Manual Técnico

Proyecto: Modelado CNC 1313 router

INFRAESTRUCTURA II

Universidad de Central del Ecuador

Facultad de Ingeniería y Ciencias Aplicadas

Sistemas de Información

Julio

Modelado CNC 1313 router

MANUAL TÉCNICO

1. Objetivos

Este documento técnico tiene como propósito describir de manera detallada, precisa y profesional el funcionamiento integral del proyecto de simulación de una máquina CNC Router 1313 utilizando Unity. La simulación está diseñada con fines pedagógicos y permite a los estudiantes comprender el funcionamiento de una máquina CNC real a través de una interfaz virtual que simula con alta fidelidad los movimientos y procesos de corte en base a coordenadas G-code.

El sistema contempla la simulación de cortes en un plano rectangular, permitiendo ingresar coordenadas manuales, cargar archivos G-code, o incluso procesar una imagen mediante visión artificial para generar rutas de corte automáticas.

1.1 Objetivos Específicos.

- Proporcionar una simulación visual y funcional de una máquina CNC Router 1313, adecuada para entornos educativos.
- Implementar la lectura e interpretación de código G (G-code), reconociendo movimientos de desplazamiento (G0) y de corte (G1).
- Desarrollar un sistema flexible que permita al usuario ingresar coordenadas manuales o cargar archivos predefinidos.
- Introducir una función innovadora de procesamiento de imágenes mediante Python y OpenCV para convertir siluetas en trayectorias G-code.
- Simular los movimientos de la máquina sobre ejes X, Y y Z y representar visualmente el trazo del corte con efecto de partículas y trail visual.
- Presentar una introducción interactiva en video para la comprensión inicial del sistema y sus funcionalidades.

2. Alcance

El sistema está orientado principalmente a estudiantes de tecnología, electrónica, ingeniería mecánica o carreras afines, permitiendo un acercamiento práctico a los conceptos de mecanizado CNC, programación G-code y control numérico computarizado.

Desde un enfoque educativo, el simulador busca desarrollar competencias en:

- Lectura e interpretación de G-code.
- Programación de coordenadas de corte.
- Planeación de trayectorias.
- Comprensión de las limitaciones físicas del espacio de corte.
- Adaptación de imágenes a formatos vectoriales.

Además, el sistema puede ser ampliado para propósitos más avanzados como generación automática de trayectorias desde diseños CAD o extensión a lenguajes G-code más completos.

3. Requerimientos Técnicos

3.1 Requerimientos Mínimos de Hardware.

- Procesador: Intel Core i3 o superior (recomendado i5+)
- RAM: 4 GB mínimo (8 GB recomendado)
- Almacenamiento: 1 GB libre
- Gráfica: Integrada compatible con DirectX 10 o superior
- Resolución mínima de pantalla: 1280x720

3.2 Requerimientos Mínimos de Software.

- Sistema Operativo: Windows 10/11 64 bits
- Unity Editor 2021.3.3f1 o superior
- Python 3.8+ (si se utiliza procesamiento de imágenes)
- Librerías Python requeridas: opencv-python, numpy

3.3 Requerimientos Mínimos de Usuario.

- Privilegios para ejecutar Unity o aplicaciones compiladas
- Permiso para ejecutar scripts externos desde Unity si se activa la función Python
- Familiaridad básica con carpetas y rutas de archivos

4. Herramientas Utilizadas para el Desarrollo.

Unity

- Motor principal de simulación
- Interfaz gráfica, animaciones y código en C#
- Sistema de lectura de archivos G-code
- Integración con efectos visuales y control de la máquina

Visual Studio

- Desarrollo del backend en C#
- Debugging y manejo de clases y scripts

Python + OpenCV

- Procesamiento de imágenes
- Detección de contornos
- Generación de coordenadas adaptadas a los límites de corte (X: -175 a 175, Z: -200 a 200)

```python
import cv2
img = cv2.imread('figura.png', 0)
ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
contornos, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for contorno in contornos:
    for punto in contorno:
        x, y = punto[0]
        print(f"G1 X{x_scaled} Y-220 Z{z_scaled}")
```

Canva / PowerPoint

- Creación de material educativo e introductorio

Otros Recursos

- Partículas para simular chispas
- G-code predefinido para figuras geométricas y artísticas (flechas, espirales, zigzag, estrella, etc.)

5. Instalación

Descargar el ejecutable del proyecto o abrir en Unity desde el proyecto fuente.

Si se desea utilizar la función de generar G-code desde imágenes:

- Instalar Python.
- Ejecutar en CMD: pip install opencv-python
- Colocar el script “imagen_a_gcode.py” dentro de la carpeta raíz.
- Asegurarse de que Unity tenga permisos para ejecutar scripts externos (configurado desde File → Build Settings → Player → Scripting Runtime).

Copiar el archivo de G-code que se desea ejecutar a la carpeta: Assets/Resources/gcodes

Ejecutar el proyecto desde Unity o el .exe si está compilado.

6. Configuración

- Coordenadas máximas de corte definidas: X [-175, 175], Z [-200, 200]
- Se utiliza el eje Y como eje fijo para simular altura.
- Archivos .gcode deben colocarse en la carpeta Assets/Resources/gcodes y tener extensión .txt (ejemplo: ejemplo.gcode.txt).
- El usuario puede elegir entre:
  - Ingreso manual de coordenadas (X, Y, Z)
  - Carga de archivo G-code predefinido
  - Conversión desde imagen (requiere Python + OpenCV)

7. Diseño de la Arquitectura Física.

- Aplicación desarrollada completamente en entorno local.
- No requiere conexión a red para ejecución.
- Procesamiento se realiza mediante componentes de Unity sin dependencia de servidores.

Equipos: PC local del usuario final.

Ubicación física: Laboratorio de informática, aula o entorno educativo.

Puertos TCP/UDP: No aplican (no requiere comunicación en red).

Dependencias: Python solo si se utiliza la función de imagen a G-code

8. Funcionamiento.

- Al iniciar el juego, se muestra una introducción en video que explica el uso de la máquina CNC Router 1313, sus funciones, objetivos, tipos de uso y manejo del entorno virtual.
- El usuario puede elegir: a. Ingresar coordenadas manuales (X, Y, Z). b. Cargar un archivo G-code desde la carpeta /Assets/Resources/gcodes (ejemplo.gcode.txt). c.
- Cargar una imagen para procesar sus contornos con IA y convertirla en G-code.
- Una vez cargada la información, se procesa cada punto mediante el script CoordinateInputManager.cs, y se envía a la clase CncSimulator.cs, que gestiona el movimiento de los ejes virtuales.
- La máquina se mueve mediante instrucciones G0 (desplazamiento sin corte) y G1 (movimiento con corte). Ejemplo:

```
G0 X0 Y-220 Z100   ; #movimiento rapido sin cortar
G1 X100 Y-220 Z0    ; #corte activo
```

- Se simula el descenso del cabezal de corte (eje Y), el movimiento horizontal (X, Z) y el ascenso posterior. El efecto visual del corte se logra mediante TrailRenderer y sistema de partículas:

```
if (esCorte)
    trail.emitting = true;
else
    trail.emitting = false;
```

- El corte se visualiza sobre un plano virtual con coordenadas cartesianas.

- El sistema emite partículas (chispas) para representar el efecto de corte activo mediante sistemas como:

```
sparkParticles.Play();
```

- Finalizado el proceso, se limpia la interfaz y se permite volver a cargar otro archivo o ingresar nuevas coordenadas.

9. Interfaz de Usuario.

- Botón "Agregar Punto": Agrega coordenadas individuales manualmente.
- Botón "Cargar y Cortar G-code": Lee el archivo y procesa los puntos.
- Botón "Mostrar Vista Previa": Traza la trayectoria sin cortar.
- Botón "Limpiar Vista Previa": Borra el trazo anterior.
- Botón "Cargar desde Imagen": Procesa una imagen con Python y genera G-code adaptado.
- Video inicial: reproduce explicación pedagógica sobre el sistema.

10. Usuarios

Usuario Educativo:

- Perfil: Estudiante o aprendiz.
- Acciones permitidas: Cargar código, visualizar trayectorias, simular corte.

Usuario Docente / Administrador:

- Perfil: Profesor, técnico o programador.
- Acciones permitidas: Modificación de scripts, ajustes de coordenadas, personalización de contenido.

11. Estructura y Funcionamiento de Scripts del CNC 1313

A continuación, se describen los scripts desarrollados en C# para Unity:

CncPreview.cs

Este componente de Unity se utiliza para mostrar con una línea verde el recorrido que realizará la herramienta de corte CNC. Esto sirve como una herramienta visual de verificación para evitar errores antes de iniciar el corte real. Usa el componente LineRenderer de Unity para dibujar la trayectoria.

Librerías utilizadas

```
1   using System.Collections.Generic;
2   using UnityEngine;
```

using System.Collections.Generic;

- Permite el uso de listas y otras estructuras de datos genéricas, como List<Vector3>.
- Importante para manejar dinámicamente conjuntos de puntos 3D.

using UnityEngine;

- Accede a todos los elementos del motor de Unity como MonoBehaviour, GameObject, Material, Shader, LineRenderer, Vector3, etc.

Funcionamiento del script

```
4    [RequireComponent(typeof(LineRenderer))]
```

- Esta línea asegura que cualquier GameObject al que se le asigne este script tenga un componente LineRenderer. Si no lo tiene, Unity lo añadirá automáticamente.
- El LineRenderer es el encargado de dibujar líneas en el mundo 3D.

```
9     private void Awake()
10    {
11        lineRenderer = GetComponent<LineRenderer>();
12        lineRenderer.positionCount = 0;
13        lineRenderer.widthMultiplier = 0.5f;
14        lineRenderer.material = new Material(Shader.Find("Sprites/Default"));
15        lineRenderer.startColor = Color.green;
16        lineRenderer.endColor = Color.green;
17    }
```

- Se ejecuta al inicio. Configura el LineRenderer:
- Se le da un ancho de línea (0.5 unidades).
- Se asigna un material simple para que pueda renderizarse correctamente.
- Se definen colores verdes al inicio y al final de la línea.
- Se inicializa sin puntos (positionCount = 0).

```
public void MostrarPreview(List<Vector3> puntos)
```

- Verifica que haya puntos válidos.
- Recorre todos los puntos de la lista.
- Convierte cada punto a una nueva posición en la que:
  - El eje X se mantiene igual.
  - El eje Z se mantiene igual.
  - El eje Y se fija en -220f para simular la posición del cabezal de corte sobre la superficie de trabajo.

```
Vector3[] posicionesConvertidas = new Vector3[puntos.Count];

for (int i = 0; i < puntos.Count; i++)
{
    // Solo trazamos sobre la pieza, es decir, en la parte baja
    posicionesConvertidas[i] = new Vector3(
        puntos[i].x,
        -220f, // Y más bajo (zona de corte)
        puntos[i].z
    );
}
```

- Activa el LineRenderer, asigna la cantidad de posiciones y dibuja la trayectoria con:

```
lineRenderer.SetPositions(posicionesConvertidas);
```

- Elimina la línea dibujada si el usuario desea limpiar la vista previa.

```
public void LimpiarPreview()
{
    lineRenderer.positionCount = 0;
}
```

CncSimulator.cs

Este script controla el movimiento de los ejes X, Y y Z del cabezal de la máquina CNC virtual. Utiliza corrutinas para simular movimientos realistas con velocidades definidas por el usuario, y además activa un rastro visual (TrailRenderer) para representar el corte cuando el eje Y está en posición baja (es decir, cuando se está cortando).

Se encuentra dentro del namespace CNC.Simulator, lo que permite encapsular su funcionalidad de forma organizada.

Librerías Utilizadas

```
using System.Collections;
using UnityEngine;

namespace CNC.Simulator
```

using System.Collections;

- Permite usar IEnumerator y corrutinas (IEnumerator MoverAHacia).
- Corrutinas permiten realizar acciones a lo largo del tiempo sin congelar el juego.

using UnityEngine;

- Proporciona acceso a clases del motor como MonoBehaviour, Transform, Vector3, Mathf, TrailRenderer, etc.
- Esencial para interactuar con componentes visuales y lógicos del juego en Unity.

namespace CNC.Simulator

- Agrupa las clases relacionadas con la simulación CNC para mejor organización y evitar conflictos de nombres.

Función principal:

- Controla el movimiento del cabezal CNC simulando descenso, desplazamiento y ascenso.
- Interpreta cuándo debe estar activo el "corte" mediante la variable esCorte.
- Controla visualmente el "rastro" de corte mediante TrailRenderer.
- Limita el movimiento del cabezal dentro de un rango definido para X, Y y Z.
- Gestiona un movimiento fluido entre puntos.

Variables y Componentes Clave

```
public Transform mov_izq_der;
public Transform mov_arrib_abajo;
public Transform movimiento_adel_atr;

public float velX = 1f;
public float velY = 1f;
public float velZ = 1f;

public Vector2 limiteX = new Vector2(-100, 100);
public Vector2 limiteY = new Vector2(-220, -180);
public Vector2 limiteZ = new Vector2(-100, 100);

public TrailRenderer trail;
```

public Transform mov_izq_der;

Movimiento horizontal sobre el eje X.

public Transform mov_arrib_abajo;

Movimiento vertical (descenso/subida del cabezal), eje Y.

public Transform movimiento_adel_atr;

Movimiento en el eje Z (profundidad en el plano de corte).

public float velX, velY, velZ;

Velocidades de desplazamiento para cada eje.

public Vector2 limiteX, limiteY, limiteZ;

Límites de movimiento para evitar que el cabezal salga del área simulada de corte.

public TrailRenderer trail;

Componente visual que deja un "trazo" o "camino" cuando se activa el corte.

Funcionamiento del Script

- El método público MoverAHacia recibe un Vector3 (coordenada destino) y un booleano esCorte.
    - Clampea las coordenadas al rango permitido (evita salirse del área).
    - Llama a una corrutina para mover secuencialmente.
- MoverSecuencia realiza el movimiento en tres etapas:

  Si esCorte es true:
    - Baja el eje Y simulando que el cabezal baja para cortar.

  Activa el trail y mueve los ejes X y Z hacia la posición deseada.

  Desactiva el trail una vez finalizado el movimiento horizontal.

  Si esCorte era true:
    - Sube nuevamente el eje Y simulando que el cabezal terminó de cortar.

```
if (esCorte)
{
    while (mov_arrib_abajo.localPosition.y > limiteY.x + 0.1f)
    {
        Vector3 posY = mov_arrib_abajo.localPosition;
        posY.y = Mathf.MoveTowards(posY.y, limiteY.x, velY * Time.deltaTime);
        mov_arrib_abajo.localPosition = posY;
        yield return null;
    }
}
```

- Este fragmento baja el cabezal simuladamente antes de cortar.

Ejemplo de uso:

Supongamos que queremos mover el cabezal a X=50, Y=-220, Z=100 con corte activo:

```
StartCoroutine(cncSimulator.MoverAHacia(new Vector3(50, -220, 100), true));
```

GcodeParser.cs

Este script es responsable de leer un archivo G-code (ejemplo.gcode ubicado en /Assets/Resources/gcodes/) y convertir sus instrucciones en una lista de puntos en el espacio tridimensional (Vector3), junto con una etiqueta que indica si el movimiento es de tipo corte (G1) o solo desplazamiento (G0). Esta lista será luego usada por el simulador para mover el cabezal CNC virtual y trazar el recorrido.

Librerías utilizadas

```
1   using System.Collections.Generic;
2   using UnityEngine;
3
4   namespace CNC.Simulator
```

using System.Collections.Generic:

Permite utilizar List<> y otras estructuras de colección como Tuple.

using UnityEngine:

Permite el uso de Vector3, Debug.LogError, TextAsset, y Resources.Load.

namespace CNC.Simulator:

Encapsula el script en un espacio lógico que organiza los componentes del simulador.

Funcionamiento detallado

- Se accede al archivo de texto usando Resources.Load<TextAsset>(), que es la manera estándar de acceder a recursos en la carpeta Resources en Unity.
- El archivo se divide en líneas y se procesa cada línea una por una.
- Si una línea empieza con G1, se considera que debe hacer un corte (esCorte = true).
- Si una línea empieza con G0, se interpreta como un desplazamiento sin corte.
- Se ignoran otras líneas que no contienen instrucciones G0 o G1.

Cada línea válida se procesa para extraer valores de X, Y y Z. Se construye un Vector3 con estas coordenadas, y se guarda junto a la bandera esCorte en una lista.

Este bloque es clave para identificar el tipo de instrucción y extraer coordenadas:

```
    foreach (string parte in partes)
    {
        if (parte.StartsWith("X"))
            float.TryParse(parte.Substring(1), out x);
        else if (parte.StartsWith("Y"))
            float.TryParse(parte.Substring(1), out y);
        else if (parte.StartsWith("Z"))
            float.TryParse(parte.Substring(1), out z);
    }

    puntos.Add((new Vector3(x, y, z), esCorte));
}
```

Posición en el flujo del sistema:

Este script es invocado desde el script de Coordinación (por ejemplo, CoordinateInputManager.cs o desde un botón) y su resultado (la lista de puntos) se entrega al CncSimulator para ser ejecutada como ruta de movimiento.

CoordinateInputManager.cs

Este script actúa como controlador principal de interacción entre el usuario y el sistema de simulación. Coordina las acciones que el usuario realiza en la interfaz gráfica (UI) de Unity, permitiendo agregar manualmente coordenadas, cargar archivos G-code, lanzar el proceso de corte, y visualizar una vista previa del recorrido sobre el área de trabajo.

En otras palabras, se comporta como el puente entre la entrada del usuario (InputFields, botones) y la ejecución del movimiento del CNC en pantalla.

Librerías utilizadas

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.UI;
5   using CNC.Simulator;
```

using UnityEngine:

Elementos gráficos, coordenadas, referencias a componentes.

using UnityEngine.UI:

Manejo de InputField, GameObject CanvasUI.

using System.Collections.Generic:

Para usar la lista de coordenadas (List<Vector3>).

using System.Collections:

Para usar corutinas (IEnumerator) que permiten esperar entre pasos de la animación de corte.

using CNC.Simulator:

Para acceder a las clases CncSimulator y GcodeParser.

Principales métodos:

AgregarPunto()

```
public void AgregarPunto()
{
    if (float.TryParse(inputX.text, out float x) &&
        float.TryParse(inputY.text, out float y) &&
        float.TryParse(inputZ.text, out float z))
    {
        puntosParaCorte.Add((new Vector3(x, y, z), true)); // Asume corte por defecto
        Debug.Log($"Punto agregado manualmente: {x}, {y}, {z}");
    }
    else
    {
        Debug.LogWarning("Entrada inválida");
    }
}
```

Permite al usuario ingresar coordenadas manuales desde la UI.

- Convierte texto a float y añade un nuevo punto como corte (G1 por defecto).
- Muestra un mensaje de advertencia si la entrada no es válida.

CargarYCortarDesdeGcode()

```
public void CargarYCortarDesdeGcode()
{
    List<(Vector3, bool)> puntos = GcodeParser.ParsearGcode();

    if (puntos.Count == 0)
    {
        Debug.LogWarning("No se cargaron puntos desde el archivo G-code.");
        return;
    }

    puntosParaCorte.Clear();
    puntosParaCorte.AddRange(puntos);

    if (CanvasUI != null)
        CanvasUI.SetActive(false);

    StartCoroutine(ProcesarCorte());
}
```

- Llama al método ParsearGcode() de GcodeParser.cs para leer el archivo G-code.
- Agrega todos los puntos extraídos a la lista puntosParaCorte.
- Oculta la interfaz (CanvasUI).
- Llama al proceso de corte usando una coroutine.

Cortar()

```
public void Cortar()
{
    if (puntosParaCorte.Count == 0)
    {
        Debug.LogWarning("No hay puntos para cortar");
        return;
    }

    if (CanvasUI != null)
        CanvasUI.SetActive(false);

    StartCoroutine(ProcesarCorte());
}
```

- Verifica si hay puntos cargados.
- Si hay puntos, ejecuta el corte con ProcesarCorte().

MostrarVistaPrevia()

```
public void MostrarVistaPrevia()
{
    List<Vector3> puntosSolo = new List<Vector3>();
    foreach (var p in puntosParaCorte)
        puntosSolo.Add(p.punto);

    cncPreview.MostrarPreview(puntosSolo);
}
```

- Extrae los puntos (solo posiciones, ignora G0/G1) y se los pasa al componente CncPreview.cs para mostrar la ruta con líneas verdes.

LimpiarVistaPrevia()

```
public void LimpiarVistaPrevia()
{
    cncPreview.LimpiarPreview();
}
```

- Llama al método LimpiarPreview() de CncPreview.cs para borrar la línea de vista previa.

ProcesarCorte() (corutina)

```
private IEnumerator ProcesarCorte()
{
    cncSimulator.trail.Clear();
    cncSimulator.trail.emitting = false;

    foreach (var paso in puntosParaCorte)
    {
        Vector3 punto = paso.punto;
        bool esCorte = paso.esCorte;

        yield return cncSimulator.MoverAHacia(punto, esCorte);
    }

    puntosParaCorte.Clear();
}
```

- Recorre la lista de puntos.
- Llama a cncSimulator.MoverAHacia(punto, esCorte) para mover el cabezal de la máquina virtual punto por punto, respetando si es G0 o G1.

Funcionamiento detallado

El script hace uso de varios componentes del proyecto:

- InputFields: Para que el usuario introduzca valores numéricos de X, Y, Z.
- CncSimulator: Clase que mueve los ejes del cabezal.
- CncPreview: Componente de línea que visualiza la ruta.
- CanvasUI: Panel general de la interfaz, que se oculta al momento de cortar.

Además, mantiene una lista interna (puntosParaCorte) con las instrucciones de corte (posición + tipo G0 o G1).

Ejemplo de uso en la interfaz

- El usuario llena campos X, Y, Z → Presiona “Agregar Punto”.
- Luego puede presionar “Mostrar Vista Previa”.
- Si está conforme, presiona “Cortar”.

```
public class CoordinateInputManager : MonoBehaviour
{
    public InputField inputX;
    public InputField inputY;
    public InputField inputZ;
    public CncPreview cncPreview;
    public GameObject CanvasUI;
```

12. Extensiones posibles a futuro

- Generación directa de G-code desde dibujos vectoriales SVG.
- Exportación del trazo generado como archivo .gcode para su uso en máquinas reales.
- Simulación de velocidades variables o cambio de herramienta.
- Ampliación del espacio de corte y simulación de colisiones o errores comunes.
- Registro de resultados o tiempos de ejecución por figura.
- Mejora del sistema de partículas según la intensidad de corte.
- Visualización del cabezal en perspectiva 3D dinámica

13. Contingencias y soluciones.

Si no se carga el G-code: Verifique que el nombre del archivo sea correcto, esté ubicado en la carpeta /Assets/Resources/gcodes/, y tenga la extensión adecuada (.gcode.txt).

Si se dibujan líneas no deseadas entre puntos: Asegúrese de que en el archivo G-code estén correctamente separados los movimientos con G0 (sin corte) y G1 (con corte).

Si la imagen no genera coordenadas: Verifique que la biblioteca OpenCV esté instalada en el entorno de Python y que no haya errores en el script.

Si el video de introducción no se reproduce: Asegúrese de que el archivo esté correctamente ubicado en StreamingAssets y que su nombre coincida con el especificado en el script.

14. Conclusiones y Recomendaciones.

CONCLUSIONES

1. El proyecto desarrollado ofrece una representación realista, interactiva y educativa del funcionamiento de una máquina CNC Router 1313, permitiendo comprender conceptos fundamentales de mecanizado computarizado sin riesgo físico ni necesidad de maquinaria real.
2. La implementación del lenguaje G-code en la lógica de Unity refleja fielmente los comandos básicos (G0 y G1), enseñando cómo se traduce una secuencia de instrucciones en movimientos físicos del cabezal y trayectorias de corte sobre un plano.
3. El sistema modular permite alternar entre ingreso manual de coordenadas, carga de archivos G-code y generación automática de trayectorias desde imágenes, demostrando versatilidad e integración con herramientas externas como Python y OpenCV.
4. La inclusión de elementos visuales como el TrailRenderer y las partículas mejora considerablemente la experiencia del usuario y refuerza el entendimiento del proceso de corte, haciendo más intuitiva la relación entre código y acción.

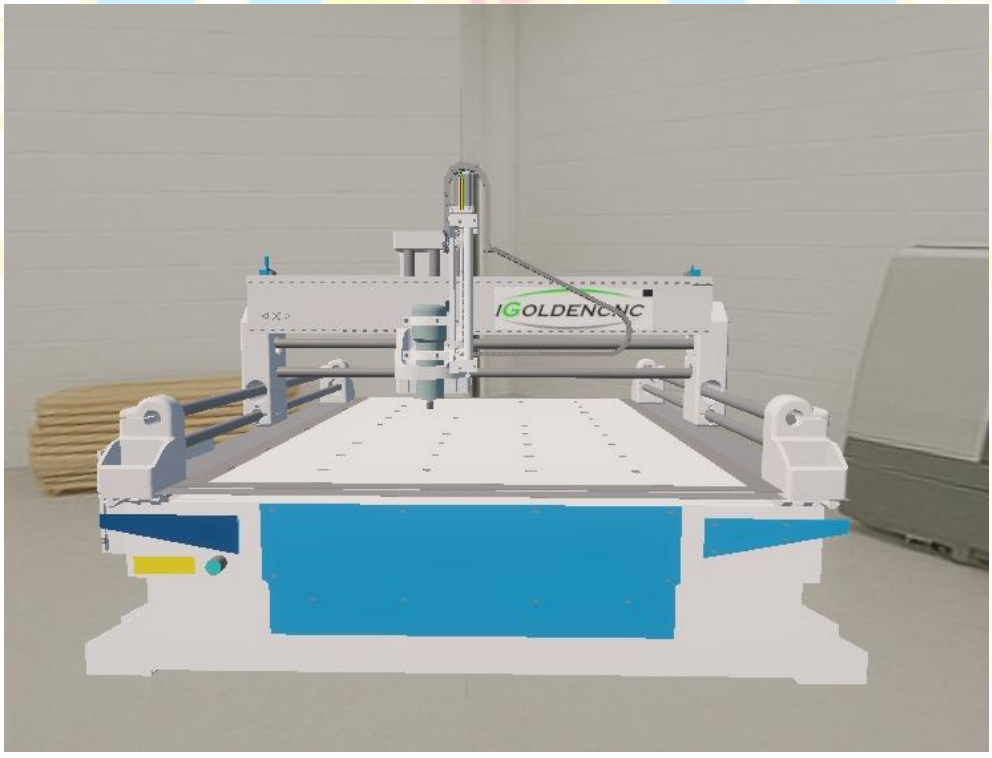
5. La interfaz gráfica es amigable y flexible, permitiendo que tanto estudiantes sin experiencia como docentes avanzados puedan utilizar, modificar o ampliar el simulador según sus necesidades pedagógicas.
6. La adaptación a los límites físicos del área de corte real (X: -175 a 175, Z: -200 a 200) aporta una dimensión práctica importante, ya que obliga a considerar restricciones reales en la planificación de trayectorias.
7. La simulación se comporta de manera determinista, replicando fielmente la secuencia de instrucciones, tiempos de espera y lógica de ascenso y descenso del eje Y, facilitando el entrenamiento paso a paso del usuario.
8. Este tipo de herramienta fomenta el autoaprendizaje, el ensayo-error y la experimentación responsable con conceptos de control numérico computarizado, sin el costo o riesgo de operar máquinas reales.

RECOMENDACIONES

1. Continuar expandiendo el repertorio de figuras disponibles en G-code, incluyendo letras, símbolos, patrones artísticos y trayectorias no lineales para enriquecer la práctica.
2. Incluir un sistema de validación o "depuración" de G-code antes de iniciar el corte, que identifique errores comunes (como trayectorias fuera del área de corte o ausencia de G0 inicial).
3. Añadir un sistema de retroalimentación visual para los errores del usuario (colores distintos, mensajes emergentes) cuando las coordenadas superen los límites o falten valores.
4. Incorporar la posibilidad de simular diferentes materiales, ajustando parámetros como la velocidad, color de partículas o intensidad del trazo.
5. Implementar un historial de corte o bitácora educativa que registre qué figuras se han practicado, con tiempo de ejecución y nivel de complejidad.
6. Expandir la lógica de generación de G-code desde imágenes para permitir múltiples capas de corte, rellenos interiores o detección de agujeros (por ejemplo, letras con "o" o "a").

7. Ofrecer la opción de exportar el G-code generado desde imágenes o desde ingreso manual, en formato estándar .gcode, para permitir pruebas en software de control real como GRBL o Marlin.
8. Documentar detalladamente todos los scripts y funciones disponibles en Unity para facilitar el trabajo de otros desarrolladores o instituciones que deseen extender el proyecto.
9. Explorar el uso de machine learning para corregir o predecir trayectorias óptimas a partir de entradas incompletas, mejorando la eficiencia y usabilidad del sistema.
10. Finalmente, se recomienda aplicar este simulador como parte de programas de formación técnica, módulos prácticos en carreras afines o como herramienta previa a prácticas con máquinas reales.

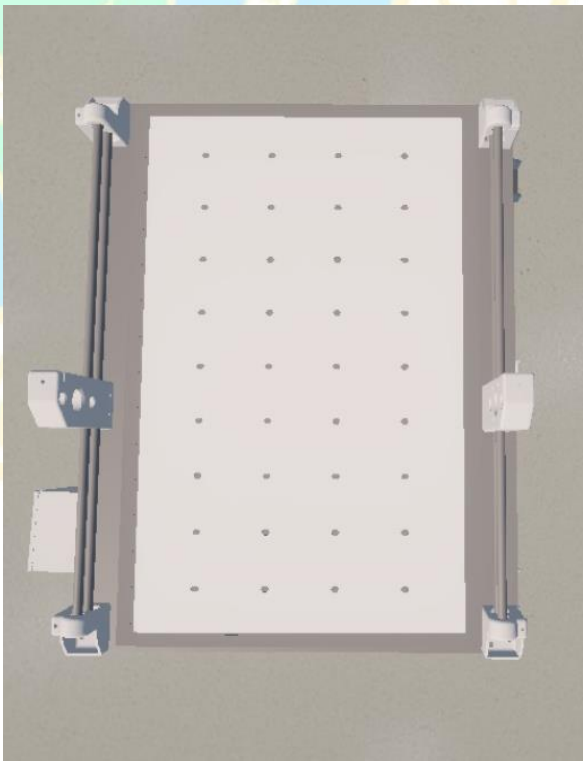
15. Anexos.



CncPreview.cs | CncSimulator (1).cs | GcodeParser.cs | CoordinateInputManager.cs | import cv2 Untitled-1

C: > Users > jorda > Downloads > CoordinateInputManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using CNC.Simulator;

public class CoordinateInputManager : MonoBehaviour
{
    public InputField inputX;
    public InputField inputY;
    public InputField inputZ;
    public CncPreview cncPreview;
    public GameObject CanvasUI;

    public CncSimulator cncSimulator;

    private List<(Vector3 punto, bool esCorte)> puntosParaCorte = new List<(Vector3, bool)>();

    public void AgregarPunto()
    {
        if (float.TryParse(inputX.text, out float x) &&
            float.TryParse(inputY.text, out float y) &&
            float.TryParse(inputZ.text, out float z))
        {
            puntosParaCorte.Add((new Vector3(x, y, z), true)); // Asume corte por defecto
            Debug.Log($"Punto agregado manualmente: {x}, {y}, {z}");
        }
        else
        {
            Debug.LogWarning("Entrada inválida");
        }
    }

    public void CargarYCortarDesdeGcode()
    {
        List<(Vector3, bool)> puntos = GcodeParser.ParsearGcode();

        if (puntos.Count == 0)
        {
            Debug.LogWarning("No se cargaron puntos desde el archivo G-code.");
            return;
        }

        puntosParaCorte.Clear();
        puntosParaCorte.AddRange(puntos);
```



CncPreview.cs | CncSimulator (1).cs | GcodeParser.cs | CoordinateInputManager.cs | import cv2 Untitled-1

C: > Users > jorda > Downloads > GcodeParser.cs

```
using System.Collections.Generic;
using UnityEngine;

namespace CNC.Simulator
{
    public static class GcodeParser
    {
        public static List<(Vector3, bool)> ParsearGcode()
        {
            TextAsset archivo = Resources.Load<TextAsset>("gcodes/ejemplo");

            if (archivo == null)
            {
                Debug.LogError("No se pudo cargar el archivo G-code desde Resources. Asegúrate de que esté en Assets/Resources/gcodes/ejemplo.gcode");
                return new List<(Vector3, bool)>();
            }

            string[] lineas = archivo.text.Split('\n');
            List<(Vector3, bool)> puntos = new List<(Vector3, bool)>();

            float x = 0, y = 0, z = 0;

            foreach (string linea in lineas)
            {
                string l = linea.Trim();

                bool esCorte = false;

                if (l.StartsWith("G1")) esCorte = true;
                else if (!l.StartsWith("G0")) continue;

                string[] partes = l.Split(' ');

                foreach (string parte in partes)
                {
                    if (parte.StartsWith("X"))
                        float.TryParse(parte.Substring(1), out x);
                    else if (parte.StartsWith("Y"))
                        float.TryParse(parte.Substring(1), out y);
                    else if (parte.StartsWith("Z"))
                        float.TryParse(parte.Substring(1), out z);
                }

                puntos.Add((new Vector3(x, y, z), esCorte));
            }
```

CncPreview.cs | CncSimulator (1).cs | GcodeParser.cs | CoordinateInputManager.cs | import cv2 Untitled-1

C: > Users > jorda > Downloads > CncPreview.cs

```
{
    private LineRenderer lineRenderer;

    private void Awake()
    {
        lineRenderer = GetComponent<LineRenderer>();
        lineRenderer.positionCount = 0;
        lineRenderer.widthMultiplier = 0.5f;
        lineRenderer.material = new Material(Shader.Find("Sprites/Default"));
        lineRenderer.startColor = Color.green;
        lineRenderer.endColor = Color.green;
    }

    // Llamar este método para mostrar el recorrido sin ejecutar el movimiento real
    public void MostrarPreview(List<Vector3> puntos)
{
    if (puntos == null || puntos.Count == 0)
    {
        lineRenderer.positionCount = 0;
        return;
    }

    Vector3[] posicionesConvertidas = new Vector3[puntos.Count];

    for (int i = 0; i < puntos.Count; i++)
    {
        // Solo trazamos sobre la pieza, es decir, en la parte baja
        posicionesConvertidas[i] = new Vector3(
            puntos[i].x,
            -220f, // Y más bajo (zona de corte)
            puntos[i].z
        );
    }

    lineRenderer.enabled = true;
    lineRenderer.positionCount = posicionesConvertidas.Length;
    lineRenderer.SetPositions(posicionesConvertidas);
}
public void LimpiarPreview()
{
    lineRenderer.positionCount = 0;
}

```

CncPreview.cs | CncSimulator (1).cs ● | GcodeParser.cs | CoordinateInputManager.cs | import cv2 Untitled-1 ●

C: > Users > jorda > Downloads > CncSimulator (1).cs

```
namespace CNC.Simulator
{
    public class CncSimulator : MonoBehaviour
    {
        public Transform mov_izq_der;
        public Transform mov_arrib_abajo;
        public Transform movimiento_adel_atr;

        public float velX = 1f;
        public float velY = 1f;
        public float velZ = 1f;

        public Vector2 limiteX = new Vector2(-100, 100);
        public Vector2 limiteY = new Vector2(-220, -180);
        public Vector2 limiteZ = new Vector2(-100, 100);

        public TrailRenderer trail;

        private Coroutine movimientoCoroutine;

        public IEnumerator MoverAHacia(Vector3 destino, bool esCorte)
        {
            Vector3 puntoClampeado = new Vector3(
                Mathf.Clamp(destino.x, limiteX.x, limiteX.y),
                Mathf.Clamp(destino.y, limiteY.x, limiteY.y),
                Mathf.Clamp(destino.z, limiteZ.x, limiteZ.y)
            );

            if (movimientoCoroutine != null)
                StopCoroutine(movimientoCoroutine);

            movimientoCoroutine = StartCoroutine(MoverSecuencia(puntoClampeado, esCorte));
            yield return movimientoCoroutine;
        }

        private IEnumerator MoverSecuencia(Vector3 destino, bool esCorte)
        {
            // Bajar si es corte
            if (esCorte)
            {
                while (mov_arrib_abajo.localPosition.y > limiteY.x + 0.1f)
                {
                    Vector3 posY = mov_arrib_abajo.localPosition;
                    posY.y = Mathf.MoveTowards(posY.y, limiteY.x, velY * Time.deltaTime);
```