODE.....	15
3.2 Comenzi GCODE modale.....	15
3.2.1 Grupul comenzilor de mișcare	16

3.2.2	Grupul comenzilor pentru selectarea planului	18
3.2.3	Grupul comenzilor pentru poziționare	19
3.2.4	Grupul comenzilor pentru setarea unității de măsură	19
3.2.5	Comenzi personale rezervate	19
3.3	Comenzi GCODE non modale	19
3.4	Comenzi GCODE auxiliare.....	20
4	Algoritmi.....	21
4.1	Formula pentru calculul distanței fizice de deplasare	21
4.2	Simularea vitezei de deplasare	22
4.3	Algoritmul pentru mișcare liniară simplă	24
4.4	Algoritmul pentru interpolare liniară	24
4.5	Algoritmul pentru interpolare circulară.....	26
5	Implementare	29
5.1	Arhitectura generală a aplicației.....	29
5.1.1	Clasa <i>Machine</i>	31
5.1.2	Clasa Motor.....	33
5.2	Fișierele de configurație	36
5.2.1	Configurația Hardware.....	36
5.2.2	Configurația CNC-ului.....	37
5.3	Implementarea comunicației	37
5.4	Problema preciziei a numerelor fracționare	38
5.5	Implementarea aplicației pentru comunicația USB.....	39
6	Utilizare.....	40
6.1	Pregătirea CNC-ului pentru desenare.....	40
6.2	Pregătirea CNC-ului pentru a sculpta/grava.....	41
7	Concluzii	43
7.1	Îmbunătățirea protocolului de comunicație.....	43
7.2	Îmbunătățirea distanței dintre axa X și de patul de desenare	44
8	Bibliografie	46

1 Resurse utilizate

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

1.1 Software

În această secțiune sunt menționate resursele software folosite pentru realizarea CNC-ului în Arduino, cât și cele folosite pentru realizarea 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.

1.1.1 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 și una dintre cele mai bune alegeri pentru programarea sistemelor *embedded*³. Oferă toate facilitățile *low-level*⁴ ale limbajului C, cât și posibilitatea de a programa la un nivel cât mai *high-level*⁵ folosind concepte de programare orientată pe obiecte.

Compilerul folosit pentru a transforma limbajul C++ în limbaj mașină pentru microcontrolerul Atmega328p este *avr-g++*. Acesta suportă aproape toate *feature*-ile limbajului C++, însă fără posibilitatea utilizării conceptului de Excepții.

Pentru operațiile de input/output pe pini cât și pentru partea de comunicație cu portul serial, am folosit biblioteca oferită de Arduino (1) și biblioteca AVR (2) a microcontrolerului.

1.1.2 Aplicația de comunicație cu Arduino

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

1.1.3 Unelte pentru conversia imaginilor

Pentru convertirea imaginilor în instrucțiuni GCODE am testat mai multe *tool*-uri. Cel mai des am folosit aplicația **Inkscape** pentru a transforma imaginea în formatul SVG, și aplicația online **JSCut** pentru a converti imaginea SVG într-un fișier GCODE.

Există însă o gamă mai largă de posibilități pentru a realiza acest lucru.

³ Sistem *embedded* este o combinație dintre hardware și software ce îndeplinește o funcție în timp real

⁴ *Low-level* se referă la programarea cât mai aproape de detaliile hardware ale platformei

⁵ *High-level* se referă la programarea la un nivel cât mai abstractizat de detaliile hardware

1.2 Hardware

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

1.2.1 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 1 – Arduino UNO

Specificații tehnice generale:

- frecvență microcontroler: **16Mhz**
- memorie flash: **32KB**
- memorie SRAM: **2KB**
- memorie EEPROM: **1KB**
- pini digitali I/O: **14**
- pini analogici: **6**
- tensiune alimentare placă: **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 în subcapitolul 1.2.2. Acest *shield* simplifică foarte mult conexiunile fizice dintre motoarele pas cu pas și *driver*-ele pentru motoare despre care voi discuta în subcapitolul 1.2.4.

Un alt avantaj este dat de faptul că interfațarea dintre laptop și 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 că *upload*-area software-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 în ajutorul acestei probleme platforma Arduino vine cu trimiterea

mesajelor de *debug* prin intermediul portului USB. Fapt ce rezolvă 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 conversie a semnalului USB către interfața serial a microcontrolerului.

Iar nu în ultimul rând legat de costuri, această platformă împreună cu celelalte componente hardware, *shield*-ul CNC și *driver*-ele pentru motoare, sunt o soluție relativ ieftină din punct de vedere financiar, iar cerințele hardware pentru implementarea CNC-ului sunt suficiente.

1.2.2 CNC shield

În figura următoare am atașat o imagine cu *shield*-ul pentru CNC folosit în acest proiect. Voi face referire la această imagine în capitolul următor de asamblare la partea de hardware.

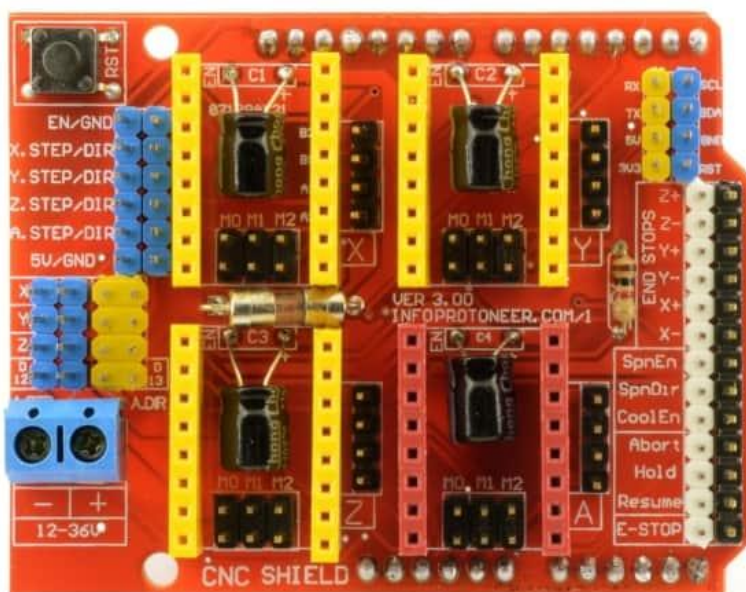


Figura 2 – CNC shield

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 și Z, respectiv 3 motoare pas cu pas. În plus se mai poate monta încă un driver și 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 semnaliza sfârșitul axei, poziția zero a unei axe sau celălalt capăt.

Tensiunea de alimentare pentru acest *shield* este între 12 și 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 maximă și 200mA curentul maxim).

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.

1.2.3 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.

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 axă, X, Y, respectiv axa Z. Modelul pe care l-am ales este: Motor pas cu pas **17HS8401S** Nema17.



Specificații tehnice:

- pas: **1.8grade, adică 200 pași in 360 grade**
- cuplu: **0.59Nm**
- curent: **1.7 A**
- dimensiuni: **specifice standardului Nema17**

Figura 3 – Motor Pas cu Pas Nema17

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

1.2.4 Drivere pentru motoarele pas cu pas

Driver-ele 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 și se poate controla direcția în care motorul se deplasează, precum și 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 opțiunilor de *microstepping* se 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*.

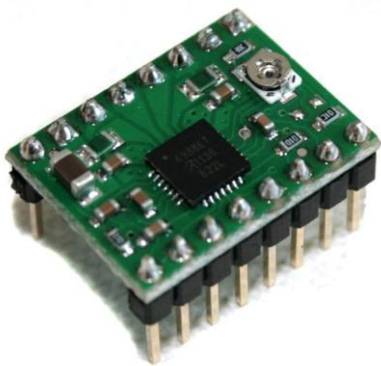


Figura 4 – Driver A4988

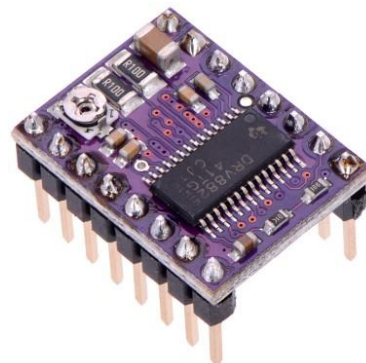


Figura 5 – Driver DRV8825

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.

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

1.2.5 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 a uneltei.

Am folosit trei astfel de *switch*-uri, câte unul pentru fiecare axă.

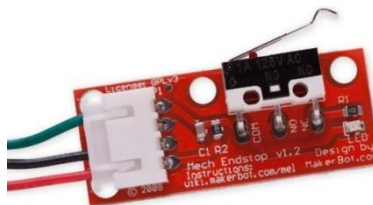


Figura 6 – Endstop switch

1.2.6 Sursă de alimentare

Shield-ul CNC primește ca input o tensiune de 12-36V. Acest curent de intrare va alimenta motoarele. Un asemenea motor pas cu pas consuma maxim 1.7A, deci avem nevoie de o sursă care să fie capabilă să furnizeze un curent de minim aproximativ 6A la tensiunea de 12V, adică o putere de minim 70W. Pentru acest proiect am ales o sursă ATX de calculator cu specificațiile din imaginea alăturată:

Figura 7 – Sursă aliminetare

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

1.3 Mecanice

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

⁶ Homing este termenul folosit pentru deplasarea uneltei în pozițiile 0 ale celor trei axe.

1.3.1 Componente printate din plastic

Componentele printate din plastic se pot găsi în *link*-ul din referință (3). Acestea formează scheletul mecanic al CNC-ului.

1.3.2 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 38cm lungime
- pentru axa Y am folosit 2 tije cromate (10mm diametru) de 40cm lungime
- pentru axa Z am folosit 2 tije cromate (8mm diametru) de 20cm lungime



Figura 8 – Tije cromate de 10, respective 8 mm diametru

1.3.3 Rulmenți

Mișcarea unei axe trebuie să se producă liniar, fără erori de deplasare în alte direcții și cu o rezistență cât 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 ușor, mai ales în popularele jucării *fidget spinner*.

Așadar am avut nevoie de:

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



Figura 9 – Rulment liniar



Figura 10 – Rulment contragreutate 608RS

1.3.4 Șuruburi trapezoidale

Pentru mișcarea celor trei axe am folosit trei șuruburi trapezoidale (8 mm diametru, 30cm lungime). O rotație completă de 360 de grade a acestui șurub mută piulița exact **8mm**. Vom folosi această specificație în formula pentru calcularea distanței de deplasare din subcapitolul 4.1.



Figura 11 – Șurub trapezoidal

1.3.5 Hub de cuplaj flexibil pentru motor

Aceasta piesă face legătura dintre motor și șurubul trapezoidal. Am avut nevoie de 3 astfel de piese.



Figura 12 - Hub de cuplaj flexibil

1.3.6 Tijele filetate și piulițe

Tije filetate și piulițele au avut rolul de prindere a cadrului. Pentru proiect am avut nevoie de următoarele materiale:

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

1.3.7 Pat suport pentru desenare

Acesta a fost printre ultimele componente montate la CNC. Am ales o suprafață dintr-un plastic transparent de 30cm lungime și 25cm lățime. Cu toate acestea doar 20cm vor fi disponibili pentru spațiul efectiv de lucru.

1.3.8 Unealtă CNC

Această parte este opțională. Pentru a desena vom vedea în capitolul de asamblare că putem atașa diferite tipuri de pixuri, însă pentru a grava am ales o unealtă multifuncțională.

2 Asamblare

În acest capitol vom discuta despre asamblarea componentelor hardware, despre unele modele 3D printate și despre starea finală a platformei mecanice pentru acest proiect.

2.1 Asamblarea componentelor electrice

În imaginile următoare avem toate componentele hardware asamblate împreună cu o parte din schematicul hardware ce va fi utilă pentru maparea pinilor în capitolul de implementare.

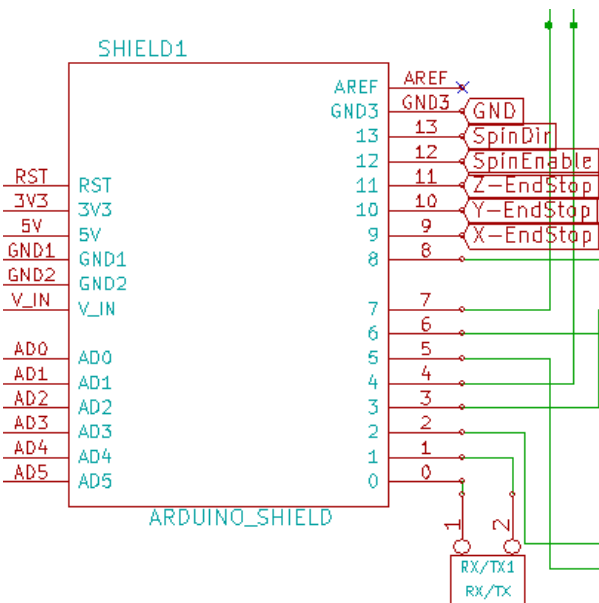


Figura 13 – Schematic hardware pini

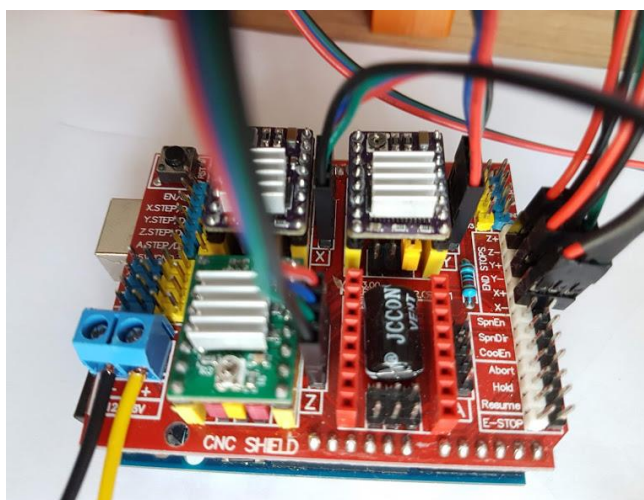


Figura 14 – Componentele hardware asamblate

Shield-ul CNC vine montat peste Arduino. Pe *shield* am montat doua *drivere* DRV8825 pentru axele X și Y, iar pentru axa Z am montat un *driver* A4988. Pentru ca motoarele să aibă o deplasare cât mai fluidă am activat opțiunea *microstepping* de 1/16 (adică 3200 de pași pe rotație) pe toate cele 3 *drivere*.

Următoarele componente montate au fost cele trei motoare pas cu pas. Dacă un motor se va deplasa invers față de felul cum am hotărât în implementare, atunci este suficient doar să punem mufa motorului invers.

Pe *shield* am fi putut monta 6 *switch-uri endstop* pentru detectarea limitelor fizice pentru ambele capete, însă cu doar 3 *endstop-uri* putem face același lucru. Vom atașa câte un *endstop* la fiecare axă, celalalt capăt fiind calculat din software.

Pentru alimentarea motoarelor am folosit de la sursa de alimentare un fir negru (GND) și unul de culoare galbenă (+12V). Sursa de alimentare nu pornește însă decât dacă firul verde al sursei este

conectat cu un fir negru de GND. Pentru a opri și porni sursa ușor fără a o scoate din priză am conectat aceste doua fire printr-un întrerupător.

2.2 Asamblarea componentelor mecanice

Pentru componentele printate la imprimantă 3D am folosit o platformă de CNC disponibilă pe *site-ul Thingiverse* (3). *Link-ul* acestui proiect se va regăsi în referințe. Nu am insistat pe descrierea părții de asamblare a componentelor mecanice deoarece în *link-ul* din referințe se află și un video cu asamblarea scheletului mecanic.

Pentru a putea atașa CNC-ului diferite tipuri de instrumente de desenat sau unealta multifuncțională, am creat în aplicația *ThinkerCad* următoarele modele 3D.

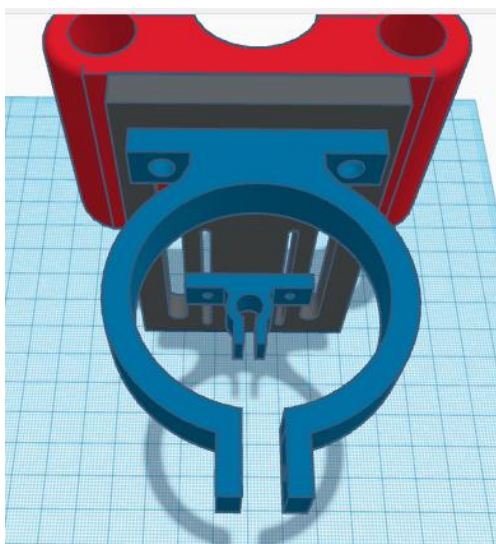


Figura 15 – Modelele 3D pentru a atașa diverse unelte

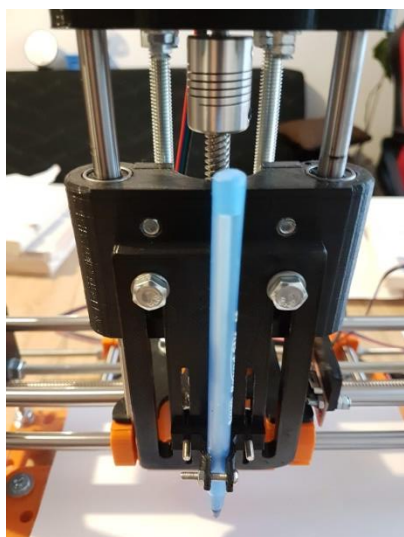


Figura 16 – Exemplu atașare pix



Figura 17 – Exemplu atașare unealtă gravat

Astfel putem ataşa şi ajusta cu uşurinţă diferite tipuri de pixuri, creioane sau în special unealta de gravat ca în figurile din imaginile de mai sus.

2.3 Starea finală platformei mecanice

În imaginea următoare avem finalizată platforma mecanică a CNC-ului după asamblarea tuturor componentelor menţionate anterior.

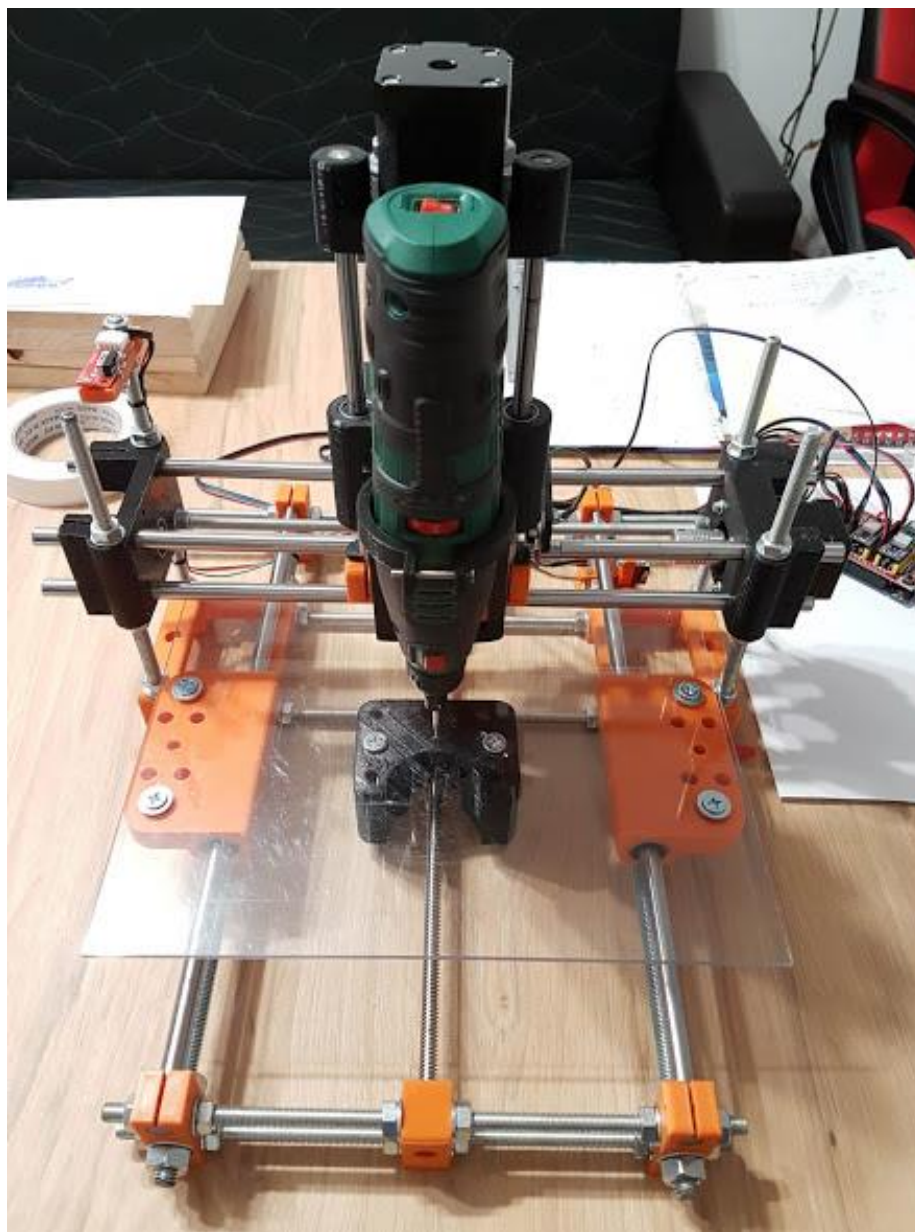


Figura 18 – Asamblarea finalizată a CNC-ului

Aşadar putem trece mai departe să discutăm despre limbajul GCODE, despre algoritmi de interpolare folosiţi, iar în final vom putea trece la partea de implementare a aplicaţiei software.

3 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 mașină 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 să le implementez în *software*-ul de pe Arduino. astfel încât CNC-ul să fie funcțional și să poată executa fișiere GCODE trimise linie cu linie prin intermediul USB-ului.

3.1 Sintaxa limbajului GCODE

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

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

- **G** – este simbolul folosit pentru a selecta o comandă GCODE
- **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, în adâncime
- **F** - setează viteza cu care unealta CNC-ului se deplasează, reprezentată în **mm/min**
- **M** - alte funcții diverse

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

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

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

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 *debug*.

3.2 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. Dacă o comandă este dintr-un anumit grup, execuția unei comenzi din același grup dezactivează comanda precedentă.

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

3.2.1 Grupul comenzilor de mișcare

În acest grup se află următoarele comenzi: G00, G01, G02 și 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 mișcările de bază ale CNC-ului. Cu ajutorul lor se poate efectua orice tip de desen, oricât de complex ar fi.

- **G00** reprezintă o mișcare liniară a axelor într-un mod sincron. De exemplu, dacă suntem la poziția $P_0(0, 0)$ și următoarea comandă urmează să fie executată: **G00 X40 Y20**
Atunci se execută 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 execută mișcarea pe cea de-a doua axă până când poziția **Y: 20** a fost atinsă.



Figura 19 – Comanda G00, mișcare liniară simplă

- **G01** reprezintă o mișcare liniară de interpolare. De exemplu dacă suntem la poziția $P_0(0, 0)$ și următoarea comandă urmează să fie executată: **G00 X40 Y20**
Atunci se execută o mișcare liniară pe dreapta formată dintre cele două puncte, punctul P_0 și punctul destinație $P_1(40, 20)$. Mișcarea obținută va fi echivalentă cu cea din imaginea următoare.



Figura 20 – Comanda G01, interpolare liniară

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

Singura diferență dintre G02 și G03 este faptul că prima comandă G02 produce cercuri **în sensul acelor de ceasornic**, iar comanda G03 produce cercuri **invers acelor de ceasornic**.

Pentru aceste două tipuri de comenzi există 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 centrul cercului. Există două metode prin care putem desena un cerc. Prima metodă este să trimitem parametrii **I** și **J**, iar cea de-a doua metodă este să trimitem doar raza cercului prin parametrul **R**, atunci ceilalți doi parametri pot să lipsească.

Pentru a desena un arc trebuie să trimitem prin parametrii **X** și **Y** coordonatele celui alt capăt al arcului. Atunci CNC-ul va trasa un arc începând de la punctul curent până la punctul destinație. Punctul destinație trebuie neapărat să se situeze pe cerc. În 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 arcul marcat cu roșu.

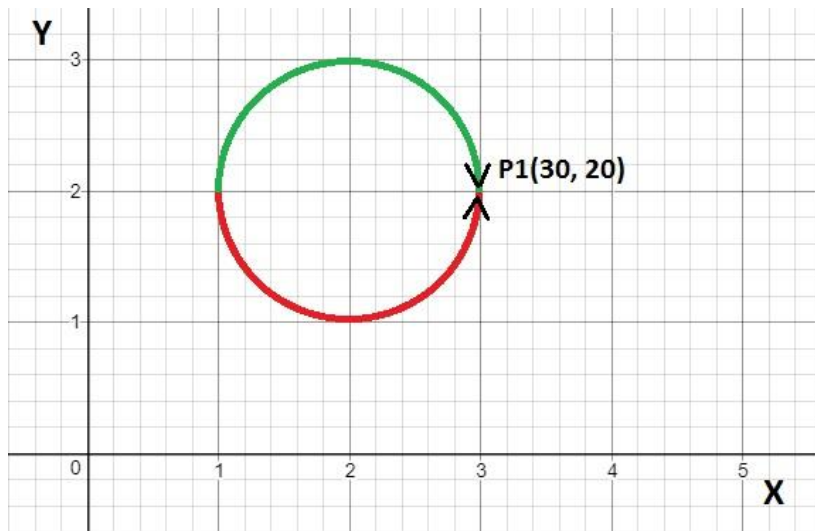


Figura 21 – Interpolare circulară, comenzile G02 și G03

3.2.2 Grupul comenzilor pentru selectarea planului

Aceste comenzi au impact doar asupra comenzilor de interpolare circulară **G02** și **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**

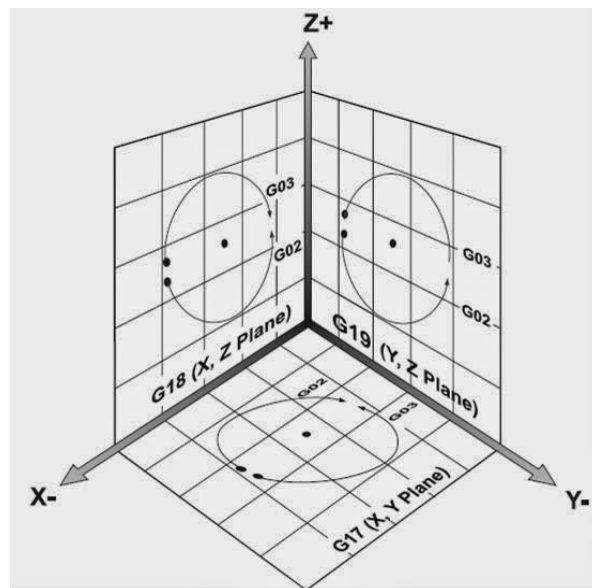


Figura 22 – Selectarea planului

3.2.3 Grupul comenzilor pentru poziționare

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

Comanda **G90** reprezintă poziționarea absolută a parametrilor X, Y și Z relativ la poziția zero a axelor. De exemplu, dacă suntem la poziția $P_0(10, 10, 10)$ și avem de executat următoarea comandă: **G01 X20 Y5 Z10** atunci CNC-ul se va poziționa la punctul $P_1(20, 5, 10)$.

Comanda **G91** reprezintă poziționarea incrementală a parametrilor X, Y și Z relativ la poziția curentă a axelor. De exemplu, dacă suntem la poziția $P_0(10, 10, 10)$ și avem de executat următoarea comandă: **G01 X20 Y5 Z10** atunci CNC-ul se va poziționa la punctul $P_1(30, 15, 20)$.

3.2.4 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.

3.2.5 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 și 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.

3.3 Comenzi GCODE non modale

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

În acest subcapitol voi discuta despre comenzile non modale.

- Comanda **G04** este o instrucțiune de așteptare, echivalentul unei funcții de *delay* din programare. Această comandă poate veni însoțită de simbolul **P** urmat de un număr ce specifică timpul în milisecunde de așteptare. Opțional, timpul specificat de așteptare poate fi trecut în parametrul **X**, semnificând același lucru.

- Comanda **G28** este o procedură în 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 uneltei 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 în mijlocul patului:

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

3.4 Comenzi GCODE auxiliare

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

- Comanda **M02** înseamnă sfârșitul scriptului GCODE trimis. Pentru această comandă se pot implementa diferite proceduri de distanțare a uneltei față de poziția curentă pentru a avea acces mai ușor la patul de desenare.
- Comanda **M17** dezactivează toate cele 3 motoare pentru a le putea deplasa manual dacă este cazul. Dacă motoarele sunt dezactivate, atunci toate instrucțiunile de mișcare executate de CNC nu vor avea nici un efect asupra motoarelor, ele vor modifica doar poziția logică internă în care se află CNC-ul.
- Comanda **M18** activează toate cele 3 motoare.

4 Algoritmi

În secțiunea curentă voi detalia cei mai importanți algoritmi folosiți în implementarea CNC-ului. Voi încerca să aleg notații cât mai sugestive pentru a avea o trasabilitate cât mai bună 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 în ultimele subcapitole voi descrie algoritmi pentru cele patru tipuri de mișcări posibile reprezentate în limbajul GCODE de G00, G01, G02 și G03 discutate în 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 aceleași notații peste tot parcursul acestei secțiuni.

În algoritmi 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

4.1 Formula pentru calculul distanței fizice de deplasare

CNC-ul primește prin intermediul simbolurilor **X**, **Y**, și **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 să deducem câteva formule.

În primul rând pentru a deduce cum calculăm 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 axă într-un pas al motorului, notăm cu **TRAVEL_DISTANCE_360** specificația șurubului trapezoidal, în cazul de față 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:

$$\text{STEP_REZOLUTION} = \text{TRAVEL_DISTANCE_360} / \text{NUMBER_OF_STEPS_360}$$

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

Opțiune <i>microstepping</i>	NUMBER_OF_STEPS_360	STEP_REZOLUTION
Fără <i>microstepping</i>	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

4.2 Simularea vitezei de deplasare

Limbajul GCODE specifică viteza cu care unealta trebuie să se deplaseze atunci când un anumit tip de mișcare este executată. Este important ca unele operațiuni de mișcare ale CNC-ului să se execute la o viteză mai mică. Spre exemplu, dorim să gravăm în metal la o viteză 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 între 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 cât 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, și 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 numărul de rotații pe minut, însă avem nevoie să convertim **RPM** în **RPS**, adică rotații pe secundă:

$$\text{RPM} = \text{SPEED} / \text{TRAVEL_DISTANCE_360}$$

$$\text{RPS} = \text{RPM} / 60$$

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

$$\text{STEPS_PER_SECOND} = \text{RPS} * \text{NUMBER_OF_STEPS_360}$$

Având la dispoziție toate aceste rezultate 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:

$$\text{DELAY_s} = 1 / \text{STEPS_PER_SECOND}$$

$$\text{DELAY_ms} = 1.000 / \text{STEPS_PER_SECOND}$$

$$\text{DELAY_us} = 1.000.000 / \text{STEPS_PER_SECOND}$$

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

În tabelul următor am trecut câteva valori orientative pentru timpul de așteptare dintre pașii 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 și pragul superior al vitezei. În tabel vor fi calculate *delay*-urile pentru toate opțiunile posibile de microstepping ale *driver*-elor în microsecunde(us).

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 simulată, motorul se învârtă cu 1 RPM, iar la viteza maximă motorul ajunge undeva la aproximativ 255 RPM. Am ales aceste limite de minim și maxim pentru a proteja driverele și motoarele de supraîncălzire.

4.3 Algoritmul pentru mișcare liniară simplă

Algoritmul pentru mișcarea liniară simplă este cel mai intuitiv și 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 și **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)
```

4.4 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 două puncte.

Avem ca *input* **p0** punctul curent de pornire și **p1** fiind punctul destinație. *Output*-ul este mișcare de interpolare liniară, de la **p0** spre **p1** mergând pe ecuația dreptei celor două 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 incrementăm poziția curentă 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 depășind valoarea constantei STEP_RESOLUTION. Din acest motiv, o astfel de linie va fi desenată ca în imaginea din figura următoare.

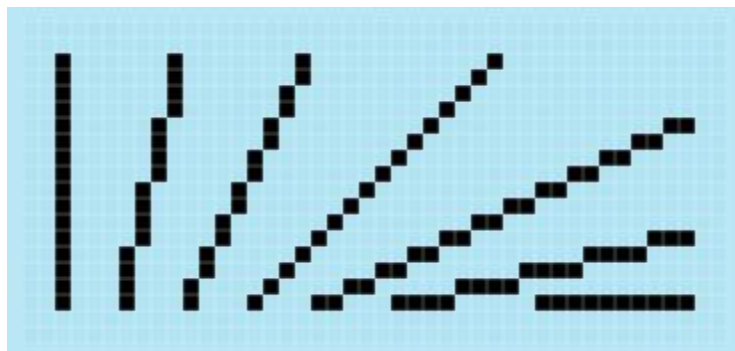


Figura 23 – Aproximare interpolare liniară

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

4.5 Algoritmul pentru interpolare circulară

În acest ultim subcapitol vom detalia algoritmul de interpolare circulară pentru desenarea cercurilor. Acest algoritm va implementa comenzile **G02** și **G03**.

Vom avea nevoie de ecuația cercului: $(x - \text{center.x})^2 + (y - \text{center.y})^2 = R^2$. Rescriem ecuația în următoarea formă: $FXY = (x - \text{center.x})^2 + (y - \text{center.y})^2 - R^2$.

FXY ne va da informația dacă punctul curent în care ne aflăm este pe ecuația cercului sau nu. Dacă $FXY < 0$, atunci punctul curent este în interiorul cercului însemnând că va trebui să ne deplasăm spre exterior, iar dacă $FXY > 0$, atunci punctul curent este în exterior și va trebui să ne deplasăm 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 să fie pe ecuația cercului, însă aceste aproximații vor fi insesizabile.

În continuare avem nevoie să știm unde ne situăm pe cerc relativ la centrul acestuia. În variabila **A** vom reprezenta cu 0 dacă suntem în stânga cercului, iar cu 1 dacă suntem în dreapta sa. În variabila **B** vom reprezenta cu 0 dacă suntem sub centrul cercului, respectiv 1 dacă suntem deasupra acestuia.

Combinând cazurile de mai sus FXY cu A și B, vom avea 8 cazuri posibile de tratat pentru luarea unei decizii de deplasare pe axele X și Y. Luând și posibilitatea în care avem de trasat cu o anumită orientare (în 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 (în 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 punctul curent se afla in interiorul sau exteriorul cercului  
    FXY = (p.x + center.x)^2 + (p.y+center.y)^2 - R^2  
    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 pe care dintre cele două axe, X sau Y, trebuie să se execute un pas și în ce direcție.

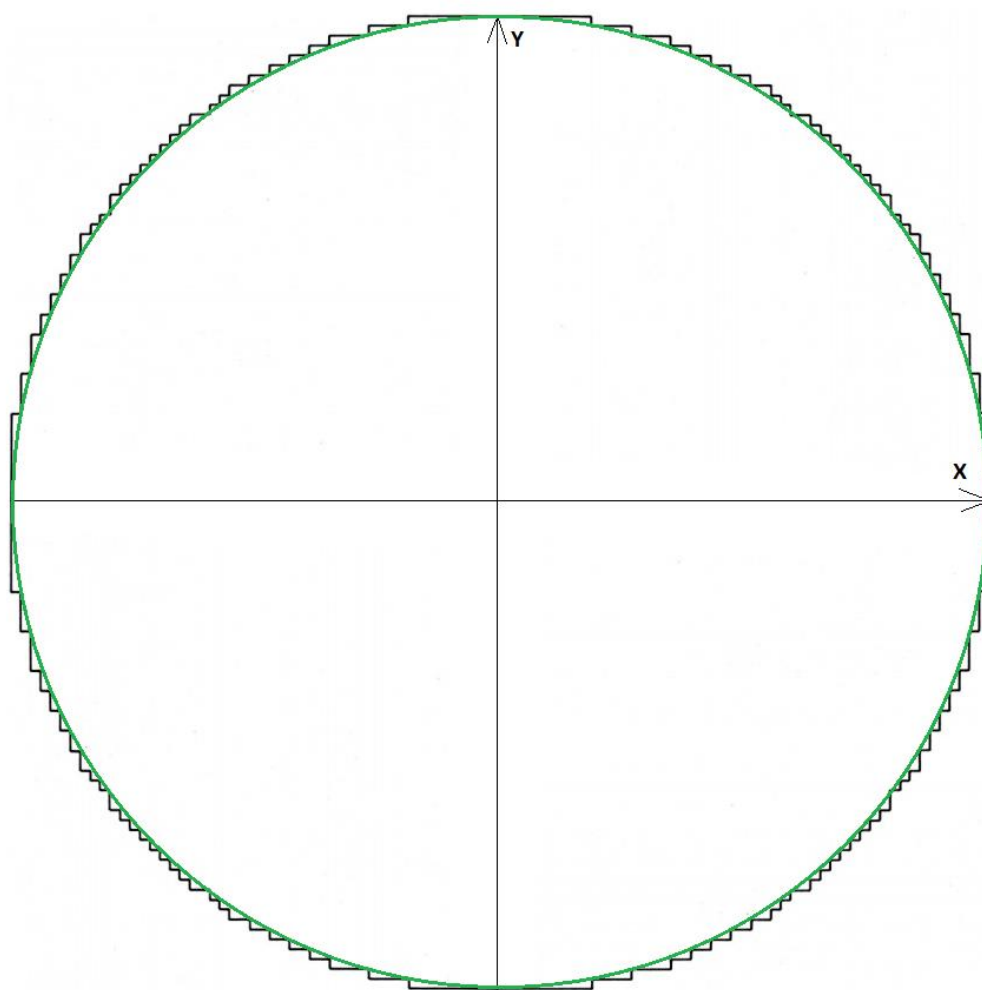


Figura 24 – Aproximare a mișcării circulare pentru desenarea unui cerc

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

5 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 în *Python* pentru a face acest lucru. Cu ajutorul acestui script voi avea posibilitatea să trimit și fișiere GCODE întregi către CNC.

Comenzile vor fi trimise linie cu linie. Dacă comanda trimisă este validă atunci aceasta va fi analizată și executată, iar în caz contrar CNC-ul va răspunde cu un mesaj de eroare.

În primul subcapitol vom discuta despre arhitectura aplicației.

5.1 Arhitectura generală a aplicației

Următoarea diagramă reprezintă descompunerea funcțională minimalistă a aplicației.

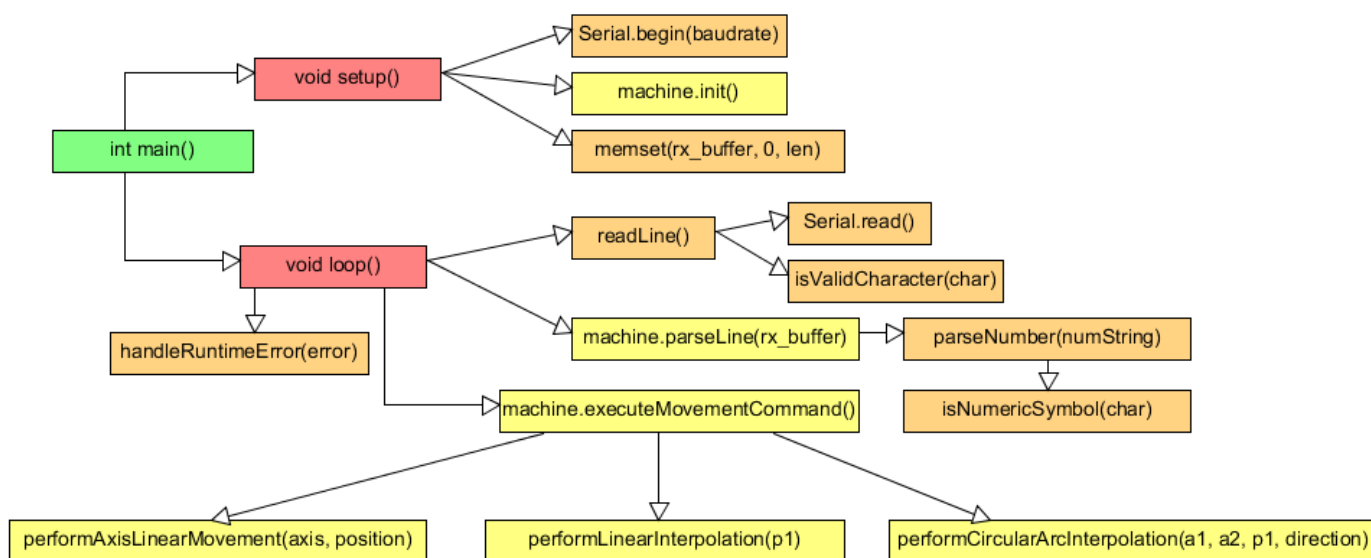


Figura 25 – Diagrama de descompunere funcțională

În punctul de intrare al aplicației, funcția *main*, se apelează cele două 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 în aplicația făcută în *Python*.

În 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 microcontrolerul se resetează. Dacă conexiunea este activă se așteaptă primirea unei linii complete în *buffer*. Primirea unei linii este semnalată printr-un *flag*. Când *flag*-ul este setat se apelează funcția de validare și analizare, *parseLine*.

Funcția *parseLine* extrage instrucțiunile GCODE din linia primită și creează o nouă 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 termină, 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:

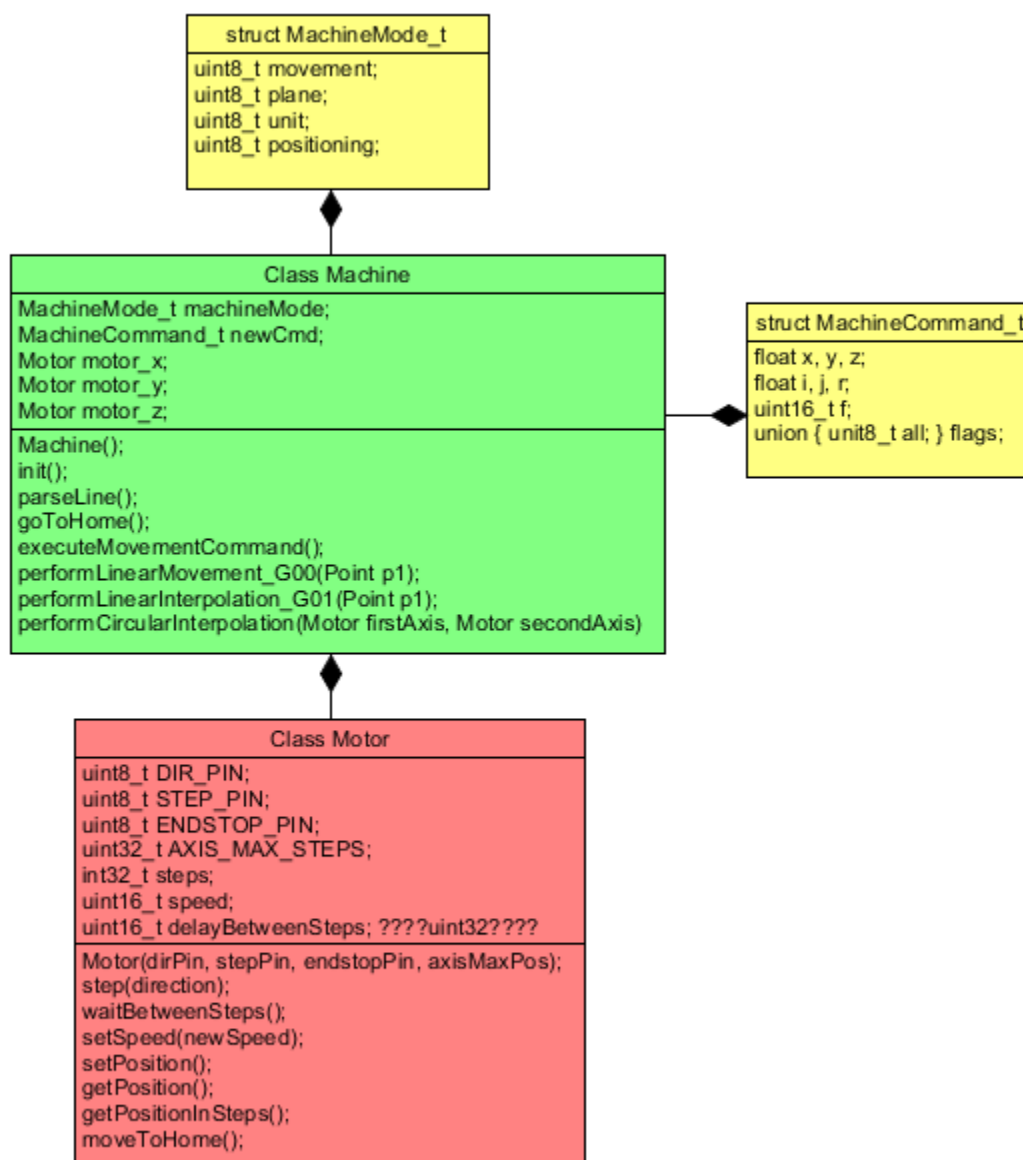


Figura 26 – Diagrama claselor

5.1.1 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 structură:

```
machine.h
65  typedef struct
66  {
67      uint8_t movement;    // movement mode should be G00 G01 G02 G03
68      uint8_t plane;       // selected plane for G02 and G03 cutting: XY XZ YZ
69      uint8_t unit;        // set unit to mm (G21) or inches G20
70      uint8_t positioning; // G90 for absolute, G91 for incremental
71  }MachineMode_t;
```

Figura 27 – Structura pentru starea comenzilor modale

Atunci când primim comenzi de deplasare a unelei 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 măsură în milimetrii
4. **X50.10 Y25.90** – mutăm 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 implementată intern ca fiind o instanță a următoarei structuri de date:

```
machine.h
76  typedef struct
77  {
78      float x, y, z; // new x y z position for 3d printer
79      float i, j, r; // for arc movement radius distance.
80      uint16_t f;    // new feed rate speed (mm/min)
81
82      union {
83          uint8_t all;
84          struct { uint8_t x:1, y:1, z:1, i:1, j:1, r:1, f:1; };
85      }flags; // movement flags
86
87  }MachineCommand_t;
```

Figura 28 – Structura internă 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 și G03. Pe lângă acestea 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 dacă 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 șapte *flag*-uri ocupă un singur byte.

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

```
machine.h
92  class Machine
93  {
94  private:
95      // private variables
96      MachineMode_t machineMode;
97      MachineCommand_t newCmd;
98      Motor motor_x;
99      Motor motor_y;
100     Motor motor_z;
101 }
```

Figura 29 – Câmpurile din clasa Machine

5.1.2 Clasa Motor

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

La inițializarea clasei se setează pinii pentru controlul motorului și pinul pentru *endstop*. De asemenea se setează poziția curentă ca fiind poziția 0 și o viteză implicită a motoarelor de 500mm/min.

De menționat faptul că la punerea în funcțiune a CNC-ului trebuie corectată poziția curentă a unelei prin comanda **G28**. Această comandă va executa procedura de *homing* ale celor trei axe. Comanda G28 va deplasa unealta pe toate cele trei axe până când toate *endstop*-urile au fost atinse. Abia atunci poziția din software va coincide cu poziția fizică a unelei.



```

20
21 class Motor
22 {
23     private:
24         // private variables
25         const uint8_t DIR_PIN; //
26         const uint8_t STEP_PIN; //
27         const uint8_t ENDSTOP_PIN; //
28         int32_t AXIS_MAX_STEPS; //
29
30         // used to compute Logical axis
31         // counts numbers of steps move
32         int32_t steps; //
33         uint16_t speed; //
34         uint16_t delayBetweenSteps; //
35

```

Figura 30 – 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 HIGH⁷, atunci motorul face un pas într-o direcție, iar pe LOW⁸ 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 limită inferioară a axei.
- Variabila **AXIS_MAX_STEPS** reprezintă numărul maxim de pași ce pot fi făcuți pe o axă pentru a ajunge din poziția 0 până în celalalt capăt.
- Variabila **steps** reprezintă poziția în pași relativă la poziția zero a axei. Inițial am ales să reprezint poziția motorului ca fiind distanța față de poziția zero a axei 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 în **mm/min**.

⁷ HIGH este notația în implementare pentru valoarea digitală 1

⁸ LOW este notația în implementare pentru valoarea digitală 0

- 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 următoarea interfață publică a motorului:

```

C++ motor.cpp
34 bool Motor::step(uint8_t direction)
35 {
36     bool endstop = digitalRead(this->ENDSTOP_PIN);
37     if((MOTOR_DIRECTION_REVERSE == direction) && (AXIS_ENDSTOP_REACHED == endstop) )
38     {
39         Serial.println("cnc>>>warning: axis reached reverse ending!"); return false;
40     }
41     if((MOTOR_DIRECTION_FORWARD == direction) && (this->steps >= this->AXIS_MAX_STEPS) )
42     {
43         Serial.println("cnc>>>warning: axis reached forward ending!"); return false;
44     }
45     // Continue if Limits were not reached
46
47     // change direction
48     digitalWrite(this->DIR_PIN, direction);
49
50     // perform step
51     digitalWrite(this->STEP_PIN, LOW);
52     this->waitBetweenSteps(ACCELERATION_DISABLED);
53     //delay_sync( this->delayBetweenSteps );
54     digitalWrite(this->STEP_PIN, HIGH);
55
56     // update position
57     if ( direction == MOTOR_DIRECTION_REVERSE) { this->steps -= 1; }
58     else if(direction == MOTOR_DIRECTION_FORWARD) { this->steps += 1; }
59     else { /* this is reached only if direction is invalid */ }
60     return true;
61 }

```

Figura 31 – Implementarea funcției step a clasei Motor

La începutul funcției se verifică mai întâi dacă putem executa pasul, adică dacă limitele nu au fost atinse. Dacă ne deplasăm în *REVERSE*, adică spre poziția 0, verificăm dacă *switch*-ul *endstop* nu a fost apăsător. Dacă ne deplasăm în direcția *FORWARD* atunci verificăm dacă poziția curentă nu depășește limita din fișierul de configurație.

Dacă putem efectua pasul atunci pinul DIR se setează în direcția dată ca input apoi se trimite către *driver* pulsul pentru a efectua pasul motorului. Între tranzițiile pinului STEP din LOW în HIGH se apelează funcția *waitBetweenSteps* pentru simularea vitezei cu ajutorul formulei discutate în subcapitolul 4.2.

După ce pasul a fost efectuat vom actualiza și poziția motorului (calculată în pași) relativă la poziția 0. Dacă ne deplasăm în *REVERSE* decrementăm cu 1, iar dacă ne deplasăm în direcția *FORWARD* atunci incrementăm cu 1.

Așadar, există 3 instanțe ale acestei clase, câte una pe fiecare dintre cele trei axe. După cum am văzut mai sus în variabila *steps* se salvează intern poziția pe o anumită axă reprezentată în pași.

Pentru a obține poziția uneltei pe o axă reprezentată în milimetrii avem la dispoziție următoarea interfață.

```
130 // get motor position in float relative to 0
131 float Motor::getPosition()
132 {
133     return this->steps * STEP_RESOLUTION;
134 }
```

Figura 32 – Implementarea interfeței *getPosition*

5.2 Fișierele de configurație

În secțiunea curentă vom discuta despre setările de configurație ale CNC-ului și configurația hardware a pinilor.

5.2.1 Configurația Hardware

În fișierul *platform_config.h* avem asociați pinii de pe Arduino la componentele montate pe *shield*-ul CNC. Pentru a face acest lucru am folosit imaginea din Figura 13, din capitolul de asamblare.

Așadar, avem nevoie de a controla pinii STEP și DIR ale driverelor pentru deplasarea celor trei motoare pas cu pas și de pinii *endstop*-urilor pentru a detecta poziția 0 a axelor.

Astfel, dacă dorim schimbarea software-ului pe altă platformă Arduino, de exemplu Arduino Mega⁹ cu *shield*-ul RAMPS¹⁰, atunci singurele modificări pe care trebuie să le facem sunt doar în acest fișier.

```
C platform_config.h
1  #ifndef PLATFORM_CONFIG_H
2  #define PLATFORM_CONFIG_H
3
4  #define PIN_ENABLE_MOTORS 8u
5
6  #define PIN_MOTOR_X_STEP 2u
7  #define PIN_MOTOR_Y_STEP 3u
8  #define PIN_MOTOR_Z_STEP 4u
9
10 #define PIN_MOTOR_X_DIR 5u
11 #define PIN_MOTOR_Y_DIR 6u
12 #define PIN_MOTOR_Z_DIR 7u
13
14 #define PIN_ENDSTOP_X 9u
15 #define PIN_ENDSTOP_Y 10u
16 #define PIN_ENDSTOP_Z 11u
17
18 // Reserved pins (not used yet)
19 #define PIN_RESERVED_12 12u
20 #define PIN_RESERVED_13 13u
21 #define PIN_RESERVED_A0 A0
22 #define PIN_RESERVED_A1 A1
23 #define PIN_RESERVED_A2 A2
24 #define PIN_RESERVED_A3 A3
25 #define PIN_RESERVED_A4 A4
26 #define PIN_RESERVED_A5 A5
```

Figura 33 – Fișierul de configurație hardware

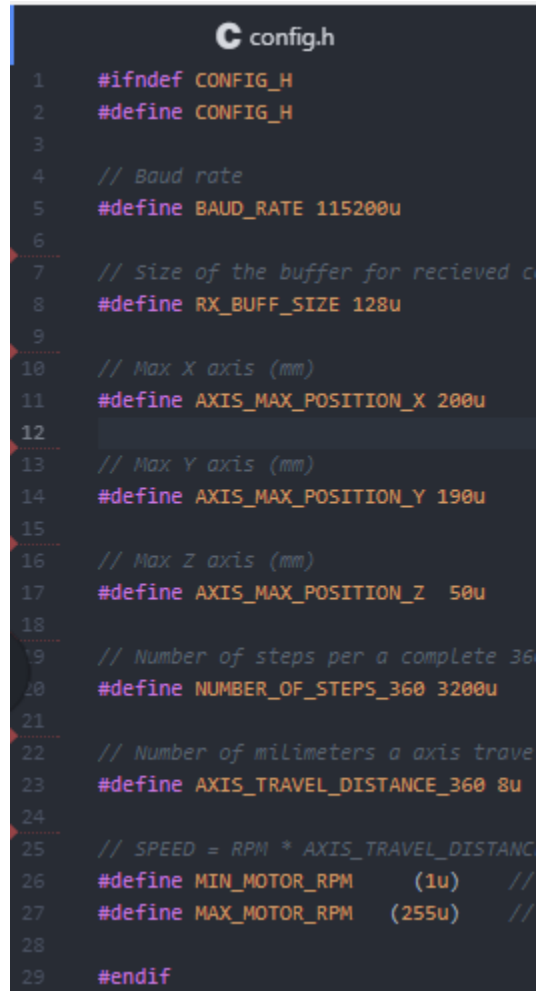
⁹ Arduino Mega este o altă platformă Arduino, aceasta are un alt microcontroler ce dispune de mai mulți pini I/O și memorie flash și RAM mai mare

¹⁰ RAMPS este un *shield* compatibil cu platforma Arduino Mega, această combinație fiind o opțiune foarte populară în imprimante 3D.

5.2.2 Configurația CNC-ului

În fișierul *config.h* avem setările de configurare generale ale CNC-ului:

- **BAUD_RATE** este viteza de comunicație serială. Pentru a se putea comunica, aceeași viteză trebuie să o avem în ambele capete.
- **RX_BUFF_SIZE** este dimensiunea *buffer*-ului de recepție a datelor.
- **AXIS_MAX_POSITION_X** este distanța maximă în cm ce poate fi parcursă de la poziția 0 a axei în partea dreaptă (orizontal).
- **AXIS_MAX_POSITION_Y** este distanța maximă în cm ce poate fi parcursă de la poziția 0 a axei în față (vertical).
- **AXIS_MAX_POSITION_Z** este distanța maximă în cm ce poate fi parcursă de la poziția 0 a axei în sus (adâncime).
- **NUMBER_OF_STEPS_360** reprezintă câți pași ale unui motor înseamnă o rotație completă. În cazul de față avem opțiunea de *microstepping* 1/16 activată.
- **AXIS_TRAVEL_DISTANCE_360** înseamnă câți mm se deplasează unealta pe o axă într-o rotație completă. Această informație este luată din specificațiile șurubului trapezoidal menționat în subcapitolul 1.3.4.
- **MIN_MOTOR_RPM** este turația minimă a motoarelor permisă din software. Pentru valoarea din imaginea alăturată viteza minimă va fi de **8mm/min**.
- **MAX_MOTOR_RPM** este turația minimă a motoarelor permisă din software. Pentru valoarea din imaginea alăturată viteza maximă va fi de **2040mm/min**.



```
1  #ifndef CONFIG_H
2  #define CONFIG_H
3
4  // Baud rate
5  #define BAUD_RATE 115200u
6
7  // Size of the buffer for recieved c
8  #define RX_BUFF_SIZE 128u
9
10 // Max X axis (mm)
11 #define AXIS_MAX_POSITION_X 200u
12
13 // Max Y axis (mm)
14 #define AXIS_MAX_POSITION_Y 190u
15
16 // Max Z axis (mm)
17 #define AXIS_MAX_POSITION_Z 50u
18
19 // Number of steps per a complete 360
20 #define NUMBER_OF_STEPS_360 3200u
21
22 // Number of milimeters a axis trave
23 #define AXIS_TRAVEL_DISTANCE_360 8u
24
25 // SPEED = RPM * AXIS_TRAVEL_DISTANC
26 #define MIN_MOTOR_RPM (1u) //
27 #define MAX_MOTOR_RPM (255u) //
28
29 #endif
```

Figura 34 – Fișierul de configurație a CNC-ului

5.3 Implementarea comunicației

Comunicația prin USB este făcută cu ajutorul clasei **Serial** din biblioteca Arduino. La pornirea aplicației se așteaptă în funcția *init* până când comunicația a fost inițializată.

După ce comunicația a fost inițializată se trimite șirul de caractere: “*cnc>>>connected*” urmând ca aplicația să intre în funcția ciclică *loop*. La fiecare iterație a acestei funcții se verifică dacă

comunicația este activă. În caz contrar microcontrolerul își va da *reset*, iar procesul de mai sus se va repeta până ce comunicația devine din nou activă.

În continuare se așteaptă până când o linie întreagă este primită. Când simbolul ASCII *newline* este recepționat înseamnă că linia întreagă a fost primită, pentru a semnala acest eveniment se setează un *flag*. Atunci când aplicația detectează că acest *flag* este setat începe procedura de verificare și analizare a liniei primite.

Dacă linia primită este validă sintactic atunci aceasta se execută. După ce execuția a fost finalizată se trimite șirul de caractere: “*cnc>>>done*” pentru a semnala faptul că execuția comenzii a fost finalizată cu succes sau în caz de eroare, șirul de caractere: “*cnc>>>error:*” urmat de un mesaj corespunzător.

5.4 Problema preciziei a numerelor fracționare

Implementând poziționarea uneltei CNC-ului, am ales inițial 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 ne deplasăm la o poziție pozitivă mai mare decât poziția curentă. Deci, ca să ajungem la noua poziție trebuia să incrementăm poziția curentă cu valoarea constantei **STEP_RESOLUTION** până 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 să parcurg o anumită distanță numărul pașilor crește. Făcând mai mulți pași, implicit asta înseamnă că trebuia să incrementez de mai multe ori cu constanta **STEP_RESOLUTION** la fiecare 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*.

STEPS	STEP_RESOLUTION	Reprezentarea exactă în float
200	0.04	0.039999998092
400	0.02	0.019999999046
800	0.01	0.009999999046
1600	0.005	0.004999999523
3200	0.0025	0.002499999761
6400	0.00125	0.001249999880

Așadar, pentru a scăpa de această problemă am ales să reprezint poziția curentă pe o anumită axă ca fiind distanța în pași relativă la poziția zero a axei. Am stocat această poziție într-o variabilă de tip `int32`.

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

Pentru a putea afla ce distanță maximă putem reprezenta în această abordare am împărțit valoarea maximă posibilă ce încapă 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. Știm că o rotație completă înseamnă 8mm, deci făcând o înmulțire obținem 2.684.352mm, adică 2684metri. Nu va fi niciodată cazul să depășim acest prag, mai ales ținând cont de faptul că limita fizică a axelor X respectiv Y nu va depăși 20cm.

Totuși este foarte important că am ales tipul de date `int32`. Dacă am fi ales o variabilă de tip `int16`, atunci făcând aceleași calcule ne-am fi putut deplasa doar 40mm fără a depăși valoarea maximă a acestui tip de date.

5.5 Implementarea aplicației pentru comunicația USB

Pentru a putea folosi CNC-ul într-un mod cât mai simplu am făcut un mic script în limbajul *Python*. Prin intermediul acestuia putem trimite comenzi către CNC direct de la linia de comandă. Însă cel mai important motiv pentru care am făcut acest script este de a putea trimite ușor fișiere GCODE.

Ideea de bază a implementării acestei aplicații este destul de simplă. Pentru că funcția de citire de la linia de comandă este blocantă, avem nevoie de doua fire de execuție.

În firul principal de execuție se așteaptă primirea de comenzi de la linia de comandă. Acestea pot fi comenzi GCODE sau comenzi interne ale *script*-ului. De exemplu am ales comanda ***quit*** pentru a închide aplicația și comanda ***file*** pentru a trimite un fișier GCODE către CNC.

În cel de-al doilea fir de execuție se așteaptă primirea răspunsurilor din partea CNC-ului. Atunci când răspunsul este primit, acesta este afișat în consolă și se setează un *flag* de notificare ce indică faptul că CNC-ul este pregătit pentru primirea comenzii următoare.

6 Utilizare

Putem folosi CNC-ul în modul interactiv, folosind scriptul implementat din capitolul anterior, trimițând comenzi GCODE direct din linia de comandă. Acest mod a fost foarte util în dezvoltarea aplicației, însă pentru utilizarea normală dorim să trimitem direct fișiere întregi într-un mod automat.

Pentru utilizarea normală a CNC-ului dorim să desenăm sau să gravăm diferite imagini sau text.

Pentru a desena o imagine, trebuie mai întâi să convertim imaginea într-un fișier GCODE. Nu voi intra detalii deoarece există o mulțime de aplicații și tutoriale pe internet cu ajutorul cărora se poate realiza ușor acest lucru.

După ce avem fișierul GCODE pregătit putem începe pregătirea CNC-ului.

Mai întâi trebuie conectată placa Arduino la portul USB al computerului. După care putem porni și scriptul pentru comunicație de pe computer. După ce inițializarea comunicației dintre CNC și computer a fost realizată, în consola va fi afișat mesajul “*cnc>>>connected*”.

În acest pas putem interacționa cu CNC-ul prin intermediul linei de comanda. Putem trimite orice tip de comandă **GCODE** sau comenzi speciale ce încep cu simbolul #. Aceste comenzi speciale au rolul de informare cu privire la starea mașinii sau de diagnosticare.

Avem următoarele comenzi speciale implementate:

- #1 este folosită pentru a afla poziția în mm la care se află unealta CNC-ului
- #2 este folosită pentru a afla aceeași poziție ca mai sus, doar că este reprezentată în pași
- #3 este folosită pentru a interoga viteza de deplasare a motoarelor.
- #4 este folosită pentru a interoga modurile în care se află CNC-ul

6.1 Pregătirea CNC-ului pentru desenare

Pentru a putea desena trebuie să calibrăm CNC-ul astfel încât la poziția 0 a axei Z, vârful instrumentului de desenat să se poziționeze perfect pe foaia de desenare.

Mai întâi trimitem comanda **G28** pentru a poziționa toate axele în poziția home.

După ce această procedură a fost terminată putem ajusta la alegere ori instrumentul de desenat ori să mutăm endstop-ul de pe axa Z mai sus sau mai jos până ce vârful o să facă contact perfect cu patul de desenare (a se vedea imaginea din Figura 35).

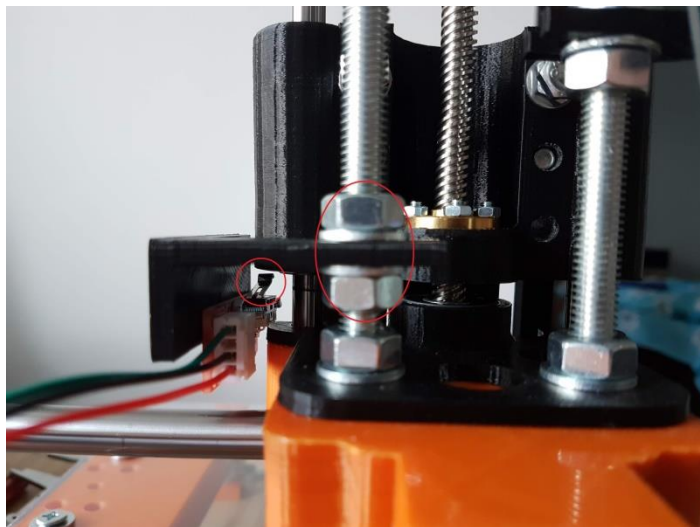


Figura 35 – Ajustarea endstop-ului de pe axa Z

Dacă folosim cea de-a doua metodă cu ajustarea endstop-ului, atunci mai executăm odată comanda G28. Dacă observăm ca vârful instrumentului încă nu face contact cu patul, atunci înseamnă ca trebuie să-l coborâm mai jos.

6.2 Pregătirea CNC-ului pentru a sculpta/grava

Pentru decuparea lemnului putem pregăti CNC-ul direct din software.

Deplasăm unealta prin comenzi GCODE în poziția de unde vrem să începem decuparea. Vârful uneltei trebuie să atingă perfect materialul.

Executăm comanda **G92**. Comanda setează poziția curentă ca fiind punctul 0 de plecare ale celor trei axe. Acest lucru înseamnă că următoarele comenzi de deplasare vor fi executate relativ la această poziție.

Pentru o poziționare cât mai bună putem dezactiva temporar toate cele trei motoare cu comanda **M18**, astfel încât să mutăm unealta manual. După ce am terminat trebuie să activăm din nou motoarele cu comanda **M17**.



Figura 36 – Poziționare unealtă

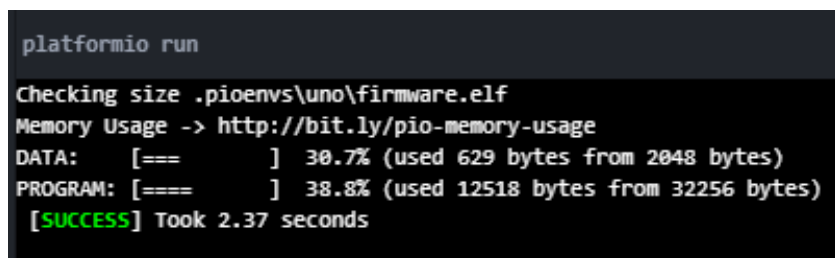
După ce am terminat de făcut toate aceste calibrări putem trimite fișierul GCODE dorit prin comanda *file* urmată de numele fișierului din consola scriptului de pe computer.

7 Concluzii

Așadar avem implementată o aplicație software complet funcțională și compatibilă cu majoritatea *tool*-urilor de conversie a imaginilor în GCODE. Deci am putea desena sau decupa în lemn orice tip de imagini.

Făcând mici ajustări în fișierele de configurație putem folosi aceeași aplicație fără nici o altă modificare în implementare pe diferite tipuri de platforme mecanice ce au la bază coordonatele carteziene.

Din resursele hardware de pe microcontroler am folosit doar 12 din totalul de 20 de pini disponibili, iar memoria rămasă pe microcontroler este suficientă pentru a mai putea adăuga și alte funcționalități.



```
platformio run
Checking size .pioenvs\uno\firmware.elf
Memory Usage -> http://bit.ly/pio-memory-usage
DATA:    [===      ]  30.7% (used 629 bytes from 2048 bytes)
PROGRAM: [====     ]  38.8% (used 12518 bytes from 32256 bytes)
[SUCCESS] Took 2.37 seconds
```

Figura 37 – Memoria utilizată de aplicație

7.1 Îmbunătățirea protocolului de comunicație

În implementarea curentă folosim o metodă sincronă de comunicație. Pentru fiecare comandă GCODE trimisă așteptăm răspunsul CNC-ului pentru a trimite următoarea comandă. Deoarece microcontrolerul are o viteză de comunicație relativ mică, se poate sesiza un timp de așteptare de câteva milisecunde între două comenzi consecutive.

Pentru a scăpa de acest timp de așteptare putem mări dimensiunea *buffer*-ului de recepție și să primim mai multe linii deodată, astfel încât să avem tot timpul o comandă de executat. Putem folosi o coadă pentru a implementa această idee.

7.2 Îmbunătățirea distanței dintre axa X și de patul de desenare

După cum se poate observa în imaginea alăturată, pe patul suport al CNC-ului nu pot să încapă obiecte mai înalte de aproximativ 2.2cm. Axa X este foarte aproape de patul suport.

Putem însă înălța un pic axa X, însă nu foarte mult, deoarece ne vom lovi de alte probleme.

Dacă aș începe un alt proiect asemănător, aș alege altă platformă mecanică pentru CNC, cel mai probabil cu un schelet făcut din metal.

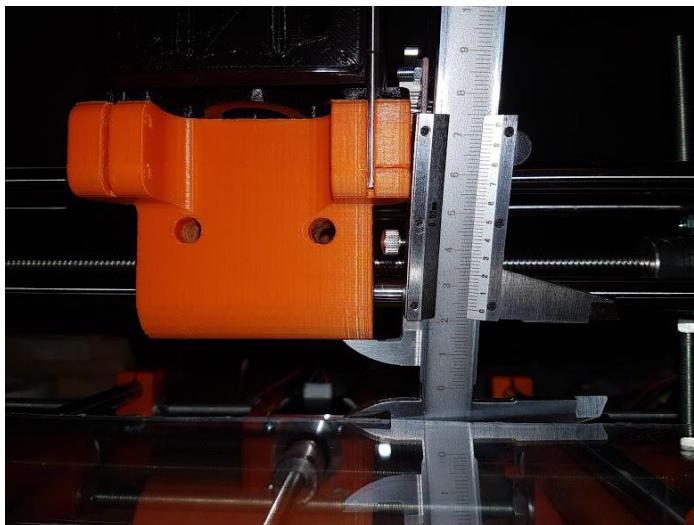


Figura 38 – Problema distanței foarte mici dintre pat și axa X

Figura 1 – Arduino UNO	5
Figura 2 – CNC shield	6
Figura 3 – Motor Pas cu Pas Nema17.....	7
Figura 4 – Driver A4988.....	8
Figura 5 – Driver DRV8825	8
Figura 6 – Endstop switch.....	9
Figura 7 – Sursă aliminetare	9
Figura 8 – Tije cromate de 10, respective 8 mm diametru	10
Figura 9 – Rulment liniar.....	10
Figura 10 – Rulment contragreutate 608RS.....	10
Figura 11 – Șurub trapezoidal.....	11
Figura 12 - <i>Hub</i> de cuplaj flexibil.....	11
Figura 13 – Schematic hardware pini	12
Figura 14 – Componentele hardware asamblate	12
Figura 15 – Modelele 3D pentru a atașa diverse unelte.....	13
Figura 16 – Exemplu atașare pix	13
Figura 17 – Exemplu atașare unealtă gravat	13
Figura 18 – Asamblarea finalizată a CNC-ului.....	14
Figura 19 – Comanda G00, mișcare liniară simplă.....	16
Figura 20 – Comanda G01, interpolare liniară	17
Figura 21 – Interpolare circulară, comenzile G02 și G03.....	18
Figura 22 – Selectarea planului.....	18
Figura 23 – Aproximare interpolare liniară	25
Figura 24 – Aproximare a mișcării circulare pentru desenarea unui cerc	28
Figura 25 – Diagrama de descompunere funcțională	29
Figura 26 – Diagrama claselor.....	31
Figura 27 – Structura pentru starea comenzilor modale	32
Figura 28 – Structura internă a unei comenzi de deplasare	32
Figura 29 – Câmpurile din clasa Machine	33
Figura 30 – Câmpurile din clasa Motor	34
Figura 31 – Implementarea funcției step a clasei Motor.....	35
Figura 32 – Implementarea interfeței <i>getPoziție</i>	36
Figura 33 – Fișierul de configurație hardware.....	36
Figura 34 – Fișierul de configurație a CNC-ului	37
Figura 35 – Ajustarea endstop-ului de pe axa Z	41
Figura 36 – Poziționare unealtă	41
Figura 37 – Memoria utilizată de aplicație	43
Figura 38 – Problema distanței foarte mici dintre pat și axa X	44

8 Bibliografie

1. **ARDUINO.** Arduino Reference. [Online] <https://www.arduino.cc/reference/en/>.
2. **MICROCHIP.** ATmega328p Datasheet. [Online] <https://www.microchip.com/wwwproducts/en/ATmega328p>.
3. **Javier, Arnedo.** Homemade CNC. *Thingiverse*. [Online] 2018. <https://www.thingiverse.com/thing:2794509>.
4. **AUTODESK.** Getting started with G-Code. [Online] <https://www.autodesk.com/industry/manufacturing/resources/manufacturing-engineer/g-code>.
5. **Ken Goldberg, Melvin Goldberg.** XY Interpolation Algorithms. *Robotics Age*. 1983.
6. **Barr, Michael.** *Programming Embedded Systems in C and C++*. 1999.