

Program pentru comanda unui utilaj cu comanda numerica



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Marian Iosif-Andrei
Grupa: 30232

Universitatea Tehnică din
Cluj-Napoca – Facultatea de
Automatică și Calculatoare

An Universitar 2023-2024

Cuprins

1. Introducere
 - 1.1. Context
 - 1.2. Specificatii
 - 1.3. Obiective
2. Studiu Bibliografic
3. Analiza
 - 3.1. Citirea Fișierului de Traiectorie
 - 3.2. Interpretarea și prelucrarea Datelor
 - 3.3. Generarea de Comenzi
 - 3.4. Crearea Interfeței Grafice (GUI)
 - 3.5. Testare și Depanare
 - 3.6. Simularea deplasării capului de tăiere
4. Proiectare
5. Implementare
 - 5.1. Algoritmul de rasterizare
 - 5.2. Algoritmul de citire din fișier și interpretare G-code
 - 5.3. Algoritmul de generare G-code
6. Testare și validare
7. Concluzii
8. Bibliografie

1.1. Context

Scopul esențial al proiectului este să dezvolte un utilaj de tăiat cu flama cu comandă numerică, capabil să execute tăieturi precise. Aceste tăieturi sunt determinate de o traiectorie specificată într-un fișier de intrare, unde traiectoria este reprezentată ca o secvență de segmente și arce de cerc. Datele de intrare sunt numere în format zecimal care indică coordonatele și unghiurile necesare pentru a descrie aceste traiectorii.

Proiectul implică mai multe etape esențiale:

- 1.1.1. Control Numeric: Implementarea unui sistem de control numeric care interpretează datele din fișierul de intrare și traduce aceste informații în comenzi precise pentru motoarele și dispozitivele de control al aparatului.
- 1.1.2. Software de Simulare: Crearea unui software de simulare care permite testarea și vizualizarea traiectoriilor de tăiere înainte de a fi efectuate pe materialul real. Acest lucru ajută la evaluarea corectitudinii și optimizarea tăierilor.
- 1.1.3. Documentație și Instruire: Elaborarea documentației pentru utilizatori, inclusiv manuale de operare și instruirea personalului care va lucra cu utilajul.

Scopul final este de a dezvolta un utilaj capabil să execute tăieri precise și eficiente pe baza traiectoriilor specificate în fișierul de intrare.

1.2. Specificatii

Acest proiect presupune simularea unui program în mediul de dezvoltare IntelliJ, utilizând Java GUI (Graphical User Interface) pentru a oferi o interfață grafică intuitivă. Scopul principal este să simuleze și să vizualizeze traiectoria parcursă de aparatul de tăiat cu flama. Datele de intrare, care reprezintă traiectoriile, vor fi introduse din fișiere, iar apoi se vor aplica operații și transformări asupra acestora pentru a genera și afișa corect traseul. Astfel, utilizatorii vor putea monitoriza și gestiona tăieturile printr-o interfață prietenoasă. Se folosesc operații cu fișiere pentru a gestiona datele și pentru a oferi o experiență simplificată utilizatorilor.

1.3. Obiective

Proiectarea și implementarea unei aplicații pentru a interpreta traseul unui aparat în mod ușor de înțeles pentru utilizatori, folosind IntelliJ și arhitectura MVC, implica:

- Crearea unei structuri de date pentru traseu (Model).
- Dezvoltarea unei interfețe grafice (View) pentru vizualizarea traseului.
- Implementarea unui controller pentru interacțiunea dintre model și vizualizare.
- Încărcarea și procesarea datelor de intrare, transformându-le în format utilizabil.
- Testarea și documentarea aplicației, oferind o interfață intuitivă pentru utilizatori.

- **Cum sunt stocate datele in fisier?**

Datele vor fi reprezentate intr-un fisier .txt sub urmatoarea forma: pe linii separate se vor afla diferite tipuri de traiectorii, segmente si arce de cerc. Segmentele vor fi reprezentate printr-un varf de forma (x, y) care va reprezenta distanta de deplasare pe axele X si Y pornind de la punctul curent si calculand punctul de final bazat pe deplasamentul dat de comanda din fisier iar arcele de cerc vor fi reprezentate printr-o lista de 3 argumente: primele doua reprezinta coordonatele (x, y) care va reprezenta distanta de deplasare pe axele X si Y pornind de la punctul curent si calculand punctul de final bazat pe deplasamentul dat de comanda, iar ultimul argument reprezinta raza cercului folosita ca si ghidaj pentru a putea reprezenta arcul de cerc. De asemenea un alt aspect important il reprezinta directia de parcurgere a arcului de cerc: clock-wise si anti clock-wise, date care se cunosc din apelul comandai, nu din attributele acesteia.

- **Cum sunt interpretate datele din fisier?**

Datele se vor citi din fisier linie cu linie, se vor folosi operatii cu fisiere si se vor retine in string-uri. Apoi cu ajutorul unui regex string-urile vor fi impartite in obiecte definite care sa reprezinte segmentele si arcele de cerc. Apoi vor fi stocate in ordinea in care apar intr-un container (o lista de obiecte) pentru a le putea folosi in reprezentarea traiectoriei pe ecran. [“Java Regular Expressions” – Bibliografie 3]

- **Interpretarea datelor cu ajutorul G-code**

Datele vor fi furnizate sub forma unei liste de comenzi de G-code intr-un fisier. Sintaxa acestor comenzi este de forma “G.. X.. Y.. *R..”. Primul argument din comanda este tipul de traiectorie dupa care se ghideaza aparatul de taiat. Folosim comanda G00 pentru setarea punctului de start (initial fiind setat la coordonatele x=0, y=0) in cazul in care acesta nu este deja setat sau la trasarea unei drepte intre punctul start si punctul de final calculat ulterior pe baza datelor din comanda. Comenzile G02 si G03 sunt folosite pentru trasarea arcelor de cerc, acestea necesita ca punctul de start sa fie deja setat, in caz contrar arcul nu va putea fi calculat si proiectat si se va afisa un mesaj de eroare. Daca punctul de start este setat se va calcula si proiecta un arc de cerc in sensul acelor de ceas pentru comanda G02 si in sens trigonometric pentru comanda G03. Urmatoarele argumente din comenzi constituie deplasamentul pe axele X, respectiv Y cu ajutorul carora se calculeaza punctul de final, care trebuie sa apartina gridului de 290x150, daca acesta depaseste gridul se va afisa un mesaj de eroare, iar traiectoria nu va fi proiectata. Ultimul argument R apare doar in cazul comenzilor pentru arcele de cerc si se refera la raza cercului din care face parte arcul care va fi desenat. [G-code Explained | List of Most Important G-code Commands – Bibliografie 9]

- **Arhitectura proiectului**

Proiectul va fi dezvoltat folosind arhitectura MVC (Model View Controller). Pachetul Model apar toate datele de intrare necesare pentru realizarea simularii, definirea obiectelor necesare, segment si arc de cerc. Clasa Controller face legatura dintre Model si View si aici se intampla toate interpretarile datelor de intrare si conexiunea cu interfata grafica. Pachetul View constituie interfata grafica care ofera un design user-friendly unde este proiectata traiectoria utilajului. In acest pachet se vor folosi functii din Java Swing. [MVC Design Pattern – Bibliografie 4]

- **Cum interpreteaza utilizatorul rezultatul?**

In cadrul pachetului View vor fi definite clase care vor crea o interfata prietenoasa pentru utilizator pe care se va proiecta o matrice de pixeli virtuali scalati astfel incat utilizatorul sa aiba un ghidaj pentru traiectoria pe care o va urmari aparatul de taiat cu flama. De asemenea traiectoria pe care acesta o va vedea este formata cu ajutorul unor functii predefinite in pachetul javaSwing, drawArc() si fillArc() care prelucreaza acele obiecte de tip arc de cerc si cu ajutorul functiei drawLine() se vor desena si obiectele de tip segment. [Utilizarea claselor legate de grafica, fonturi si culori – Bibliografie 1]

Pentru a crea un program Java cu o interfață grafică (GUI) care să simuleze deplasarea unui cap de tăiere pe o traiectorie citită dintr-un fișier, acest proiect poate fi împărțit în mai multe părți tehnice cheie:

3.1. Citirea Fișierului de Traiectorie:

Pentru a citi fișierul care conține traiectoria de tăiere, se utilizează clasa `File` pentru a specifica fișierul și un `bufferedReader` pentru a citi conținutul fișierului rând cu rând. Pentru acest lucru se utilizează operații pe fișiere precum `.hasNextLine()` și `.nextLine()` pentru a citi într-un string un rând întreg. Se creează noi tipuri de date, obiecte care reprezintă tipurile de traiectorii. [Java Read Files – Bibliografie 5]

3.2. Interpretarea și prelucrarea Datelor:

Trebuie interpretată fiecare linie din fișier pentru a extrage informații despre segmente și arce de cerc. Comenzile generate depind de modul în care mașina de tăiat trebuie să execute această traiectorie. Sirurile de caractere în care se salvează datele citite din fișier sunt împartite în numere întregi prin intermediul unui regex și al unui pattern pentru a verifica potrivirea datelor introduse de utilizator. Datele citite din fișier vor fi salvate într-o listă de liste, adică vor fi mai multe liste în care se vor salva datele despre un tip de traiectorie (segment sau arc de cerc) așa cum apar în fișier și încă o listă care va conține toate aceste traiectorii. Pentru crearea listelor se folosesc metode specifice pentru liste, precum `.add()` pentru adăugarea elementelor în listă, `.size()` pentru aflarea numărului de elemente din listă, această metodă este folosită la determinarea tipului de traiectorie care urmează să fie parcurs, `.get()` pentru accesarea elementelor. [Java ArrayList – Bibliografie 6]

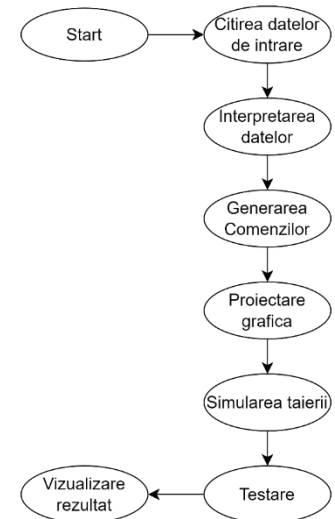


Figura 1: Diagrama de stări

3.3. Generarea de Comenzi:

În mare parte comenzile pentru simularea utilajului sunt reprezentate de colorarea pixelilor care aparțin traiectoriei primite ca dată de intrare. Pentru realizarea acestui fapt se parcurge lista de date iar pentru fiecare tip de traiectorie și se va calcula traiectoria respectivă în funcție de pixelii generați de interfața urmând ca aceștia să fie colorați. Tehnica de abordare al acestui fapt va fi rasterizarea (Figura 3). Principiul de funcționare al acesteia pentru linie este de a împărți spațiul de desenare în 8 cadrane, pentru fiecare cadran tehnica este aceeași, diferind doar modul în care cresc coordonatele x și y . Pentru primul cadran care este de la 0 grade la 45 grade se verifică pixelii prin care ar trece o linie imaginară între cele 2 puncte, aici ne interesează doar pixelii de pe axa y și dacă linia intersectează 2 dintre aceștia, moment în care se calculează distanța pe axa x până la punctul de intersecție cu linia imaginară, și se colorează pixelul care conține distanța mai mare (dacă distanța este mai mare până la intersecție se colorează pixelul de dedesubt, în caz contrar se colorează cel de deasupra). Parcurgerea este asemănătoare și pentru restul cadranelor, dar și pentru generarea arcelor de cerc. [“Line Rasterization – Bibliografie 7]

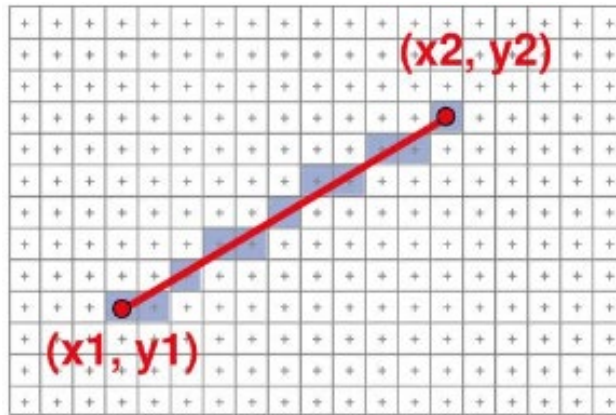


Figura 2: Rasterizarea unei linii

3.4. Crearea Interfeței Grafice (GUI):

Pentru a crea o interfață grafică în Java, se folosesc elemente din Java Swing. Interfața grafică ar trebui să conțină elemente precum un panou de desen, în care se va proiecta o matrice de de pixeli scalati (nu reprezinta pixeli adevarati). Pentru proiectarea matricei de pixeli se folosesc metode deja implementate in pachetele swing si awt. Pentru desenarea traiectoriei in matricea de pixeli urmand modelul primit ca si date de intrare se vor folosi emementele legate de grafica, fonturi si culori. Toate acestea fac parte tot din pachetul awt, precum drawLine(), drawArc(), fillArc() care sunt operatii de trasare a unor linii si arce de cerc. Corespondenta intre interfața grafica si modelele de date se face prin intermediul unui controller care dupa generarea comenzilor transmite interfetei grafice datele care corespund matricei pentru parcurgerea traiectoriei urmata de aparat. [Drawing on a JButton[][] grid in a Java GUI – Bibliografie 10]

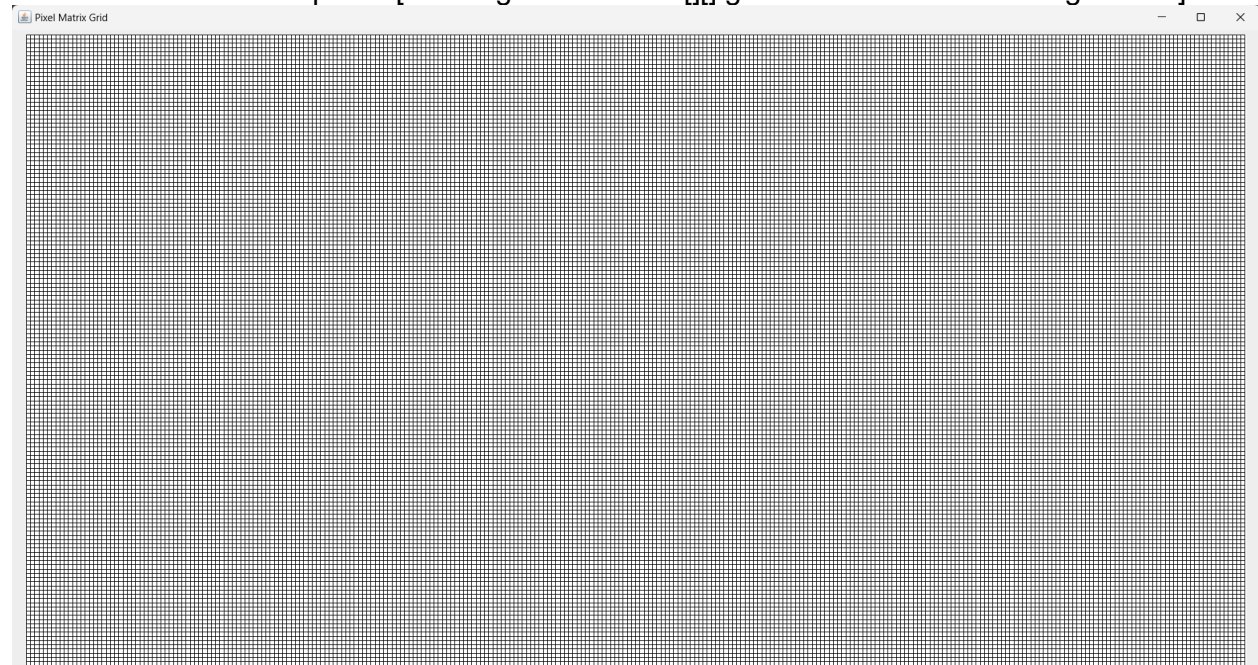


Figura 3: Interfața grafica

3.5. Testare și Depanare:

Programul va fi testat folosind JUnit, se vor face teste legate atat de de rezultatele asteptate cat si legate de mesaje de eroare in anumite cazuri (introducerea unor date de intrare care depasesc gridul, introducerea unor date de intrare care nu respecta pattern-ul impus). De asemenea se vor face si teste grafice pentru verificarea acoperirii oricarei celula din gridul format. Pentru testarea rezultatelor asteptate se va folosi functia assert din cadrul JUnit. [JUnit – Test Framework – Bibliografie 8]

3.6. Simularea deplasării capului de tăiere:

Se utilizează un component grafic pentru a desena simularea traiectoriei capului de tăiere. Capul de tăiere se va deplasa pe ecran în funcție de comenzile generate, astfel încât să reflecte traiectoria reală. Modul în care se parcurge traiectoria este desenarea celulelor din matrice pornind de la primul element grafic stocat în listă și parcurgându-se celula cu celula în matricea de pixeli până la finalizarea modelului, apoi la finalizarea traiectoriei. Utilizatorul va avea pe ecran două butoane “START” și “RESTART” prin care pornește simularea, respectiv o restartează și va putea vedea pe ecran direcția de parcurgere al utilajului.

4. Proiectare

Interacțiunea cu interfața de simulare funcționează după cum urmează: utilizatorul selectează operația pe care dorește să o verifice pe ecran, dacă acesta apasă pe butonul de start va porni simularea iar trecerea dintr-o stare în alta din diagrama de stări din figura 1 se realizează automat, trecând prin stările de citire, interpretare a datelor, generare de comenzi și proiectării interfeței grafice în ordine. Butonul de restart resetează simularea din orice stare în starea inițială. Prin această simulare utilizatorul poate verifica de asemenea precizia aparatului de a urmări traiectorii introduse de acestea, poate porni și opri simularea după cum este prezentat în diagrama de use case din Figura 3.

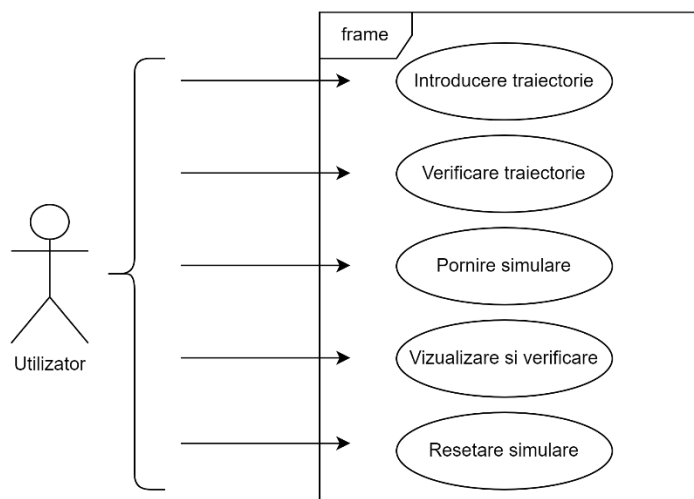


Figura 4: Diagrama de use case

În partea de proiectare se definește următorul model de diagrama de clase și de pachete prezentat în Figura 4. Pentru definirea obiectelor de prelucrare se definesc clase pentru fiecare tip de traiectorie, clasa Line și clasa CircularArc cu ajutorul cărora se instantiază obiecte de tipurile respective care au ca și atribute coordonate, atât în cazul liniilor cât și a arcelor de cerc, iar pentru arcele de cerc mai apar în plus câteva atribute cum ar fi înălțimea și lățimea arcului, unghiul și numărul de grade al acestuia. De asemenea se definesc metode care sunt folosite la simplificarea codului, astfel se pot accesa date și prelucra obiectele mult mai ușor din clasa Controller, aceasta făcând legătura dintre model și interfața grafică. Clasa controller nu este folosită pentru instantierea obiectelor ci doar pentru generarea comenzilor și prelucrarea datelor care sunt transmise interfeței grafice. Încă o clasă importantă este clasa View care creează framework-ul pentru vizualizarea simulării și simplificarea utilizatorilor. Există posibilitatea implementării unor clase ajutoare de legătură între acestea.

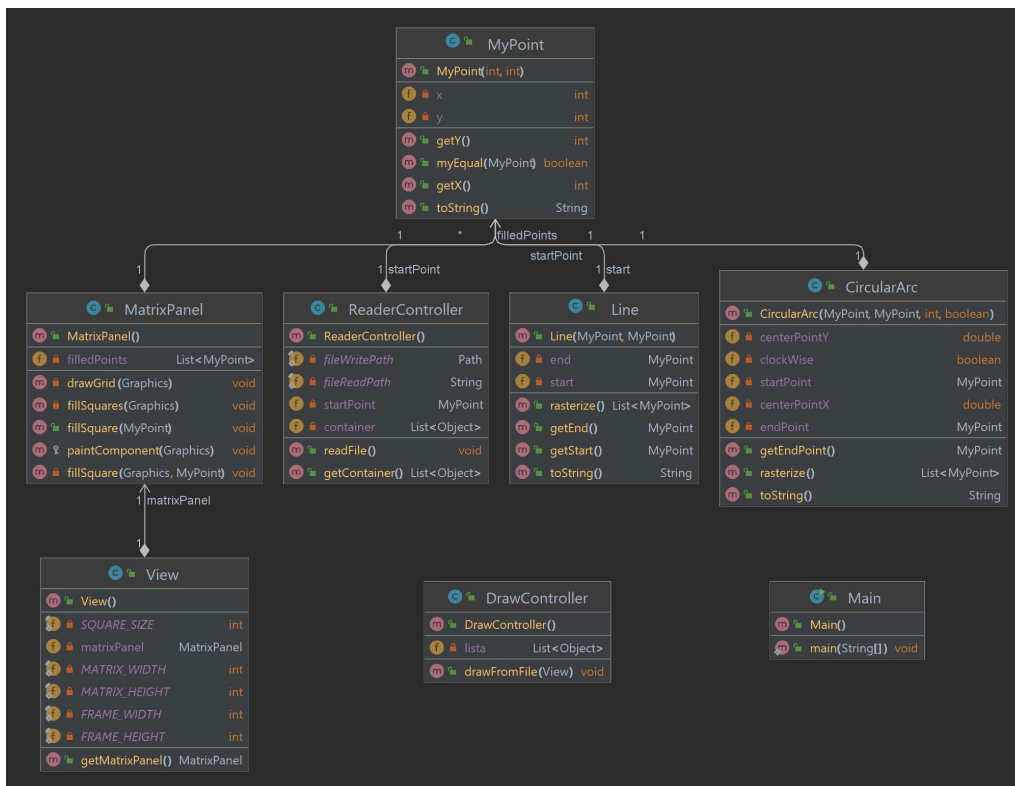


Figura 5: Diagrama UML de clase si de pachete

5. Implementare

5.1 Algoritmul de rasterizare:

- Implementat într-o metoda rasterize în clasa fiecărui tip de traiectorie, Line și CircularArc, acest algoritm returnează o listă de puncte care reprezintă coordonatele pixelilor din matricea creată în interfață, puncte consecutive de la punctul de start până la cel de final create astfel încât traiectoria desenată să fie cât mai apropiată de una reală
- În cazul liniei, funcționarea algoritmului de rasterizare este realizat în felul următor: se calculează o eroare pe baza distanței dintre dx, distanța pe x și dy, distanța pe y dintre punctele de start și de final, în funcție de care se stabilește dacă următorul punct din linie se află la coordonate +1 sau -1 pe x sau pe y, și se adaugă punctul în lista de returnat. (Figura 6)
- În cazul arcului de cerc, acest algoritm începe prin calcularea unghiurilor de start și final, și determinând unghiurile în raport cu centrul cercului. Ulterior, se ajustează aceste unghiuri în funcție de sensul în care se dorește desenarea arcului. Raza cercului este calculată prin aplicarea teoremei lui Pitagora pe baza și înălțimea triunghiului format de centrul cercului și punctul de start. Numărul de puncte de pe arc este determinat de o valoare constantă înmulțită cu raza, iar unghiul de

```

public List<MyPoint> rasterize() {
    List<MyPoint> points = new ArrayList<>();

    int x1 = start.getX();
    int y1 = start.getY();
    int x2 = end.getX();
    int y2 = end.getY();

    int dx = Math.abs(x2 - x1);
    int dy = Math.abs(y2 - y1);
    int sx = (x1 < x2) ? 1 : -1;
    int sy = (y1 < y2) ? 1 : -1;

    int err = dx - dy;

    while (true) {
        points.add(new MyPoint(x1, y1));

        if (x1 == x2 && y1 == y2) {
            break;
        }

        int e2 = 2 * err;
        if (e2 > -dy) {
            err = err - dy;
            x1 = x1 + sx;
        }
        if (e2 < dx) {
            err = err + dx;
            y1 = y1 + sy;
        }
    }

    return points;
}

```

Figura 6: Metoda de rasterizare pentru linie

incrementare este calculat pentru a distribui uniform aceste puncte. Prin intermediul ecuațiilor parametrice ale cercului, se generează coordonatele (x, y) pentru fiecare punct de pe arc într-o buclă. Punctul de sfârșit al arcului este actualizat cu ultimul punct generat, iar lista rezultată de puncte de pe arc este returnată. Acest algoritm oferă o aproximare discretă a arcului de cerc, fiind util în contextul grafic sau de desenare, și permite controlul asupra nivelului de detaliu prin ajustarea numărului de puncte utilizate în rasterizare. (Figura 7)

```
public List<MyPoint> rasterize() {
    List<MyPoint> pointsOnArc = new ArrayList<>();

    double startAngle = Math.atan2(startPoint.getY() - centerPointY, startPoint.getX() - centerPointX);
    double endAngle = Math.atan2(endPoint.getY() - centerPointY, endPoint.getX() - centerPointX);

    double radius = Math.sqrt(Math.pow(centerPointX - startPoint.getX(), 2) + Math.pow(centerPointY - startPoint.getY(), 2));

    if (!clockWise && startAngle < endAngle) {
        startAngle += 2 * Math.PI;
    } else if (clockWise && startAngle > endAngle) {
        endAngle += 2 * Math.PI;
    }

    int numPoints = (int)(5 * radius);
    double angleIncrement = (endAngle - startAngle) / numPoints;

    for (int i = 0; i <= numPoints; i++) {
        double currentAngle = startAngle + i * angleIncrement;
        int x = (int)(centerPointX + radius * Math.cos(currentAngle));
        int y = (int)(centerPointY + radius * Math.sin(currentAngle));
        pointsOnArc.add(new MyPoint(x, y));
    }

    this.endPoint = pointsOnArc.get(pointsOnArc.size() - 1);
    return pointsOnArc;
}
```

Figura 7: Metoda de rasterizare pentru arcul de cerc

5.2 Algoritm de citire din fisier si interpretare G-code:

- Cu ajutorul unui buffer reader se citesc comenzile G-code din fisier, iar cu ajutorul unui regex se verifica daca comenzile primite sunt conforme cu cele asteptate. Pattern-ul care trebuie respectat arata in felul urmator: "(G[0-9]*)(X-?[0-9]*)(Y-?[0-9]*)(R-?[0-9]*)?" Si acesta verifica comenzi de tipul "G00 X.. Y.." si "G02 X.. Y.. R.." sau "G03 X.. Y.. R.." unde ".." pot fi inlocuite de numere. Pentru fiecare nerespectare a pattern-ului se va afisa un mesaj de eroare intr-un fisier cu ce a mers gresit.
- Interpretarea comenzilor consta in verificarea tipului de comenzi, determinarea punctului de final pentru crearea obiectului necesar si adugarea acestuia intr-un container. Pentru crearea punctului final se verifica apartenenta acestuia in grid dupa realizarea comenzilor din G-code. Daca obiectul creat are ca si punct final un punct care nu apartine gridului se va trimite un mesaj in Log.txt cu eroarea intampinata.
- Crearea unui obiect de tipul arc de cerc pe baza comenzilor din fisier se realizeaza in felul urmator: se citește comanda, se creaza punctul de final prin adunarea deplasamentelor pe X si pe Y, apoi se retine raza, iar in final se creaza obiectul respectiv tinand cont si de signatura comenzii: G02 sau G03. Pentru G02 se creaza un arc de cerc care este parcurs in sensul acelor de ceas, iar pentru G03 un arc de cerc parcurs in sens trigonometric (Figura 8). Exact la fel se procedeaza si in cazul liniilor, dar se pastreaza doar punctul final fara raza sau sensul de parcurgere.


```

} else if ((match.group(1).equals("G02") || match.group(1).equals("G03")) && match.group(4) != null && startPoint != null) {
    int d = 0;
    scanner = new Scanner(match.group(4)).useDelimiter("R");
    if (scanner.hasNextInt()) {
        d = scanner.nextInt();
    }

    x = startPoint.getX() + x;
    y = startPoint.getY() + y;
    if (x >= 0 && x <= 289 && y >= 0 && y <= 149) {
        MyPoint endPoint = new MyPoint(x, y);
        CircularArc c = null;
        if (match.group(1).equals("G02")) {
            c = new CircularArc(startPoint, endPoint, d, clockWise: true);
        } else {
            c = new CircularArc(startPoint, endPoint, d, clockWise: false);
        }

        container.add(c);
        startPoint = c.getEndPoint();
        try {
            Files.write(fileWritePath, (" " + c.toString() + '\n').getBytes(), StandardOpenOption.APPEND);
        } catch (IOException e) {
            e.printStackTrace();
        }
    } else {
        String err;
        if (match.group(1).equals("G02")) {
            err = "! Circular arc out of the grid " + new CircularArc(startPoint, new MyPoint(x, y), d, clockWise: true) + "\n";
        } else {
            err = "! Circular arc out of the grid " + new CircularArc(startPoint, new MyPoint(x, y), d, clockWise: false) + "\n";
        }
        try {
            Files.write(fileWritePath, err.getBytes(), StandardOpenOption.APPEND);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
}

```

Figura 8 : Citirea, interpretarea datelor din G-code și crearea obiectelor de tip arc de cerc

5.3 Algoritmul de generare G-code:

- Implementarea pentru generarea G-code în acest cod include trei clase principale: CNCCCommandController, CNCCCommandModel, și CNCCCommandView. Iată o descriere pe scurt a rolurilor fiecărei clase:
- CNCCCommandController:
 - Este clasa controlor care gestionează interacțiunea dintre model și vizualizare.
 - Se ocupă de evenimentul de apăsare al butonului "Generate" din interfața grafică și inițiază generarea G-code.
 - Verifică dacă datele introduse în interfață sunt valide (numere) și afișează un mesaj pop-up în consecință.
 - În funcție de comanda selectată, construiește comanda G-code corespunzătoare și o scrie într-un fișier ("Trajectory5.txt").
 - Utilizează clasa PopupMessageFrame pentru afișarea mesajelor pop-up cu diferite culori și dimensiuni ale textului.
- CNCCCommandModel:
 - Este clasa model care reprezintă starea aplicației.
 - Păstrează comanda G-code generată.
- CNCCCommandView:
 - Este clasa vizualizare care se ocupă de interfața grafică (GUI) a aplicației.
 - Folosește Java Swing pentru a crea interfața cu utilizatorul (JFrame, JComboBox, JLabel, JTextField, JButton).
 - Dispune de un ComboBox pentru selectarea comenzii dorite, câmpuri de introducere pentru coordonatele (X, Y, R), și un buton pentru generarea G-code.

- Include un mesaj pop-up pentru afișarea rezultatului generării G-code.
- PopupMenuFrame:
 - Este o fereastră (JFrame) specializată pentru a afișa mesaje pop-up.
 - Poate fi personalizată pentru a afișa mesaje de succes (verde) sau de eroare (roșu).
 - Dimensiunile ferestrei și colțurile pot fi ajustate.
 - Mesajele dispar automat după un interval de timp (în acest caz, 5 secunde).
 - Împreună, aceste clase implementează un sistem simplu pentru generarea și afișarea G-code, interacționând în conformitate cu arhitectura Model-View-Controller (MVC).

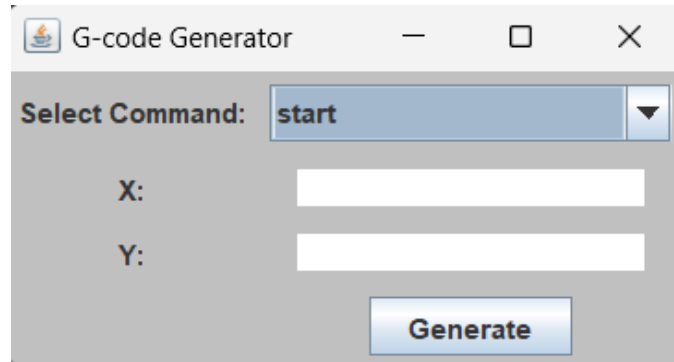


Figura 9: Interfața de utilizare a programului de generare G-code

6. Testare și validare

Validarea soluției am realizat-o prin crearea mai multor teste, în fișiere .txt care conțin comenzi pentru simularea unor traiectorii pe care aparatul urmează să le parcurgă și după modelul cărora să poată tăia. Conținutul fișierelor constă în tipuri de traiectorii care acoperă toate cazurile în care aparatul poate ajunge.

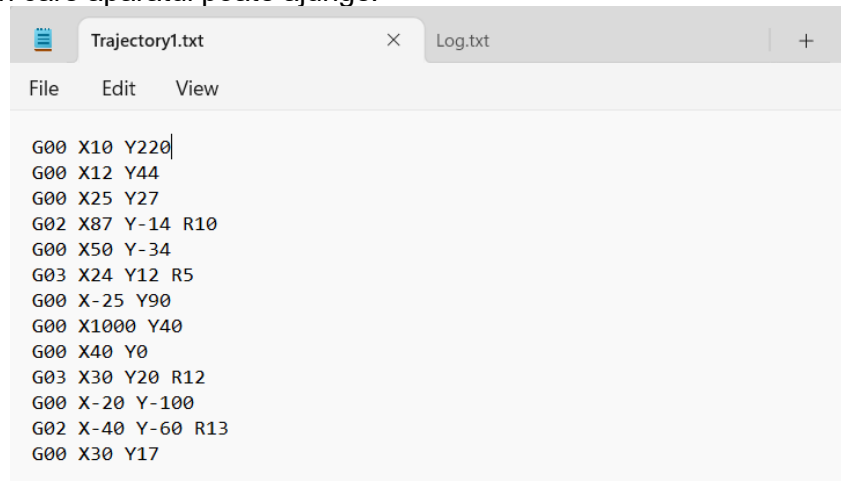


Figura 10: Formatul unui fișier de testare a simulării

În fișierul Trajectory1.txt se află un test în care apar toate mesajele de eroare pe care le poate întâmpina și toate cazurile posibile, cum ar fi: Setarea punctului de start în afara gridului, trasarea unei linii care are ca punct final un punct în afara gridului, și un arc de cerc care are ca și punct final un punct în afara gridului. Pentru aceste cazuri aparatul își continuă execuția și ignora aceste tipuri de traiectorii continuând cu următoarele din fișier și din punctul în care acesta a rămas. Un alt caz acoperit este cel de a desena un arc care depășește gridul dar punctele de început și de final sunt în interiorul gridului, tip de traiectorie pentru care algoritmul își continuă execuția, aparatul continuă să taie dar fără a depăși gridul, punctele din afara acestuia fiind ignorate.

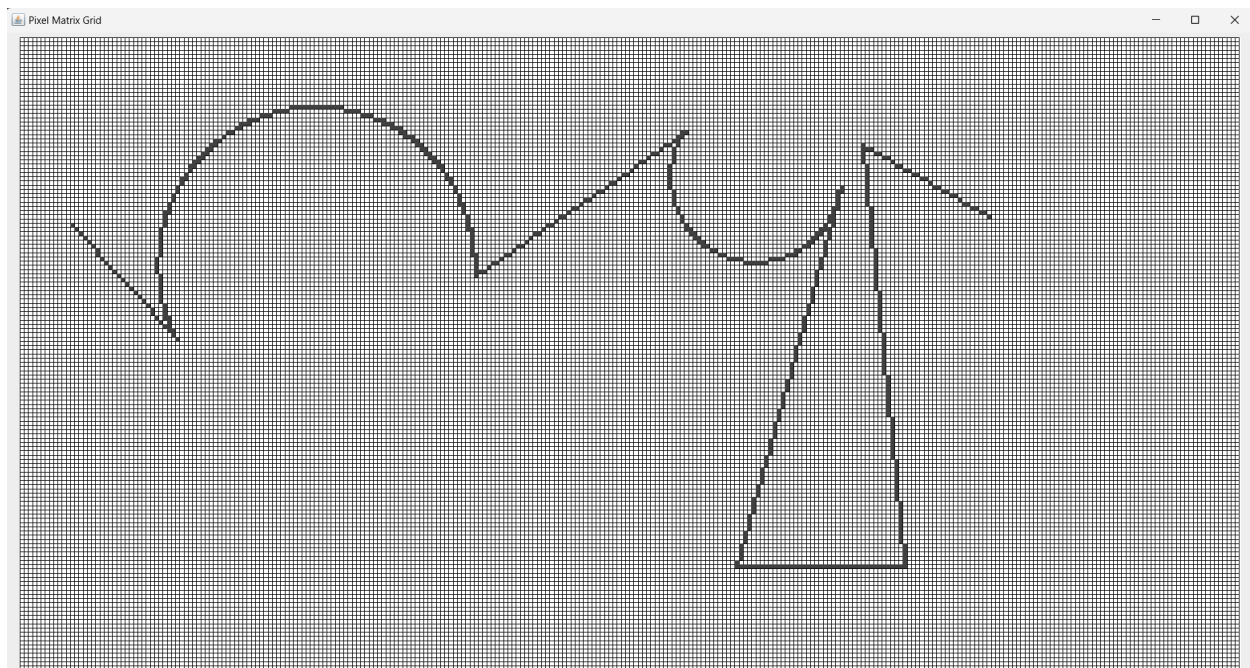


Figura 11: Simulare pentru comenzile din fisierul Trajectory1.txt

In fisierele Trajectory2.txt si Trajectory3.txt se afla teste cu aplicabilitate, definesc traiectorii de taiere sub diferite forme. In fisierul Trajectory2.txt se afla forma unui brad unde se testeaza simetria si rasterizarea diferitelor linii. In Trajectory3.txt apare forma unei mese de biliard unde apar si arce de cerc tot simetrice pe masa si se testeaza rasterizarea lor.

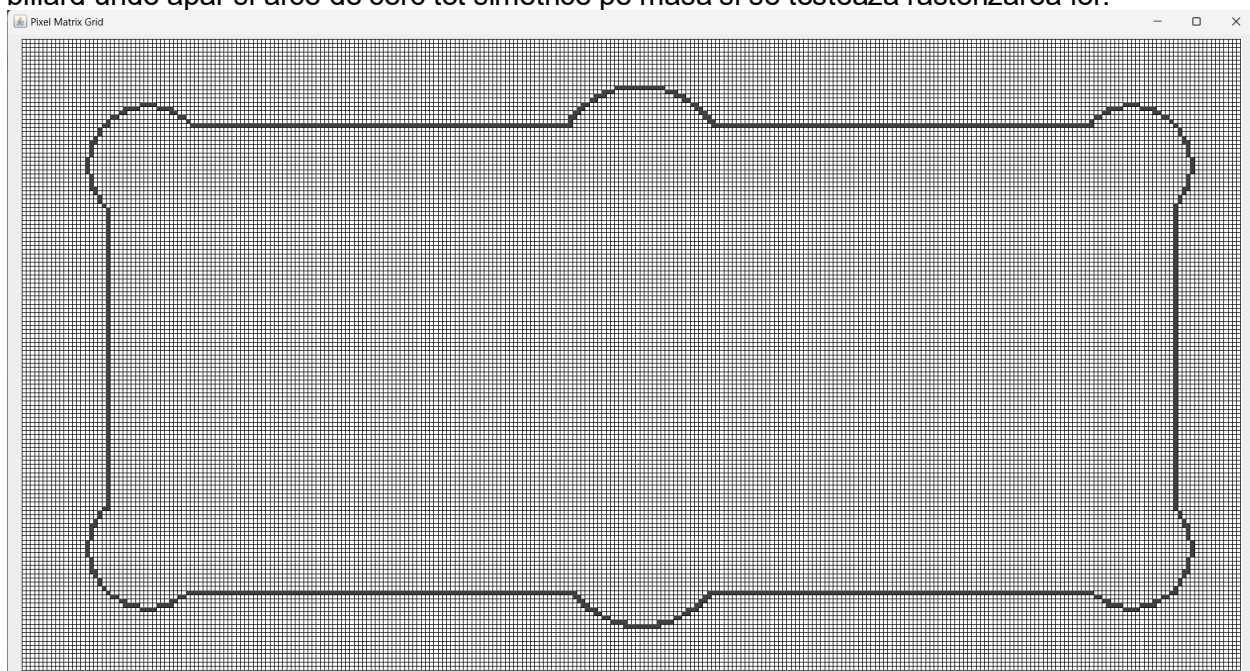
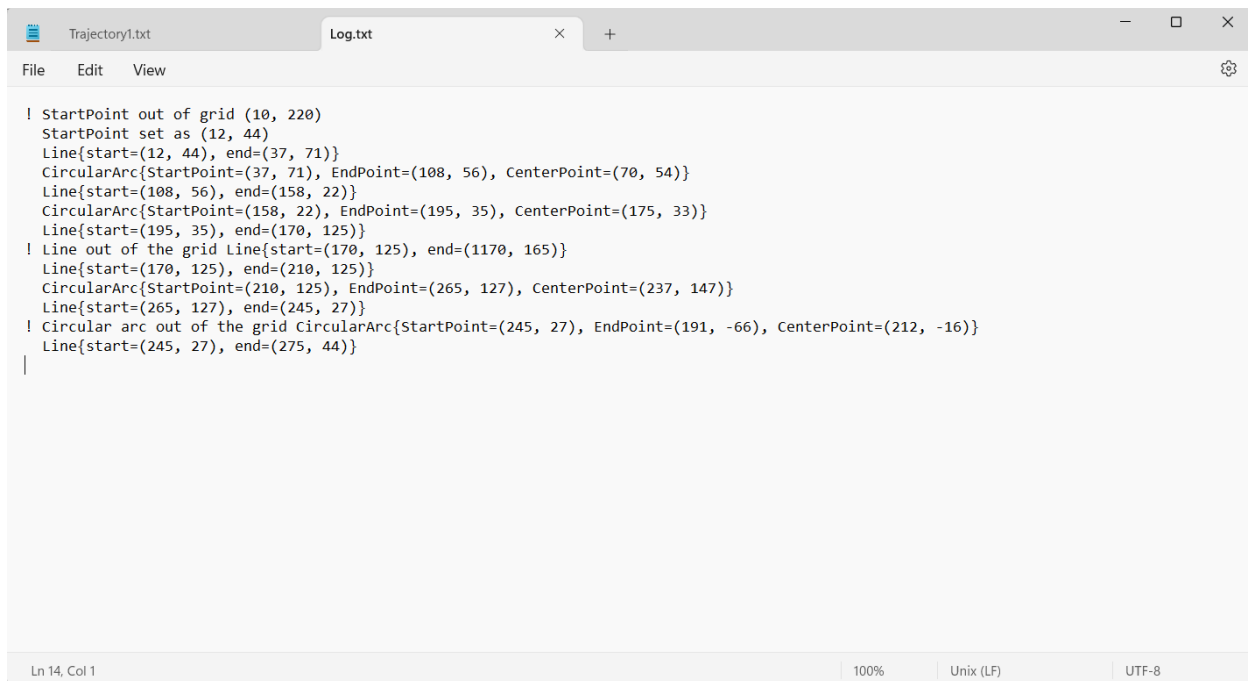


Figura 12: Simulare pentru comenzile din Trajectory3.txt – masa de biliard

Toate mesajele sunt afisate intr-un alt fisier Log.txt unde apar traiectoriile desenate si toate mesajele de eroare intampinate in realizarea taieturii. Pentru a le distinge mai usor, erorile apar cu un "!" inainte la inceput de rand.



```
! StartPoint out of grid (10, 220)
StartPoint set as (12, 44)
Line{start=(12, 44), end=(37, 71)}
CircularArc{StartPoint=(37, 71), EndPoint=(108, 56), CenterPoint=(70, 54)}
Line{start=(108, 56), end=(158, 22)}
CircularArc{StartPoint=(158, 22), EndPoint=(195, 35), CenterPoint=(175, 33)}
Line{start=(195, 35), end=(170, 125)}
! Line out of the grid Line{start=(170, 125), end=(1170, 165)}
Line{start=(170, 125), end=(210, 125)}
CircularArc{StartPoint=(210, 125), EndPoint=(265, 127), CenterPoint=(237, 147)}
Line{start=(265, 127), end=(245, 27)}
! Circular arc out of the grid CircularArc{StartPoint=(245, 27), EndPoint=(191, -66), CenterPoint=(212, -16)}
Line{start=(245, 27), end=(275, 44)}
```

Figura 13: Fisierul Log.txt – fisierul de verificare a simulării cu mesaje de validare și erori

7. Concluzii

- Proiectul propus vizează dezvoltarea unui utilaj de tăiat cu flama cu comandă numerică, cu scopul de a realiza tăieturi precise pe baza traiectoriilor specificate într-un fișier de intrare. În acest context, s-au stabilit obiectivele principale ale proiectului, inclusiv implementarea unui sistem de control numeric, dezvoltarea unui software de simulare.
- S-a analizat aspecte esențiale ale proiectului, precum stocarea datelor în fișier, interpretarea acestora cu ajutorul G-code, utilizarea arhitecturii MVC, și modul în care utilizatorul interpretează rezultatele.
- S-a subliniat, de asemenea, importanța testării și depanării, precum și simularea deplasării capului de tăiere în cadrul interfeței grafice.
- S-a propus o structură clară a proiectului, evidențiind interacțiunea dintre modulele Model, View și Controller, iar diagrama UML de clase și pachete a oferit o imagine de ansamblu asupra structurii aplicației.
- Implementarea a inclus algoritmul de rasterizare pentru traiectorii, algoritmul de citire din fișier și interpretare G-code, precum și dezvoltarea interfeței grafice cu ajutorul Java Swing.
- Pentru testare și validare, au fost create scenarii de test care acoperă diverse situații, inclusiv cazuri de eroare și forme complexe de traiectorii. Testarea a fost efectuată atât pentru rezultatele așteptate, cât și pentru mesajele de eroare în cazurile neconforme.
- În final, proiectul a fost implementat și testat cu succes, demonstrând eficiența și funcționalitatea sistemului de tăiere cu flama cu comandă numerică. Soluția oferă o interfață intuitivă pentru utilizatori și aduce beneficii semnificative în domeniul prelucrării materialelor cu ajutorul utilajului propus.

8. Bibliografie

1. "Programare Java 7 – Utilizarea claselor legate de grafica, fonturi si culori":
<https://www.scribub.com/stiinta/informatica/java/Programare-Java33947.php>
2. "Fisiere text in Java" – Doru Anastasiu Popescu:
<https://www.dopopan.ro/liceu/F3.pdf>
3. "Java Regular Expressions" – W3schools:
https://www.w3schools.com/java/java_regex.asp
4. "MVC Design Pattern" – GeeksforGeeks:
<https://www.geeksforgeeks.org/mvc-design-pattern/>
5. "Java Read Files" – W3Schools:
https://www.w3schools.com/java/java_files_read.asp
6. "Java ArrayList" – W3Schools:
https://www.w3schools.com/java/java_arraylist.asp
7. "Line Rasterization" – Frédo Durand and Seth Teller:
https://groups.csail.mit.edu/graphics/classes/6.837/F02/lectures/6.837-7_Line.pdf
8. "JUnit" - Test Framework:
https://www.tutorialspoint.com/junit/junit_test_framework.htm
9. G-code Explained | List of Most Important G-code Commands – Dejan:
https://howtomechatronics.com/tutorials/g-code-explained-list-of-most-important-g-code-commands/?utm_content=cmp-true
10. Drawing on a JButton[][] grid in a Java GUI – StackOverflow:
<https://stackoverflow.com/questions/59942354/drawing-on-a-jbutton-grid-in-a-java-gui>