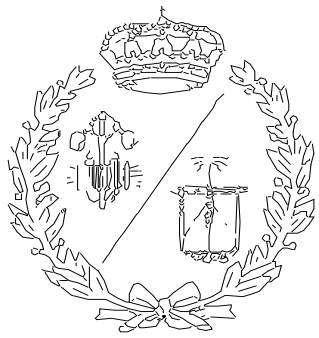
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

DESARROLLO DE UN ENTORNO INMERSIVO PARA EL MANEJO DE UN TORNO FAGOR-CNC

(Development of an immersive environment for the management of a Fagor-CNC lathe)

Para acceder al Título de

GRADUADO EN INGENIERÍA MECÁNICA

Autor: Hernán Rodrigo García Mira

Febrero - 2025

RESUMEN

Este trabajo pretende explorar una de las muchas aplicaciones que la realidad virtual puede ofrecer a la ingeniería, y concretamente a través del desarrollo de una aplicación de realidad virtual para simular un torno modelo FAGOR CNC 8025/8030 en un entorno inmersivo, tomando como referencia un equipo situado en el Área de Procesos de Fabricación de la ETS de Ingenieros Industriales de la Universidad de Cantabria.

Inicialmente, se recopilaron las medidas esenciales de la máquina para modelarla en el software CAD 3D Autodesk Inventor. Con este programa, se recrearon los componentes clave y parte del entorno del laboratorio, generando un ensamblaje para su posterior exportación. Seguidamente, el modelo se integró en Unity 3D, una plataforma de desarrollo de videojuegos multiplataforma que cuenta con la versión Unity Personal, un software de licencia gratuita destinado a estudiantes, desarrolladores independientes y pequeñas empresas.

El resultado del TFG permitió crear un entorno que da la oportunidad de formación a operarios o alumnos en el manejo del torno de forma segura y sin necesidad de poseer o estar físicamente con dicho equipo, simplemente con los periféricos de realidad virtual y el software que simula el uso de la máquina y sus operaciones básicas.

ABSTRACT

This project aims to explore one of the many applications that virtual reality can offer to engineering, and specifically through the development of a vr application to simulate a FAGOR CNC 8025/8030 lathe model in an immersive environment, taking as a reference a piece of equipment located in the Manufacturing Processes Area of the ETS of Industrial Engineers of the University of Cantabria.

Initially, essential machine measurements were collected for modeling in Autodesk Inventor 3D CAD software. With this program, the key components and part of the laboratory environment were recreated, generating an assembly for subsequent export. The model was integrated into Unity 3D, a cross-platform video game development software that features the Unity Personal version, a free-license software for students, independent developers, and small businesses.

The result of the TFG makes it possible to create an environment that gives the opportunity to train operators or students in the handling of the lathe safely and without the need to own or be physically with the equipment, simply with the virtual reality peripherals and the software that will simulate the use of the machine and its basic operations.

Índice de contenidos

Índice de figuras

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

La evolución de la tecnología en la industria abre multitud de oportunidades para mejorar los procesos de formación y capacitación. Uno de los aspectos más críticos es la manipulación de todo tipo de maquinaria CNC (Control Numérico por Computadora), que requiere precisión y destreza, además de que constituye un riesgo considerable. Tradicionalmente, la formación para el uso de estas máquinas ha requerido mucho tiempo y recursos, ya que depende del equipo físico para poder practicar con ellas. Con la implementación de un entorno virtual se consigue disminuir este coste al mínimo. Además, la creación de un entorno inmersivo virtual también responde a la necesidad de preparar a los futuros profesionales para un mercado laboral cada vez más digitalizado y automatizado. Proveer a los alumnos y operarios de herramientas avanzadas de aprendizaje que simulen situaciones reales contribuirá a su desarrollo técnico y a una mejor comprensión de la industria 4.0.

1.2 OBJETIVO

El objetivo principal de este proyecto es la creación de un entorno virtual inmersivo que permita a los estudiantes y operarios aprender a manipular un torno CNC de manera segura y eficiente. A través de la realidad virtual, este entorno proporcionará una experiencia realista evitando riesgos asociados al uso de maquinaria real. Esto también supone reducir los costos y aumentar la flexibilidad del proceso educativo. A través de este proyecto, se espera demostrar que un entorno virtual inmersivo puede ser una herramienta educativa eficaz y rentable para la formación en el uso de tornos CNC, contribuyendo a la modernización y mejora de la enseñanza de los procesos industriales.

Asimismo, cabe mencionar que este trabajo constituye la primera fase de un proyecto más ambicioso del director de este TFG, que se espera sea completada por otro alumno en el futuro, ampliando y mejorando las funcionalidades, implementando todas las características relacionadas con la programación de rutinas en código G y otras funcionalidades propias del torno Fagor CNC 8025, asegurando que el entorno virtual sea lo más funcional y realista posible.

2 LA REALIDAD VIRTUAL

2.1 ¿QUÉ ES LA REALIDAD VIRTUAL?

Según el diccionario de la lengua española, la realidad virtual es una ‘’representación de escenas o imágenes de objetos producida por un sistema informático, que da la sensación de su existencia real’’.[1]

Para dar una explicación más aclaratoria podemos decir que la realidad virtual (RV) es una tecnología que utiliza dispositivos informáticos para simular un entorno tridimensional en el que el usuario puede interactuar de manera inmersiva. En esta realidad virtual, el usuario es transportado a un entorno digital generado por computadora, donde puede moverse, explorar e interactuar con objetos virtuales como si estuviera físicamente presente. La inmersión en esta tecnología se logra mediante el uso de diferentes tipos de accesorios como son los visores, los controladores de movimiento, o los guantes hápticos.

2.1.1 Diferencias entre realidad virtual y aumentada.

La realidad aumentada puede verse como el siguiente paso evolutivo de la realidad virtual. Es por ello que pueden llegar a confundirse debido a sus múltiples similitudes y al hecho de que ambas son aplicaciones de una misma base tecnológica. Aunque en principio pueden parecer conceptos intercambiables, cada una ofrece experiencias y posibilidades únicas, que pueden ser aplicadas para la transformación de diversos sectores como la educación, el entretenimiento y la industria.

Realidad Virtual (RV): La Realidad Virtual recrea un entorno real o imaginario de manera completamente digital en el que el usuario se sumerge, interactuando con objetos y escenarios digitales. En la RV el usuario puede conseguir una inmersión completa en el mundo virtual, perdiendo la percepción del entorno físico real con la ayuda de los visores que cubren el campo de visión del usuario y con auriculares que proporcionan sonido espacial.

Sus usos más comunes son videojuegos, simuladores de entrenamiento o aprendizaje, o el diseño de prototipos en ingeniería y arquitectura.

Figura 1. Videojuego en un entorno de realidad virtual [2]

Realidad Aumentada (RA): La Realidad Aumentada superpone elementos digitales sobre el entorno físico real, permitiendo que el usuario vea e interactúe con ambos de manera simultánea. Con la realidad aumentada el usuario mantiene la percepción de la realidad mientras interactúa con elementos digitales añadidos. Los dispositivos de RA, como gafas transparentes o smartphones y tabletas con cámara permiten esta integración.

Algunos ejemplos de usos comunes pueden ser aplicaciones móviles como Pokémon GO, herramientas de asistencia de mantenimiento y reparación en tiempo real, o el turismo interactivo.

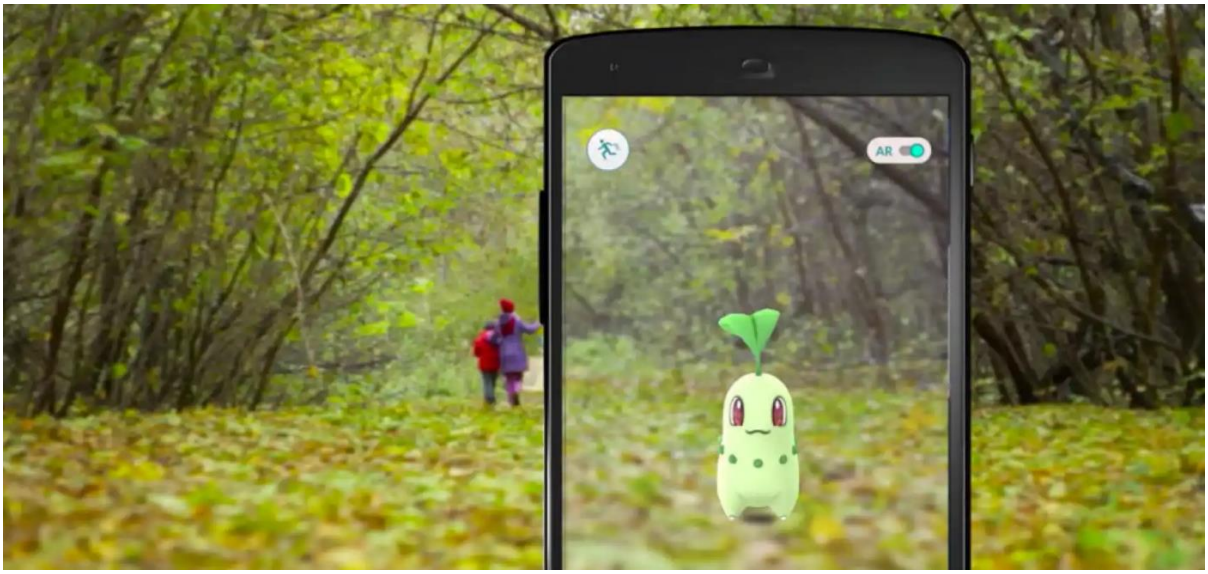


Figura 2. Realidad aumentada en Pokémon GO [3]

2.2 HISTORIA Y EVOLUCIÓN

2.2.1 Orígenes

La realidad virtual tiene su origen en la Segunda Guerra Mundial, cuando la Marina de Guerra de Estados Unidos en colaboración con el MIT crearon un simulador de vuelo para entrenar a los pilotos de bombarderos. Este proyecto, conocido como Whirlwind, se completó en 1951. Ocho años después, la Fuerza Aérea de los Estados Unidos retomó el proyecto bajo el nombre de "Claude Project", dando comienzo al uso civil de este tipo de tecnología 3D. [4]

Figura 3. Componentes del simulador Whirlwind [5]

2.2.2 Evolución tecnológica y proyectos emblemáticos

A lo largo del siglo XX esta tecnología se fue mejorando y se desarrollaron varios sistemas de realidad virtual, destacando algunos hitos importantes:

- **1962:** Morton Heilig construyó el 'Sensorama', una máquina que mostraba imágenes estereoscópicas tridimensionales con sonido estéreo, efectos de viento y aromas, y asiento móvil. [4]

Figura 4. Sensorama [6]

- **1968:** Ivan Sutherland diseñó The Sword of Damocles, un casco de realidad virtual que mostraba imágenes estereoscópicas con modelos wireframe. [4]

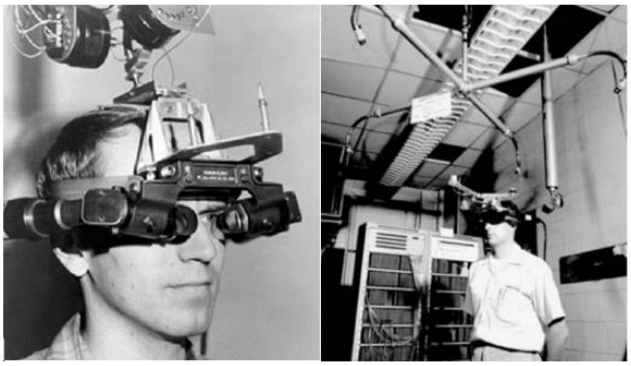


Figura 5. The Sword of Damocles [7]

- **1978:** Andrew Lippman y su equipo del MIT crearon el Aspen Movie Map, un programa que permitía recorrer las calles de Aspen mediante filmaciones reales e interactuar con algunos edificios. [4]

Figura 6. Aspen Movie Map [8]

- **1984:** La cadena de parques temáticos Six Flags, en Baltimore estrenó The Sensorium, una sala de 'cine 4D' con proyecciones estereoscópicas, asientos vibratorios y efectos aromáticos. [4]
- **1987:** Nintendo lanzó el Famicom 3D System y Sega el Master System 3D Glasses, ambos dispositivos de realidad virtual para sus respectivas consolas. [4]

Figura 7. Izquierda: Nintendo Famicom 3D System. Derecha: Sega Master System 3D Glasses [9],[10]

A lo largo de los años noventa, se produjeron algunos avances significativos en la realidad virtual:

- **1991:** Sega anunció el Sega VR, un casco de realidad virtual con pantalla LCD y auriculares estéreo para máquinas arcade y consolas de videojuegos. Aunque fue presentado en 1993, este nunca llegó a comercializarse a pesar de haberse anunciado a un precio relativamente asequible de 200 dólares. [4] Recientemente se ha descubierto un juego perdido de esta máquina (el Nuclear Rush) y la Fundación para la Historia de los Videojuegos lo ha emulado para que pueda ser jugado con gafas de realidad virtual modernas. [11]

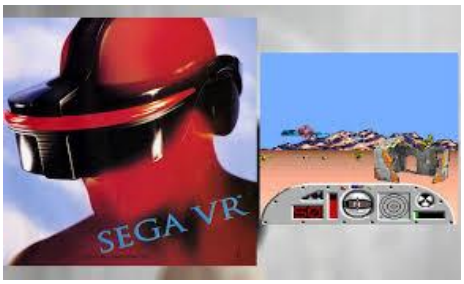


Figura 8. Prototipo del Sega VR y emulación de Nuclear Rush [11]

- **1994:** Sega en colaboración con Virtuality, lanzó en el parque temático Joypolis en Yokohama, Japón el Sega VR-1, un simulador de movimiento que incorporaba un casco con gráficos tridimensionales poligonales y seguimiento de movimientos de la cabeza. [4]

Figura 9. Atracción Sega VR-1 [12]

- **1995:** Nintendo presentó la Virtual Boy, un casco de realidad virtual con pantalla monocromática de paralaje. Ese mismo año, Forte lanzó el VFX1, un casco de realidad virtual con imagen estereoscópica, seguimiento de movimientos de cabeza y auriculares estéreo. [4]

Figura 10. Izquierda: Nintendo Virtual Boy. Derecha: Forte VFX1 [13] [14]

2.3 ESTADO ACTUAL DE LA REALIDAD VIRTUAL

2.3.1 Revolución en la industria

En 2012 se produjo un 'boom' en el mundo de la realidad virtual, cuando Palmer Luckey presentó el primer prototipo del casco Oculus Rift. Este dispositivo revolucionario inició una nueva era en la realidad virtual. En 2015 salió al mercado la versión comercial, marcando un hito significativo en la accesibilidad a la realidad virtual para el consumidor medio. [4]

Figura 11. Versión de desarrollo del Oculus Rift [15]

Este auge continuó en 2016 con el lanzamiento de las PlayStation VR por parte de Sony, ampliando el alcance de la realidad virtual al mundo de las consolas de videojuegos. Ese mismo año, HTC y Valve colaboraron para lanzar el HTC Vive, un dispositivo que ofrecía una experiencia inmersiva con seguimiento de movimientos a escala de habitación, elevando aún más el estándar de la realidad virtual. [4]

Figura 12. Izquierda: PlayStation VR. Derecha: HTC Vive[16][17]

Este periodo marcó un punto de inflexión en la adopción masiva de la realidad virtual, estableciendo las bases para innovaciones y aplicaciones futuras en multitud de ámbitos e industrias. [4]

2.3.2 Hardware, Software y Principios de funcionamiento

La realidad virtual funciona mediante la integración de diversos dispositivos de hardware junto al software necesario para crear experiencias inmersivas. Juntos, estos elementos permiten simular la presencia física en entornos virtuales y ofrecer experiencias envolventes.

El hardware de realidad virtual incluye componentes esenciales como son los visores, controladores y sensores. Los visores del Oculus Rift, PlayStation VR, HTC Vive y demás marcas del mercado actual, permiten a los usuarios situarse en entornos virtuales inmersivos que responden a los movimientos de la cabeza del usuario. Los controladores permiten interactuar con los objetos presentes en el entorno virtual mediante los botones y sensores de movimiento. Las cámaras externas y las estaciones base, capturan los movimientos y la ubicación del usuario para integrarlo en el entorno virtual y asegurar una interacción lo más realista posible.

Figura 13. Controladores del HTC Vive [18]

El software de realidad virtual es el encargado de generar estos los entornos virtuales. Los motores gráficos, como Unity y Unreal Engine, permiten desarrollar gráficos detallados y realistas. Las plataformas como SteamVR y Oculus Home, facilitan la distribución y el acceso a aplicaciones y contenidos de realidad virtual y permiten a los desarrolladores crear experiencias interactivas.

Figura 14. Logotipos de Unity y Unreal Engine, principales desarrolladores gráficos [19][20]

2.4 APLICACIONES DE LA REALIDAD VIRTUAL

2.4.1 Entretenimiento y videojuegos

La realidad virtual ha transformado la industria de los videojuegos, ofreciendo experiencias inmersivas y realistas a los jugadores que pueden sumergirse en mundos virtuales e interactuar con personajes y escenarios de manera más natural y envolvente. Además, la realidad virtual también se utiliza en el cine y la música para crear experiencias interactivas y de 360 grados, permitiendo a los espectadores sentirse parte de la acción.

2.4.2 Educación y formación

Con la realidad virtual los estudiantes pueden visitar lugares históricos, explorar entornos científicos o realizar experimentos en un laboratorio virtual, sin la necesidad de salir de las aulas. En la formación profesional, se puede utilizar para simular situaciones de riesgo y entrenar a trabajadores en operaciones complejas, como el pilotar aeronaves o el manejar y reparar maquinaria industrial pesada, mejorando sustancialmente la seguridad en el proceso de aprendizaje.

2.4.3 Medicina y salud

En el campo de la medicina, la realidad virtual tiene aplicaciones tanto en el diagnóstico como en el tratamiento. Sus aplicaciones incluyen la formación de médicos con simulaciones que permiten adquirir habilidades sin riesgo para los pacientes, el tratamiento de fobias mediante entornos controlados o hacer las rehabilitaciones más amenas evitando ejercicios repetitivos. Incluso se puede emplear para las propias cirugías para mejorar la precisión y la eficiencia de los procedimientos, permitiendo a los cirujanos visualizar en 3D la anatomía del paciente y planificar las intervenciones con mayor exactitud. [21]

2.4.4 Arquitectura e industria

La realidad virtual es una herramienta poderosa en la arquitectura y el diseño industrial, permitiendo a los profesionales visualizar y experimentar sus proyectos en tres dimensiones antes de proceder a la construcción de edificios, obras públicas o naves industriales. Permite recorrer virtualmente los diseños, modificar detalles al instante y presentárselos a los clientes de manera más comprensible. Esto permite la detección y prevención de errores y la toma de decisiones antes de iniciarse la construcción, optimizando todo el proceso.

2.4.5 Turismo

La realidad virtual también puede ofrecer nuevas formas de hacer turismo y descubrir nuevos lugares sin la necesidad de salir de casa. Las personas que tienen problemas para desplazarse de sus hogares pueden usar esta tecnología para visitar lugares famosos como museos, monumentos históricos y paisajes naturales de manera inmersiva. Las agencias de viaje también pueden aprovechar la realidad virtual para promocionar los destinos turísticos, permitiendo mostrar a sus clientes virtualmente esos lugares antes de hacer la reserva.

2.4.6 Impacto en la Sociedad

La realidad virtual ha transformado significativamente la manera en que interactuamos con la tecnología. Ha pasado de ser una herramienta de nicho a convertirse en una plataforma accesible y utilizada en múltiples sectores permitiendo experiencias inmersivas y cambiando la forma en que aprendemos, trabajamos, jugamos y nos comunicamos. Ofrece numerosos beneficios y oportunidades. En educación, facilita el aprendizaje. En medicina, mejora el entrenamiento, planificación y tratamiento. En el ámbito industrial y empresarial, permite simulaciones realistas y colaboraciones remotas. Además, abre nuevas posibilidades en multitud de ámbitos como el entretenimiento o el turismo y es capaz de mejorar la calidad de vida. Por todo esto, podemos afirmar que la realidad virtual mejora una gran cantidad de aspectos en nuestras vidas y, probablemente, en un futuro no muy lejano se podría integrar de manera natural en nuestro día a día.

3 DISEÑO Y MODELADO DEL TORNO EN CAD

3.1 TORNO FAGOR CNC 8025/8030

Como se ha indicado previamente, el presente trabajo se centrará en el diseño y modelado de un torno de control numérico para posteriormente crear un entorno de realidad virtual en el cual se podrán utilizar sus funciones básicas.

Para ello se toma como referencia un torno modelo FAGOR CNC 8025 ubicado en el Área de Procesos de Fabricación de la ETS de Ingenieros Industriales de la Universidad de Cantabria.

Figura 15. Torno real en la Universidad de Cantabria

En concreto este torno se trata de un modelo fabricado por la empresa Alecop de Mondragón (Gipuzkoa) en 1997, según la placa de identificación de la máquina.

Figura 16. Placa de identificación.

Características principales de la serie FAGOR CNC 8025/8030:

- **Modelos disponibles:** Existen dos modelos diferentes dentro de esta serie, el 8025 y el 8030, que son similares en funcionalidad, pero varían en tamaño y configuración.
- **Interfaz de usuario:** Incluye un monitor/teclado/panel de mando único para el modelo 8025 y un monitor/teclado y panel de mando separado para el 8030.
- **Modos de operación:** Ofrece 10 modos diferentes de operación, incluyendo algunos como automático, que ejecuta programas en ciclo continuo, bloque a bloque que significa que el programa ejecuta una línea de código cada vez en lugar de ejecutarse todo el programa de principio a fin (útil para fines de prueba y depuración), o modo en vacío, que sirve para probar los programas antes de ejecutar las piezas.
- **Programación:** Soporta programación paramétrica y digitalización de modelos.
- **Conexión:** Permite la comunicación vía RS232C, DNC y RS485 (Red FAGOR).
- **Funciones preparatorias:** Incluye funciones como el acoplamiento-desacoplamiento electrónico de ejes, tratamiento de bloque único, y creación automática de bloques.
- **Seguimiento de perfil:** Capacidad de seguir el perfil de la chapa en máquinas láser.
- **Visualización de herramientas:** Ayudas geométricas para la programación y visualización de la punta de la herramienta.

Estas características vienen detalladas en el manual de usuario que puede ser descargado o consultado online. [22]

A continuación, se hará un repaso de los principales elementos del torno que van a ser diseñados en CAD. El resultado del diseño y modelado de cada elemento queda reflejado en el anexo de este documento.

3.1.1 Bancada

La bancada es la base de cualquier torno, ya que proporciona el soporte y la estabilidad necesarios para que la maquinaria y las herramientas de trabajo puedan realizar operaciones de mecanizado de alta calidad. Esta debe ser capaz de soportar tanto el peso de la pieza a mecanizar como los componentes móviles del torno sin deformarse, asegurando un funcionamiento óptimo.

En el caso del FAGOR CNC 8025, el diseño de la bancada se caracteriza por ser recio a la vez que compacto. La estructura de la bancada está construida con fundición de hierro, un material robusto y con capacidad para absorber las vibraciones. Esto garantiza estabilidad y precisión durante el

mecanizado, minimizando las desviaciones y aumentando la calidad del producto final. Esta rigidez también otorga capacidad de carga al torno, que es un factor importante para el funcionamiento.

Figura 17. Bancada

3.1.2 Interruptor

En el lateral derecho de la bancada se encuentra el interruptor del torno. Este es el primer elemento con el que se interactúa al operar con la máquina. Este interruptor, que es fundamental para la seguridad y el funcionamiento del equipo, permite conectar y desconectar la máquina de la red de corriente eléctrica de manera controlada y segura.

Al activar el interruptor, se suministra la energía a todos los componentes eléctricos del torno, asegurando que todos los sistemas estén operativos y listos para realizar las funciones programadas. Igualmente, al apagar el interruptor, se corta el suministro de energía deteniendo la máquina completamente y de forma segura.

El diseño del interruptor es ergonómico y está estratégicamente ubicado en el lateral hacia la parte trasera de la bancada para evitar su activación o desactivación accidentalmente, manteniendo un entorno de trabajo seguro y evitando paradas o arranques no deseados durante el uso.

Figura 18. Interruptor y ubicación en el torno

3.1.3 Botonera

En la parte frontal de la bancada nos encontramos con la botonera. La botonera contiene los controles principales del torno, permitiendo al operador interactuar con la máquina. La botonera incluye una serie de botones que permiten realizar funciones básicas del torno. Además, está equipada con indicadores luminosos que proporcionan información visual sobre el estado de este.

Esta botonera está compuesta principalmente por tres botones. Estos al tener la misma forma no es necesario crear los tres, sino que creamos uno solo y después al realizar el ensamblaje se incluye múltiples veces en la escena:

- **Botón verde (Botón 'ON'):** Es el encargado de encender el torno. Si el torno está conectado a la corriente eléctrica, al presionar este botón el torno se enciende. Además, en su interior se ilumina una luz para indicar al operario que el torno está encendido. Esta proporciona una indicación visual clara del estado del torno, ayudando a evitar accidentes y garantizando un entorno de trabajo seguro.
- **Botón rojo (Botón 'OFF'):** Al presionar este botón se apaga el torno (si se encontraba encendido previamente), apagando también la luz del botón verde.
- **Botón amarillo (Botón 'Puerta'):** Este botón controla la apertura y cierre de la tapa de seguridad. Es un componente que garantiza que el operario tenga acceso seguro al área de trabajo del torno. Contiene una luz interior que se mantiene encendida mientras el torno está conectado a la corriente eléctrica.

Además, también cuenta con otros elementos adicionales:

- **Seta de seguridad:** La seta de seguridad es un elemento crucial en cualquier máquina industrial. Este dispositivo, también conocido como botón de parada de emergencia, se utiliza para detener el torno de inmediato en caso de emergencia. Su diseño con forma de 'seta' y de color rojo intenso lo hace fácil de identificar y accionar rápidamente, asegurando que el operador pueda reaccionar de la manera más inmediata posible ante cualquier situación peligrosa. Al presionarle, se interrumpe el suministro de energía al torno, deteniendo todas las operaciones y garantizando la seguridad del entorno.
- Espacios para la instalación de botones de ajuste automático de las pinzas triples del husillo: Estos botones no están presentes en nuestro modelo, por lo que este ajuste se realiza de forma manual. No obstante, la bancada dispone de los espacios necesarios para la instalación de dichos botones, permitiendo una posible actualización y automatización del sistema de ajuste de las piezas a mecanizar en las pinzas triples del husillo.

Figura 19. Botonera de la bancada

3.1.4 Tapa de seguridad

La tapa de seguridad de un torno es un componente esencial diseñado para garantizar la protección de los operarios y mantener el entorno de trabajo tanto seguro como limpio. Esta tapa está compuesta por dos partes: una parte fija y una parte móvil.

- **Parte Fija:** La parte fija de la tapa de seguridad está anclada a la bancada del torno proporcionando una barrera constante de seguridad que protege de los componentes móviles

y las virutas que se generan durante el mecanizado. Esta sección fija también asegura que ciertas áreas permanezcan inaccesibles, reduciendo el riesgo de accidentes.

- **Parte Móvil:** La parte móvil de la tapa de seguridad se puede abrir y cerrar mediante un mecanismo controlado por el botón amarillo. Esto permite al operario acceder al área de trabajo de manera segura cuando es necesario realizar ajustes en la pieza, cambios de herramienta o mantenimientos en el propio torno.

El diseño de la tapa de seguridad, ofrece una protección efectiva sin sacrificar la accesibilidad. Gracias al sistema de bloqueo, se evita la apertura accidental durante el funcionamiento del torno, garantizando así la seguridad durante las operaciones de mecanizado.

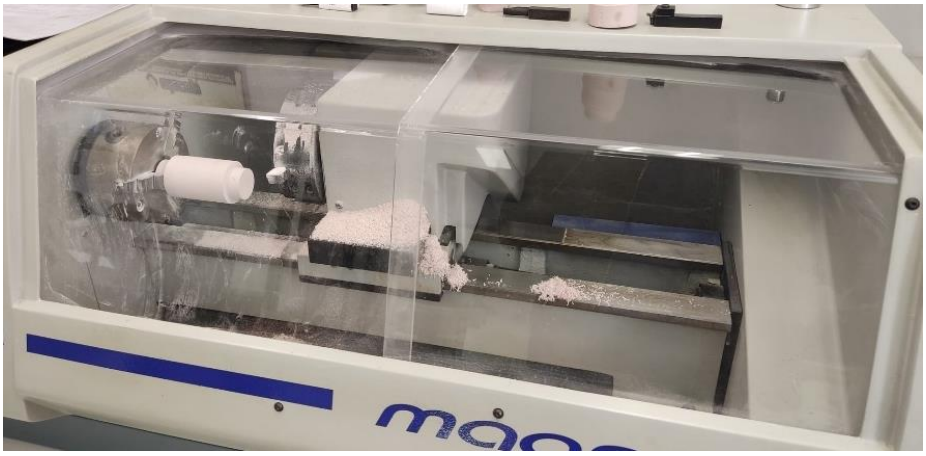


Figura 20. Tapa de seguridad cerrada

Figura 21. Solape de la parte móvil bajo la parte fija de la tapa de seguridad

3.1.5 Guías de deslizamiento

El torno está equipado con dos guías sobre las que se asienta el carro portaherramientas. También conocidas como guías lineales, son esenciales para el movimiento del carro y el cabezal móvil, asegurando un movimiento suave y de alta precisión.

Estas guías consisten en dos vías paralelas de sección rectangular, atornilladas directamente a la bancada para garantizar estabilidad. Están fabricadas con materiales de alta resistencia, como acero templado, y recubiertas con lubricante para reducir la fricción y proteger del desgaste, minimizando las deformaciones y prolongando la vida útil de la máquina y sus componentes.

La disposición y la calidad de las guías de deslizamiento son elementos clave para mantener la exactitud y la eficiencia en el mecanizado debiendo estas, estar alineadas con el eje de giro del husillo con el mínimo margen de error.

Figura 22. Guías de deslizamiento y anclaje a la bancada

3.1.6 Carro portaherramientas

El torno cuenta con dos elementos móviles esenciales para su funcionamiento, el husillo y el carro portaherramientas que van montados dentro de la 'caja' que conforma la bancada.

El carro portaherramientas es el componente encargado de mover y posicionar las herramientas de corte con para realizar las diversas operaciones de mecanizado. Este se compone de tres partes principales:

- **Carro longitudinal:** El carro longitudinal o deslizadera, va montado sobre las guías de deslizamiento y es el encargado de desplazarse a lo largo del eje Z del torno proporcionando el movimiento de avance y retroceso. Este movimiento del carro longitudinal es crucial para realizar cortes longitudinales precisos en la pieza de trabajo, como pueden ser el cilindrado, vaciado o roscado.

Figura 23. Carro longitudinal sobre las guías

- **Carro transversal:** El carro transversal va montado sobre el carro longitudinal y se desliza sobre este. Es el encargado de proporcionar movimientos precisos en la dirección perpendicular al eje del husillo. Se desplaza a lo largo del eje X del torno, realizando cortes transversales y ajusta la profundidad de corte. Gracias a este movimiento se consiguen realizar operaciones como el refrentado, acanalado o tronzado.

- **Portaherramientas:** Este elemento permite sujetar, posicionar y seleccionar las herramientas de corte durante el mecanizado. Su principal función es montar y cambiar estas herramientas de manera rápida y eficiente, permitiendo diferentes posiciones que se adapten a las diversas operaciones. Va montado en el carro transversal y su eje de giro es paralelo a la deslizadera. Un buen diseño del portaherramientas asegura una sujeción firme para evitar vibraciones y una alineación correcta para unos acabados de alta calidad.

Figura 24. Conjunto del carro portaherramientas

3.1.7 Husillo

El husillo es el elemento principal en el funcionamiento de cualquier torno, ya que es el encargado de hacer rotar a la pieza a mecanizar. Está compuesto por varios componentes, entre los cuales se encuentran el eje central, las pinzas triples y las piezas de ajuste de las pinzas.

- **Eje central:** Es el componente principal del husillo, encargado de la rotación. Debe ser robusto y estar alineado con precisión, tanto con la bancada como con el carro portaherramientas para asegurar un mecanizado lo más exacto posible.
- **Pinzas triples:** Las pinzas triples se utilizan para fijar la pieza de trabajo al husillo. Estas pinzas están diseñadas para adaptarse a diferentes tamaños y formas de piezas, asegurando una sujeción firme y segura durante el mecanizado. Su diseño en tres partes a 120º cada una, permite un agarre uniforme y estable, evitando deslizamientos o movimientos no deseados.
- **Ajuste de las pinzas triples:** Estas piezas se utilizan para ajustar y asegurar las pinzas triples a la pieza de trabajo. Permiten ajustar la presión y el posicionamiento de las pinzas, asegurando que la pieza de trabajo esté correctamente alineada y sujeta.

Tanto las pinzas como las piezas de ajuste solo es necesario crear en CAD una única pieza de cada y posteriormente se ensamblan tres veces.

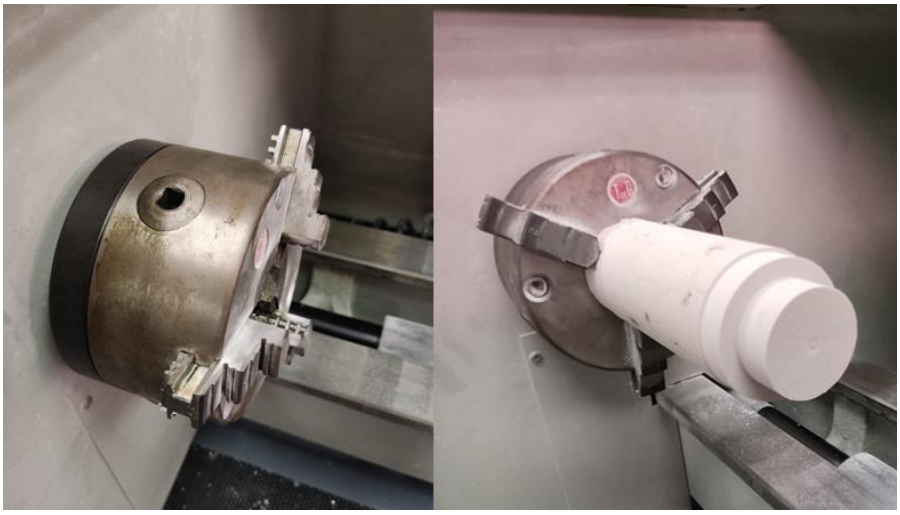


Figura 25. Husillo en vacío y husillo cargado con una pieza de trabajo

3.1.8 Mesa soporte

Para la correcta instalación del torno en el laboratorio, es fundamental situarlo sobre una mesa firme y estable, que garantice la absorción de las vibraciones generadas durante el mecanizado. Además, la altura de la mesa debe ser la adecuada, para que el operario trabaje de manera ergonómica y segura, contribuyendo tanto a la seguridad como a la comodidad del usuario.

Figura 26. Bancada sobre la mesa soporte

3.1.9 Panel de control

El panel de control de un torno CNC es el elemento que permite interactuar con la máquina y programar las diversas operaciones de pruebas y mecanizados.

El panel de control de nuestro torno incluye una pantallita donde se ve toda la información necesaria, como los parámetros del mecanizado, los programas que hay en ejecución, y los posibles mensajes de error. También cuenta con un teclado alfanumérico que permite al operario introducir los datos y navegar entre los distintos menús del sistema. También incorpora diversos controles para ajustar las velocidades de giro y avance.

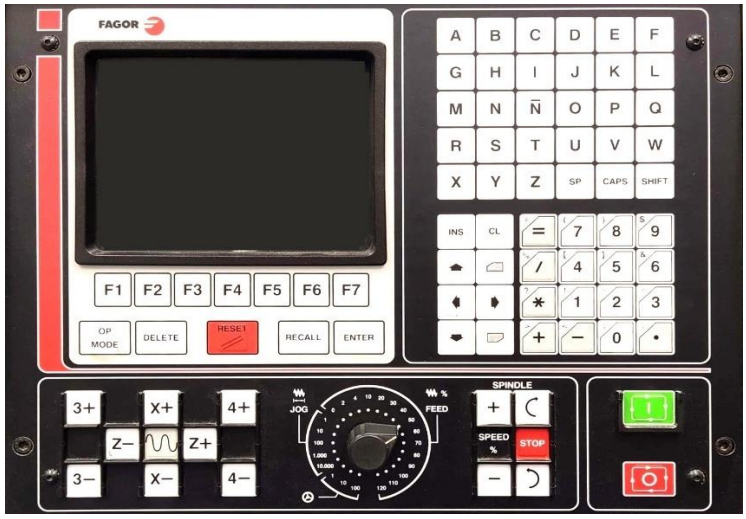


Figura 27. Teclado y pantalla del panel de control.

Este dispositivo es un elemento independiente de la bancada, lo que permite al usuario colocarlo según sus preferencias y/o necesidades para optimizar el entorno. Al igual que la bancada, el panel contiene una seta de seguridad que permite una rápida reacción ante situaciones de riesgo. Esto permite al usuario poder acceder a los controles de manera cómoda, eficiente y rápida sin estar limitado por la posición del torno.

Figura 28. Dispositivo del panel de control

3.2 DISEÑO DE LOS ELEMENTOS EN CAD Y POSTERIOR ENSAMBLAJE

Antes de poder comenzar a dibujar en CAD los elementos del torno, fue necesario desplazarse al aula donde se encuentra el equipo. Utilizando una cinta métrica, una regla y un calibre pie de rey, se tomaron las medidas de los componentes mencionados en el anterior apartado.

Figura 29. Toma de medidas in situ de los elementos del torno.

Con este proceso nos aseguramos que las dimensiones y detalles sean lo más iguales posible a la realidad, permitiendo replicar el torno de manera exacta en el diseño y modelado CAD.

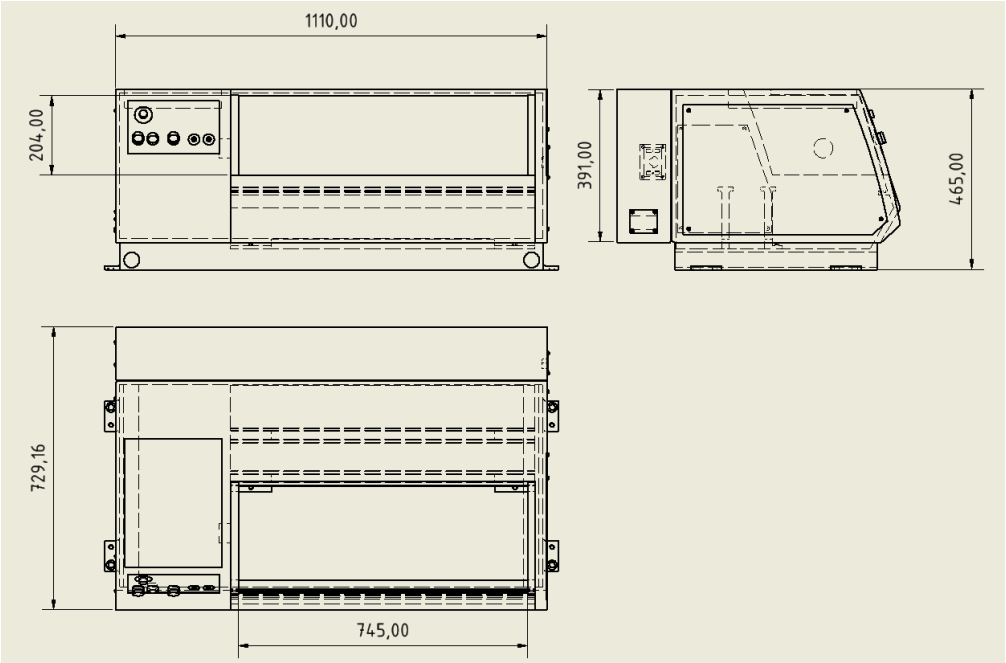


Figura 30. Principales medidas exteriores de la bancada (cotas en milímetros).

3.2.1 Introducción a Autodesk Inventor

Para el diseño y modelado de todos los elementos, se decidió utilizar el programa Autodesk Inventor, un software de diseño asistido por ordenador (CAD) desarrollado por Autodesk, que permite tanto a ingenieros y diseñadores como a estudiantes dar forma sus ideas antes de la fabricación.

Figura 31. Logotipos de Autodesk (empresa matriz) y Inventor (programa de CAD). [23]

Aunque Inventor se trata de un software de pago, dispone de una versión gratuita para estudiantes, lo que permite el acceso a sus herramientas para fines educativos sin incurrir en grandes costos. Esta versión para estudiantes de Autodesk Inventor tiene ciertas limitaciones, que no son funcionales, sino más bien administrativos, para evitar que el software sea utilizado con fines comerciales. Esto incluye restricciones en la duración de la licencia, la cantidad de dispositivos en que se puede instalar, o la inclusión de marcas de agua por lo cual es perfectamente compatible con el uso que le daremos. [23]

Autodesk Inventor ofrece un entorno de modelado paramétrico para la creación de modelos 3D a partir de bocetos. Algunas de las principales herramientas para crear piezas 3D a partir de estos bocetos son la de extrusión, que permite generar sólidos a partir de perfiles bidimensionales; la de revolución, que crea formas tridimensionales girando un perfil alrededor de un eje; o la herramienta de barrido, que permite crear geometrías complejas desplazando un perfil a lo largo de una trayectoria definida. También se pueden editar estas geometrías con las herramientas de corte, chaflán y redondeo, que permiten modificar y detallar los modelos para obtener las formas deseadas.

Posee una interfaz intuitiva que permite a los usuarios diseñar piezas individuales con facilidad, así como ensamblajes complejos que integran múltiples componentes. Además se pueden generar planos 2D detallados y acotados a partir de las piezas 3D para la fabricación de estos diseños.

Inventor también tiene la capacidad de simular el comportamiento físico de los diseños con análisis de esfuerzos y movimientos bajo diferentes condiciones de carga. Esto optimiza el diseño y también reduce el coste y tiempo de desarrollo, al disminuir la necesidad de elaborar prototipos físicos. Además, Autodesk Inventor es capaz de integrarse con otras de las múltiples herramientas con las que cuenta Autodesk, como por ejemplo AutoCAD o Revit. [24]

También es compatible con aplicaciones de terceros, lo que facilita la colaboración y el intercambio de datos entre diferentes plataformas. Esta compatibilidad nos es especialmente útil para exportar los modelos a software de animación y desarrollo de videojuegos, como Blender y Unity3D.

Al trabajar en proyectos que requieren de diferentes disciplinas de software, la capacidad de exportar modelos de Autodesk Inventor a Blender permite aprovechar al máximo las herramientas de modelado, texturizado y renderizado avanzadas que ofrece Blender. Además, la integración fluida con Unity3D facilita la creación de entornos de juego y objetos interactivos con alta precisión y detalle.

En un contexto educativo, Autodesk Inventor es una herramienta perfecta para los estudiantes de ingeniería y diseño industrial. Su uso académico no solo permite a los estudiantes desarrollar habilidades técnicas avanzadas, sino que también les prepara para los desafíos del mundo profesional, permitiendo simular el ciclo de desarrollo, desde la idea inicial hasta la validación final del producto.

En resumen, podemos decir que Autodesk Inventor es una plataforma poderosa y versátil capaz de optimizar los procesos de diseño y mejorar la colaboración entre equipos, así como de otorgar las herramientas de aprendizaje para enfrentar los retos de la industria 4.0.

3.2.2 Boceto

Una vez que tenemos todas las medidas necesarias de cada pieza del torno, seguiremos un proceso similar para la elaboración de cada una de ellas para recrearlas digitalmente.

Figura 32. Toma de la medida exterior del eje principal del husillo.

Para comenzar, empezamos creando un boceto. Primero, se selecciona un plano de trabajo en el que hacer el dibujo. Dentro del entorno de boceto, se puede dibujar figuras geométricas básicas como líneas, circunferencias, rectángulos o arcos con las herramientas disponibles en la pestaña 'Boceto'.

A estas figuras se les puede (o más bien debe) aplicar distintos tipos de cotas para definir las dimensiones del boceto, tales como cotas de longitud, radio, ángulo y diámetro. Junto con las restricciones de paralelismo, coincidencia o tangencia se tiene todo lo necesario para definir correctamente el boceto. Además, Inventor permite el uso de cotas de referencia, que no afectan al boceto, sino que le sobre restringen. Estas cotas son fáciles de identificar ya que siempre aparecen entre paréntesis. También se puede convertir una cota normal en cota de referencia y viceversa según sea necesario.

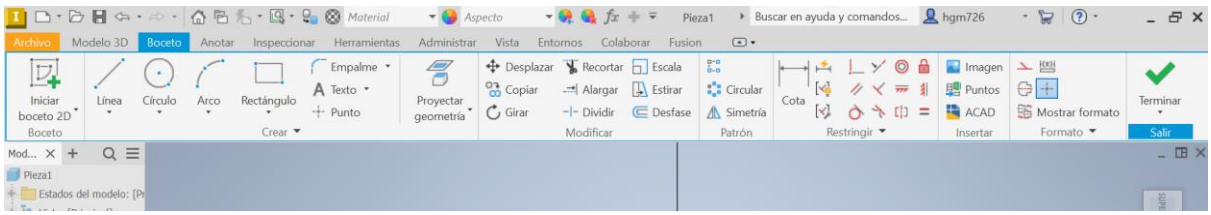


Figura 33. Pestaña 'Boceto' para el dibujo 2D.

El software también permite incorporar elementos de construcción, como líneas auxiliares y puntos de referencia, que no forman parte del boceto final, pero ayudan al diseño. Estos elementos de construcción son especialmente útiles para alinear y posicionar componentes, permitiendo un mayor control del boceto.

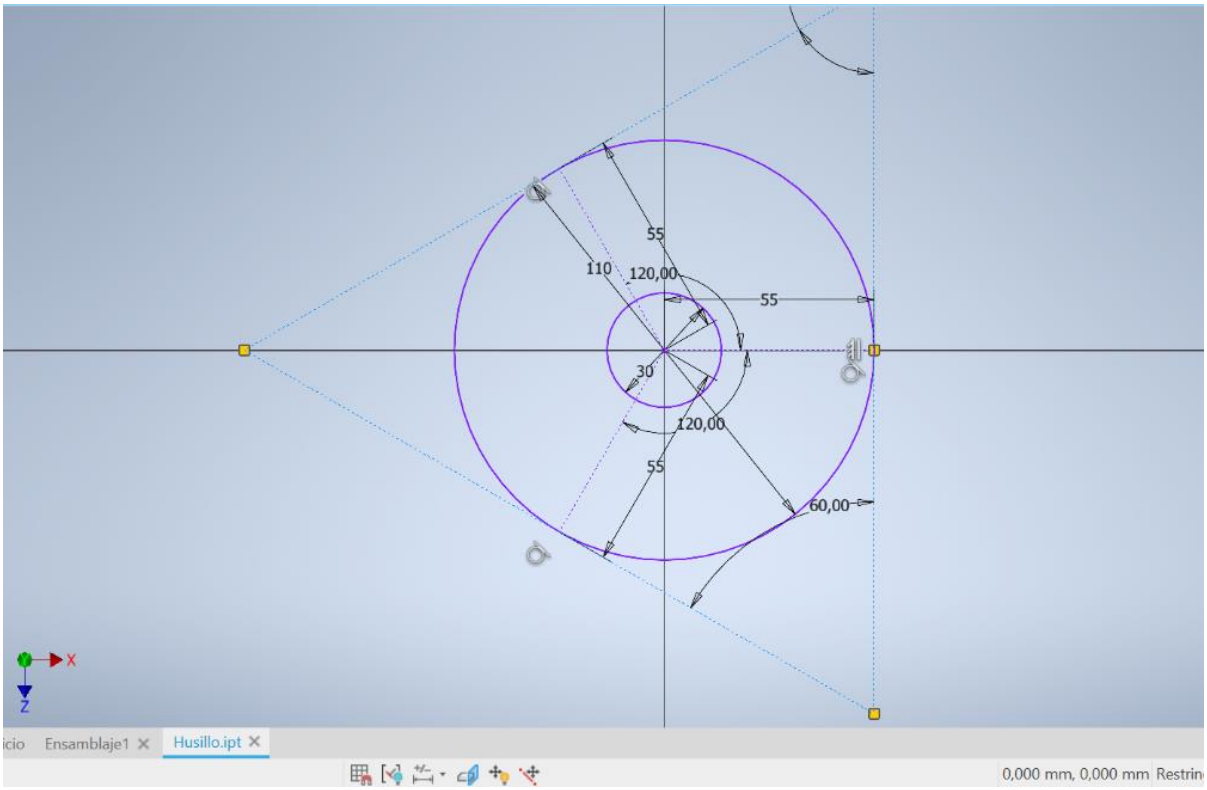


Figura 34. Primer boceto de la geometría de eje central del husillo.

3.2.3 Geometría 3D

Una vez tenemos el boceto, podemos crear la pieza en 3D utilizando la pestaña 'Modelo 3D'. El primer paso es seleccionar el boceto que hemos y elegir la herramienta de modelado más adecuada en función del tipo de pieza que vayamos a hacer.

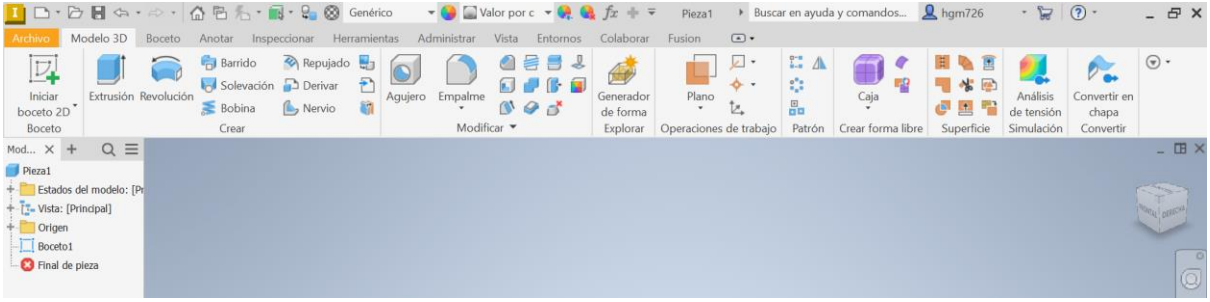


Figura 35. Pestaña 'Modelo 3D'.

Entre las herramientas más comunes se encuentran extrusión, revolución y barrido:

- **Extrusión**: Esta herramienta permite convertir el perfil 2D del boceto en una forma tridimensional al extenderlo en una dirección perpendicular al plano del boceto. Podemos especificar la distancia de extrusión, la creación de cortes, la adición de material incluso otorgar cierto ángulo a la propia extrusión.
- **Revolución**: Utilizamos esta herramienta para crear piezas de revolución rotando el boceto alrededor de un eje. Esta técnica es especialmente útil para crear piezas cilíndricas o esféricas.
- **Barrido**: Con esta herramienta, podemos generar geometrías desplazando un perfil a lo largo de una trayectoria. Es ideal para crear piezas con formas curvilíneas y más complejas.

Una vez hemos aplicado la herramienta de modelado, podemos editar la pieza mediante operaciones adicionales, como empalme, chaflán y vaciado:

- **Empalme:** Redondea las aristas del modelo creando una transición suave y curvada entre las superficies adyacentes. Mejora la estética y reduce concentraciones de esfuerzo.
- **Chaflán:** Hace un bisel en ángulo en las aristas del modelo, creando un plano inclinado. Mejora la estética eliminando bordes afilados y sirve para facilitar el ensamblaje posterior
- **Vaciado:** Vacía el interior de una pieza, manteniendo un espesor de las paredes. Es útil para reducir el peso del modelo o realizar cavidades internas.

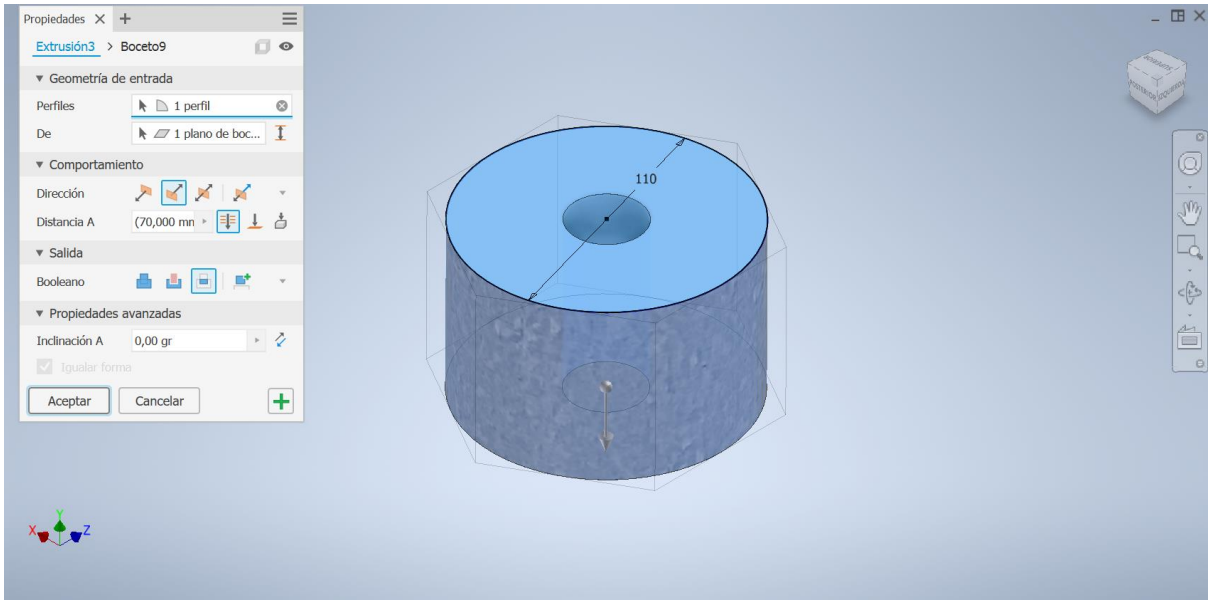


Figura 36. Extrusión de la geometría base del husillo a partir del boceto 2D.

Con todas estas herramientas podemos convertir las medidas tomadas en un dibujo a mano, en un boceto 2D y después en una pieza tridimensional, lista para su posterior ensamblaje y fabricación.

Para la creación de la geometría completa del torno se representaron las principales piezas que lo conforman, generándose 16 archivos CAD, que de forma individual o ensamblada contienen todos los elementos esenciales del torno y se muestran en el Anexo.

3.2.4 Ensamblaje

El ensamblaje en Inventor es un proceso fundamental para crear productos complejos que constan de múltiples componentes.

Inventor nos permite unir diversas piezas individuales para formar un modelo completo y funcional. Para comenzar, es necesario crear un archivo de ensamblaje nuevo (con extensión .iam) donde se irán insertando las distintas piezas (archivos .ipt) previamente creadas.

Una vez dentro del entorno de ensamblaje, se siguen los siguientes pasos:

1 - Inserción de los componentes: Se insertan las piezas individuales en la escena de ensamblaje. Esto se puede hacer seleccionando la opción 'Insertar' en la pestaña 'Ensamblaje' y navegando hasta la ubicación de los archivos de las piezas en el navegador o simplemente arrastrándolas hasta la escena.

2 - Restricciones: Para ensamblar las piezas, se utilizan diferentes restricciones que definen cómo se relacionan entre sí. Las restricciones más comunes incluyen coincidencia (alinea las superficies de las piezas para que queden en contacto directo), superposición (alinea las superficies de las piezas para que queden paralelas y en el mismo plano), eje (alinea los ejes de dos superficies cilíndricas alineándolas en el mismo eje) o ángulo (define el ángulo entre dos superficies o ejes de las piezas)

3 - Movimiento y ajuste: Una vez aplicadas las restricciones, las piezas pueden moverse dentro del conjunto de ensamblaje para asegurar el ajuste adecuado. Podemos simular el movimiento de las piezas y añadir límites para que el ensamblaje funcione correctamente.

4 - Subensamblajes: En casos de ensamblajes complejos o con una cantidad de elementos considerable como es el nuestro, es posible crear subensamblajes, que son conjuntos de piezas que se ensamblan por separado antes de integrarse en el ensamblaje principal.

5 – Análisis: Inventor ofrece herramientas para verificar el ensamblaje, como la detección de colisiones y la comprobación de interferencias para identificar posibles problemas.

6 - Documentación: si es necesario, una vez que el ensamblaje está completo, se pueden generar planos 2D del conjunto y de los subensamblajes que son de gran ayuda si se va a realizar la fabricación y el montaje de un producto.

Para simplificar el proceso de ensamblaje y facilitar posibles modificaciones de todos los componentes de este proyecto, se crearon tres subensamblajes. En primer lugar, se realizó el subensamblaje del husillo con las pinzas triples y las piezas de ajuste, y el subensamblaje del portaherramientas con el carro y la deslizadera. Posteriormente, se integraron en el subensamblaje de la bancada junto con las tapas de seguridad (fija y móvil), el interruptor y todos los botones. Finalmente, se unieron en un ensamblaje general junto con la mesa y el panel de control.

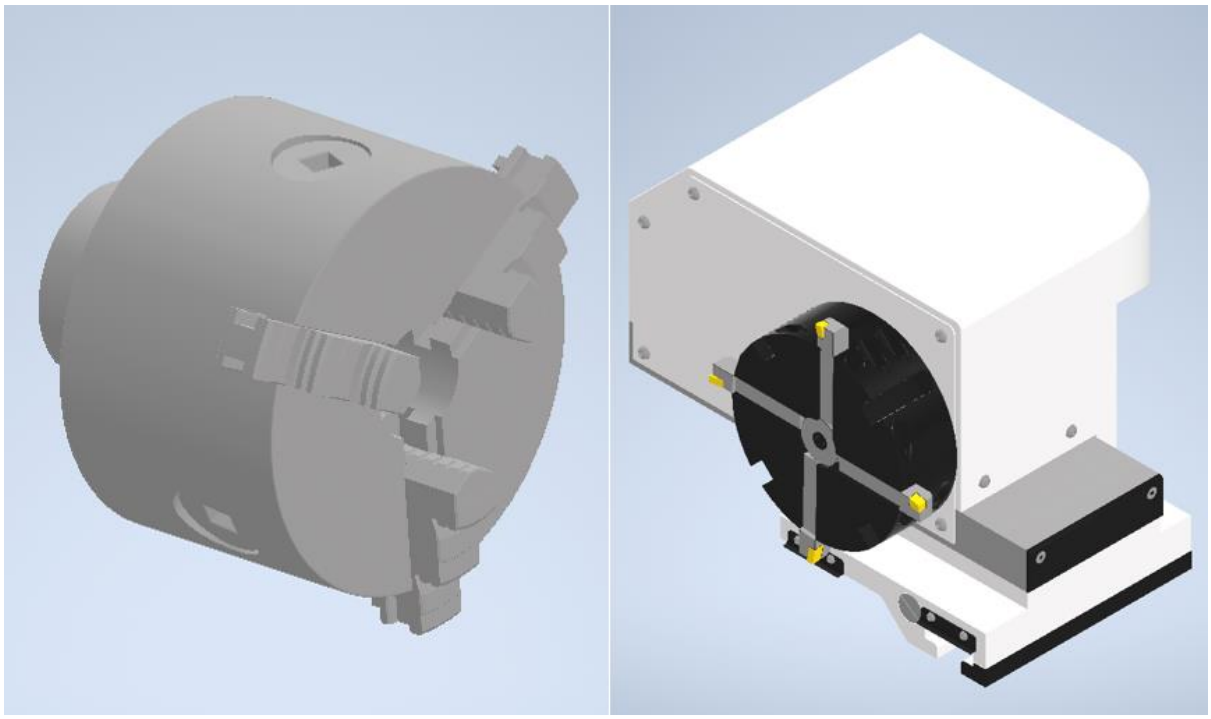


Figura 37. Subensamblajes del husillo y carro portaherramientas.

Figura 38. Ensamblaje completo del modelo.

4 EXPORTACIÓN DEL MODELO

Una vez creados todos los elementos y realizado el ensamblaje en Autodesk Inventor, es el momento de exportar el modelo. Dado que no podemos exportar directamente desde Inventor a Unity3D (ya que de hacerlo se perderían ciertas propiedades como las texturas o relaciones entre objetos), es necesario utilizar una aplicación intermedia para convertir el archivo generado por Inventor a uno compatible con Unity.

El programa que se utilizará será Blender, un software de código abierto y de uso gratuito, para la creación de contenido 3D. Desarrollado por la Fundación Blender, es una de las herramientas más poderosas y de las más usadas en la industria del modelado y animación 3D ya que ofrece una amplia gama de funcionalidades como el modelado, escultura, texturizado, animación, renderizado y simulación de físicas entre otras.

Una de las características más destacadas de Blender y que nos resulta especialmente útil para este proyecto, es su capacidad para manejar una gran variedad de formatos, lo que lo hace ideal para la conversión y exportación de nuestro modelo entre las diferentes plataformas. Además, gracias a su comunidad activa, Blender recibe actualizaciones y mejoras constantes, manteniéndose a la vanguardia de la tecnología 3D.

En este proyecto, Blender se utilizará para abrir el modelo exportado desde Autodesk Inventor y convertirlo a un formato compatible con Unity3D. Este proceso garantiza que se mantengan la configuración del ensamblaje y la jerarquía de los diferentes objetos que componen el modelo, permitiendo integrarlo perfectamente en el entorno de Unity para continuar con el desarrollo y visualización del proyecto.[25][26]

En los siguientes apartados se realizará una explicación detallada del proceso que hemos seguido para llevar nuestro modelo desde Inventor hasta Unity3D.

4.1 EXPORTACIÓN DESDE INVENTOR

El primer paso para exportar el modelo desde Inventor consiste en ir a la pestaña 'Archivo'. A continuación, debe seleccionarse la opción 'Exportar', lo que despliega un submenú con diversos formatos que seleccionaremos según el uso previsto del archivo. En este caso, se elige la opción 'Formato de CAD', para asegurarse que el modelo se guarda en un formato compatible con otros programas de diseño y modelado 3D, lo que abrirá una ventana del explorador de Windows (o el sistema operativo que se utilice), en la que elegimos la carpeta de destino para guardar el exportado.

También se deberá seleccionar el tipo de archivo correspondiente, eligiendo en este caso 'Archivos OBJ (*.obj)'. El formato OBJ es uno de los formatos más utilizados y versátiles para modelos 3D. Algunas de las principales razones por las cuales se elige este formato son su compatibilidad, ya que es soportado por una gran variedad de programas de modelado y animación 3D incluyendo Blender. A diferencia de algunos formatos que solo funcionan con softwares específicos, el formato OBJ es independiente de este, lo que significa que se puede importar y exportar entre diferentes aplicaciones sin problemas. Además, es un formato que puede almacenar información detallada sobre la geometría del modelo y las coordenadas UV necesarias para mapear texturas.

También es fácil de convertir a otros formatos 3D si es necesario. Esto es útil cuando se necesita trabajar con diferentes software o formatos para diferentes etapas del proyecto y lo que nos permitirá convertirlo más adelante para continuar con el desarrollo en Unity.

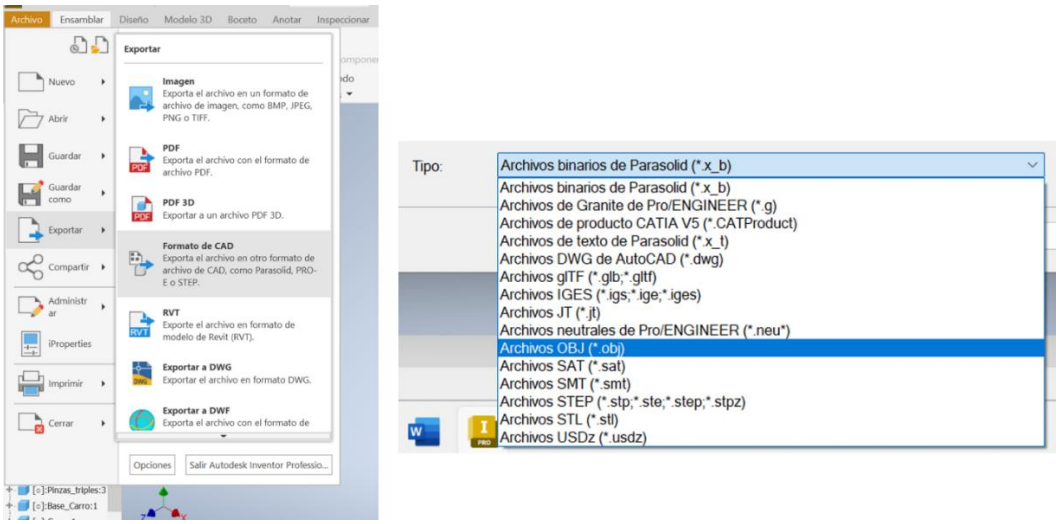


Figura 39. Selección del tipo de archivo para la exportación a Blender.

Antes de guardar el proyecto, es necesario entrar en la casilla 'opciones' para asegurarnos de que el modelo se exportará correctamente. Al pulsar sobre esta casilla se abre una ventana en donde se deberán ajustar parámetros como las unidades, las cuales seleccionaremos como 'metros' ya que son las que utiliza Blender. De no hacerlo, es posible que el tamaño del modelo no sea correcto al importarlo en Blender. También debemos ir a la sección donde se indica la estructura y en el desplegable seleccionar 'un archivo por ejemplar de pieza', para que cada pieza del ensamblaje se exporte como un archivo individual, manteniendo la jerarquía original. De no seleccionar esta opción, el ensamblaje se exportaría como un único archivo, lo que complicará la manipulación de cada componente por separado y se perdería la jerarquía del modelo. Además, trabajar con un único archivo demasiado grande podría afectar el rendimiento de Blender o de nuestro ordenador.

Por último, es recomendable ajustar la resolución a 'baja' para mejorar el rendimiento, especialmente cuando se trabaja con proyectos mecánicos muy complejos o con muchas partes. Al reducir la resolución, se disminuye el procesamiento necesario, permitiendo que el programa funcione más fluido. Esto sobre todo es útil en la fase de edición y visualización del modelo, ya que facilita las modificaciones sin comprometer el rendimiento.

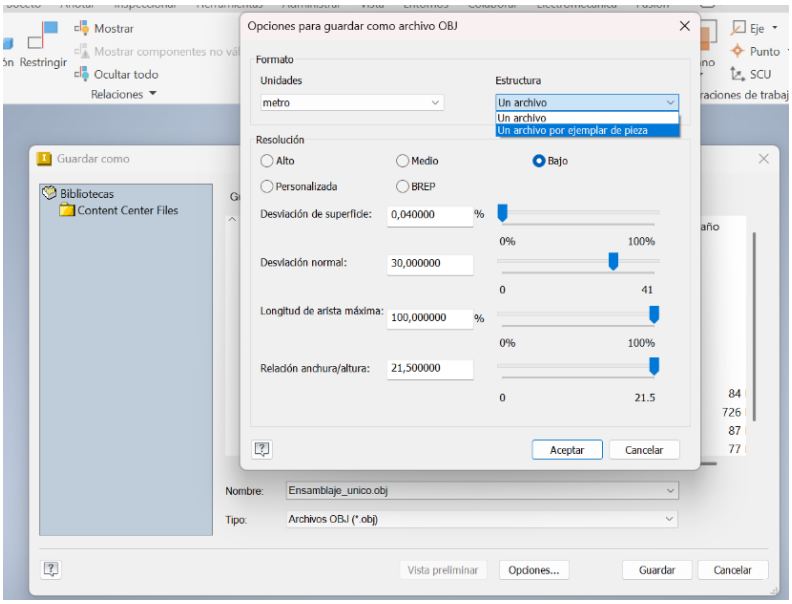


Figura 40. Ajuste de parámetros de la exportación.

Una vez que se han hecho estos ajustes, el proyecto ya está listo para ser guardado correctamente.

4.2 IMPORTACIÓN A BLENDER Y EXPORTACIÓN A UNITY3D

Una vez que se ha exportado el modelo desde Autodesk Inventor en un formato compatible, el siguiente paso es importarlo a Blender para realizar las modificaciones necesarias para exportarlo a Unity3D. Para este proyecto utilizaremos Blender exclusivamente como un puente entre Autodesk Inventor y Unity3D. El propósito es convertir el archivo OBJ de Inventor en un formato adecuado para su uso en Unity. Este proceso garantiza que se mantenga la integridad del modelo, incluyendo la jerarquía y las texturas, asegurando una correcta transición al entorno de Unity3D.

Lo primero que necesitamos es instalar en Blender la extensión 'io_batch_import_objs.py'. Esta extensión es un complemento de Python para Blender que permite la importación en lote de múltiples archivos OBJ al mismo tiempo, lo que ahorra tiempo y esfuerzo, especialmente cuando se trabaja con grandes archivos 3D. Además, gestiona los materiales asociados a los archivos OBJ, asegurando que las texturas del modelo se conserven durante la importación. Es compatible con diferentes versiones de Blender, desde la 2.65 hasta la más reciente.[27]

Por defecto, Blender soporta la importación ‘Wavefront (.obj)’, un modo de importación muy utilizado en la industria 3D. Sin embargo, esta importación por defecto solo permite importar un archivo OBJ a la vez, lo que puede ser tedioso con múltiples archivos. La extensión **io_batch_import_objs.py** nos facilita la importación en lote, mejorando el flujo de trabajo y la calidad del resultado. Es una extensión gratuita como el propio programa, que podemos descargar directamente desde el sitio web oficial de Blender o a través de la comunidad de desarrolladores de Blender y su instalación es sencilla y rápida.[28]

Con esta extensión instalada ya podemos importar el modelo a Blender. Para ello, abrimos el programa y nos vamos a la pestaña 'Archivo', seleccionamos la opción 'Importar' y elegimos 'Wavefront Batch (.obj)'. Se nos abre una ventana del visor de archivos de Blender donde buscaremos la carpeta en la que tenemos guardado nuestro modelo. Si la exportación la hicimos correctamente, dentro de esta carpeta, encontraremos dos archivos por cada objeto del modelo creado en Inventor: un archivo OBJ y un archivo MTL. El OBJ contiene la geometría del modelo 3D, es el archivo principal que define la forma del objeto. Por su parte el archivo MTL (Material Template Library, que en español se traduce como Biblioteca de Plantillas de Materiales), contiene información sobre los materiales y las texturas del modelo. El MTL se asocia con el OBJ para importar tanto la geometría como las texturas del modelo, garantizando una representación fiel. [32]

En esta ventana también podemos ajustar los ejes de coordenadas para que coincidan con los utilizados en Inventor y evitar que el modelo se importe rotado. Para ello, seleccionamos 'Eje de avance' como ‘-Z’ y 'Eje superior' como ‘Y’. Esta configuración es necesaria porque Inventor y Blender usan diferentes orientaciones por defecto, y así importamos el modelo con la orientación deseada.

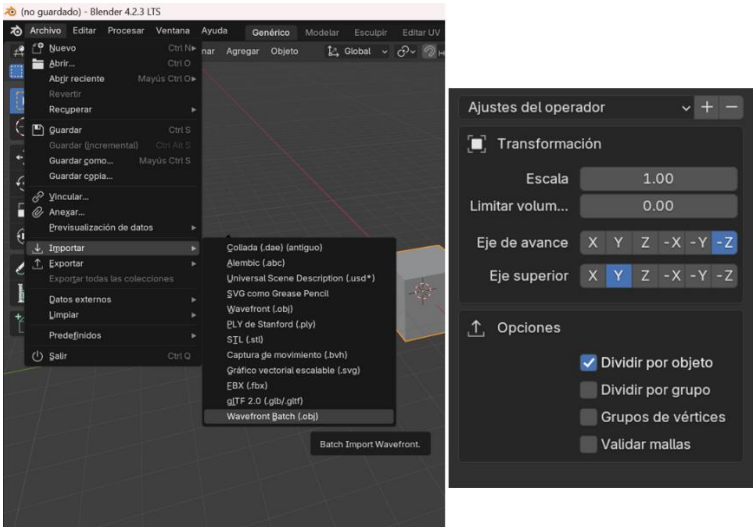


Figura 41. Ajuste de parámetros del archivo importado en Blender.

A continuación, debemos seleccionar a la vez todos estos archivos OBJ y MTL. Es importante asegurarnos de que todos los archivos estén seleccionados para que tanto la geometría como los materiales y texturas de cada elemento se importen correctamente.

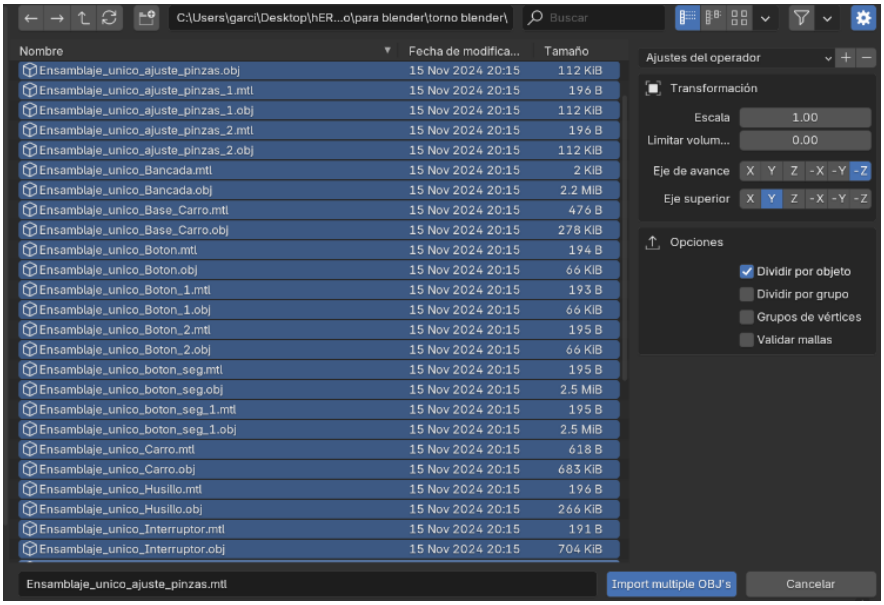


Figura 42. Selección de todos los archivos generados.

Para acabar de importar el modelo, pulsamos en 'Import Multiple OBJ's', lo que hará que aparezca en la escena de Blender. Se puede observar que todos los elementos del modelo se han importado dentro de la misma 'Collection', en la que cada elemento importado tiene asociado su archivo de texturas.

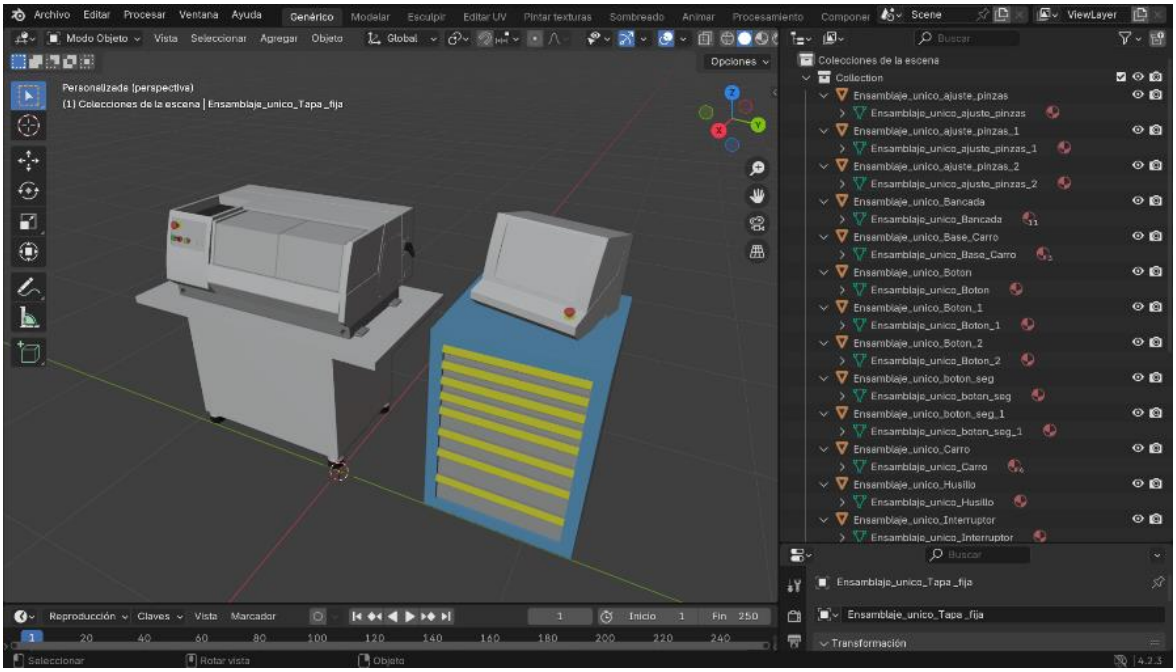


Figura 43. Modelo en la escena de Blender.

Una vez comprobado que todo se ha importado correctamente y cada objeto del ensamblaje está en su sitio, pasaremos a hacer la exportación a Unity. Para ello, volvemos a la pestaña 'Archivo', seleccionamos 'Exportar' y luego elegimos 'FBX (.fbx)'.

FBX es un formato de archivo utilizado para la transferencia de modelos 3D, animaciones y datos relacionados entre diferentes aplicaciones de diseño y modelado 3D. Este formato fue desarrollado por Kaydara y aunque actualmente es propiedad de Autodesk, numerosas aplicaciones de modelado y animación son capaces de abrir este tipo de archivo.

Esto es especialmente útil en este proyecto ya que Unity tiene soporte nativo para archivos FBX, lo que facilita la importación y manipulación de modelos 3D dentro del motor de juego. El archivo FBX puede almacenar una amplia gama de datos, incluyendo geometría, texturas o materiales, además de animaciones, luces y cámaras. Esto asegura que todos los detalles del modelo se mantengan intactos. Además, la estructura del archivo FBX permite una transferencia eficiente de datos entre diferentes softwares, reduciendo el tiempo y minimizando los posibles errores en el proceso de exportación e importación.[29]

Después de seleccionar el tipo de archivo, en la ventana del visor de archivos de Blender buscamos la carpeta en la que queremos guardar el archivo exportado, le asignamos el nombre deseado y pulsamos en ‘Exportar FBX’ para guardarlo. Ahora el modelo ya está listo para ser importado y utilizado en Unity.

5 CREACIÓN DEL ENTORNO DE REALIDAD VIRTUAL EN UNITY3D

Para crear la aplicación propiamente dicha de la que trata este TFG, utilizamos Unity3D. Unity es un motor de juego multiplataforma desarrollado por Unity Technologies, ampliamente utilizado en la industria del desarrollo de videojuegos y aplicaciones interactivas. Unity fue lanzado en 2005 con el objetivo de permitir a los desarrolladores de todos los niveles crear juegos y experiencias interactivas de alta calidad gracias a su facilidad de uso desde principiantes a profesionales. Es muy utilizado en la industria de los videojuegos debido a su robustez, versatilidad y facilidad de uso.[30]

Su motor de juego multiplataforma permite crear aplicaciones y juegos que pueden ser desplegados en una amplia variedad de plataformas, incluyendo PC, consolas, móviles, tabletas, webs y sistemas de realidad virtual y aumentada. Esto es una de las razones clave de su gran popularidad.
Unity también tiene un motor de gráficos potente que soporta renderizado en tiempo real y gráficos de alta calidad, lo que permite comprobar que todos los elementos se muestren de forma correcta. Esta capacidad es esencial para garantizar que cualquier ajuste se refleje de inmediato, facilitando la detección y corrección de errores. El editor visual es una herramienta intuitiva que facilita el diseño y desarrollo de juegos. Unity utiliza C# como lenguaje principal para los scripts, un lenguaje flexible y muy usado por los programadores, aunque también soporta scripts escritos en otros lenguajes de programación. Cuenta con una extensa biblioteca conocida como 'Unity Asset Store'. Aquí, se pueden encontrar modelos 3D, texturas, sonidos, scripts y otros recursos para utilizarles en los proyectos, ahorrando tiempo y esfuerzo. También es compatible con el desarrollo de aplicaciones de realidad aumentada y realidad virtual lo que nos permitirá crear una aplicación exportable a esta tecnología.
Unity Technologies proporciona documentación, tutoriales y soporte técnico para ayudar en el proceso de creación, además de tener una gran comunidad de desarrolladores que comparten conocimientos, y soluciones a problemas cotidianos.
Tanto por su versatilidad, potencia y accesibilidad para los inexpertos en el mundo de la programación de videojuegos, Unity es la mejor opción para la creación y programación básica de este proyecto. Además, permite que el desarrollo de esta aplicación se pueda ampliar en el futuro y nos asegura que el proyecto no solo sea fácil de iniciar, sino también adaptable a múltiples plataformas.[30][31]

5.1 CREACIÓN DEL SUELO, IMPORTACIÓN DEL MODELO Y CONFIGURACIÓN DE LOS BOX COLLIDERS.

Para empezar a crear la aplicación en Unity abrimos el programa y creamos un nuevo proyecto 3D. Al hacerlo, se abre una escena por defecto que incluye una cámara y una luz. También se crea un 'global

volume’ que podemos eliminar, ya que no lo necesitamos. Esta configuración inicial proporciona un punto de partida para construir nuestro entorno 3D. A continuación, lo que hacemos es añadir un suelo a la escena principal. Para ello, creamos un gameobject de tipo 'plane’ con el botón derecho del ratón en el panel 'Hierarchy' y seleccionando '3D Object > Plane'. La Hierarchy (jerarquía) es el panel donde se listan todos los objetos presentes en la escena actual de Unity y cada objeto puede incluir otros objetos hijos, permitiendo organizar y estructurarla de manera lógica.

Una vez creado el plano, ajustamos tanto su tamaño y posición desde el Inspector, que es la ventana donde se muestran las propiedades de los objetos seleccionados en cada momento. Esta ventana permite cambiar aspectos como la escala, la posición y la rotación, así como ajustar componentes adicionales que se pueden añadir, como materiales, scripts o físicas. Desde aquí podemos centrar el plano para que se sitúe en el origen de coordenadas.

Esto se verá reflejado en la ventana 'Scene' (escena), que es donde visualizamos y editamos la disposición de los objetos en nuestra escena. La escena permite una vista interactiva del entorno 3D, donde podemos mover, rotar y escalar los objetos para ajustarlos según nuestras necesidades.

Con el plano creado, ya tenemos un objeto sobre el que podremos colocar el modelo del torno, así como un suelo sobre el que se mueva nuestro objeto que hará las veces del jugador u operario del torno y con el que poder interactuar con el entorno.

A continuación, importamos el torno a unity. Para ello solo tenemos que abrir la carpeta en donde hemos guardado el modelo y lo arrastramos hasta la ventana ‘Project’ que es donde gestionamos todos los recursos y archivos de nuestro proyecto y lo soltamos en la carpeta ‘Assets’ que es donde se almacenan todos los archivos y recursos del proyecto (modelos 3D, texturas, imágenes, scripts, etc.) Con esto hecho nuestro modelo ya forma parte de unity y podemos añadirlo a la escena directamente arrastrándolo a la escena o a la jerarquía. Todos los cambios que vamos haciendo se pueden ver automáticamente reflejados tanto en la ventana ‘Game’. Esta ventana es una vista previa de cómo se verá y funcionará la aplicación al ejecutarla y nos permite probar el proyecto para que todo funcione correctamente antes de la compilación final.

Con el torno seleccionado nos dirigimos al inspector y en el botón ‘add component’ pulsamos y buscamos ‘box collider’. Esto crea un componente en forma de caja alrededor del modelo, que ajustamos para definir un área invisible que colisiona con otros objetos físicos. La función principal del Box Collider es evitar que los objetos traspasen el modelo y nos aseguramos que cualquier interacción física con el torno, como el personaje al caminar, se comporte de manera realista, respetando los límites en vez de atravesar el modelo.

Cualquier cambio, error o evento que ocurra durante el desarrollo se reflejará en la ventana 'Console', un panel donde se muestran mensajes de depuración, advertencias y errores. Esto nos permite identificar y solucionar problemas rápidamente evitando contratiempos.

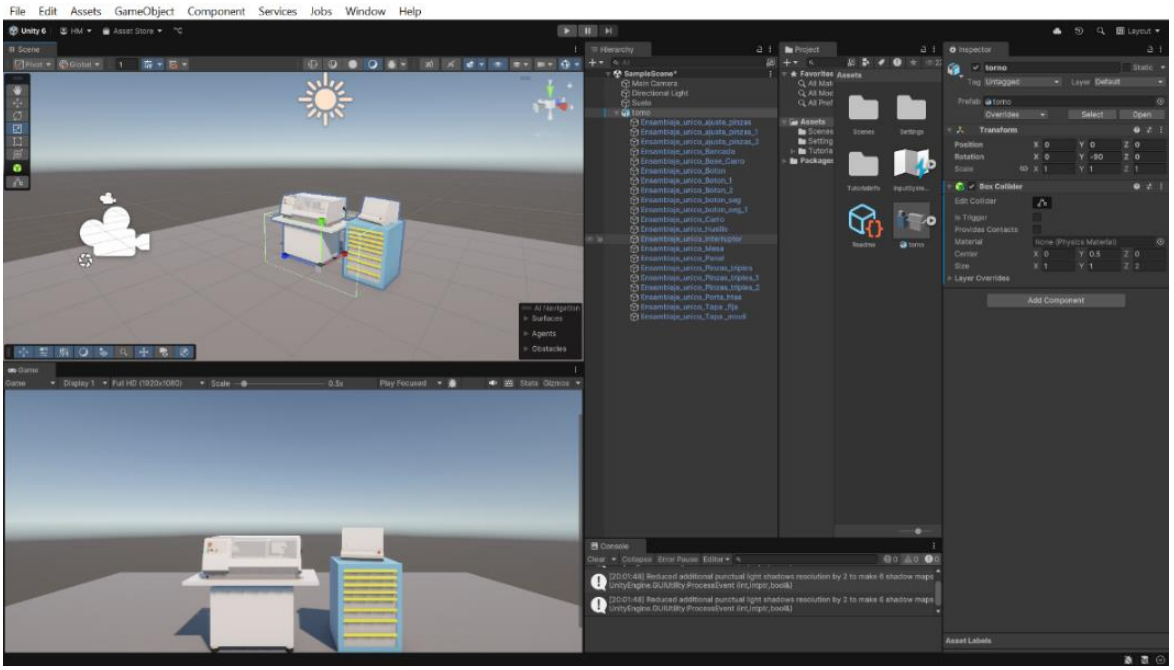


Figura 44. Modelo del torno en la escena de Unity con el box collider.

5.2 MOVIMIENTO DEL JUGADOR Y CONFIGURACIÓN DEL PUNTERO

Continuamos creando lo que será nuestro jugador. Para ello creamos un ‘GameObject’ 3D de tipo ‘capsule’, que incluye un componente llamado ‘capsule collider’. Este componente funciona similar al ‘box collider’ del torno definiendo una zona de colisión alrededor del jugador asegurando que, al desplazarnos con él, no traspasemos otros objetos y podamos interactuar con ellos.

Le añadimos un componente 'Rigidbody' que sirve para dotar a la cápsula de propiedades físicas, como la gravedad y nos permite configurar el objeto para que reaccione a las fuerzas y colisiones de manera realista dentro del entorno.

Después, en la jerarquía arrastramos la cámara a este objeto para que sea su 'hijo'. De este modo, cuando nos desplacemos con la cápsula, la cámara seguirá sus movimientos proporcionando una vista en primera persona consiguiendo más esa sensación de inmersión en el entorno.

5.2.1 Configuración del movimiento del jugador

Para permitir el movimiento del jugador dentro del entorno virtual, creamos un script en C# llamado ‘PlayerMovement’. Este script, adjunto al gameobjet del jugador, emplea las flechas de dirección o las

teclas ‘WASD’ para mover al personaje por el escenario. La velocidad de desplazamiento se controla mediante una variable pública ‘speed’, ajustable desde el Inspector.

Para evitar que el jugador se vuelque durante el desplazamiento, es necesario ajustar las ‘Constrains’ del componente ‘Rigidbody’ del jugador que consiste en bloquear las rotaciones en los ejes ‘X’ y ‘Z’, permitiendo únicamente la rotación en el eje ‘Y’. De esta forma, evitamos que el jugador se desplace de manera incontrolada o vuelque debido a las inercias.

```
using UnityEngine;

public class PlayerMovement : MonoBehaviour
{
    public float speed = 5.0f;
    private Rigidbody rb;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void Update()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);

        // Transformar la dirección del movimiento según la dirección de la cámara
        movement = Camera.main.transform.TransformDirection(movement);
        movement.y = 0.0f; // Evitar que el jugador se mueva en el eje Y

        rb.MovePosition(transform.position + movement * speed * Time.deltaTime);
    }
}
```

Figura 45.Script PlayerMovement.

5.2.2 Control con el ratón

Continuamos creando un script llamado ‘CameraController’ que adjunto al objeto de la cámara, facilita la rotación de esta dependiendo de los movimientos del ratón, añadiendo la capacidad de orientación del jugador. Adicionalmente, ajustamos el script anterior para que el movimiento de avance del jugador siga la dirección en la que apunta la cámara.

```
using UnityEngine;

public class CameraController : MonoBehaviour
{
    public Transform player;  // Referencia al jugador
    public Vector3 offset;    // Distancia de la cámara al jugador
    public float sensitivity = 5.0f;  // Sensibilidad del ratón

    private float rotationY = 0.0f;
    private float rotationX = 0.0f;

    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        offset = transform.position - player.position;
    }

    void Update()
    {
        rotationY += Input.GetAxis("Mouse X") * sensitivity;
        rotationX -= Input.GetAxis("Mouse Y") * sensitivity;
        rotationX = Mathf.Clamp(rotationX, -90f, 90f);

        transform.localRotation = Quaternion.Euler(rotationX, rotationY, 0);
        transform.position = player.position + offset;
    }
}
```

Figura 46. Script CameraController.

5.2.3 Interacción con Objetos

Para permitir la interacción con los objetos, se creó un 'gameobject' vacío denominado 'Hand', el cual se posicionó como hijo de la cámara para que se mueva en sincronía con ella. Posteriormente, se desarrolló un script llamado 'HandController', que lance un rayo, o raycast, desde la posición de la cámara hacia adelante para detectar aquellos objetos que sean interactivos, a los cuales los etiquetamos en el Inspector con un tag que también creamos y llamamos 'Interactable'.

Para facilitar esta interacción, se creó un pequeño objeto de esfera llamado 'HandPointer' que actúa como puntero. El script se ajustó para actualizar la posición del puntero según la dirección del rayo y se le otorgó la capacidad de cambiar el color de azul a verde, en función de si el raycast golpea un objeto con el tag 'interactable' o no.

También se añadió una lógica para asegurar que el puntero mantenga un tamaño constante en la pantalla, independientemente de la distancia a la que se encuentre del jugador para evitar que mengüe el tamaño si está muy lejos o aumente muy cerca.

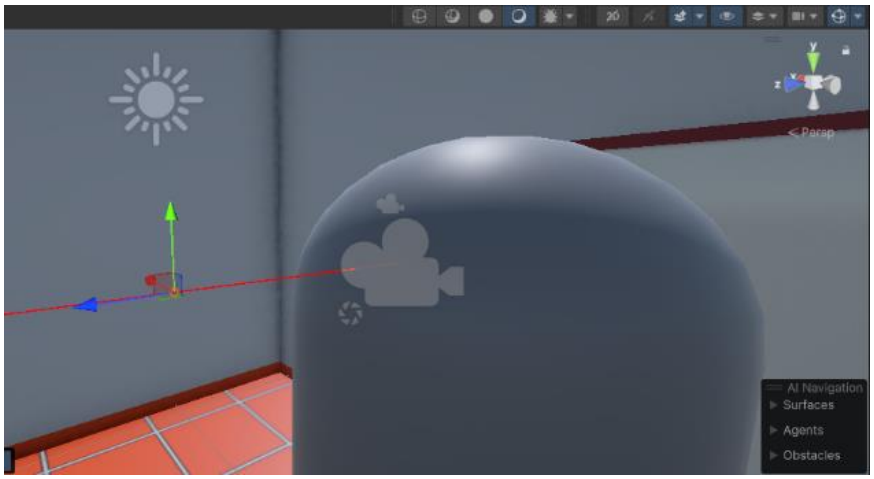


Figura 41. Raycast y puntero del jugador.

```
using UnityEngine;

public class HandController : MonoBehaviour
{
    public float interactionDistance = 2.0f;
    public float defaultPointerDistance = 0.5f; // Distancia cuando no toca nada
    public GameObject handPointer;
    private Renderer handPointerRenderer;
    public float initialScale = 0.004f; // Escala inicial
    public float scaleFactor = 0.02f; // Factor de escala

    void Start()
    {
        // Obtener el Renderer del handPointer
        handPointerRenderer = handPointer.GetComponent<Renderer>();
        // Ajustar la escala inicial del handPointer
        handPointer.transform.localScale = new Vector3(initialScale, initialScale, initialScale);
```

Figura 47. Inicio del script HandController.

5.3 CONFIGURACIÓN DEL INTERRUPTOR Y LOS BOTONES

5.3.1 Interruptor

El interruptor del torno es responsable de conectar o desconectar el torno de la fuente de electricidad. Es crucial configurarlo de manera que nos garantice que, cuando el interruptor esté en la posición de apagado, el torno no pueda operar en ninguna circunstancia.

Para gestionar el estado del interruptor, se desarrolló un script llamado 'SwitchController'. Este script permite alternar entre dos posiciones (encendido y apagado) mediante clics con el ratón.

Se implementó una rotación de 90º en sentido antihorario para indicar la posición de encendido y una la inversa para regresar a la posición inicial, indicando apagado con rotaciones locales para asegurar que rote correctamente alrededor del mismo punto independientemente de su orientación en el escenario virtual.

```
IEnumerator RotateSwitch(float angle)
{
    Quaternion startRotation = rotationPoint.rotation;
    Quaternion endRotation = startRotation * Quaternion.Euler(Vector3.right * angle);
    float elapsedTime = 0;

    while (elapsedTime < rotationDuration)
    {
        rotationPoint.rotation = Quaternion.Slerp(startRotation, endRotation, elapsedTime / rotationDuration);
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    rotationPoint.rotation = endRotation;
}
```

Figura 48. Lógica de rotación del interruptor en el script SwitchController.

Para ajustar el pivote respecto el cual se produce el giro del interruptor, se creó una esfera denominada 'RotationPoint'. Decidimos crear una esfera ya que, a diferencia de un objeto vacío, esta es mucho más fácil de posicionar visualmente en la escena y permite un mejor ajuste para conseguir el movimiento que queremos. Esta esfera, que posteriormente ocultamos para que no se vea en la aplicación final, actúa como nuevo pívot, la cual se colocó en la jerarquía como padre del interruptor y hacer que rote alrededor de este punto.

Figura 49. Esfera de referencia (visible y oculta) para el movimiento del interruptor.

5.3.2 Control de las luces

Cuando en el torno real encendemos el interruptor, se enciende la luz del botón amarillo, indicando que el torno está conectado a la corriente eléctrica. Para esta función se creó un script denominado como 'YellowButtonController', que se encarga de encender y apagar la luz de este botón según el estado del interruptor. También, hubo que modificar el script 'SwitchController' para que informase a este nuevo script y a futuros scripts que creemos, sobre el estado del interruptor para que se activen o desactiven según el estado de este.

También añadimos objetos de luz al botón amarillo, así como al verde, cuya programación futura asegura que funcione con el interruptor. Al activar el interruptor, solo se enciende la luz amarilla, mientras que todas las luces que hubiese encendidas se apagan al desactivar el interruptor. Esta lógica se implementó garantizando que el estado de cada luz sea coherente con la posición del interruptor, proporcionando así una indicación visual clara al usuario.

Figura 50. Luz amarilla en respuesta al interruptor.

5.3.3 Añadir movimiento a los botones

Para mejorar la experiencia de usuario y evitar que la interacción con los botones se limite solo a un clic, se añadió un movimiento en los botones cuando sean presionados. Este cambio hace que la interacción sea más inmersiva y fiel a la realidad.

Para ello, se desarrolló un nuevo script en C# 'ButtonMovement' para gestionar el desplazamiento suave de los botones. Usa corrutinas que son una característica que permite ejecutar partes del código de manera secuencial en durante un tiempo, suspendiendo y reanudando su ejecución en diferentes puntos. En lugar de ejecutar todo el método de una vez, una corrutina puede realizar una pausa (por ejemplo, para esperar un período de tiempo) y luego continuar desde el punto donde se quedó. Esto es especialmente útil para tareas que requieren múltiples pasos o para animaciones, o la realización de cálculos sin pausar el programa entero. Este script ajusta la posición del botón para simular una pulsación y luego lo devuelve a su estado original de manera fluida.

Se añaden unas variables públicas para los valores de la distancia (Move Distance), velocidad (Move Speed) y ángulo (Angle) que se pueden configurar desde el Inspector para que el movimiento sea perceptible y proporcional a la inclinación de los botones, ya que estos están situados y orientados según un plano con cierta inclinación del propio torno.

```
using UnityEngine;
using System.Collections;

public class ButtonMovement : MonoBehaviour
{
    public float moveDistance = 0.01f; // Distancia que el botón se moverá hacia dentro
    public float moveSpeed = 0.1f; // Velocidad del movimiento
    public float angle = 70.98f; // Ángulo de inclinación del botón
    public AudioSource audioSource;
    public AudioClip clickSound;

    private Vector3 initialPosition;
    private Vector3 moveDirection;

    void Start()
    {
        initialPosition = transform.localPosition;
        float radianAngle = angle * Mathf.Deg2Rad;
        moveDirection = new Vector3(Mathf.Sin(radianAngle), -Mathf.Cos(radianAngle), 0);
```

Figura 51. Fragmento del script ButtonMovement.

5.4 ENCENDIDO DEL TORNO Y CONTROL DE LA TAPA DE PROTECCIÓN

5.4.1 Simulación del encendido del torno

Para gestionar el encendido del torno de manera controlada, se desarrolló un script llamado 'LatheController'. El estado inicial del torno está configurado en apagado, y este script hace que el solo se active mediante la interacción con el botón ON (botón verde), siempre que el interruptor esté en la posición de encendido. Si el interruptor se encuentra en dicha posición de encendido, al presionar ON, la luz asociada a este se enciende y el torno puede comenzar a funcionar. En contraste, al presionar el botón OFF (botón rojo), el torno se apaga, apagando simultáneamente la luz verde. El funcionamiento del encendido y apagado de esta luz se configuró con dos scripts 'GreenButtonController' y 'RedButtoncontroller' encargados de la interacción con estos.

```
using UnityEngine;

public class LatheController : MonoBehaviour
{
    public bool isSwitchOn = false; // Estado del interruptor

    public void SetSwitchState(bool switchState)
    {
        isSwitchOn = switchState;
        Debug.Log("Interruptor: " + (isSwitchOn ? "Encendido" : "Apagado"));
    }

    public void PressGreenButton()
    {
        if (isSwitchOn)
        {
            Debug.Log("Torno encendido.");
            // Aquí podría ir la lógica para encender el torno
        }
        else
        {
            Debug.Log("No se puede encender el torno, el interruptor está apagado.");
        }
    }

    public void PressRedButton()
    {
        Debug.Log("Torno apagado.");
        // Aquí podría ir la lógica para apagar el torno
    }
}
```

Figura 52. Script LatheController.

```
using UnityEngine;

public class GreenButtonController : MonoBehaviour
{
    public Light greenLight;
    public CanvasController canvasController;
    public LatheController latheController;

    void Start()
    {

        greenLight.enabled = false;
    }

    void OnMouseDown()
    {
        if (latheController != null && latheController.isSwitchOn)
        {
            SetGreenLightAndCanvasState(true);
            latheController.PressGreenButton(); // Notificar al LatheController que el botón verde ha sido presionado
        }
        else
        {
            Debug.Log("El interruptor no está en la posición de encendido.");
        }
    }

    public void SetGreenLightAndCanvasState(bool state)
    {
        greenLight.enabled = state;
        if (canvasController != null)
        {
            canvasController.SetCanvasState(state);
        }
    }

    public bool IsGreenLightOn()
    {
        return greenLight.enabled;
    }
}
```

Figura 53. Script GreenButtonController.

```
using UnityEngine;

public class RedButtonController : MonoBehaviour
{
    public GreenButtonController greenButtonController; // Referencia al controlador del botón verde
    public LatheController latheController; // Referencia al LatheController

    void OnMouseDown()
    {
        if (latheController != null)
        {
            latheController.PressRedButton(); // Notificar al LatheController que el botón rojo ha sido presionado
            greenButtonController.SetGreenLightAndCanvasState(false); // Apaga  la luz verde y el Canvas
        }
    }
}
```

Figura 54. Script RedButtonController.

Figura 55. Luz verde en respuesta a los botones ON y OFF.

5.4.2 Movimiento de la tapa de seguridad

Para controlar la tapa de seguridad, se adaptó el script 'YellowButtonController'. Se añadió una función que desplaza la tapa de seguridad 330mm en el eje Z negativo de gradualmente al presionar el botón amarillo, siempre y cuando la luz verde esté encendida. Además, al presionar nuevamente el botón amarillo, la tapa de seguridad vuelve a su posición inicial de la misma manera.

Se implementó una lógica adicional en el script para asegurar que la tapa complete su desplazamiento antes de permitir otro movimiento, evitando así problemas de código y asegurando una simulación de las medidas de seguridad.

```
void OnMouseDown()
{
    // Verificar si la luz verde está encendida y si la tapa no está en movimiento
    if (greenButtonController != null && greenButtonController.IsGreenLightOn() && !isMoving)
    {
        if (movementCoroutine != null)
        {
            StopCoroutine(movementCoroutine);
        }

        if (isCoverOpen)
        {
            // Cerrar la tapa
            movementCoroutine = StartCoroutine(MoveCover(Vector3.forward * movementDistance));
            isCoverOpen = false;
        }
        else
        {
            // Abrir la tapa
            movementCoroutine = StartCoroutine(MoveCover(Vector3.back * movementDistance));
            isCoverOpen = true;
        }
    }
    else
    {
        if (isMoving)
        {
            Debug.Log("La tapa está en movimiento, espera a que termine para hacer otra acción.");
        }
        else
        {
            Debug.Log("La luz verde no está encendida, no se puede mover la tapa.");
```

Figura 56. Lógica para la apertura y cierre de la tapa en el script YellowButtonController.

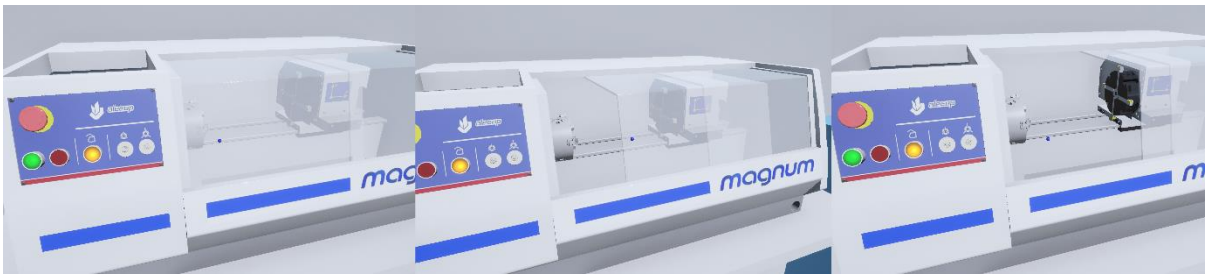


Figura 57. Transición de apertura de la tapa de seguridad.

5.4.3 Sonido del Interruptor

Cuando el interruptor del torno real se activa, este emite un pitido para indicar al operario que se ha encendido. Para implementar esta función se añadió un componente de sonido al interruptor. Se añadió un 'AudioSource' al objeto del interruptor y se configuró un 'AudioClip' para el sonido del pitido. El 'AudioSource' es un componente de Unity que reproduce un sonido y tiene diferentes opciones para controlar cómo se reproduce ese sonido, como el volumen, la distorsión, o la frecuencia. Por otro lado, el 'AudioClip' es el propio archivo de audio que se va a reproducir. Para crear este 'AudioClip' se buscó en una biblioteca de sonidos en la web y se descargó un archivo que más se ajustara al sonido real. Para hacerlo aún más realista del sonido se modificó usando el programa de edición de audio 'Audacity'. También se agregó al script 'SwitchController' una función para reproducir el sonido al activar el interruptor, indicando el encendido.

Se verificó que el pitido solo se emita cuando el interruptor pasa a la posición de encendido, evitando así la emisión del sonido al apagar el interruptor.

5.5 PANTALLA DEL PANEL

5.5.1 Añadir una pantalla de Información

Cuando arrancamos el torno y lo encendemos con el botón ON, se enciende la pantalla del panel de mando que sirve para controlar el funcionamiento y los programas de este. Al encenderse muestra el nombre del torno y el estado de la puerta de seguridad. Cuando se abre y cierra la puerta esto ve reflejado en el panel. Para proporcionar esta información visual en tiempo real, se decidió utilizar un canvas en Unity. Este canvas es un GameObject que contiene elementos UI que muestran datos relevantes, como el estado de la puerta (abierta o cerrada).

El canvas se crea en la jerarquía de Unity mediante la opción UI > canvas. Dentro del este, se añadieron dos objetos de texto (UI > Text), denominados 'DoorOpenText' y 'DoorClosedText'. Estos objetos se configuran para mostrar los mensajes 'PUERTA ABIERTA' y 'PUERTA CERRADA' respectivamente.

Para que el canvas se integre de manera coherente en el entorno del torno, se ajustaron la posición y la escala del canvas para alinearse con la pantalla del panel, asegurando una visualización clara. También se ajustaron los textos para que sean legibles desde cualquier perspectiva del jugador. Además, se colocó el canvas en el inspector como hijo del panel, para garantizar que mantenga su posición relativa si se modifica la posición del panel.

5.5.2 Control de la visibilidad del canvas

A continuación, se desarrollaron dos scripts: 'UIController' y 'CanvasController'. El 'UIController' se encarga de actualizar los cuadros de texto según el estado de la tapa de seguridad. Este script permite que el texto mostrado en el canvas cambie entre 'PUERTA ABIERTA' y 'PUERTA CERRADA' en función de si la tapa está abierta o cerrada.

```
public class UIController : MonoBehaviour
{
    public GameObject PuertaA; // Referencia al objeto de texto para "Puerta Abierta"
    public GameObject PuertaC; // Referencia al objeto de texto para "Puerta Cerrada"
    public YellowButtonController yellowButtonController; // Referencia al controlador del botón amarillo
    public Light greenLight; // Referencia a la luz verde

    void Update()
    {
        // Actualizar los cuadros de texto según el estado de la tapa y la luz verde
        if (yellowButtonController != null && greenLight != null && greenLight.enabled)
        {
            if (yellowButtonController.isCoverOpen)
            {
                PuertaA.SetActive(true);
                PuertaC.SetActive(false);
            }
            else
            {
                PuertaA.SetActive(false);
                PuertaC.SetActive(true);
            }
        }
        else
        {
            PuertaA.SetActive(false);
            PuertaC.SetActive(false);
        }
    }
}
```

Figura 58. Script UIController.

Por otro lado, el 'CanvasController' gestiona la visibilidad del canvas basado en el estado de la luz verde. Este script se asegura de que el canvas solo sea visible cuando la luz verde está encendida. Para ello, se le añadió una referencia al 'GreenButtonController', el cual controla la luz verde. El 'CanvasController' actualiza la visibilidad del canvas según el estado de la luz verde, asegurando que la pantalla se apague y encienda con el funcionamiento del torno.

Para garantizar que el canvas esté apagado al iniciar el programa (simulando que el torno aún no está encendido), se desactivó por defecto en el Inspector de Unity. Esta configuración asegura que permanezca apagado hasta que el interruptor esté en la posición de encendido y el botón ON sea presionado.

Figura 59. Arriba: Estados del canvas según en estado del torno (apagado y encendido). Abajo: Estados del canvas según el estado de la tapa.

5.6 SONIDO DE LOS BOTONES Y LA PUERTA

En las aplicaciones interactivas, los efectos sonoros pueden mejorar considerablemente la experiencia del usuario proporcionando feedback auditivo que completa las acciones realizadas por el usuario, haciendo que el uso sea más intuitivo. En este proyecto, se implementaron efectos sonoros para los botones y la puerta móvil, asegurando que estos sonidos se integren sin interferir con la funcionalidad existente. Estos sonidos debían de reproducirse al presionar los botones y durante el movimiento de la puerta.

5.6.1 Sonido de los botones

Inicialmente, se añadieron componentes de 'AudioSource' y 'AudioClip' a los scripts de los botones. Esto permite que cada vez que se hiciera clic en un botón, se reprodujera un sonido específico.

La reproducción del sonido se integró en el método 'OnMouseDown()'. El método 'OnMouseDown()' es un evento de Unity que se activa cuando el usuario hace clic en el objeto con el que está asociado. Este método se utiliza comúnmente para detectar clics del ratón y ejecutar código en respuesta a la acción de los usuarios. Con esto nos aseguramos que el sonido se reproduzca en el momento exacto en que se produce el clic, proporcionando una respuesta inmediata.

Posteriormente, se adaptaron los scripts de los botones para asegurar que el sonido se reproduzca correctamente al ser presionados, sin interferir con la funcionalidad existente.

5.6.2 Sonido de la puerta móvil

Para la puerta móvil, se siguió un enfoque similar. Se añadieron referencias a un 'AudioSource' y un 'AudioClip' en el script que la controla la puerta. El objetivo era que el sonido se reproduzca cada vez que la puerta comience a moverse, proporcionando un sonido continuo durante todo el movimiento. Este efecto se consiguió con el método 'MoveCover()', asegurando que el sonido acompañara al movimiento de la puerta en todo momento. Este método se encarga de actualizar la posición de la puerta en cada iteración de manera suave y fluida, utilizando interpolaciones para garantizar movimientos precisos y continuos. Además, el método se asegura de que el 'AudioSource' comience a reproducir el 'AudioClip' al inicio del movimiento y lo detenga al finalizar, proporcionando un efecto realista.

Finalmente, se realizaron pruebas para asegurar que los sonidos se reprodujeran correctamente durante las interacciones con los botones y la puerta. Durante estas pruebas, se ajustaron los volúmenes y se modificaron los clips de sonido en Audacity, para lograr el efecto deseado.

5.7 POSICIONAMIENTO ALEATORIO INICIAL

El posicionamiento aleatorio de objetos dentro de una escena es una técnica común en los desarrollos de juegos y aplicaciones interactivas. Permite que los elementos tengan ubicaciones iniciales variadas, lo que puede añadir realismo a la experiencia. En este proyecto de Unity, se implementó el posicionamiento aleatorio inicial para los objetos del carro portaherramientas, permitiendo así practicar la función de búsqueda de la posición cero.

5.7.1 Puntos de referencia

El objetivo era configurar estos objetos para que, al iniciar el programa, se posicionaran aleatoriamente dentro de unos límites definidos. El carro consta de dos partes, el carro transversal que se mueve perpendicular a las vías del torno y la deslizadera que se desplaza sobre estas, paralelas al eje del husillo. Al iniciar el programa, el conjunto debe situarse aleatoriamente a lo largo de sus ejes 'Z' (equivalente al eje X del entorno) y 'X' (equivalente al eje Z del entorno). Con el fin de establecer puntos de referencia para el movimiento, se crearon dos esferas, 'ZPTAHTAS' y 'XPTAHTAS', que se ubicaron en la jerarquía para ser seguidas por los objetos del portaherramientas. Esto permitió que cada objeto pudiera seguir su trayectoria de manera bien coordinada.

Los límites del movimiento de ambos objetos también estaban definidos por otros objetos de referencia: ‘CEROCERO’, situado en el cero del husillo, ‘100Z’ en el punto de máximo del eje ‘Z’, y ‘100X’ en el punto de máximo del eje ‘X’.

Estas esferas posteriormente se ocultan para no verse en el resultado final.

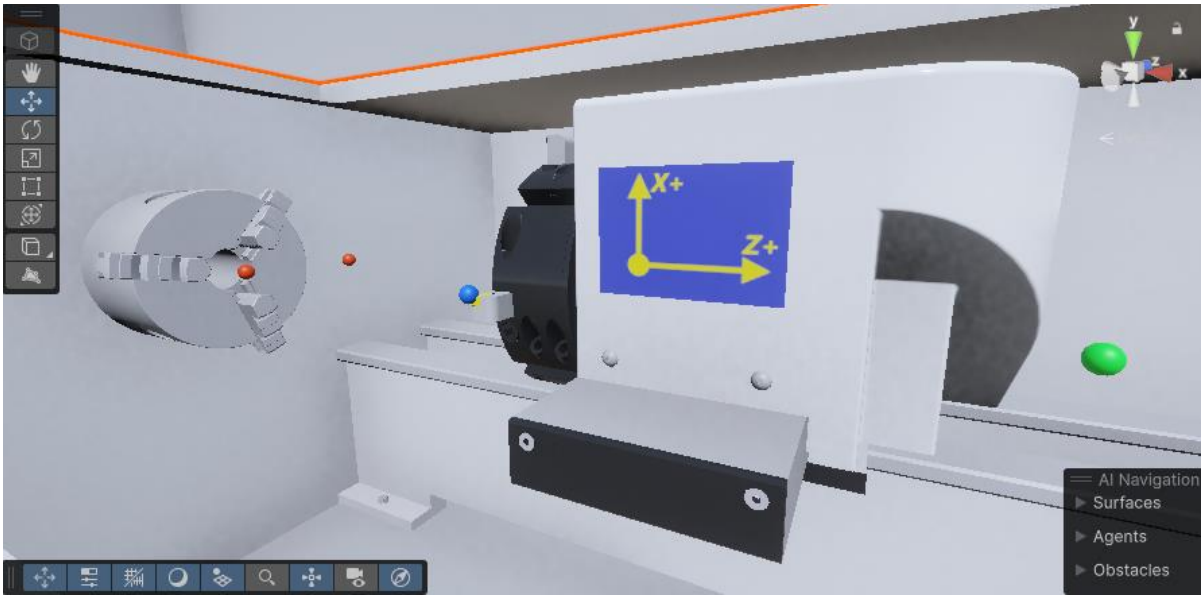


Figura 60. Esferas para los puntos de referencia del posicionamiento del carro.

5.7.2 Asignación de la posición

Se creó un script, ‘MovementLimiter’ que contiene la lógica para el posicionamiento aleatorio. Este script se asignó a uno de estos objetos presentes en la escena, y se configuraron las referencias a los objetos en el Inspector de Unity.

Empezamos configurando el objeto ‘ZPTAHTAS’ para que, al iniciar el programa, se posicionara aleatoriamente en su eje ‘Z’, manteniendo fijas sus coordenadas ‘X’ e ‘Y’. Los límites de este movimiento se establecen utilizando las posiciones en ‘X’ de los objetos ‘CEROCERO’ y ‘100Z’.

De manera similar, se configuró el objeto ‘XPTAHTAS’ para que al iniciar el programa se posicionara de forma aleatoria únicamente en su eje ‘X’. Los límites de este movimiento se definen con las posiciones en ‘Z’ de los objetos ‘CEROCERO’ y ‘100X’.

Para implementar estas configuraciones en el script, se calcula los límites en el eje ‘Z’ (‘zMin’ y ‘zMax’) y en el eje ‘X’ (‘xMin’ y ‘xMax’) basándose en las posiciones de ‘CEROCERO’, ‘100Z’ y ‘100X’. Luego, se les asigna una posición aleatoria a ‘ZPTAHTAS’ y ‘XPTAHTAS’ dentro de estos límites.

```
using UnityEngine;

public class MovementLimiter : MonoBehaviour
{
    public Transform ZPTAHTAS;
    public Transform XPTAHTAS;

    public Transform CEROCERO;
    public Transform LIMIT_100Z;
    public Transform LIMIT_100X;

    private float zMin;
    private float zMax;
    private float xMin;
    private float xMax;

    void Start()
    {
        // Calcular los límites basados en los puntos de referencia
        zMin = CEROCERO.position.x;
        zMax = LIMIT_100Z.position.x;
        xMin = CEROCERO.position.z;
        xMax = LIMIT_100X.position.z;

        Debug.Log("Límites Z (en términos del entorno): " + zMin + " a " + zMax);
        Debug.Log("Límites X (en términos del entorno): " + xMin + " a " + xMax);

        // Asignar posición aleatoria a ZPTAHTAS en su eje Z (equivalente al eje X del entorno)
        Vector3 newPositionZPTAHTAS = ZPTAHTAS.position;
        newPositionZPTAHTAS.x = Random.Range(zMin, zMax); // Movimiento aleatorio en X del entorno (eje Z de ZPTAHTAS)
        ZPTAHTAS.position = newPositionZPTAHTAS;

        // Asignar posición aleatoria a XPTAHTAS en su eje X (equivalente al eje Z del entorno)
        Vector3 newPositionXPTAHTAS = XPTAHTAS.position;
        newPositionXPTAHTAS.z = Random.Range(xMin, xMax); // Movimiento aleatorio en Z del entorno (eje X de XPTAHTAS)
        XPTAHTAS.position = newPositionXPTAHTAS;

        Debug.Log("ZPTAHTAS Inicial: " + ZPTAHTAS.position);
        Debug.Log("XPTAHTAS Inicial: " + XPTAHTAS.position);
    }
}
```

Figura 61. Script MovementLimiter.

Figura 62. Diferentes estados aleatorios de inicialización del torno.

5.8 BÚSQUEDA DEL CERO DE LA MÁQUINA

Uno de los pasos que hay que hacer en el torno real antes de iniciarle, es el de posicionar el carro en la "posición de cero máquina". Este procedimiento implica pulsar una secuencia específica de botones en el panel de control, lo cual permite que el carro se ubique en una posición predeterminada para las operaciones de mecanizado.

5.8.1 Definir la interacción con los botones

El primer paso fue crear unos GameObjects que simulasen los botones del panel de control. Estos se crearon como planos con las dimensiones y posiciones adecuadas, y se les asignaron imágenes que imitan la apariencia de los botones originales. Además, se les etiquetó con la etiqueta 'Interactable' para permitir la interacción con clics del ratón. Estos botones son: 'F1', 'X', 'Y' y el botón verde del panel, llamado dentro de Unity 'VERDE'.

5.8.2 Posicionamiento en el eje X

Empezamos programando la posición según el eje X, con las interacciones con los botones 'X', 'F1', y 'VERDE'. La idea era que al clicar en ese orden exacto, el objeto 'XPTAHTAS' se desplazara lentamente hasta alcanzar el máximo en su eje X.

Para lograr esta funcionalidad, se creó el script 'PosicionamientoHerramienta'. Este script detectaría cuando se interactúa con cada uno de estos GameObjects en el orden correcto y, al completarse la secuencia, inicia el movimiento de 'XPTAHTAS'.

El script debe monitorear las interacciones con los botones 'X', 'F1', y 'VERDE'. Usando el método 'OnTriggerEnter' de Unity, se puede comprobar si el objeto con el que se está interactuando tiene el tag 'Interactable'. Para cada interacción, se incrementa un contador que rastrea las interacciones (interactionStep). Solo si estas ocurren en el orden correcto, el contador llega al valor para el movimiento de 'XPTAHTAS'.

Una vez realizadas las interacciones en dicho orden ('X', 'F1' y 'VERDE'), debe moverse 'XPTAHTAS' al máximo en su eje X arrastrando con ella el carro transversal. Se utiliza una interpolación lineal con 'Mathf.Lerp' para sea un movimiento suave. La velocidad se puede ajustar directamente desde el Inspector de Unity.

Este script 'PosicionamientoHerramienta' debe ser asignado a un GameObject en la escena (puede ser 'XPTAHTAS' o un objeto vacío, pero decidimos asignárselo al mismo objeto para que la configuración sea más intuitiva) y todas las referencias hay que configurarlas en el Inspector.

```
void MoveXPTAHTAS()
{
    if (XPTAHTAS == null)
    {
        return;
    }

    Vector3 position = XPTAHTAS.position;
    position.z += velocidadMovimiento * Time.deltaTime; // Usar la variable velocidadMovimiento para controlar la velocidad

    // Verificar si ha alcanzado el límite
    if (position.z >= xMax)
    {
        position.z = xMax;
        shouldMoveX = false; // Detener el movimiento cuando se alcanza el límite máximo
    }

    XPTAHTAS.position = position;
}
```

Figura 63. Lógica para el movimiento en X del script PosicionamientoHerramienta.

5.8.3 Actualizar el script 'HandController'

El tercer paso consistió en actualizar el 'HandController' para que llamara al script 'PosicionamientoHerramienta'. Esto permite que las interacciones con los objetos 'X', 'F1', y 'VERDE' desencadenen el movimiento de 'XPTAHTAS' de manera correcta y fluida.

A continuación, verificamos que el script estuviera asignado correctamente en la escena y que las referencias estuvieran bien configuradas en el Inspector. Luego, ejecutamos el juego para comprobar que, al interactuar con los botones en el orden correcto, 'XPTAHTAS' se moviera adecuadamente y observamos la consola para asegurarnos de que no hubiera errores.

Durante estas pruebas los resultados eran correctos. Sin embargo, la consola mostraba un mensaje de advertencia que indicaba que un método utilizado en el script 'HandController' (Object.FindObjectOfType<T>) había quedado obsoleto. Aunque el programa funcionaba correctamente con este método, su uso podría generarnos problemas a futuro, especialmente si queremos exportar el programa a otras plataformas o profundizar más en su desarrollo, ya que este método podría ser eliminado en versiones futuras de Unity y su uso no es recomendado.

Para evitar posibles inconvenientes, consultamos la página oficial de Unity en busca de alternativas. Encontramos una versión más actualizada de este método (Object.FindFirstObjectByType<T>), que no solo nos evitará problemas futuros, sino que también ofrece mejoras en términos de eficiencia y rendimiento.

5.8.4 Ajuste de la velocidad

El siguiente paso fue ajustar la velocidad de movimiento para que pudiera ser controlada desde el Inspector de Unity. Para lograr esto, se añadió una nueva variable pública en el script 'PosicionamientoHerramienta', para el ajuste de la velocidad directamente sin necesidad de modificar el código.

Se creó la variable pública 'velocidadMovimiento' en el script. Esta nos permite cambiar la velocidad desde el Inspector sin necesidad de modificar el código.

5.8.5 Posicionamiento según el eje Z

Habiendo configurado el posicionamiento en X y comprobado que su funcionamiento era el adecuado, el paso posterior consistió en añadir esta misma funcionalidad para que el objeto 'ZPTAHTAS' se moviera al punto medio de 'CEROCERO' y '100Z' cuando se interactúe con 'Z', 'F1' y 'VERDE' en dicho orden. El funcionamiento es casi idéntico por lo que solo fue necesario actualizar el script e implementar las modificaciones pertinentes referidas a este objeto.

```
void MoveZPTAHTAS()
{
    if (ZPTAHTAS == null)
    {
        return;
    }

    Vector3 position = ZPTAHTAS.position;
    position.x = Mathf.MoveTowards(position.x, zMidPoint, velocidadMovimiento * Time.deltaTime); // Movimiento hacia el punto medio

    // Verificar si ha alcanzado el punto medio
    if (position.x == zMidPoint)
    {
        shouldMoveZ = false; // Detener el movimiento cuando se alcanza el punto medio
    }

    ZPTAHTAS.position = position;
}
```

Figura 64. Lógica para el movimiento en Z del script PosicionamientoHerramienta.

5.8.6 Reinicio de las secuencias

El último paso fue añadir la función de restablecer los contadores si se interactuaba con algún botón diferente a los del movimiento en los ejes X o Z. Esta característica garantiza que las secuencias de interacción se reinicien en caso de una interacción no deseada.

Se añadieron condiciones en el método para detectar si se interactuaba con cualquier botón fuera de la secuencia correcta. Esto se hizo para asegurarnos de que el sistema pudiera identificar interacciones no deseadas y que el movimiento no se produzca cuando no o deseamos. Por ejemplo, si la secuencia esperada para 'XPTAHTAS' era interactuar con los botones 'X', 'F1', y 'VERDE' en ese orden, y el usuario interactúa con un botón diferente en cualquier punto de la secuencia, el sistema lo detecta y reinicia

las variables ‘interactionStepX’ e ‘interactionStepZ’. Estas variables se utilizan para rastrear el progreso de las secuencias de interacción para los movimientos de ‘XPTAHTAS’ y ‘ZPTAHTAS’, respectivamente.

Resultado obtenido:

- Al interactuar con los botones en el orden correcto el carro y la deslizadera se mueven según lo esperado hasta la posición de ‘cero máquina’.
- Si se interactúa con cualquier otro botón fuera de las secuencias válidas, los contadores de interacción se reinician, garantizando que las secuencias se mantengan en el orden correcto.

```
public void InteractuarConObjeto(string nombreObjeto)
{
    // Interacción para XPTAHTAS
    if (interactionStepX == 0 && nombreObjeto == "X")
    {
        interactionStepX = 1;
    }
    else if (interactionStepX == 1 && nombreObjeto == "F1")
    {
        interactionStepX = 2;
    }
    else if (interactionStepX == 2 && nombreObjeto == "VERDE")
    {
        interactionStepX = 3;
        shouldMoveX = true; // Habilitar el movimiento de XPTAHTAS
    }
    else
    {
        interactionStepX = 0; // Reiniciar los pasos de interacción
        shouldMoveX = false;
    }

    // Interacción para ZPTAHTAS
    if (interactionStepZ == 0 && nombreObjeto == "Z")
    {
        interactionStepZ = 1;
    }
    else if (interactionStepZ == 1 && nombreObjeto == "F1")
    {
        interactionStepZ = 2;
    }
    else if (interactionStepZ == 2 && nombreObjeto == "VERDE")
    {
        interactionStepZ = 3;
        shouldMoveZ = true; // Habilitar el movimiento de ZPTAHTAS
    }
    else
    {
        interactionStepZ = 0; // Reiniciar los pasos de interacción
        shouldMoveZ = false;
    }
}
```

Figura 65. Lógica para el reinicio de las secuencias del script PosicionamientoHerramienta.

5.9 SONIDO PARA EL DESPLAZAMIENTO DE LA HERRAMIENTA

Para indicar el movimiento de la herramienta, se añadió un efecto sonoro que se reprodujera de manera continua mientras los objetos del carro (XPTAHTAS) y la deslizadera (ZPTAHTAS) estuviesen en movimiento. Este sonido se debe empezar a reproducir al inicio del movimiento y detenerse al final, ofreciendo un sonido continuo durante el mismo.

En primer lugar, se seleccionó el GameObject que contenía el script PosicionamientoHerramienta, para añadirle los efectos de audio. En el Inspector, se buscó y se seleccionó un 'AudioSource'. Tras añadir este componente, se descargó un clip de audio desde una biblioteca de sonidos y se ajustó utilizando Audacity para adaptarlo. Este clip de audio se asignó al 'AudioSource', permitiendo que se reproduzca cuando se active dicho componente.

A continuación, se actualizó el script 'PosicionamientoHerramienta' para controlar el 'AudioSource' y gestionar la reproducción del sonido mientras los objetos se movían. Se añadió una variable pública para asignar el componente desde el Inspector, y se añadió lógica para reproducir el sonido cuando el objeto esté en movimiento y detenerlo cuando se detenga. Todas las referencias fueron correctamente configuradas en el inspector y se ajustó la velocidad de movimiento

Al probar el juego, se comprobó que el sonido se reproducía en el momento adecuado, pero sin embargo, se reproducía de manera errática como distorsionado. Esto era porque estaba reiniciándose en cada iteración del programa. Para solucionar este problema, se ajustó el script para evitar que el audio se reiniciara en cada iteración. En lugar de utilizar los métodos 'Play()' y 'Stop()', se optó por 'PlayOneShot()', que es un método de Unity que permite reproducir un clip de audio una sola vez, asegurando una reproducción sin interrupciones.

Con estos ajustes, se garantizó que el sonido se reprodujera de manera continua durante el movimiento de los objetos, sin reiniciarse constantemente y se detuviese al finalizar el movimiento.

5.10 CONDICIONES DE INICIACIÓN DEL SCRIPT

En el desarrollo de aplicaciones, es necesario establecer condiciones específicas para la ejecución de ciertos scripts. Esto garantiza que las acciones solo se ejecuten bajo ciertas circunstancias, evitando comportamientos indeseados. En este proyecto, se implementaron condiciones de iniciación para el script encargado del movimiento de los objetos del carro portaherramientas.

El objetivo de esta implementación fue asegurar que el script 'PosicionamientoHerramienta' se ejecutara solo si se cumplían dos condiciones específicas: si la luz verde está encendida (indicando que

tanto el interruptor como el torno están encendidos) y si la tapa de seguridad está cerrada (evitando que el carro se desplace en situaciones de riesgo). Esto nos garantiza que los movimientos de los objetos solo ocurrieran en situaciones controladas y predecibles, mejorando la simulación de las medidas de seguridad del sistema.

Primero, se añadieron referencias a la luz verde y al controlador de la tapa en el script. Estas referencias permitieron verificar el estado de los componentes antes de cualquier acción.

Se desarrolló un método denominado 'CheckConditions()' que verificaba si ambas condiciones (luz verde encendida y tapa cerrada) se cumplían. Este método devolvía un valor (true o false) indicando si el script podía continuar con su ejecución.

En el Inspector de Unity, se asignaron las referencias a la luz verde (greenLight) y al controlador de la tapa (yellowButtonController) en el script 'PosicionamientoHerramienta'. Posteriormente, se realizaron pruebas para asegurar que el script solo se ejecutara cuando la luz verde estuviera encendida y la tapa estuviera cerrada. Durante estas pruebas, se comprobó que el script no realizaba ninguna acción si alguna de las condiciones no se cumplía.

```
public class PosicionamientoHerramienta : MonoBehaviour
{
    public Transform XPTAHTAS;
    public Transform ZPTAHTAS;
    public Transform CEROCERO;
    public Transform LIMIT_100X;
    public Transform LIMIT_100Z;
    public Light greenLight; // Referencia a la luz verde
    public YellowButtonController yellowButtonController; // Referencia al script de la tapa
    public float velocidadMovimiento = 1.0f; // Variable pública para ajustar la velocidad
    public AudioSource movimientoAudioSource; // Variable pública para el AudioSource
    public AudioClip movimientoClip; // Clip de audio para reproducir el sonido

    private float xMax;
    private float zMidPoint;
    private int interactionStepX = 0;
    private int interactionStepZ = 0;
    private bool shouldMoveX = false; // Variable para controlar el movimiento de XPTAHTAS
    private bool shouldMoveZ = false; // Variable para controlar el movimiento de ZPTAHTAS
    private bool isPlayingSound = false; // Variable para controlar el estado del sonido
```

Figura 66. Variables implicadas en el script PosicionamientoHerramienta.

5.11 PRUEBA DE GIRO DEL HUSILLO

Se espera que este proyecto continue desarrollándose en el futuro, con el objetivo de programarlo para que funcione con secuencias de Código G, permitiendo la simulación de los procesos de mecanizado de las piezas. Por ello, y para terminar con la programación en Unity del torno, se decidió añadir una demostración visual para mostrar cómo se vería el husillo girando durante el

funcionamiento de este. Para ello, se decidió dotar al husillo de la capacidad de girar a tres velocidades diferentes (lenta, media y rápida) y en ambos sentidos de giro (horario y antihorario). La velocidad y el sentido de rotación se seleccionan desde el panel de control, y se reflejan en la pantalla de este utilizando el canvas previamente creado.

5.11.1 Configuración de los botones

Se empezó por crear los botones para seleccionar tanto el sentido de giro como la velocidad. Al igual que en ocasiones previas, para cada botón que estuviese involucrado en esta función, se crearon gameobjects del tipo plano con la posición y escala adecuados a los que se les asignaron imágenes para que tuvieran la apariencia de los botones del panel real. Estos botones se les etiquetó con el tag 'Interactable' para poder tener interacción con ellos y que se apliquen los cambios. Estos botones son los de aumentar la velocidad ('+speed'), disminuir la velocidad ('-speed'), sentido horario ('horario'), sentido antihorario ('antihorario'), comienzo del giro ('ENTER') y pausa ('STOP').

5.11.2 Control de las velocidades

En esta sección se implementó un mecanismo para controlar las velocidades del sistema utilizando un script llamado 'SpeedController'. Este script fue diseñado para gestionar los textos del canvas 'VELOCIDAD: 1', 'VELOCIDAD: 2' y 'VELOCIDAD: 3', representando las distintas velocidades operativas y permitiendo que el husillo gire a la velocidad reflejada en pantalla.

Se implementaron los métodos específicos 'IncreaseSpeed' y 'DecreaseSpeed' en el script que modifican la variable 'currentSpeed', que detecta la velocidad a la que gira el husillo, y actualizan la el canvas según la velocidad actual.

Además, se añadieron verificaciones dentro de los métodos de control de velocidad para asegurar que las acciones solo se ejecuten si el canvas está activo. Esto evita que el husillo pueda girar si el torno está apagado, simulando así un correcto funcionamiento.

También se actualizó el script 'CanvasController' para manejar la visibilidad del Canvas y los objetos asociados, según el funcionamiento del husillo.

```
using UnityEngine;

public class SpeedController : MonoBehaviour
{
    public GameObject VEL1; // Referencia al GameObject VEL1
    public GameObject VEL2; // Referencia al GameObject VEL2
    public GameObject VEL3; // Referencia al GameObject VEL3
    public CanvasController canvasController; // Referencia al CanvasController

    private int currentSpeed = 2; // Comienza en VEL2

    void Start()
    {
        // Inicializar mostrando solo VEL2 si el Canvas está activo
        if (canvasController.canvas.enabled)
        {
            UpdateSpeedDisplay();
        }
        else
        {
            VEL1.SetActive(false);
            VEL2.SetActive(false);
            VEL3.SetActive(false);
        }
    }

    public void IncreaseSpeed()
    {
        if (canvasController.canvas.enabled && currentSpeed < 3)
        {
            currentSpeed++;
            UpdateSpeedDisplay();
        }
    }

    public void DecreaseSpeed()
    {
        if (canvasController.canvas.enabled && currentSpeed > 1)
        {
            currentSpeed--;
            UpdateSpeedDisplay();
        }
    }

    private void UpdateSpeedDisplay()
    {
        if (canvasController.canvas.enabled)
        {
            VEL1.SetActive(currentSpeed == 1);
            VEL2.SetActive(currentSpeed == 2);
            VEL3.SetActive(currentSpeed == 3);
        }
    }
```

Figura 67. Script SpeedController.

```
using UnityEngine;

public class CanvasController : MonoBehaviour
{
    public Canvas canvas; // Referencia al Canvas
    public GameObject VEL1; // Referencia al GameObject VEL1
    public GameObject VEL2; // Referencia al GameObject VEL2
    public GameObject VEL3; // Referencia al GameObject VEL3
    public GameObject GIROH; // Referencia al GameObject GIROH
    public GameObject GIROA; // Referencia al GameObject GIROA

    private bool wasVEL1Active;
    private bool wasVEL2Active;
    private bool wasVEL3Active;
    private bool wasGIROHActive;
    private bool wasGIROAActive;
    private bool isFirstActivation = true;

    void Start()
    {
        // Asegurarse de que el Canvas y los GameObjects estén deshabilitados inicialmente
        canvas.enabled = false;
        SetSpeedObjectsState(false);
    }

    public void SetCanvasState(bool state)
    {
        if (state)
        {
            if (isFirstActivation)
            {
                // La primera vez, activar solo VEL2 y GIROH
                VEL1.SetActive(false);
                VEL2.SetActive(true);
                VEL3.SetActive(false);
                GIROH.SetActive(true);
                GIROA.SetActive(false);
                isFirstActivation = false;
            }
            else
            {
                // Restaurar el estado de los GameObjects
                VEL1.SetActive(wasVEL1Active);
                VEL2.SetActive(wasVEL2Active);
                VEL3.SetActive(wasVEL3Active);
                GIROH.SetActive(wasGIROHActive);
                GIROA.SetActive(wasGIROAActive);

                // Si ningún GameObject está activo, activar VEL2 y GIROH por defecto
                if (!wasVEL1Active && !wasVEL2Active && !wasVEL3Active && !wasGIROHActive && !wasGIROAActive)
                {
                    VEL2.SetActive(true);
                    GIROH.SetActive(true);
                }
            }
        }
        else
        {
            // Guardar el estado actual de los GameObjects solo si alguno de ellos está activo
            if (VEL1.activeSelf || VEL2.activeSelf || VEL3.activeSelf || GIROH.activeSelf || GIROA.activeSelf)
            {
                wasVEL1Active = VEL1.activeSelf;
                wasVEL2Active = VEL2.activeSelf;
                wasVEL3Active = VEL3.activeSelf;
                wasGIROHActive = GIROH.activeSelf;
                wasGIROAActive = GIROA.activeSelf;
            }

            // Desactivar todos los objetos
            SetSpeedObjectsState(false);
        }

        // Cambiar el estado del Canvas
        canvas.enabled = state;
    }

    public void SetGiroState(bool isHorario)
    {
        if (isHorario)
        {
            GIROH.SetActive(true);
            GIROA.SetActive(false);
        }
        else
        {
            GIROH.SetActive(false);
            GIROA.SetActive(true);
        }
    }

    private void SetSpeedObjectsState(bool state)
    {
        // Desactivar todos los objetos
        VEL1.SetActive(state);
        VEL2.SetActive(state);
        VEL3.SetActive(state);
        GIROH.SetActive(state);
        GIROA.SetActive(state);
    }
```

Figura 68. Script CanvasController.

5.11.3 Control de la rotación

Para manejar la interacción con los botones de rotación (horario y antihorario), se creó un nuevo script llamado 'RotationController'.

Al igual que con las demás interacciones de objetos y botones programadas, el script 'RotationController' hace uso del raycast del puntero para detectar interacciones con los objetos etiquetados como 'Interactable', específicamente horario y antihorario. Dependiendo del objeto interactuado, se llama al 'CanvasController' para activar los textos de giro horario o antihorario. Esta lógica asegura que solo uno de los textos se muestre en pantalla.

Como en apartados anteriores, se añadió una verificación en el 'RotationController' para asegurarse de que las interacciones solo tengan efecto si el Canvas está activo.

Una vez que se configuró cómo debía mostrarse la pantalla en cada estado, se procedió a ajustar el husillo para que respondiera adecuadamente a las diferentes configuraciones y girara en función de la información mostrada por pantalla.

5.11.4 Inicialización del husillo

El primer paso fue definir las velocidades de operación para tres configuraciones distintas, con las variables publicas 'vel1RPM', 'vel2RPM' y 'vel3RPM' ajustables desde el inspector. Estas velocidades nos permiten ajustar la rotación del husillo en revoluciones por minuto (RPM), permitiendo ver como funcionaría en diferentes escenarios de mecanizado. La elección de estas velocidades garantiza un rango de operaciones que abarque desde bajas hasta altas velocidades.

Para la configuración del comportamiento de rotación del husillo, se implementó una lógica que actualiza tanto la velocidad como la dirección de giro basándose en los objetos visibles en el canvas de la interfaz de usuario. La lógica se diseñó para convertir la velocidad en RPM a grados por segundo, aplicando esta velocidad para rotar el objeto del husillo.

Se añadió un mecanismo para que los cambios de velocidad y sentido de giro fuesen lo más suaves y realista posibles. Se utilizó 'Mathf.SmoothDamp' para suavizar las transiciones entre velocidades, evitando cambios bruscos y garantizando fluidez.

Además, se implementaron efectos de sonido diferentes que sonasen para cada velocidad de forma parecida a anteriores apartados de este proyecto.

Por último, se implementaron verificaciones para que se cumplan todas las condiciones necesarias para que el husillo pueda girar. Esto incluye la comprobación de que la luz verde está encendida (torno encendido) y que la tapa de seguridad está cerrada.

```
using UnityEngine;

public class RotationController : MonoBehaviour
{
    public CanvasController canvasController; // Referencia al CanvasController

    void Start()
    {
        // Verificar si el CanvasController se encontró
        if (canvasController == null)
        {
            // CanvasController no encontrado en la escena. Asegúrate de que existe un CanvasController en la escena.
        }
    }

    void Update()
    {
        // Lanza un raycast desde la posición de la cámara hacia adelante
        Ray ray = new Ray(Camera.main.transform.position, Camera.main.transform.forward);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit, 2.0f)) // Distancia de interacción de 2.0f
        {
            // Mueve el indicador al punto de impacto
            if (hit.collider.CompareTag("Interactable"))
            {
                // Aquí puedes añadir el código para interactuar con el objeto
                if (Input.GetMouseButtonDown(0)) // Detecta el clic izquierdo
                {
                    Interact(hit.collider.gameObject);
                }
            }
        }
    }

    void Interact(GameObject obj)
    {
        // Interacción con: " + obj.name);

        // Control de giro
        if (canvasController != null && canvasController.canvas.enabled)
        {
            if (obj.name == "horario")
            {
                canvasController.SetGiroState(true);
            }
            else if (obj.name == "antihorario")
            {
                canvasController.SetGiroState(false);
            }
        }
    }
}
```

Figura 69. Script RotationController.

5.12 DETALLES FINALES Y DISEÑO DEL ESCENARIO

Después de configurar el torno con las funciones básicas, el siguiente paso fue añadir algunos elementos decorativos para enriquecer la escena y para hacer la experiencia más inmersiva, emulando el ambiente de un laboratorio universitario o un taller de procesos industriales. Comenzamos asignando las imágenes características a las distintas partes del torno, como el logo de la marca, las pegatinas de riesgo eléctrico y la placa con el número de serie. Aunque esto detalles no aportan funcionalidad al torno, añaden un toque de realismo y una sensación de autenticidad.

A continuación, se procedió a crear las paredes y el techo de la habitación. Con esto se consigue un espacio confinado en el que caben todos los elementos y también se evita que el jugador se salga de los límites de la escena y pueda caer al vacío. Esto se consiguió con cinco nuevos planos de la misma manera en que se creó el suelo al inicio y colocándolos en la posición adecuada. Los planos en Unity están configurados para ser visibles solo desde una de las dos direcciones. Es decir, si se observan

desde el frente, el plano será visible, pero si se miran por detrás, serán transparentes, permitiendo ver lo que hay al otro lado. Por esta razón, los nuevos planos que representan las paredes deben ser rotados 90 grados según su ubicación para que miren hacia el interior de la escena. De la misma manera, el plano que hace de techo debe ser rotado 180 grados. Esto permite que durante la configuración de la escena desde el exterior, sea posible ver a través de las paredes y visualizar lo que hay dentro, y sin embargo, una vez que la aplicación se ejecuta, el jugador queda confinado dentro de la escena, evitando su caída y asegurando que no se pueda ver el entorno exterior.

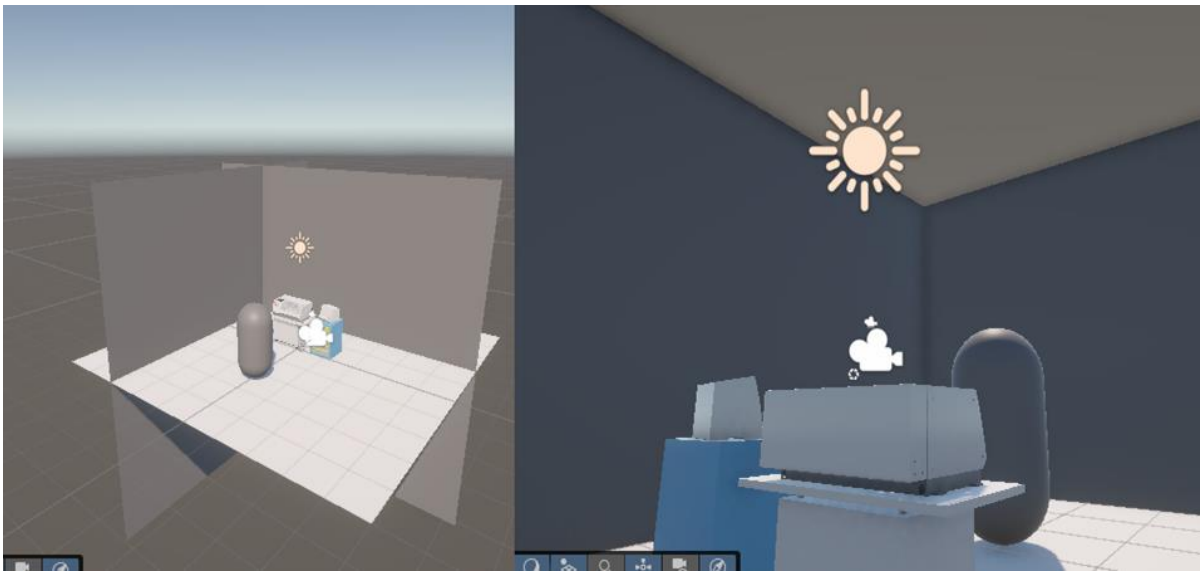


Figura 70. Visionado del entorno desde puntos opuestos fuera y dentro de las paredes.

Para ambientar la escena, se crearon en Inventor diversos objetos que imitaban armarios y estanterías típicos de cualquier taller. Además, se diseñó una puerta y unas lámparas para simular la iluminación. Estos objetos fueron importados a Unity siguiendo el mismo proceso que se utilizó para el torno, aunque más sencillo ya que eran objetos simples en lugar de ensamblajes complejos. Una vez en la carpeta de Assets de Unity, se arrastraron y se colocaron en la escena.

Dado que estos elementos son meramente decorativos, solo fue necesario aplicarles box colliders simples para evitar que el jugador los traspasase. A continuación, se añadieron puntos de luz a las lámparas para imitar bombillas. La intensidad, el rango y el alcance de estas luces se ajustaron para que funcionaran únicamente como decoración, evitando así que generasen sombras con el resto de los objetos y ahorrando capacidad de procesamiento, ya que la iluminación de toda la escena se consigue con la luz general (directional light) que hay al crear una nueva escena 3D por defecto. Para finalizar, se ajustaron algunas texturas para mejorar el aspecto general de la escena.

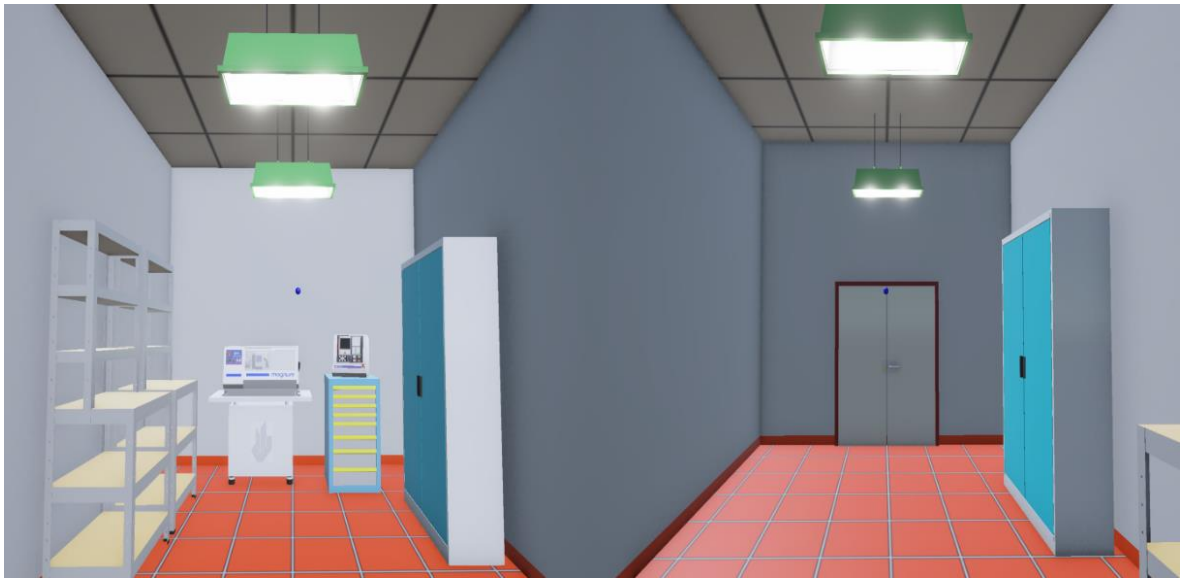


Figura 71. Aspecto final del entorno.

5.13 CONFIGURACIÓN DEL MENÚ PRINCIPAL

Para poder iniciar y cerrar la aplicación de forma simple como cualquier otro programa, fue necesario crear un menú principal que se muestre al abrir la aplicación, en lugar de mostrar directamente el simulador del torno. Para ello, se creó una nueva escena en Unity que aparece al iniciar la aplicación, proporcionando una navegación más intuitiva.

5.13.1 Creación de la Interfaz del menú principal

El menú principal se configuró en una nueva escena que creamos, llamada 'MainMenu'. El primer paso fue la creación de un canvas al que se le asignó un objeto (TextMeshPro) a modo de título con el texto: 'Simulador Fagor CNC 8025/8030'.

Durante la ejecución del juego, es posible cambiar de una escena a otra utilizando funcionalidades integradas de Unity. Este cambio puede ser provocado por diversas acciones del jugador, como completar un nivel o seleccionar una opción en un menú. Para esto, al canvas también se le agregaron dos objetos que tiene Unity preinstalados que son los objetos de botón. Estos botones pueden modificarse a gusto para que incluyan un texto, así como variar el color para los distintos estados: normal, resaltado cuando se sitúa el puntero del ratón sobre él, pulsado y seleccionado al dejar de pulsar. Así se indica visualmente al usuario la interacción con estos. Los dos botones creados son el botón 'Play' (que incluye el texto 'Iniciar simulador') para abrir la escena con el torno y el botón 'Quit' (con el texto 'Salir') para cerrar la aplicación.

5.13.2 Script para el menú principal

Se desarrolló un script con el mismo nombre que la escena, 'MainMenu', para manejar las interacciones de los botones del menú. Este script contiene métodos para cargar la escena del torno ('SampleScene') y para salir del juego.

El script se asignó al Canvas de la interfaz de usuario en la escena 'MainMenu'. Los botones 'Play' y 'Quit' se configuraron en el Inspector de Unity para llamar a los métodos que se encargan de su funcionamiento:

- **Método '**PlayGame'**:** Su función es cargar la escena con el índice '1' en el Build Settings de Unity al pulsar el botón de 'Play', que normalmente sería el primer nivel o escena del juego.
- **Método '**QuitGame'**:** Este método se utiliza para cerrar la aplicación al interactuar con el botón de salir. Cuando se ejecuta en el editor de Unity durante la configuración del programa, desactiva el modo de juego para volver al editor.

5.13.3 Retorno al menú principal

Se incluyó un modo para volver al menú principal desde la escena del torno. Para ello, se desarrolló un script llamado 'Escenas' para detectar cuando se pulse la tecla 'Escape' en el teclado del ordenador y volver a cargar la escena 'MainMenu'. En lugar de crear un gameObject vacío que lo contenga, este script se asignó a uno de los objetos de la decoración en la escena lo que permite ahorrar recursos y tener la jerarquía menos saturada.

Mientras se probaban los botones el cursor del ratón no estaba visible al retornar al menú principal desde la escena del torno. Para resolver esto, se aseguró que el cursor estuviera visible y desbloqueado en ambos scripts ('MainMenu' y 'Escenas') al cambiar de escena.

El script 'MainMenu' se configuró para que el cursor esté visible y desbloqueado cuando se carga el menú principal con el método 'Start'. Este método se llama automáticamente al iniciar la escena y se asegura de que el cursor esté visible y desbloqueado.

En el script 'Escenas', el método 'Cursor.visible = true', se activa al pulsar la tecla 'Escape' y se asegura que el cursor esté visible y desbloqueado antes de cambiar al menú principal.

Estos ajustes nos aseguran la visibilidad del cursor en ambos scripts para garantizar que se pueda interactuar correctamente con los elementos de la interfaz de usuario al cambiar de escena.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    void Start()
    {
        // Asegurarse de que el cursor esté visible
        Cursor.visible = true;
        Cursor.lockState = CursorLockMode.None;
    }

    // Método para cargar la escena del juego
    public void PlayGame()
    {
        SceneManager.LoadScene(1); // Cargar la escena con el índice 1 (SampleScene)
    }

    // Método para salir del juego
    public void QuitGame()
    {
        Application.Quit(); // Salir del juego
#if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false; // Solo para editor de Unity
#endif
    }
}
```

Figura 72. Script MainMenu

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Escenas : MonoBehaviour
{
    void Update()
    {
        // Detectar si se ha presionado la tecla "Escape"
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            // Asegurarse de que el cursor esté visible antes de cambiar de escena
            Cursor.visible = true;
            Cursor.lockState = CursorLockMode.None;
            SceneManager.LoadScene(0); // Cargar la escena con el índice 0 (MainMenu)
        }
    }
}
```

Figura 73. Script Escenas.

Figura 74. Escena del menú principal.

5.14 COMPILACIÓN DEL EJECUTABLE

Para finalizar el proyecto, se generó un ejecutable de la aplicación de Unity. El proceso para crear este ejecutable comenzó con la apertura de la configuración del 'Build' desde el menú principal de Unity, accediendo a 'File > Build Profiles'.

El orden de las escenas en Unity se usa para controlar el flujo de la aplicación. Cada escena en Unity puede representar un nivel de juego, una pantalla de menú o cualquier otra sección del proyecto. El primer paso para gestionar el orden es asegurarse de que todas las escenas necesarias estén añadidas a la lista dentro del menú 'Build Profiles'. Este menú permite organizar las escenas en el orden de carga. La primera escena de la lista será la que se cargue al ejecutar la aplicación. Este orden nos garantiza un flujo lógico y también mejora la experiencia del usuario.

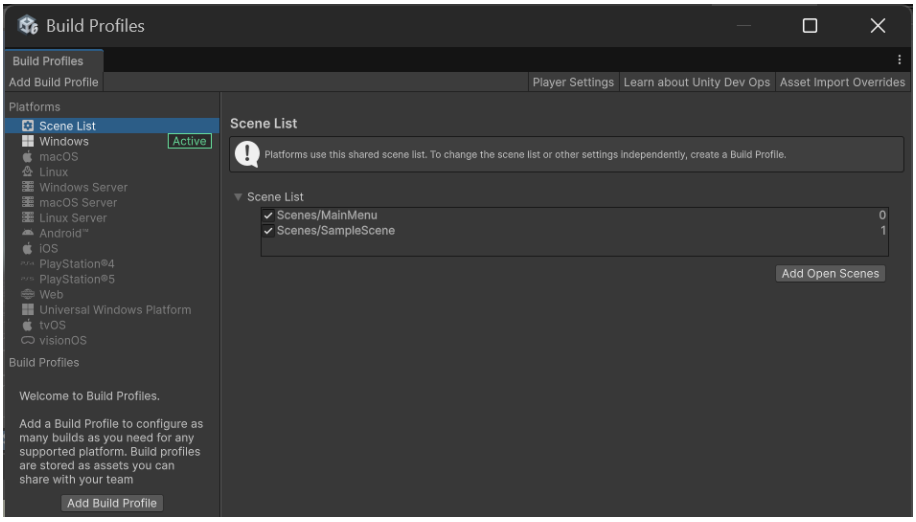


Figura 75. Lista de escenas en orden de prioridad.

A continuación, se seleccionó la plataforma deseada para el ejecutable en la sección 'Platforms', eligiendo en este caso Windows, que es el sistema operativo del que disponemos. Si se desea ejecutar la aplicación en otra plataforma, habría que dirigirse de nuevo a la lista de plataformas para instalar un módulo distinto y crear un nuevo ejecutable. Luego, se accedió a la ventana de configuración del jugador mediante el botón 'Player Settings', donde se puede configurar el nombre del producto, la versión, los iconos y la configuración de resolución.

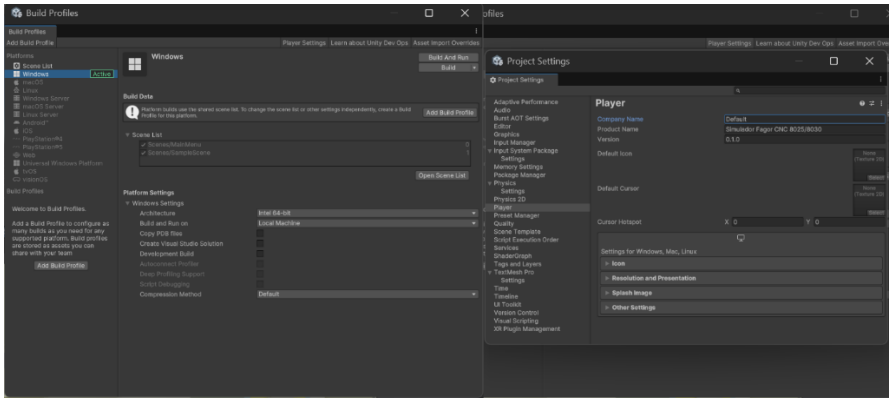


Figura 76. Selección de plataforma y configuración del 'Player Settings'.

Una vez configuradas las opciones, se procedió a hacer clic en 'Build And Run' dentro de la ventana 'Build Profiles'. Se eligió una carpeta de destino para guardar el ejecutable y Unity comenzó el proceso de compilación del proyecto, generando el archivo ejecutable en la ubicación seleccionada.

Finalmente, se ejecutó para comprobar que todas las funciones añadidas funcionaran como se esperaba y que todas las escenas necesarias estuvieran incluidas, garantizando el funcionamiento del ejecutable para obtener feedback de otros usuarios antes de su presentación.

5.15 EL EJECUTABLE POR DENTRO

Al crear el ejecutable de un proyecto en Unity, se generan varios archivos y carpetas esenciales para su funcionamiento:

La carpeta '**D3D12**' contiene archivos relacionados con 'Direct3D 12', una API gráfica (Interfaz de Programación de Aplicaciones Gráficas) utilizada para renderizar gráficos en el proyecto de Unity.

La carpeta '**MonoBleedingEdge**' alberga el runtime de Mono y las bibliotecas asociadas. Esto permite que los scripts escritos en C# funcionen en diferentes sistemas operativos.

La carpeta '**Simulador Fagor CNC 80258030_BurstDebugInformation_DoNotShip**' incluye información de depuración. El nombre "DoNotShip" indica que estos archivos son útiles durante el desarrollo para identificar problemas, pero no están destinados a ser incluidos en la versión final.

La carpeta '**Simulador Fagor CNC 80258030_Data**' contiene los archivos de datos del proyecto de Unity, como assets, escenas y otros recursos para que el ejecutable funcione. Sin estos archivos, el juego no tendría acceso a los recursos esenciales para su ejecución.

El archivo '**Simulador Fagor CNC 80258030.exe**' es el archivo principal del proyecto. Este es el archivo que se debe ejecutar para iniciar el juego. Al hacer doble clic en este archivo, se inicia la aplicación.

El archivo '**UnityCrashHandler64.exe**' se utiliza para generar informes de fallos. En caso de que el juego se cierre inesperadamente, este archivo recopila esa información que puede ser utilizada para corregir los problemas.

Por último, el archivo '**UnityPlayer.dll**' es una biblioteca de enlace dinámico (DLL) que contiene el runtime de Unity. Este archivo es necesario ya que proporciona el motor básico de juego.

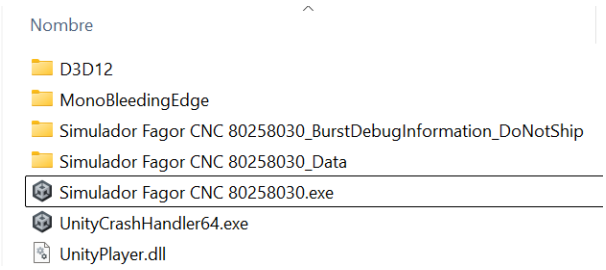


Figura 77. Archivos componentes del compilado del ejecutable.

Todos estos elementos son indispensables para que el juego funcione, permitiendo la renderizar los gráficos, la ejecución del código y gestionar errores. Sin todos estos archivos y carpetas, el ejecutable podría no operar de manera correcta y efectiva. Por lo tanto, no deben ser eliminados u olvidados si se desea enviar el ejecutable a otro dispositivo o usuario.

6 FUNCIONES IMPLEMENTADAS EN EL SIMULADOR

A continuación, se explica a modo de un manual de usuario sencillo, la lista completa de funciones implementadas en el torno virtual:

Menú principal y navegación

- Al abrir el programa se muestra el menú inicial, que incluye el título, y los botones para iniciar el simulador o salir.
- Al pulsar 'Iniciar el simulador' se carga la escena del torno, y cada vez que se inicia, el carro del torno se posiciona de forma aleatoria.
- Al pulsar 'Esc' en el teclado se regresa desde la escena del torno al menú inicial.
- Al seleccionar 'Salir' en el menú principal se cierra el programa

Figura 78. Iniciación de la escena del torno.

En la escena del torno

Estado inicial del simulador: Al comenzar (con el interruptor apagado), todas las luces y la pantalla del panel estarán apagadas, la puerta del torno permanecerá cerrada y el husillo se mantendrá inactivo.

La cámara se maneja con el ratón y las teclas W, A, S, D o las flechas del teclado.

La interacción con objetos y botones se realiza mediante el clic izquierdo del ratón.

Secuencia de encendido y apagado

- 1. Encender el Interruptor: Al activar el interruptor, ubicado en la parte derecha de la bancada, se ilumina el botón amarillo.

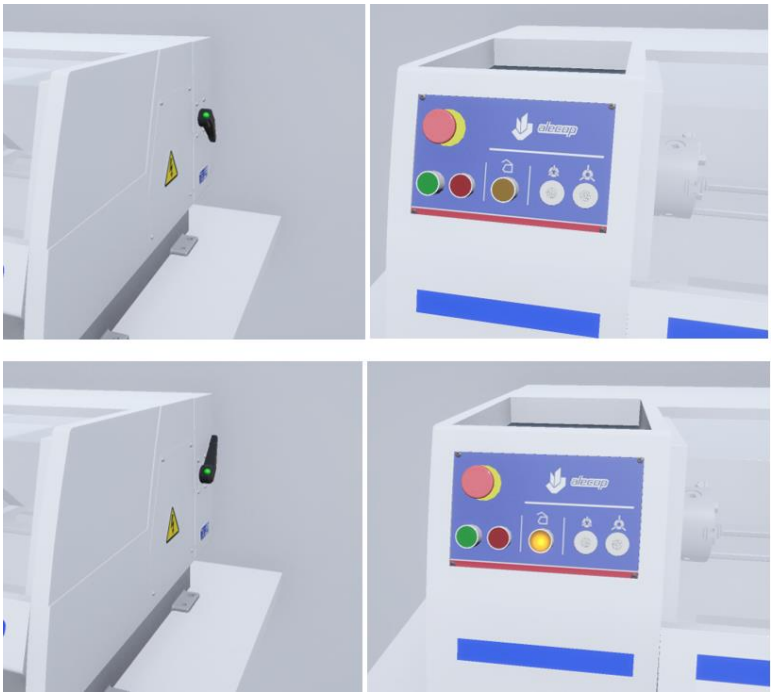


Figura 79. Secuencia de encendido del interruptor

- 2. Encender el Torno: Al pulsar el botón verde de la botonera de la bancada, se ilumina el botón verde y se activa la pantalla del panel.

- 3. Funciones con el Torno Encendido

Puerta: El botón amarillo permite abrir y cerrar la puerta, reflejándose el estado en la pantalla del panel.

Figura 80. Torno encendido y con la tapa abierta

Búsqueda del punto cero (con la tapa cerrada):

- La combinación 'X' + 'F1' + botón ‘VERDE’ del panel, busca el punto cero del eje X.
- La combinación 'Z' + 'F1' + botón ‘VERDE’ del panel, busca el punto cero del eje Z.

Giro del husillo (con la tapa cerrada):

- Al pulsar ‘Enter’ se inicia el giro del husillo.
- Los botones **'+'** y **'-'** controlan la velocidad.
- Los botones con flechas curvadas determinan el sentido de giro.
- La velocidad (1, 2 o 3) y el sentido (horario o antihorario) seleccionados se muestran en la pantalla del panel.
- Al pulsar la tecla ‘STOP’ del panel se detiene el giro.

Figura 81. Distintos modos de la pantalla según el estado del torno

- 4. Apagar el Torno: Al presionar el botón rojo de la bancada, se apaga la luz verde y la pantalla, así como cualquier función que estuviese activa, excepto la luz amarilla.

- 5. Apagar el interruptor: Al desactivar el interruptor, se apaga cualquier luz o función que estuviese activa.

Esta estructura proporciona una guía clara y ordenada para familiarizarse con todas las funciones del torno virtual.

7 CONCLUSIONES

7.1 OBJETIVOS ALCANZADOS

El resultado de este proyecto ha sido muy satisfactorio ya que se ha conseguido crear una aplicación de realidad virtual y lograr los objetivos planteados. Con Autodesk Inventor se ha recreado con detalle del torno Fagor CNC 8025 quedando un modelo muy similar al modelo real. Gracias a ello, los alumnos podrán reconocer visualmente la máquina y sus componentes. La programación en Unity 3D ha hecho posible funciones básicas de operación del torno. Acciones como el encendido, la apertura y cierre de la tapa o la búsqueda del cero máquina se han implementado de manera fiel al comportamiento real del torno. Además, la incorporación de los sonidos y texturas han mejorado la percepción del usuario, aumentando la sensación de presencia y el realismo de la escena.

Todos estos elementos han dado lugar a una aplicación funcional que facilita a alumnos y operarios un aprendizaje del manejo básico del torno en un entorno seguro antes de usar el equipo real.

7.2 LIMITACIONES Y EXPECTATIVAS

Como se ha comentado previamente, este trabajo es la primera fase de un proyecto más ambicioso del tutor de este TFG orientado a la simulación avanzada del torno Fagor CNC 8025. Si bien se han programado las funciones básicas, todavía se pueden ampliar muchas características en futuras versiones, como son la integración de operaciones de mecanizado con código G que permitirán una experiencia aún más completa.

Además, aunque el entorno desarrollado ofrece una simulación interactiva, el nivel de realismo podría aumentar significativamente si esta aplicación fuera exportada a un dispositivo de realidad virtual como Oculus Rift, HTC Vive o PlayStation VR. La incorporación de manos o controladores hápticos, así como el aislamiento sensorial que ofrecen estos dispositivos, permitiría una interacción más natural, proporcionando una experiencia aún más inmersiva y cercana a la realidad.

8 BIBLIOGRAFÍA

[1] REAL ACADEMIA ESPAÑOLA. *Diccionario de la lengua española*, 23.ª ed., [versión 23.8 en línea]. [Consulta: 2 de febrero de 2025]. Disponible en: https://dle.rae.es/realidad.

[2] G.R. Rocío. *Cómo funciona y en qué se usa la realidad virtual* [en línea]. Adslzone.net, 2024. [Consulta: 2 de febrero de 2025]. Disponible en: https://www.adslzone.net/reportajes/tecnologia/realidad-virtual-rv/.

[3] LINARES, Iván. *La realidad aumentada de Pokémon Go es tan buena que Niantic la comparte* [en línea]. El Español, 2018. [Consulta: 2 de febrero de 2025]. Disponible en: https://www.elespanol.com/elandroidelibre/noticias-y-novedades/20180628/realidad-aumentada-pokemon-go-buena-niantic-comparte/318469797_0.html.

[4] WIKIPEDIA. *Realidad virtual* [en línea]. Wikipedia.org, 2025. [Consulta: 2 de febrero de 2025]. Disponible en: Realidad virtual - Wikipedia, la enciclopedia libre

[5] WIKIPEDIA. *Whirlwind* [en línea]. Wikipedia.org, 2025. [Consulta: 2 de febrero de 2025]. Disponible en: https://es.wikipedia.org/wiki/Whirlwind

[6] NORMAN, Jeremy. *The Sensorama: One of the First Functioning Efforts in Virtual Reality* [en línea]. Historyofinformation.com, 2025. [Consulta: 2 de febrero de 2025]. Disponible en: https://www.historyofinformation.com/detail.php?id=2785

[7] SERVIN, Claudio. *Espada de Damocles* [en línea]. Proyectoidis.org, 2025. [Consulta en: 2 de febrero de 2025]. Disponible en: https://proyectoidis.org/espada-de-damocles/

[8] CAVALCANTI VIVEIROS, Lucas. *Aspen Movie Map* [en línea]. Researchgate.net, 2018. [Consulta en: 2 de febrero de 2025]. Disponible en: https://www.researchgate.net/figure/Aspen-Movie-Map-11_fig5_340144636

[9] FAMICOM WORLD. *3D System* [en línea]. Famicomworld.com, 2025. [Consulta en: 2 de febrero de 2025]. Disponible en: https://famicomworld.com/system/other/3d-system/

[10] AMOS, Evan. *Sega Masters Sys 3D Glasses* [en línea]. Wikipedia.org, 2011. [Consulta en: 2 de febrero de 2025]. Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Sega-Masters-Sys-3D-Glasses.jpg

[11] PASCUAL ESTAPÉ, Juan Antonio. *Descubren un juego perdido de realidad virtual de Megadrive, y ya puedes jugarlo con tus gafas de PC* [en línea]. Computer Hoy, 2020. [Consulta en: 2 de febrero de

2025]. Disponible en: https://computerhoy.20minutos.es/noticias/gaming/realidad-virtual-sega-vr-megadrive-nuclear-rush-760579

[12] SEGA. *SEGA VR-1 en Joypolis Yokohama* [en línea]. Facebook.com, 2014. [Consulta en: 2 de febrero de 2025]. Disponible en: https://www.facebook.com/SEGA/photos/this-is-the-vr-1-which-made-its-appearance-at-joypolis-yokohama-it-was-a-8-perso/10152482232191796/

[13] *Seeing Red: Analyzing the Pitfalls of the Virtual Boy* [en línea]. Megacatstudios.com, 2019. [Consulta en: 2 de febrero de 2025]. Disponible en: https://megacatstudios.com/blogs/gaming-news/seeing-red-analyzing-the-pitfalls-of-the-virtual-boy

[14] WIKIPEDIA. *Forte VFX1 Headgear* [en línea]. Wikipedia.org, 2012. [Consulta en: 2 de febrero de 2025]. Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Forte_VFX1_Headgear.jpg

[15] WIKIPEDIA. *Oculus Rift - Developer Version - Front* [EN LÍNEA]. Wikipedia.org, 2013. [Consulta en: 2 de febrero de 2025]. Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Oculus_Rift_-_Developer_Version_-_Front.jpg

[16] PLAYSTATION. *PlayStation VR* [en línea]. Playstation.com, 2025. [Consulta en: 2 de febrero de 2025]. Disponible en: https://www.playstation.com/es-es/ps-vr/

[17] HERNÁNDEZ CASTILLO, Álvaro. *HTC saca a la venta su paquete de realidad virtual* [en línea]. Elpais.com, 2016. [Consultado en: 2 de febrero de 2025]. Disponible en: https://elpais.com/eventos/2016/02/21/mwc/1456087165_739120.html

[18] AMAZON. *HTC VIVE Pro 2 Full Kit - PC Virtual Reality System* [en línea]. Amazon.com, 2025. [Consultado en: 2 de febrero de 2025]. Disponible en: https://www.amazon.es/Realidad-VIRUTAL-VIVE-Garantia-DOMESTICA/dp/B09H3BMZH3

[19] WIKIPEDIA. *Unity Technologies* [en línea]. Wikipedia.org, 2025. [Consultado en: 2 de febrero de 2025]. Disponible en: https://es.wikipedia.org/wiki/Unity_Technologies

[20] WIKIPEDIA. *Unreal Engine* [en línea]. Wikipedia.org, 2025. [Consultado en: 2 de febrero de 2025]. Disponible en: https://es.wikipedia.org/wiki/Unreal_Engine

[21] IMASCONO. *Realidad Virtual y medicina: aplicaciones en el sector sanitario* [en línea]. Imascono.com, 2025. [Consultado en: 2 de febrero de 2025]. Disponible en: https://imascono.com/realidad-virtual-y-medicina/

[22] FAGOR. *Fagor CNC 8025/8030 modelos T, TG, TS. Manual de operación* [en línea]. Etitudela.com, 1997. [Consultado en: 1 de febrero de 2025]. Disponible en: CNC 8025T USER

[23] AUTODESK. *Inventor Professional. Planes de suscripción* [en línea]. Autodesk.com, 2025. [Consultado en: 3 de febrero de 2025]. Disponible en: Acceso de estudiantes de Autodesk a descargas de educación

[24] AUTODESK. *Funciones clave de Inventor* [en línea]. Autodesk.com, 2025. [Consultado en: 3 de febrero de 2025]. Disponible en: Funciones

[25] BLENDER. *Blender Foundation* [en línea]. Blender.org, 2025. [Consultado en: 6 de febrero de 2025]. Disponible en: Blender Foundation — blender.org

[26] BLENDER. *Blender 4.3 Reference Manual* [en línea]. Blender.org, 2025. [Consultado en: 6 de febrero de 2025]. Disponible en: Blender 4.3 Manual

[27] BLENDER. *Blender batch import wavefront obj* [en línea]. Blender.org, 2025. [Consultado en: 6 de febrero de 2025]. Disponible en: GitHub - p2or/blender-batch-import-wavefront-obj: Import multiple OBJ files, their UV's and materials

[28] BLENDER. *Wavefront OBJ* [en línea]. Blender.org, 2025. [Consultado en: 6 de febrero de 2025]. Disponible en: Wavefront OBJ — Blender Manual

[29] CASTAÑEDA, Francisco. *Son usados para el diseño 3D, conoce a fondo los archivos FBX* [en línea]. Softzone.es, 2024 [Consultado en: 9 de febrero de 2025]. Disponible en: https://www.softzone.es/windows/como-se-hace/archivos-fbx/

[30] WIKIPEDIA. *Unity (motor de videojuego)* [en línea]. Wikipedia.org, 2025. [Consultado en: 6 de febrero de 2025]. Disponible en: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[31]. UNITY TECHNOLOGIES. *Unity.* [en línea]. Unity.com/es, 2025. [Consultado en: 6 de febrero de 2025]. Disponible en: Plataforma de desarrollo en tiempo real Unity | Motor 3D, 2D, VR y AR

[32] FILEFORMAT. *¿Qué es un archivo MTL?* [en línea]. Fileformat.com, 2025. [Consultado en: 9 de febrero de 2025]. Disponible en: Formato de archivo MTL: archivo de biblioteca de plantillas de materiales OBJ

ANEXO

PIEZAS CREADAS POR CAD EN AUTODESK INVENTOR

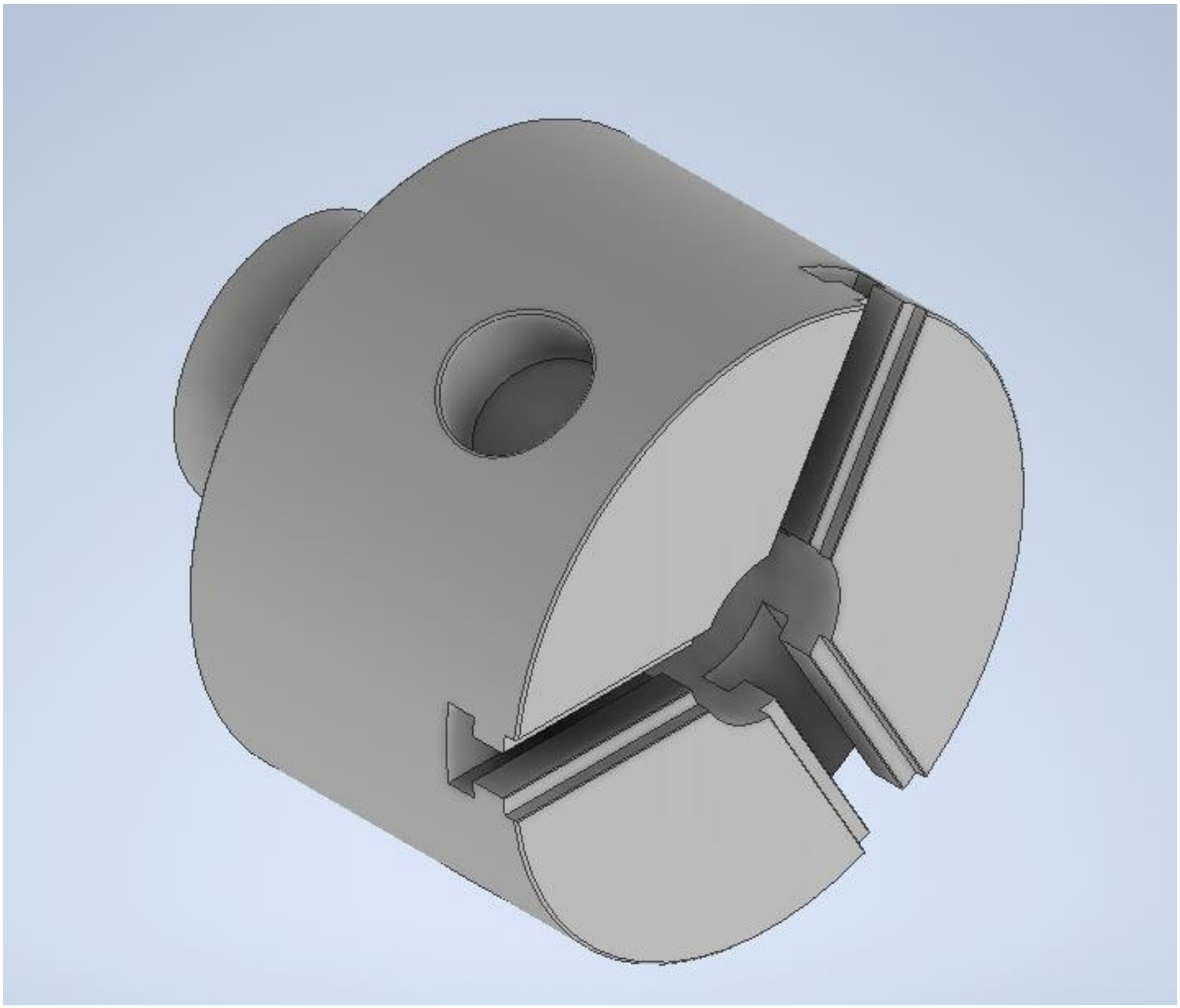


Figura 82. Eje central del husillo.

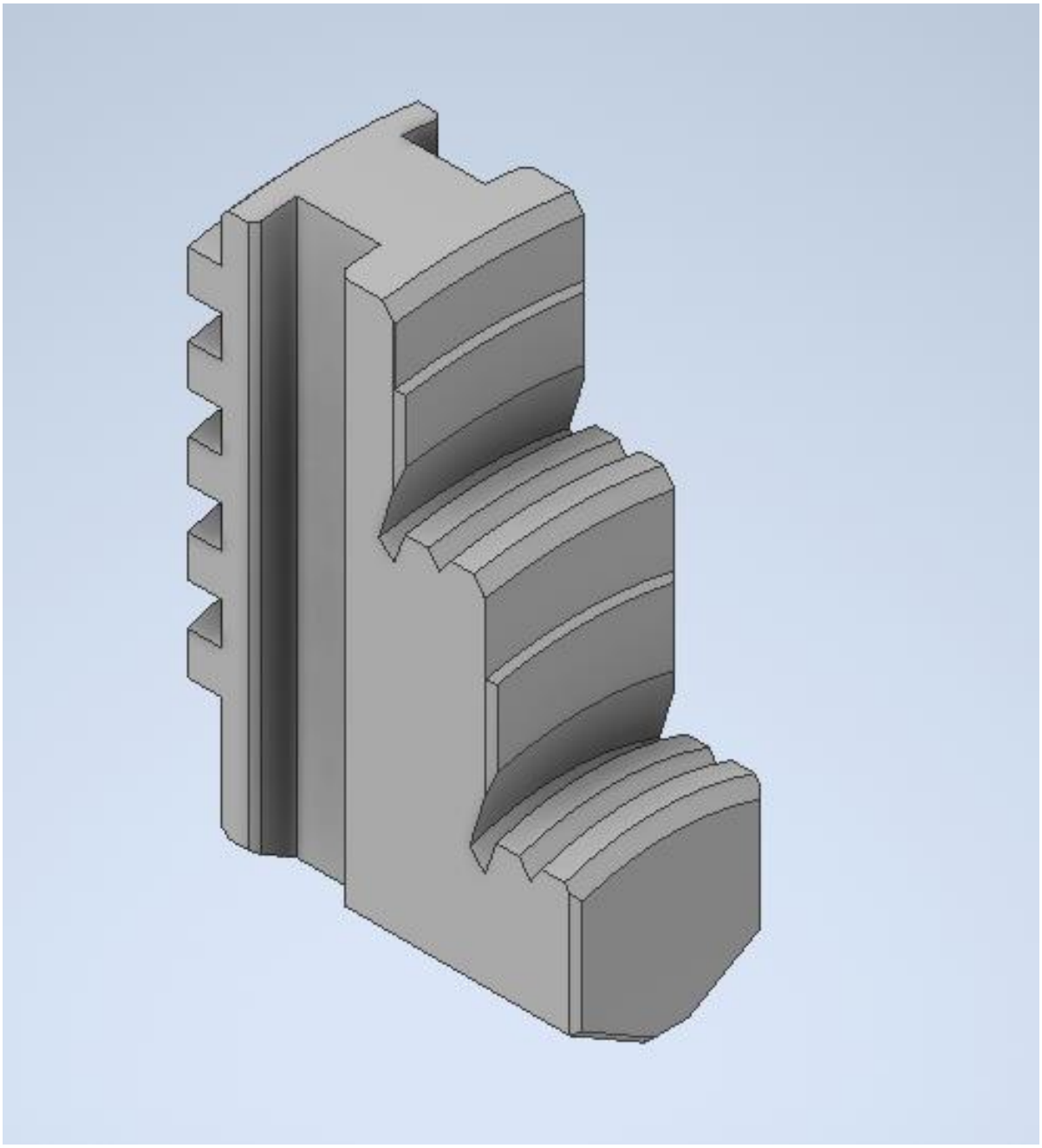


Figura 83. Garra de las pinzas triples

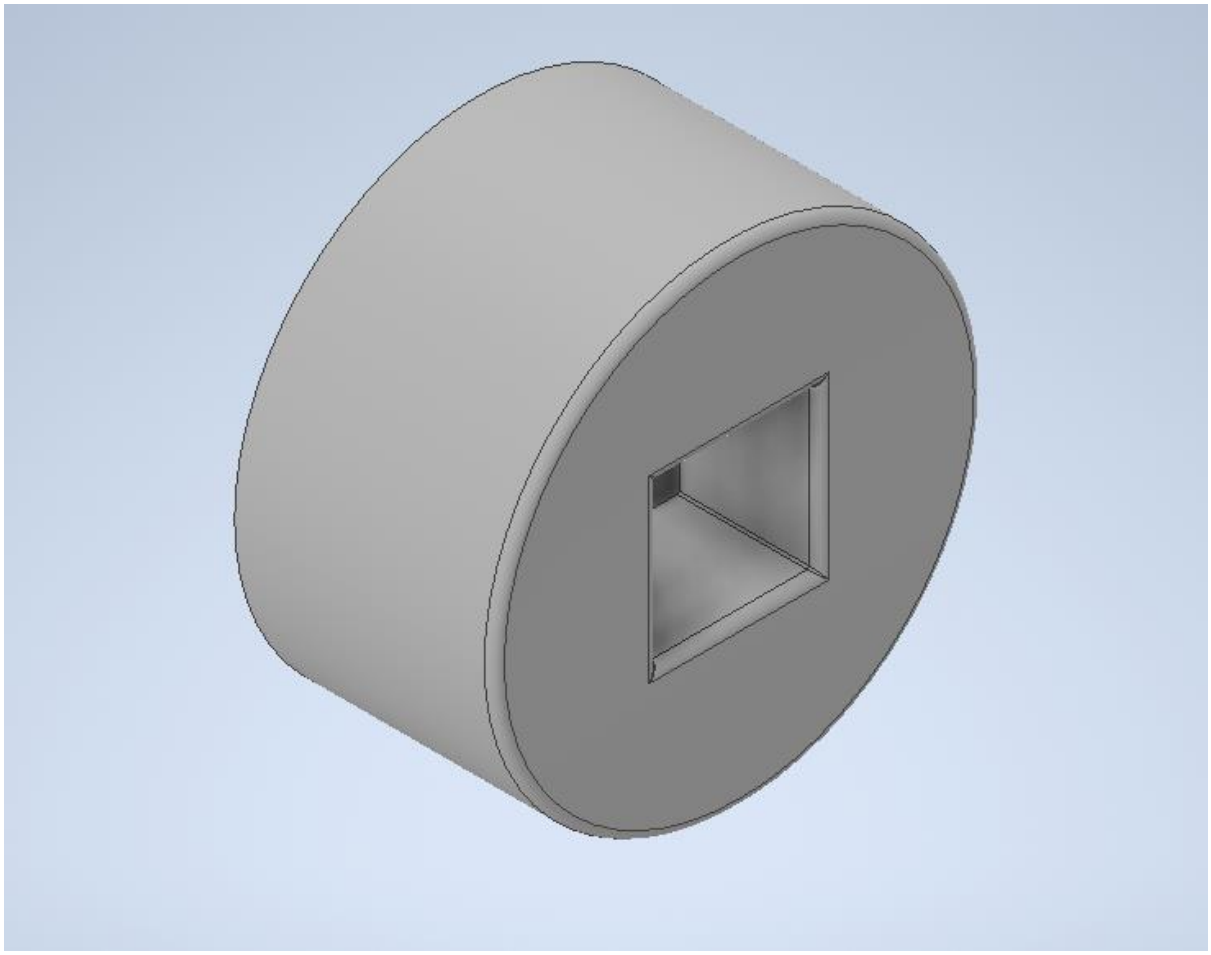


Figura 84. Pieza para el ajuste de las pinzas triples

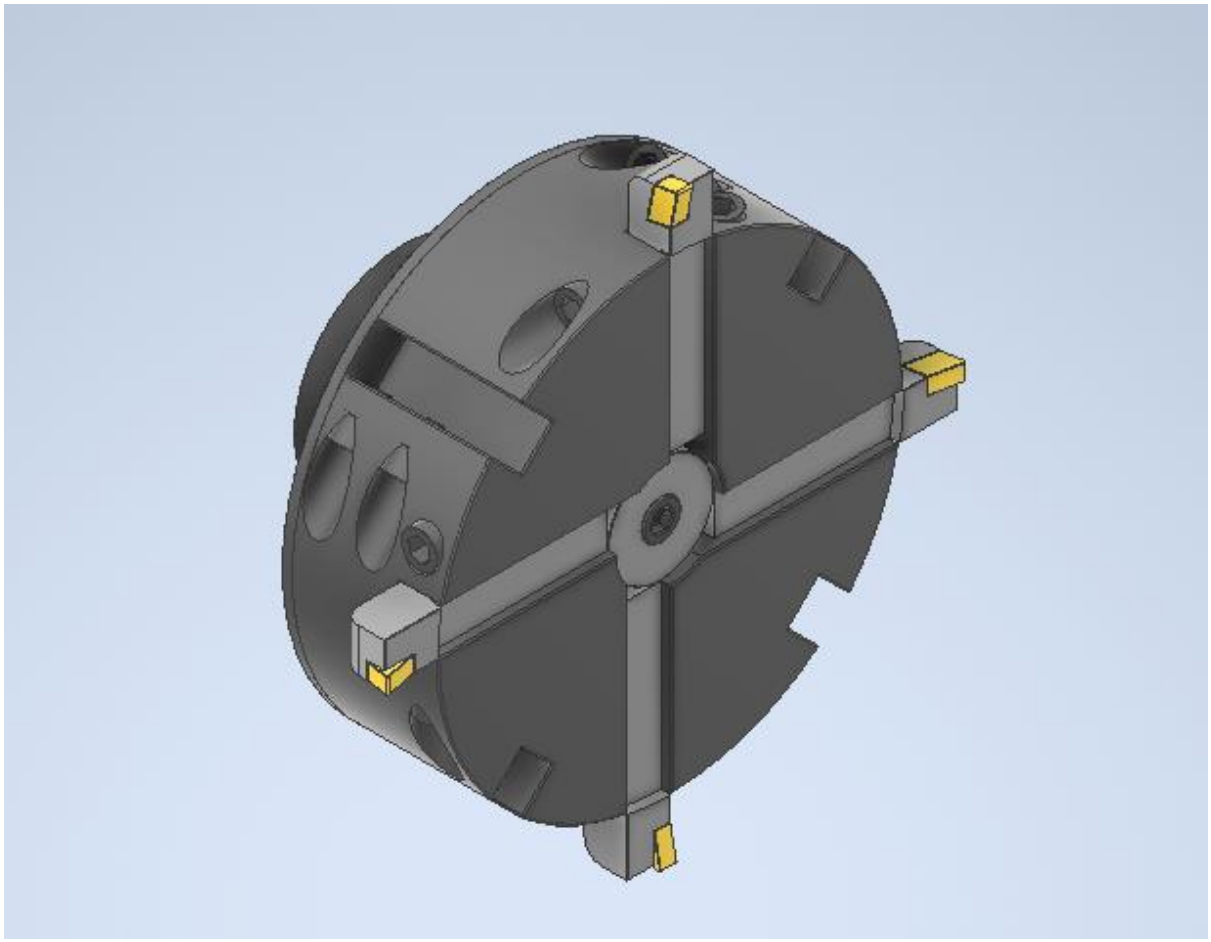


Figura 85Portaherramientas y herramientas de corte

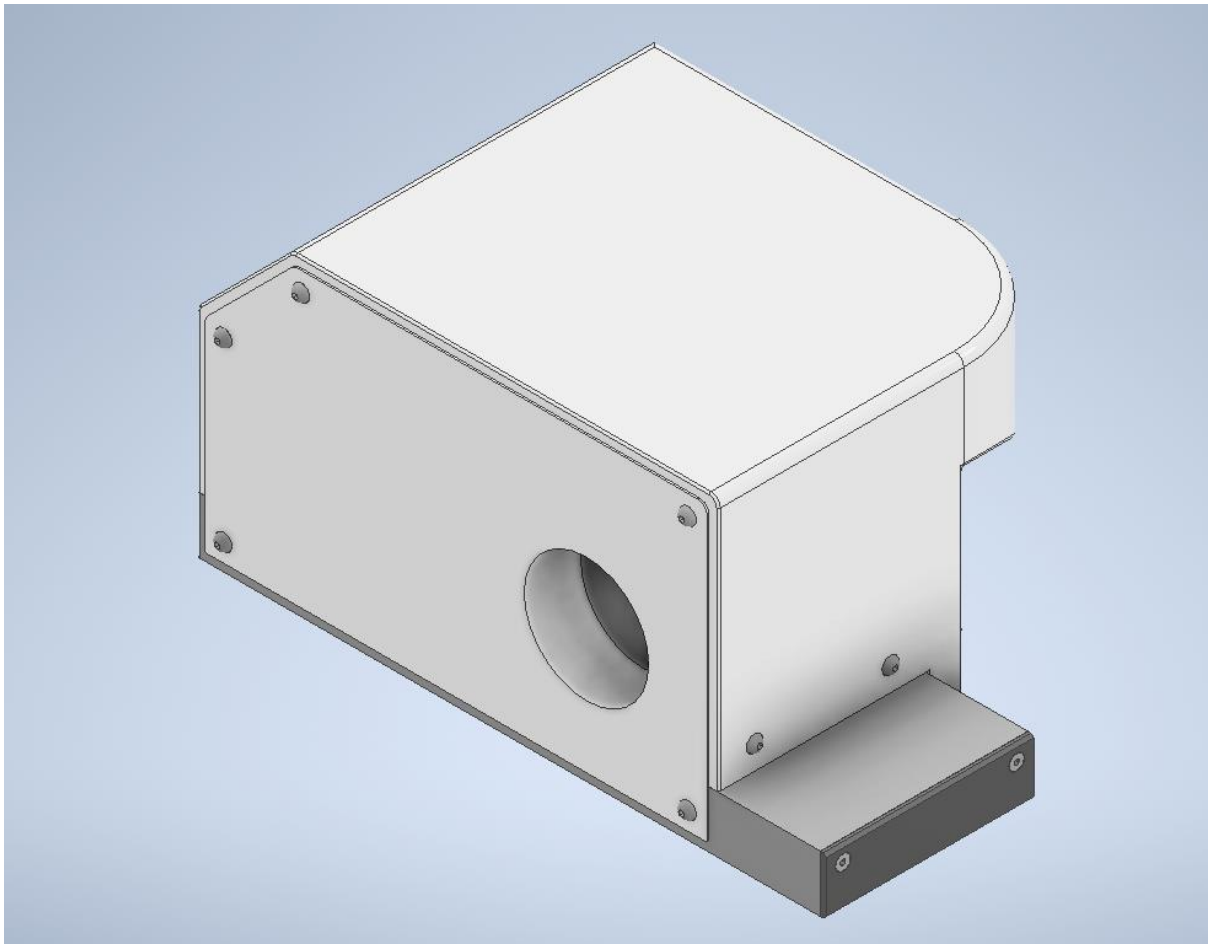


Figura 86. Carro transversal

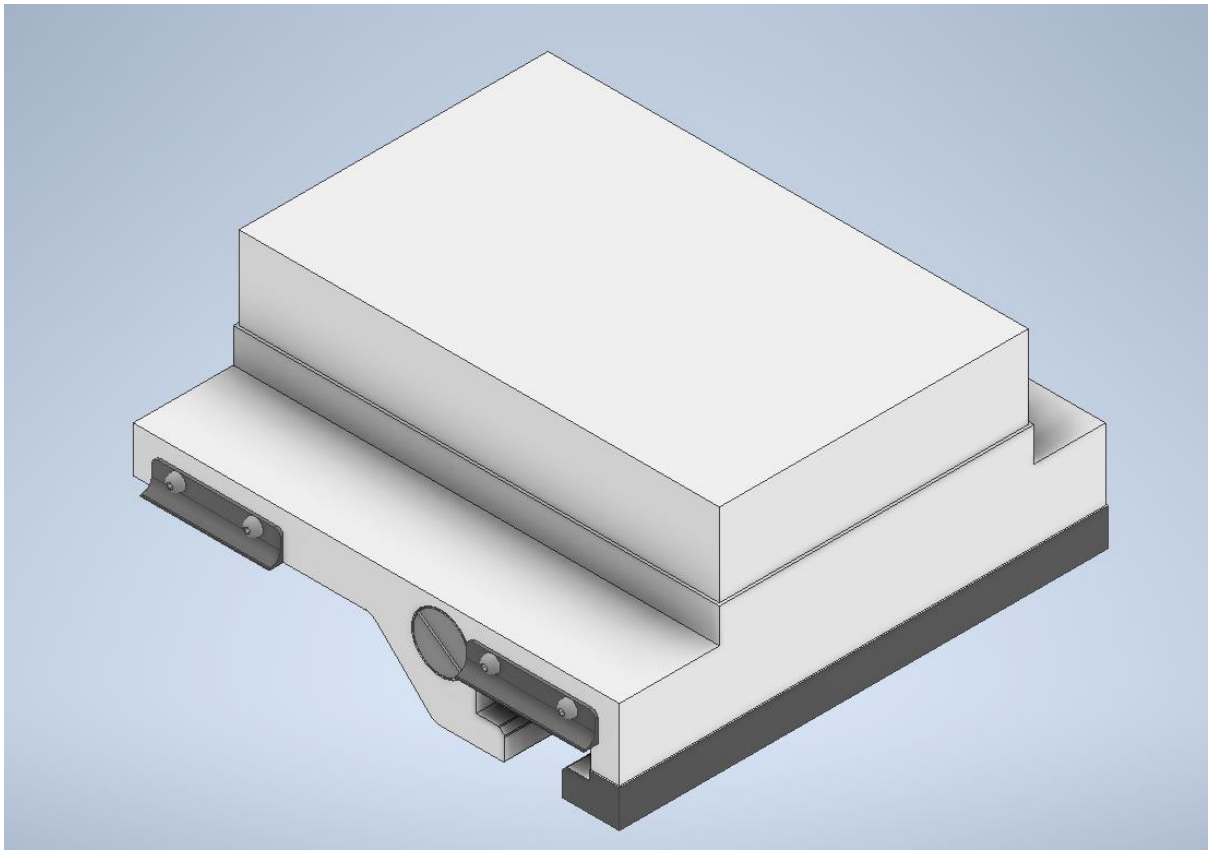


Figura 87. Carro longitudinal

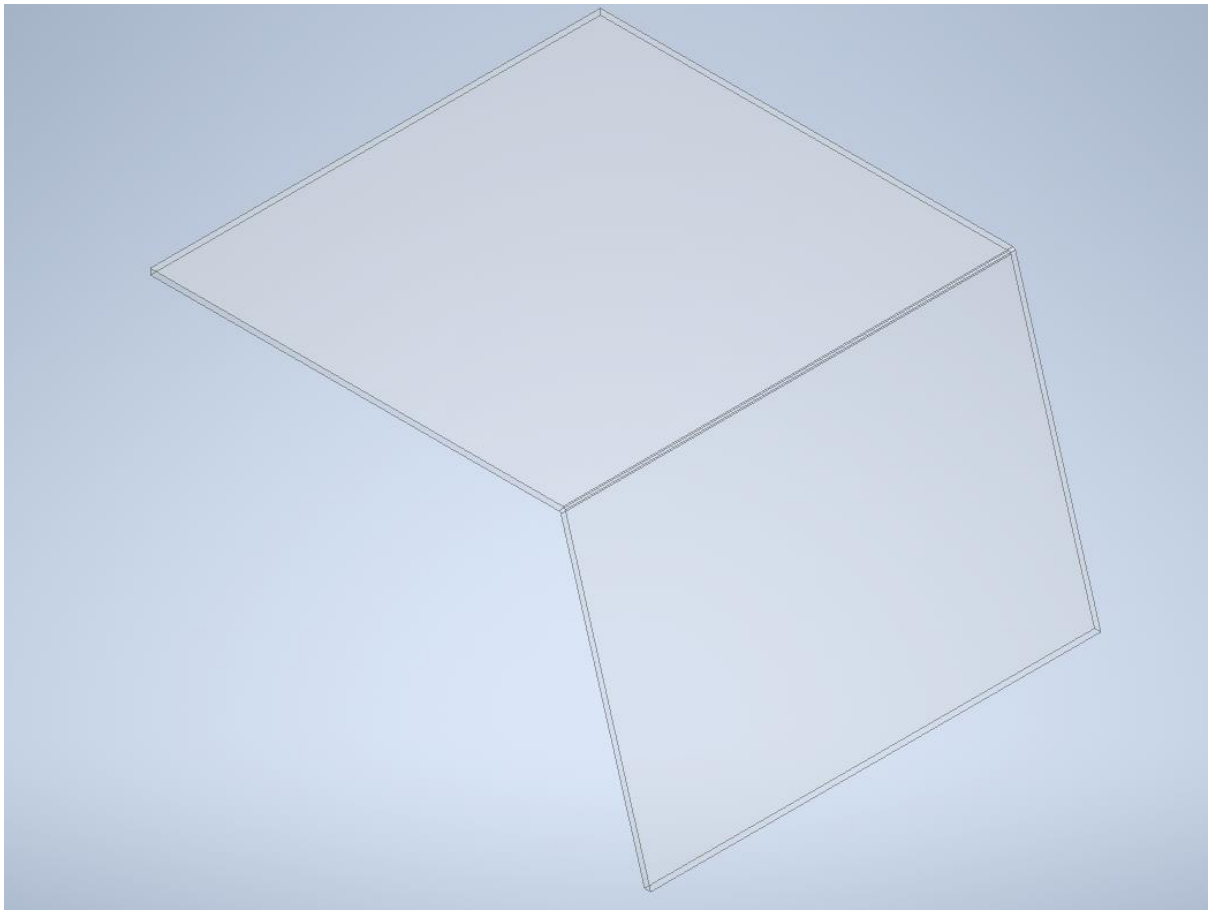


Figura 88. Tapa fija

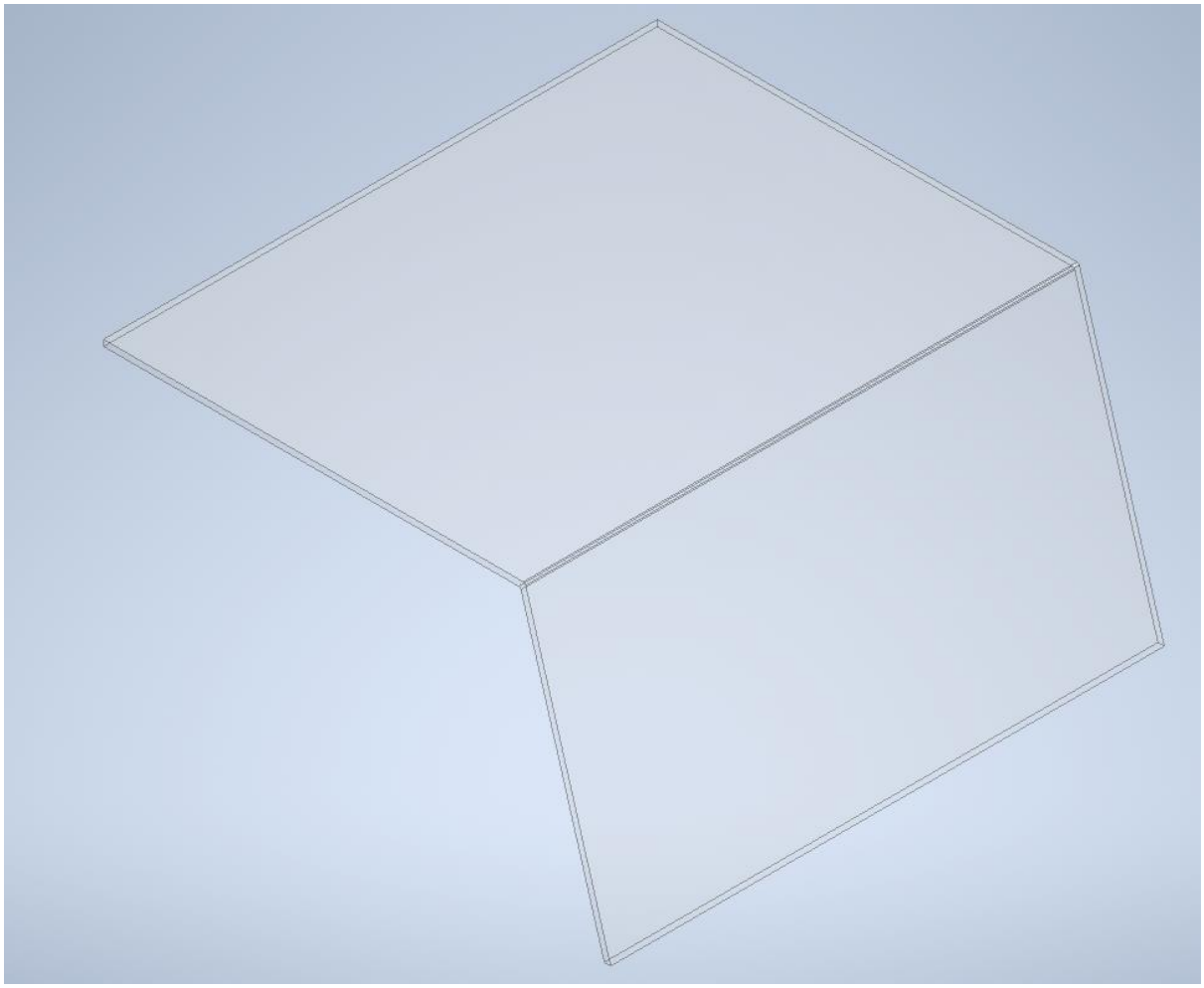


Figura 89. Tapa móvil

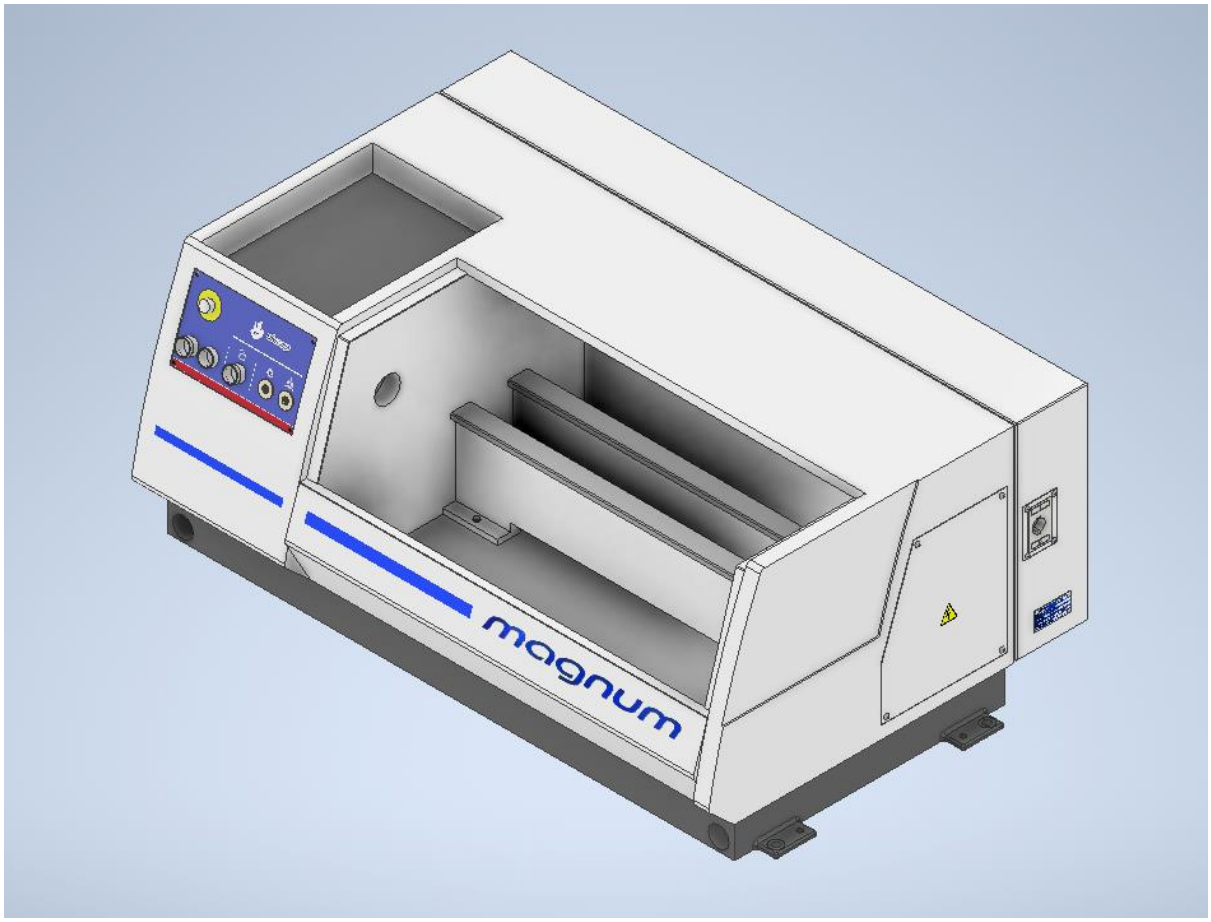


Figura 90. Bancada

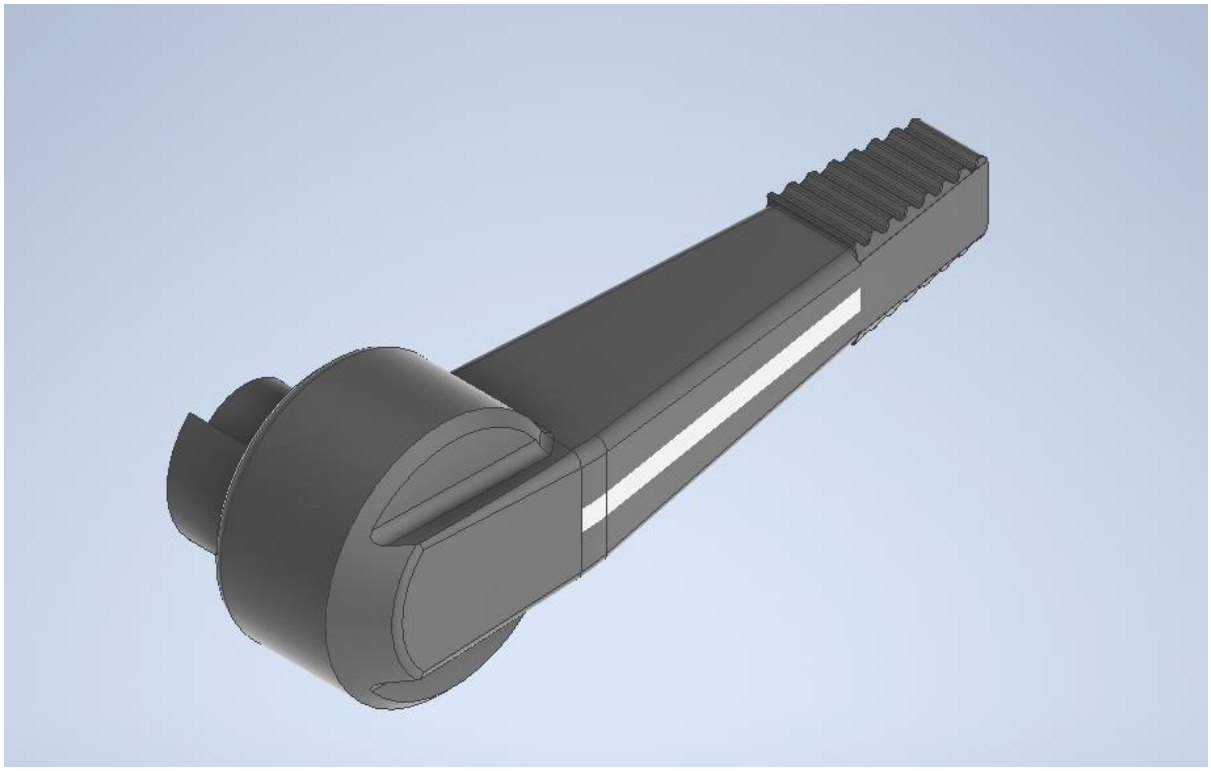


Figura 91. Interruptor

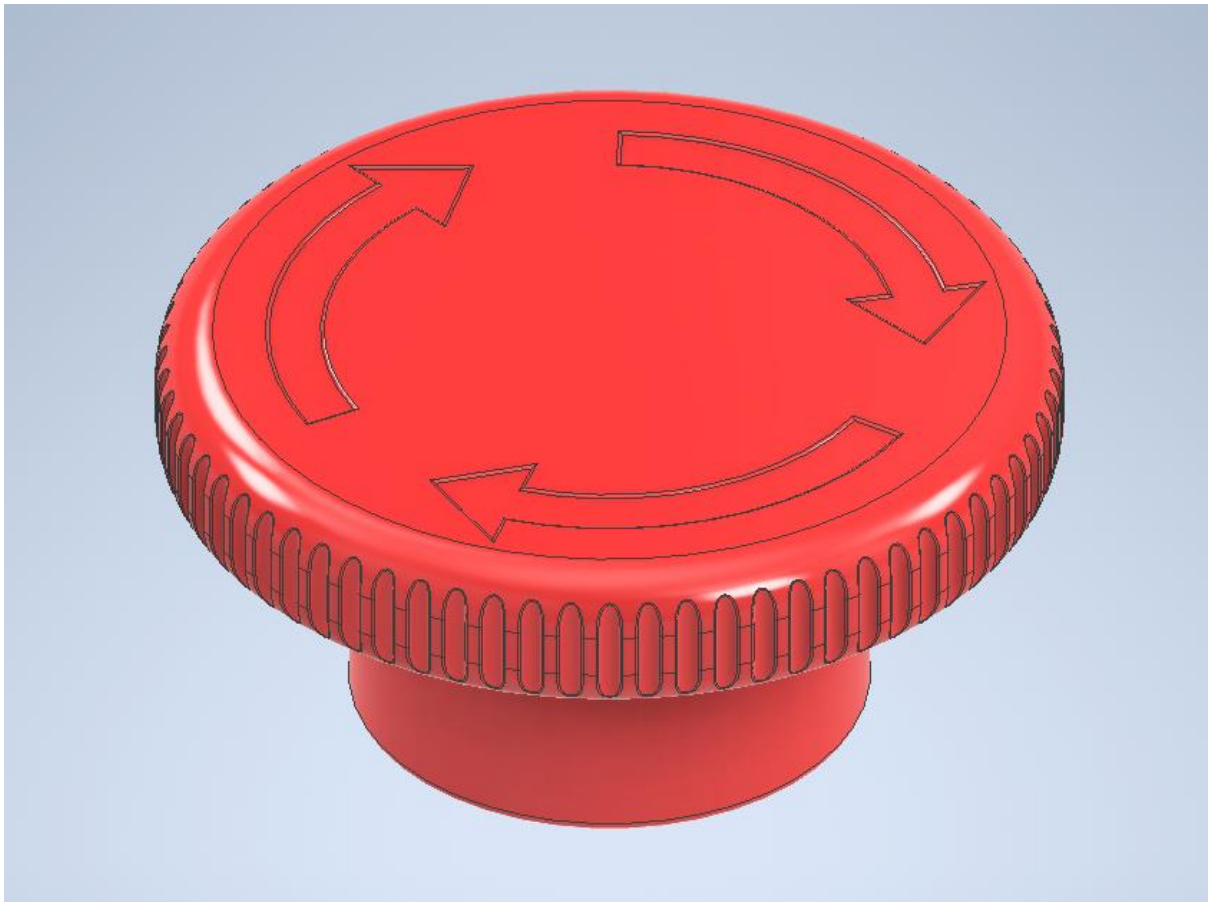


Figura 92. Seta de seguridad.

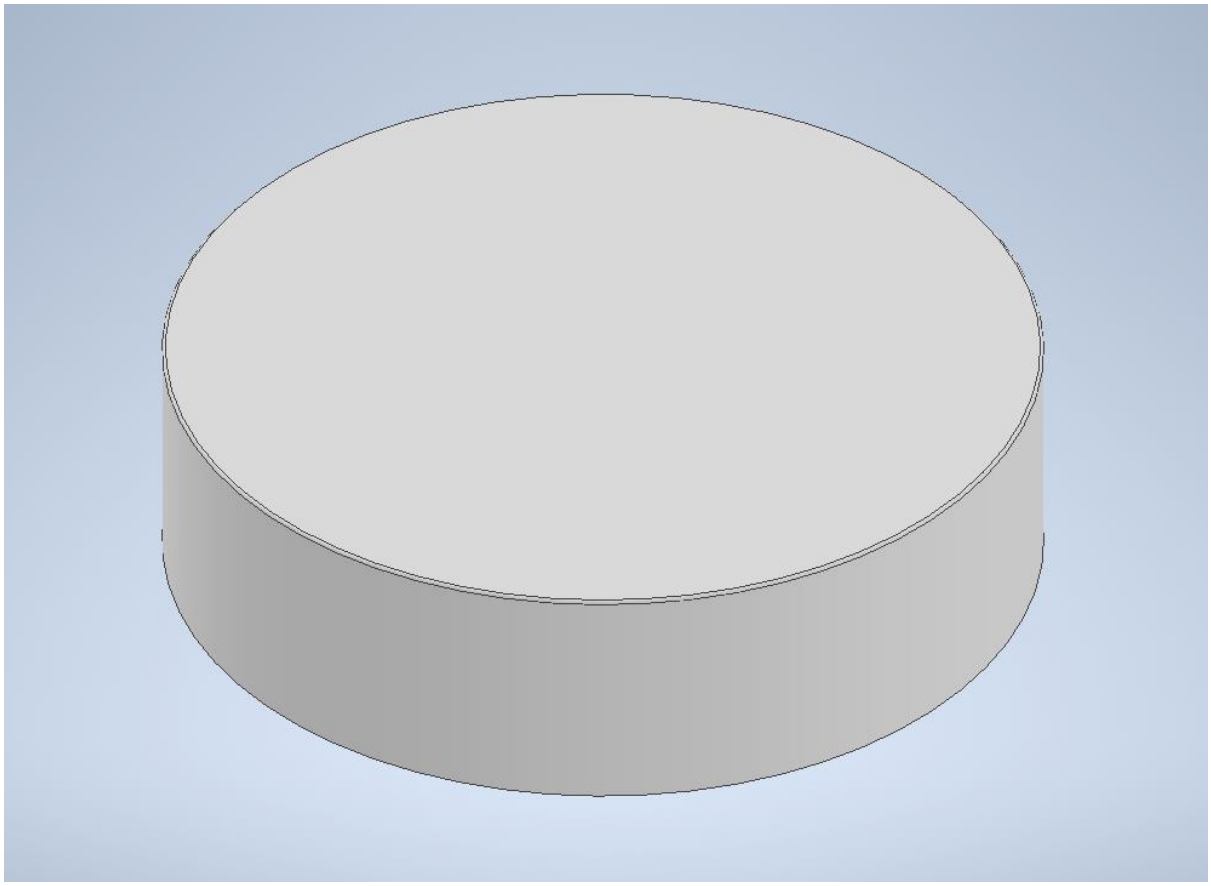


Figura 93. Pieza general para los botones

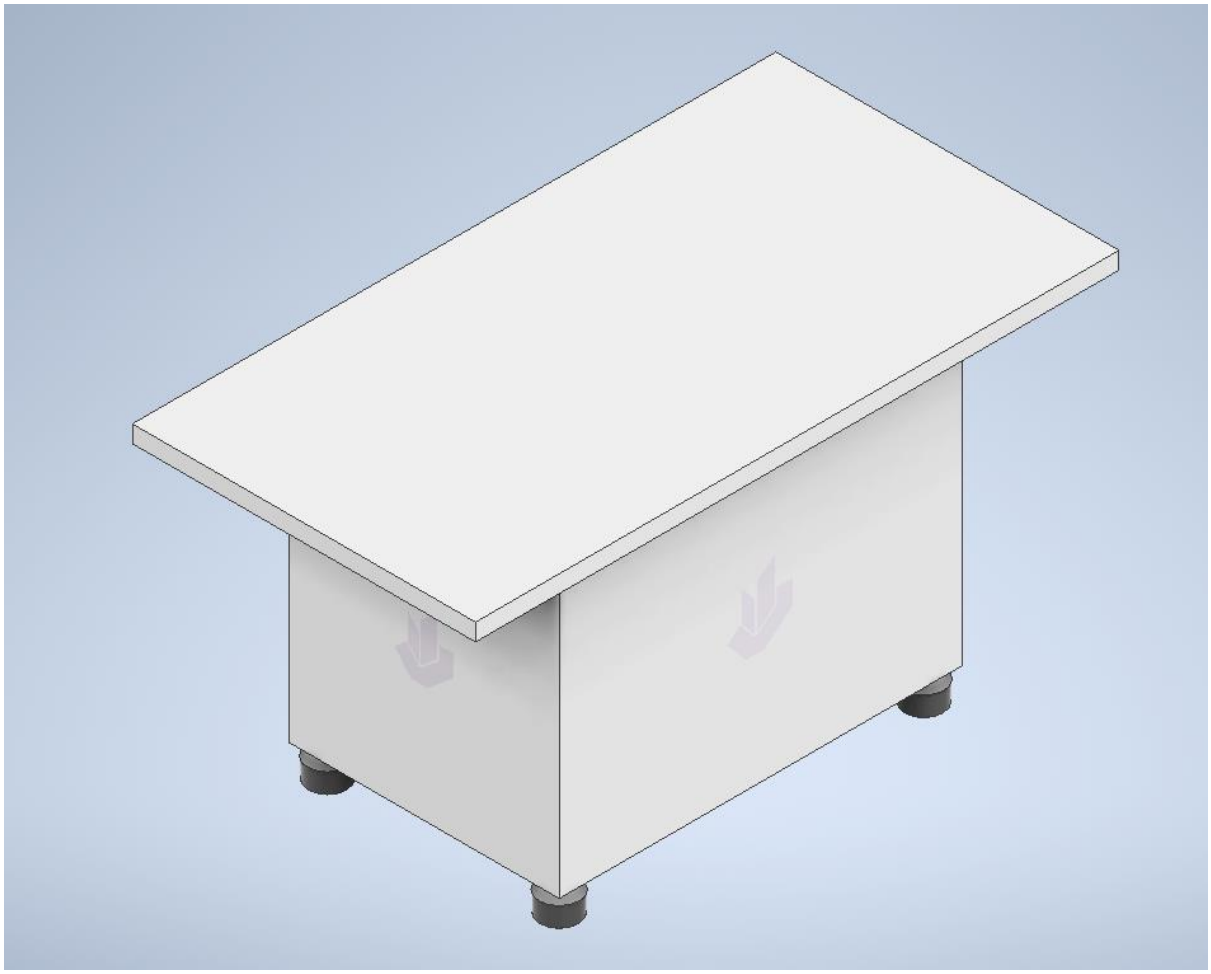


Figura 94. Mesa soporte

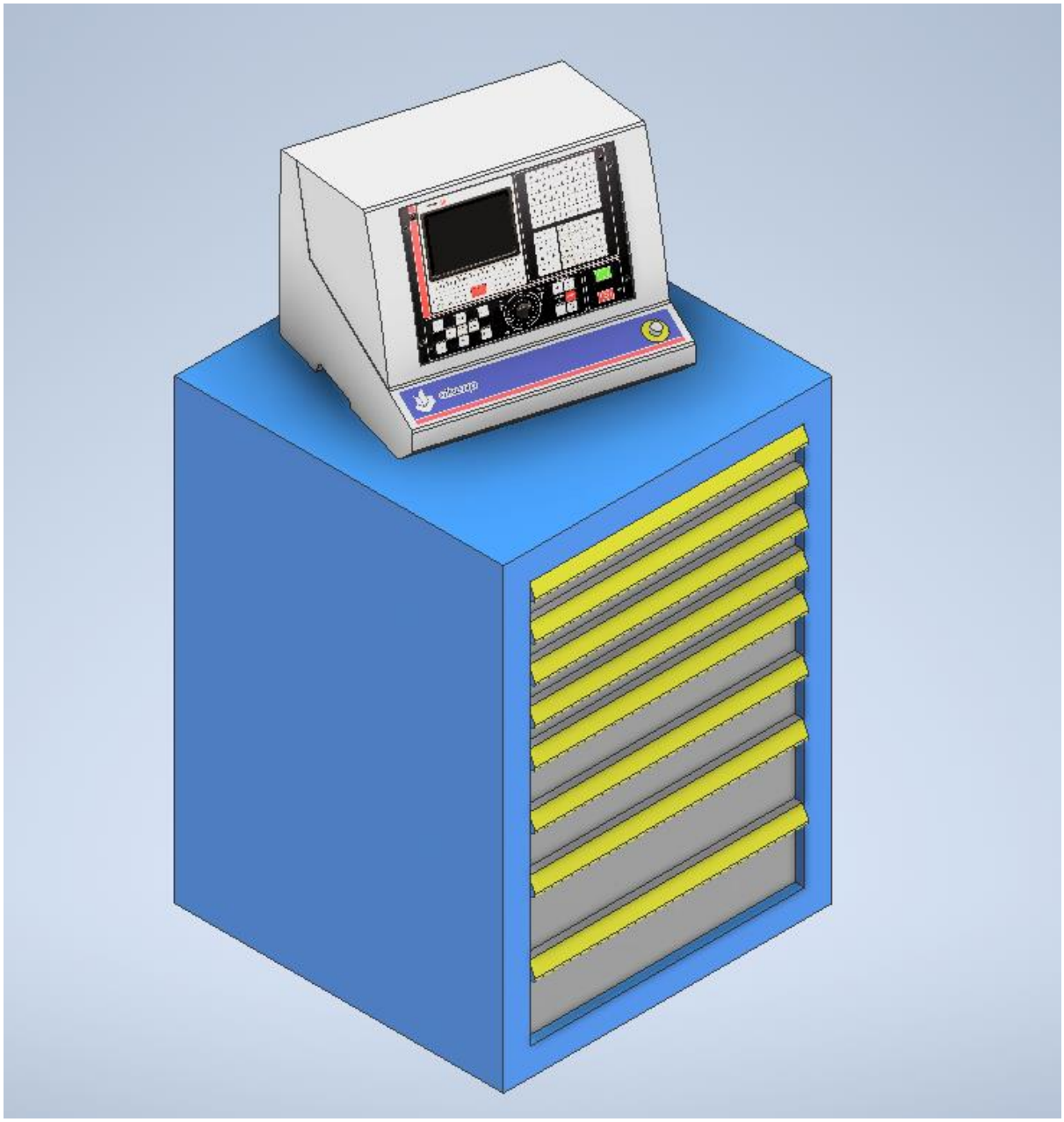


Figura 95. Panel de control y armario de herramientas