Simulación de un sistema de manufactura en lazo cerrado

Jesús Ernesto Díaz Díaz

Victor Hugo Fernández Muñoz

Trabajo de grado en Automática Industrial.

Director:
PhD. Mariela Muñoz Añasco

Universidad del Cauca
Facultad de Ingeniería en Electrónica y Telecomunicaciones
Departamento de Electrónica Instrumentación y Control
Popayán, mayo 2018

Jesús Ernesto Díaz Díaz
Víctor Hugo Fernández Muñoz

Simulación de un sistema de manufactura en lazo cerrado

Trabajo de grado presentado a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para la obtención del
Título de

Ingeniero en Automática.

Director:
Mariela Muñoz Añasco, PhD.

Popayán
2018

< En este espacio está el acta de sustentación firmada por los jurados en el momento de aceptar el trabajo, que está en el archivo "*PA-GA-4.2-FOR-13 Acta para sustentación pública de trabajo de grado.docx" >

Agradecimientos

Los autores quisieran agradecer a la Universidad del Cauca y a la PhD. Mariela Muñoz Añasco por su guía, asesoría y consejo en el desarrollo de este trabajo de grado.

Resumen

En la actualidad, la diversidad de procesos a nivel de ingeniería es cada vez mayor, los diseñadores de sistemas de manufactura pretenden evolucionar sus procesos para hacerlos más eficientes y funcionales, con el fin de abastecer la demanda de sus productos y reducir costos de proceso, esto ha ocasionado que los sistemas sean más complejos y, por lo tanto, más difíciles de diseñar. Un buen diseño del sistema permite que los procesos alcancen la eficiencia deseada, por tal motivo, muchas investigaciones se han dado en el área del diseño, sin embargo, la validación de estos diseños es el reto que tienen los autores ya que en ella se comprueba la veracidad que tiene el modelo con el sistema real.

La simulación es una técnica apropiada para la validación del modelado y muchos softwares han sido creados para llevar a cabo la representación gráfica y dinámica de los modelos, pero, aunque la simulación tenga como objetivo presentar una idea animada del proceso es necesario también evaluar los resultados de esa representación visual con señales simuladas con el fin de verificar la respuesta del sistema ante eventualidades que se puedan presentar en el sistema real.

Por lo anterior, el enfoque investigativo del presente trabajo es realizar una evaluación de un método de diseño aplicado a un proceso real, a través de un software simulación apropiado que simule dinámica en la parte de la planta, aplicando un controlador para automatizar el proceso y, además, obtener del sistema simulado las señales de proceso para verificar el funcionamiento correcto y la respuesta de la planta ante señales de control.

La estructura de este trabajo inicia con una revisión bibliográfica de las investigaciones enfocadas al diseño y simulación de sistemas de manufactura, una implementación del método de diseño al caso de estudio, la creación del entorno grafico en el software de simulación seleccionado y, por último, el análisis de resultados de simulación.

Palabras claves: Diseño, simulación, sistemas de manufactura, sistemas de eventos discretos.

Contenido

Lista de Figuras

Lista de Gráficas

Lista de Tablas

Lista de Anexos

Introducción

Los sistemas de manufactura a través de los años, se han ido desarrollando y optimizando en todos los campos de trabajo, bien sea en la parte administrativa gerencial o al nivel de la planta donde se lleva a cabo el proceso de transformación de la materia prima [1]. Debido a la alta demanda de productos y a la innovación continua de los procesos, los sistemas de manufactura han aumentado su complejidad ya que esto requiere que los procesos de fabricación sean altamente flexibles [2]; para esto, los sistemas de manufactura han fomentado la realimentación de señales de proceso para visualizar, monitorear y controlar dichos sistemas [3], lo que ha ocasionado que los sistemas sean más complejos de diseñar y simular, debido a las dinámicas del proceso y a las grandes cantidades de datos que se deben tratar.

La industria de la manufactura para mejorar la eficiencia de los procesos de fabricación, debe hacer un diseño óptimo; este le permitirá tener un excelente rendimiento general de los sistemas de manufactura; para ello algunos autores [4];[5];[6] optan por técnicas de modelado y simulación que permitan evaluar y analizar un proceso en detalle y a la vez tomar decisiones acertadas frente a él. Los sistemas de fabricación se han apropiado de la simulación como una herramienta potencial de observación, control y verificación de sus procesos. A pesar de su alto costo, y complejidad al momento de realizar todas las dinámicas de un sistema, se ha presentado un crecimiento de implementación en los últimos años, especialmente en sistemas complejos.

La simulación para sus procesos, se ha utilizado para resolver diferentes problemas, en [7] utilizan un modelo de simulación para identificar los cuellos de botella de una planta de fusión de acero, en [8] presentan un método basado en la simulación para desarrollar un diseño de distribución de la instalación eficiente, en condiciones de incertidumbre, en la demanda del producto, en [9] desarrollan varios diseños de una línea de producción de alimentos y usan la simulación para comparar los diseños propuestos con el de la línea actual, en [10] proponen un enfoque de optimización de simulación basado en el algoritmo genético (GA) y la simulación para resolver el problema de distribución de la instalación, en [11] introducen el formalismo DEVS paralelo (especificación de sistemas de eventos discretos) que permite la especificación modular y jerárquica de los sistemas de eventos discretos.

A través de los años se ha ido presentando una tendencia respecto al modelado y simulación de sistemas de manufactura. [2], a partir de una revision de 281 documentos, muestra que una de las tècnicas más utilizadas para analizar y comprender los procesos de manufactura es la simulacion de eventos discretos (DES), ya que es una herramineta altamente flexible, la cual, permite evaluar diferentes alternativas de configuraciones de sistemas y estrategias operativas, con

el fin de apoyar la toma de decisiones en el contexto de la fabricaciòn. Los DES han sido aplicados en una gran variedad de industrias para una amplia gama de aplicaciones de gestión operativa incluyendo la programación, planificación de la producción, ingeniería de procesos, control y gestión de inventario, entre otros, lo que implica su adaptabilidad en la industria, siendo apropiado para los niveles de toma de decisiones tácticas y operativas [12].

El objetivo de diseñar un modelo representativo de un proceso, es predecir los diferentes acontecimientos que se pueden presentar a través del mismo, así como la reducción de costos de fabricación y la flexibilidad del sistema. Es por esta razón que los modelos deben ser lo más aproximados a la realidad, para luego utilizar ese modelo y hacer una representación del sistema real [13]; [14].

El tema general de los DES, la simulación y el uso de diferentes softwares de simulación es abordado por varios autores. En [15], mostraron que la simulación de eventos discretos es el método más adaptable para evaluar los sistemas de fabricación o de elegir una organización de gestión de la producción, en [8] hacen un diseño conceptual basado en eventos discretos como ayuda para el desarrollo de una planta virtual, para la cual proponen un manual de integración para la simulación y el diseño, en [6] utilizan la simulación de eventos discretos para diseñar los sistemas de producción tales como los (FMS) en particular para diseñar y dimensionar el tamaño del hardware de un sistema de manufactura flexible (FMS), en [16] incluyen una revisión de aplicaciones discretas de simulación de eventos en la programación de (FMS).

La simulación de eventos discretos aparentemente es una técnica idónea para representar los sistemas de manufactura. Por tal motivo el enfoque del presente trabajo será realizar un diseño general de un proceso de manufactura en lazo cerrado (realimentación de señales), el cual se pueda simular con una herramienta software gráfico, que valide la propuesta de diseño.

Capítulo 1

Conceptos y definiciones básicas

1.1. Sistemas de manufactura en lazo cerrado

Dentro de los sistemas de manufactura, varios autores concuerdan en que la globalización ha afectado considerablemente el sector manufacturero de la ingeniería [17]. Debido a la demanda de nuevos productos, la alta calidad y un mercado en constante cambio, los sistemas de manufactura han ido evolucionando y cambiando sus métodos de operación y diseño [18].

Se han propuesto varios enfoques de sistemas de manufactura, por ejemplo: manufactura delgada, manufactura ágil, manufactura flexible, manufactura concurrente, manufactura sostenible, manufactura global, etc. debido al desarrollo de la información, la comunicación, la gestión, la detección y otras tecnologías [19] y para adaptarse los requerimientos que exige hoy en día el mercado.

Para supervisar las operaciones de un sistema de manufactura, se tiene que capturar una gran cantidad de datos del sistema, por lo que se necesitarán elementos eficientes que recopilen los datos a través de todo el proceso industrial, un medio de transporte eficiente para trasladar los datos detectados y un lugar de almacenamiento de gran capacidad, que sea capaz de guardar esas cantidades de datos. La capacidad de los ordenadores es fundamental ya que ellos deben ser capaces de soportar, analizar y administrar esa información [17].

Los sistemas de manufactura en lazo cerrado se caracterizan por tener una retroalimentación de los datos de salida del proceso con el fin de determinar los fallos del sistema, predecir cuándo es necesario hacer el respectivo mantenimiento a los equipos, predecir la vida útil de los componentes del sistema y obtener información sobre la dinámica que lleva el proceso, es decir, una supervisión del mismo, y con esto generar un control del proceso de manufactura [3].

Sin embargo, estos sistemas en lazo cerrado a pesar de ser muy útiles en la industria, presentan algunas desventajas, como son la gran cantidad de datos que se recolectan para realimentar el sistema, la información se deben tomar cuando la planta está en funcionamiento para realizar el control de supervisión realimentado, esto implica que la instrumentación debe ser óptima para un trabajo en campo arduo que por lo general se presenta en los procesos de producción, tener una excelente

red de comunicación o protocolo de comunicación para que los datos recopilados lleguen a su destino y se debe poseer un espacio de almacenamiento de información.

1.2. Método de diseño

El diseño de un sistema de manufactura es un aspecto fundamental al momento de simular un proceso [20], esto implica la composición de un modelo del sistema real, es decir, la identificación y comunicación de todos los elementos que hacen posible el proceso de manufactura, así como las diferentes situaciones que pueden afectar la dinámica del sistema [21]. Un diseño efectivo del sistema ayuda a que se aumente la efectividad y la eficiencia de las operaciones de fabricación, ocasionando una reducción de los costos de fabricación y mejorar el rendimiento del sistema [14].

Existen diferentes enfoques para diseñar un sistema de manufactura, cuyo objetivo sea el de simular su comportamiento: Diseñar la estructura del sistema de manufactura, diseñar el comportamiento del sistema de manufactura o desarrollar un lenguaje de simulación.

En el presente trabajo se hace necesario definir un diseño que responda tanto a la estructura física del sistema, como al comportamiento del sistema, con el objeto de probar diferentes modos de funcionamiento del sistema; pero que a su vez permita una simulación del proceso.

Para ello se ha generado, a partir de diferentes contribuciones, un procedimiento básico:

Primero la definición de los puntos de vista para su representación [22] [23] [24], denotados como dimensiones según [20]: Dimensión estática y dinámica; Dimensión del dominio de diseño; Dimensión de los niveles abstractos.

Dentro de la dimensión estática y dinámica se tiene un punto de vista estructural, el cual describe las entidades del sistema, sus arreglos e interacciones; y otro punto de vista del comportamiento, en donde se representa la dinámica del sistema a través de diferentes escenarios (funcionamiento, mantenimiento, evolución). En la dimensión del dominio de diseño hace referencia a un punto de vista contextual, donde se ilustra la finalidad del sistema es decir las interacciones del mismo. También se describe un punto de vista funcional o lógico, el cual expresa lo que el sistema debe hacer sin especificar como. Por último, el punto de vista físico el cual recopila la información, de cómo el sistema realiza las funciones requeridas y se especifican los componentes del sistema y sus disposiciones. El dominio de los niveles abstractos contiene dos modelos que son: el modelo independiente que representa una clase abstracta del sistema y el modelo específico el cual se deriva de una modelo independiente, aprovechando la reutilización del modelo ya implementado.

Una vez se ha definido la representación, el siguiente paso es definir el alcance del diseño [25], [26], [27], para ello en [14], se propone dividir el diseño en: diseño general del sistema y diseño de instalación, diseño del sistema de manejo de materiales, diseño del sistema de fabricación celular y el diseño de sistemas de fabricación flexible [2]. Esto indica que el diseño no solamente es utilizado en la industria para modelar los procesos de planta o transformación de la materia; sino que también es utilizado para la planificación y programación de actividades.

El siguiente paso es la simulación mediante software, según [12] los autores de diseño de sistemas de manufactura se han encargado de estudiar la estructura estática y dinámica de los procesos, y se han propuesto varios métodos de diseño que sirven para la simulación mediante software, [27], identifica cuatro pasos: la especificación del propósito del modelo, especificación de componentes, especificación de parámetros y variables, y la especificación de la relación entre variables, componentes y parámetros. En [23] se argumenta que la comprensión de la situación del problema, determinación de los objetivos del modelo, identificación de parámetros de entrada, identificación de respuestas de salida del sistema y la determinación de contenido del modelo son descripciones principales para el modelo conceptual de sistemas de fabricación. En [6], se propone un modelo de simulación el cual integra el modelo físico y el modelo lógico de un sistema de manufactura flexible.

Por otro lado en [28], se presentan cinco elementos que son necesarios para modelos de simulación: el sistema real, el marco experimental, el modelo base, el modelo agrupado y el modelo por computadora, en [29] diseña un modelo de un sistema de fabricación discreto, teniendo en cuenta los elementos de la planta que la componen: pre-actuadores, actuadores y sensores, haciendo un modelo con autómatas de Moore para cada elemento de la planta, para simulación.

Como se evidencia, el diseño de sistemas de manufactura presenta distintos enfoques; por lo tanto, es necesario tener claro el propósito del modelo de sistema que se va a construir y el tipo de proceso que se va a simular, si es un proceso de transformación de materia o por el contrario es planificación de actividades del sistema administrativo. Como el propósito de este proyecto es simular la dinámica productiva de un sistema de manufactura en lazo cerrado, se realiza una revisión más a profundidad de algunos métodos de diseño anteriormente mencionados:

Modelado jerárquico y modular de eventos discretos en un entorno orientado a objetos [30].

[30], realiza un estudio de sistemas modulares para eventos discretos donde a partir de modelos generales organiza un esquema de diseño según el grado de generalidad, donde se tienen en cuenta atributos y métodos de los objetos que definen características de los sistemas de eventos discretos para la simulación orientada a objetos. El método que se define es el siguiente: Marco experimental del

modelo, donde se abordan las restricciones deseadas para el funcionamiento del sistema con respecto al entorno simulado, además se identifican las entradas y salidas, que comprende la simulación, como configuraciones de inicio y respuesta del proceso a dichas configuraciones. Modelo físico base de simulación que son aquellos modelos realizados en 3D correspondientes a la instrumentación o maquinaria del proceso, los cuales permiten tres aspectos *primero*; Realizar configuraciones y pruebas individuales, para observar el funcionamiento de la pieza, *segundo*; Conformar ensambles con otros elementos del proceso y tercero; Configuraciones físicas de simulación. Modelo agrupado de simulación; Son aquellos modelos acoplados realizados a partir de base, cuyo conjunto permite verificar modelos más grandes, es decir el proceso simulado en el entorno gráfico, en este sentido es aquí donde se observa la fluidez del proceso y las configuraciones realizadas tanto en el modelo base como en el acoplado. Modelado de pre-actuadores, actuadores y sensores; Para modelar los sensores, pre-actuadores y actuadores es necesario verificar todos los posibles estados de los instrumentos, en cuanto a su accionar en el proceso, es decir todas las posibilidades que dicho instrumento pueda presentar, estas características, se convierten en lógica de programación, para dar un funcionamiento en particular. Modelado del controlador. El modelo del controlador se desarrolla a partir de los códigos de programación de cada instrumento, donde la lógica de funcionamiento en simulación del proceso de cada uno de ellos, es representada en un diagrama de clases para facilitar la modificación del performance del entorno. Evaluación: a partir de las configuraciones y cambios realizados, se evalúa; Coordinación del proceso, Modos de funcionamiento, Registro de señales.

Diseño sistemático de sistemas de fabricación basados en el enfoque de diseño axiomático [2].

Combinando el diseño axiomático [31], y el diseño sistemático [32]. En [2] se propone un método de diseño para los sistemas de manufactura flexible. El enfoque del diseño axiomático, define el diseño como una construcción de asignaciones entre cuatro dominios que son: el dominio del cliente (atributos del cliente), el dominio funcional (requisitos funcionales), el dominio físico (parámetros de diseño), y el dominio del proceso (variables de proceso). Por esta razón el diseño axiomático al abarcar estos cuatro dominios ayuda al diseñador a analizar y evaluar soluciones de diseño. En cuanto al diseño sistemático se divide el proceso de diseño en varias etapas principales que son: conceptuales, configuración y diseño detallado.

La metodología de diseño propuesta [2], divide el proceso de diseño de sistemas de manufactura en cuatro fases:

- Fase 1: Definición de los requisitos del diseño del sistema de manufactura.
- Fase 2: Diseño conceptual del sistema de manufactura.
- Fase 3: Diseño de configuración de los sistemas de manufactura.
- Fase 4: Diseño detallado del sistema de manufactura

La fase 1, define los requisitos de diseño para los sistemas de fabricación, teniendo en cuenta las descripciones de los productos, la planificación de proceso y los costos de producción. La fase 2, es donde se determinan las operaciones de proceso, la selección de maquinaria para proporcionar las operaciones requeridas, la definición del tipo de sistema de manufactura y la identificación de posibles sistemas de manejo de materiales. Dentro de la fase 3, se refina el diseño conceptual descrito en la fase 2, se considera una descripción más detallada de la maquinaria y se consideran estrategias de producción; de igual forma se tiene en cuenta un sistema de manipulación de materiales adecuado para el proceso que mejore la productividad y la flexibilidad del sistema, y el agrupamiento de máquinas en las que se establece requisitos de espacio y disponibilidad; la salida de esta etapa es un sistema de manufactura con máquinas principales, herramientas de producción e instalaciones de manipulación de materiales. Por último, el diseño detallado que corresponde a la fase 4 refina los parámetros de diseño y evalúa el diseño final para su implementación, dentro de los parámetros de diseño se tiene las especificaciones detalladas de las máquinas y su ubicación, capacidad a amortiguamiento, el control y las estrategias de producción. [2] proporciona un caso de estudio para verificar la utilización de la metodología propuesta para sistemas de manufactura reales, concluyendo que el diseño de sistemas de manufactura es muy complejo, ya que depende de la perspectiva de cada diseñador al momento de evaluar un proceso.

Modelado de un sistema de fabricación discreto por partes de la planta:

Los autómatas de Moore [33] son máquinas de estado finito, donde las salidas están determinadas por el estado actual, mientras la transición al siguiente estado depende del estado en que se encuentre y de la entrada introducida. Los autómatas de Moore son utilizados por [29], para representar las partes de una planta y obtener un modelo representativo del sistema. Estos autores [34]; [35]; [36], afirman que la construcción del modelo depende de varios factores entre ellos el objetivo deseado para el diseño. [29] diseña un modelo de un sistema de fabricación discreto, teniendo en cuenta los elementos de la planta que son: pre-actuadores, actuadores y sensores, haciendo un modelo con autómatas de Moore para cada elemento de la planta.

Presentando una descripción detallada de cada uno de los elementos divididos en pre-actuadores, actuadores y sensores. Se realiza el estudio del comportamiento de cada elemento y construyendo un autómata de Moore que simbolice el estado de ellos. El dividir los elementos de una planta por su característica o aporte al sistema, hace que se realice un modelo detallado del comportamiento de cada una de las partes del sistema. El caso de estudio que presenta [29], es un proceso real de llenado de botellas y una simulación cada elemento en el software *state flow* de Matlab. Sin embargo, no presentan una validación del sistema global.

Modelado y simulación del marco de control en un sistema de fabricación flexible [6].

En [6], se propone un modelo de simulación el cual integra el modelo físico y el modelo lógico de un sistema de manufactura flexible. Debido a que un FMS (sistema de manufactura flexible), depende en gran parte de la tecnología que se utiliza en la planta y también del sistema de control [37], el modelo físico hace referencia a los elementos de hardware, características físicas e interacciones del proceso, mientras que el modelo lógico, tiene en cuenta el modelado del sistema de control y su interacción con el proceso (marco de control y red).

Para que el sistema sea altamente flexible se propone un marco de control descentralizado jerárquico, el cual contiene un controlador central que recibe señales de entrada sobre las órdenes de producción y gestiona controladores locales dentro del proceso para cumplir con los pedidos requeridos.

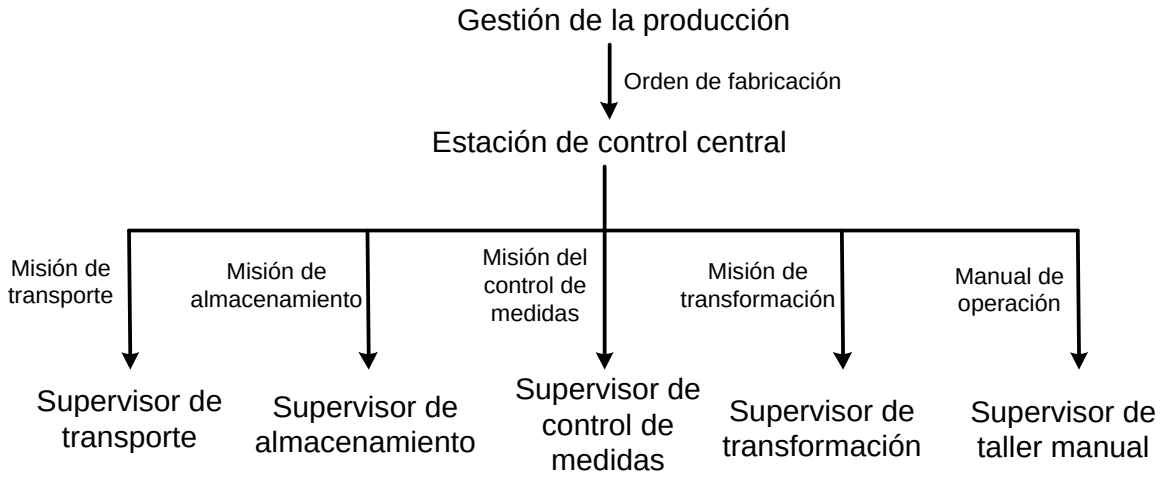


Figura 1. Estructura de control descentralizado y jerárquico *[6]*

Para modelar el control del sistema, se considera un modelo de flujo lógico que a la vez está asociado a un modelo de flujo físico, cada acción en el sistema es modelado por una entidad llamada entidad física y para cada entidad física existe una entidad lógica. Estas entidades lógicas y físicas, están sincronizadas mediante un proceso que define [6] como “espera / señal”, el cual permite bloquear una entidad con una transición de espera.

[6], selecciona el software SIMAN/ ARENA, para simular y validar su método de diseño de control. Se realiza un caso de estudio donde se divide el proceso en cinco estaciones diferentes y generando para cada una de ellas un supervisor encargado

exclusivamente de monitorear y operar lo correspondiente a su estación; y gestionan un control central que coordina los cinco supervisores y se comunica a través de una red local de Ethernet industrial.

Marco de modelado conceptual para la simulación de eventos discretos, usando estructuras de control jerárquico [38]**.**

Un marco para el modelado conceptual de procesos de manufactura ha sido propuesto por [38]. Con una revisión exhaustiva sobre la definición de modelo conceptual de sistemas de manufactura, [38] concluyen que el modelo conceptual es una descripción no especifica del software del modelo de simulación, que describe los objetivos, los productos, el contenido, los supuestos, los insumos y las simplificaciones del modelo. El modelo conceptual aborda aspectos de diseño, identificación del comportamiento del sistema y el control. [23] expone, que la identificación del problema es fundamental para realizar un modelo que simule el sistema real. Esta identificación del problema no es nada más que una descripción del sistema real donde se tiene en cuenta los objetivos principales del proceso, los factores de entrada - salida, y el alcance del modelo. El marco propuesto por [38], se basa en otros dos marcos de modelo conceptual especificados en [23] y [24].

El modelo conceptual propuesto por [23] contiene cinco pasos principales que son: la comprensión del problema o definición del proceso que se va a modelar; la determinación de los objetivos del modelo divididos en objetivos generales que se ocupan de la flexibilidad, visualización y reutilización del modelo, y objetivos del modelado; la identificación de los factores de entrada del modelo; la identificación de los resultados o salidas del modelo, los cuales se usan para determinar si se han cumplido los objetivos de diseño; y por último, la determinación del contenido del diseño, el cual se divide en la identificación del alcance del modelo y su nivel de detalle.

El modelo conceptual descrito en [24], consiste estructuras de entidades y construcciones de comportamiento. Las estructuras de identidades se encargan de las instancias de entidad en un modelo y se identifican en cuatro roles principales: consumidores, recursos, grupos y colas. Las construcciones de comportamiento se clasifican en actividades y acciones, donde las actividades se definen como una interacción entre entidades y se caracterizan por tener una condición inicial, una lista de cambios de estado al inicio, una duración y una lista de cambios de estado al finalizar y se clasifican en actividades programadas, condicional y desencadenada; por otro lado, las acciones denotan los cambios de estado similares a eventos comunes.

[38] propone un marco para el modelo conceptual unificando las propuestas de los marcos conceptuales anteriormente mencionado, añadiendo a ellos una estructura de control jerárquico. [38] argumenta que las estrategias y políticas de control deben incluirse explícitamente en cualquier método de diseño, ya que el modelo conceptual

debe denotar el proceso de abstracción de un sistema del mundo real a un modelo simplificado teniendo en cuenta sus componentes y el comportamiento del sistema. Este método de diseño es denotado por los autores como un modelo conceptual de control jerárquico (HCCM). Los pasos del marco de HCCM incluyen pasos del marco propuesto por [23], así como la separación de componentes estructurales y de comportamiento de [24]. Los pasos generales para definir el modelo HCCM se ilustran en la figura 2.

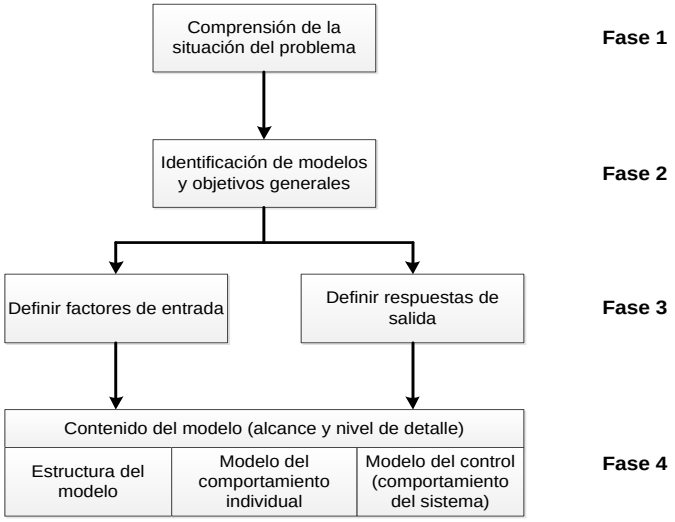


Figura 2. Estructura del modelo conceptual de control jerárquico [38]

Como se observa en la Figura 2. El modelo conceptual de control jerárquico considera las propuestas de modelos conceptuales de [23] y [24]; agregando una descripción detallada del control para el sistema a modelar. A continuación, se explica cada una de las fases del modelo:

Fase 1 Comprender la situación del problema: como se propone en [23], la comprensión del problema es una recopilación de la situación del sistema o de lo que se pretende modelar, se debe tener en cuenta situaciones supuestas del proceso. Al finalizar esta fase, se debe tener una descripción textual del proceso de producción.

Fase 2 Identificación de los objetivos: en esta fase se distinguen dos tipos de objetivos; los objetivos generales que denotan los requisitos del modelo de simulación, por ejemplo, el tiempo de ejecución, flexibilidad para cambios en el sistema y requisitos de visualización; los objetivos de modelo los cuales deben en términos de lo que se pueda lograr a partir del desarrollo y uso del modelo.

Fase 3 Definición de factores de entrada y salida: los factores de entrada son generalmente datos que pueden cambiar la dinámica del sistema; y las salidas, se

definen como los resultados del modelo y se pueden expresar en forma de valores numéricos o datos transmitidos, los datos de salida se utilizan para evaluar los objeticos del modelo.

Fase 4 Contenido del modelo: esta fase contiene la estructura del modelo, el comportamiento individual y el modelo del control. Para la estructura del modelo se define cada elemento del sistema como una entidad y se especifica las variables de estado que están asociadas a ellas, las estructuras de identidades se clasifican en activas y pasivas; en las entidades activas se considera aquellas entidades que tienen un comportamiento particular dentro del sistema pueden ser recursos, consumidores o aquellas que presentan alguna dinámica en el sistema, y las entidades pasivas no están asociadas a un comportamiento dinámico del sistema por ejemplo una entidad pasiva puede ser el área necesaria para la planta. El comportamiento individual de las entidades es una descripción del comportamiento de cada entidad asociada al sistema, se puede incluir una representación gráfica como diagramas de flujo o tablas de comportamiento. El modelo del control está asociado al comportamiento global del sistema y se determina por entidades de control, que incluyen un conjunto de reglas de funcionamiento y un conjunto de atributos y variables de estado que especifiquen el comportamiento del sistema.

Supervisores de red de Petri para DES en presencia de transiciones incontrolables y no observables [39]**.**

Un diseño de control para sistemas de eventos discretos modelados en redes de Petri es propuesto por [39], donde describen un método para diseñar controladores a partir de cálculos matemáticos, generando nuevos lugares a la red de Petri (supervisores) que modela el sistema, los cuales, impiden el disparo de transiciones que llevan a estados no deseados.

Como el objetivo del control es evitar estados no deseados del sistema, se debe hacer un estudio de restricciones físicas del sistema a controlar, estas restricciones son transformadas a una igualdad, por ejemplo: si los lugares de una red de Petri P1 y P2 no pueden estar simultáneamente activadas se genera una restricción de la forma $m(P1) + m(P2) \leq 1$; la cual indica, que la suma de las marcas de los lugares P1 y P2 debe ser menor o igual a uno. Cada una de las restricciones puestas al sistema será representada por un nuevo lugar añadido a la red de Petri del sistema total.

Cada restricción se lleva a una representación matricial $L \leq b$; donde b es un vector de enteros que representa la suma ponderada máxima de las marcas, y L es una matriz donde cada una de las filas representa cada restricción de sistema y las columnas son los lugares de la red del sistema.

La ecuación que genera la matriz de incidencia del controlador según los autores es:

$$C_c = -L * C_{'p} \qquad (1)$$

donde C_p es la matriz de incidencia de la red de Petri del proceso. El marcado inicial de los nuevos lugares es establecido por:

$$M_{c0} = b - L * M_{p0} \quad (2)$$

Donde M_{p0} es el marcado inicial del sistema. Se realiza para cada uno de los supervisores añadidos a la red de Petri del sistema un análisis de admisibilidad, donde se busca evitar que los supervisores estén relacionados con transiciones que no son controlables, es decir, que su disparo no depende exclusivamente del controlador. Para esto, se analiza la admisibilidad del sistema utilizando la ecuación $L * C_{uc} \leq 0$; donde L es la matriz de restricciones del sistema, y C_{uc} es una matriz compuesta por las transiciones no controlables del sistema, al multiplicar estas dos matrices, cada uno de sus componentes debe ser menor o igual a cero, de lo contrario el sistema no es admisible.

Si el sistema no es admisible, se construye un arreglo de matrices de la siguiente manera:

$$\begin{bmatrix} C_{uc} & I & 0 \\ LC_{uc} & R1 & R2 \end{bmatrix} \quad (3)$$

Donde, R1 y R2 son matrices resultantes a partir de operaciones entre filas para llevar a los componentes de la matriz LC_{uc} a valores admisibles, es decir a valores menores o iguales a cero.

Con la obtención de las matrices R1 y R2, se obtiene una nueva matriz de incidencia del controlador con ayuda de la ecuación:

$$C_c = -(R_1 + R_2 L) C_p \quad (4)$$

Y su marcado inicial a partir de:

$$M_{c0} = R_2(b + \mathbf{1}) - \mathbf{1} - (R_1 + R_2 L) M_{p0} \quad (5)$$

Tabla 1. Comparación de los métodos de diseño

	Diseño sistemático de sistemas de fabricación basados en el enfoque de diseño axiomático	Modelado de un sistema de fabricación discreto por partes de la planta	Modelado y simulación del marco de control en un sistema de fabricación flexible	Un marco de modelado conceptual para la simulación de eventos discretos, usando estructuras de control jerárquico
Idea principal	La metodología propuesta tiene como objetivo proporcionar pautas generales para el desarrollo de enfoques de diseño para sistemas de fabricación. Se divide el diseño en cuatro fases: Fase 1: definición de los requisitos del sistema. Fase 2: diseño conceptual del sistema Fase 3: diseño de configuración del sistema Fase 4: diseño detallado del sistema.	Presenta un enfoque para modelar un sistema de manufactura, tomando información distribuida en los elementos de la planta. Estos elementos son modelados por autómatas de Moore temporizados para permitir la comunicación entre ellos los modelos.	Realizan un diseño de un sistema de fabricación flexible combinando el modelo físico correspondiente a materiales y características propias de los FMS y el modelo lógico corresponde al modelado del sistema de control computarizado y su interacción con la parte de hardware.	Propone un nuevo marco para el modelado conceptual de sistemas de eventos discretos, el cual aborda principalmente los aspectos de diseño, definición de los requisitos del modelo y políticas de control explicitas y centralizadas. El modelo es validado a través de un sistema de proceso real.
Ventajas	Divide el diseño en fases de desarrollo, lo cual permite independizar las tareas de diseño, haciendo que se genere una metodología de diseño por etapas que de claridad al diseño y facilidad el diseñador de interpretar.	El modelado distribuido introducido en este documento permite validar partes de plantas locales, lo que permite que evitar la combinación de todos los elementos de la planta en un solo modelo. Siendo así, el modelo global del sistema está construido en base a los modelos locales que deben comunicarse juntos.	Esta integración permite evaluar la incidencia del marco de control en el rendimiento del sistema de producción. También probar diferentes marcos de control en un sistema de fabricación existente y seleccionar el que está más adaptado con respecto a las limitaciones físicas y los objetivos de producción.	Construye un modelo conceptual donde denota un sistema real a un modelo simplificado, con componentes y comportamiento individual de cada uno de ellos, añadiendo además estructuras y políticas de control explicitas dentro del modelo.
Desventajas	No presenta un formalismo de modelado de sistemas de eventos discretos.	El comportamiento global de la planta no está validado en el documento, no presenta un modelado del producto.	Se encierran específicamente en sistemas de manufactura flexibles (FMS)	Al introducir un modelo de control del sistema, vuelve el método de diseño extenso, por lo se requiere un detalle especifico de comportamiento cuidadoso a diferencia de los demás métodos de diseño.

Fuente: elaboración propia

En la

Tabla 1. Comparación de los métodos de diseño, se observa las diferentes ventajas y desventajas de cada uno. Se concluye que el método de diseño propuesto por [38], contribuye al diseño deseado para el caso de estudio, ya que utiliza el marco de estructura HCCM, para establecer la situación del problema a desarrollar en el sistema de manufactura, crear objetivos del modelo y detallar los específicamente los elementos que forman la planta. Además, utilizan los sistemas de eventos discretos como herramienta para modelar el proceso y añaden una estructura de control jerárquico dentro del modelo conceptual.

Teniendo en cuenta lo anterior, se propone seguir el siguiente procedimiento para realizar el diseño y simulación de un sistema de manufactura en lazo cerrado:

1. Definición del punto de vista estructural, comportamiento, interacciones del sistema y componentes del sistema de manera global [20].
2. Alcance del diseño: Diseño general, diseño de instalación y diseño del manejo de materiales [23].
    a. Comprensión de la situación del problema.
    b. Identificación de objetivos.
    c. Definición de factores de entrada y salidas.
    d. Contenido del modelo:
        i. Estructura especifica del modelo.
        ii. Modelo del comportamiento individual.
        iii. Modelo del control [39].
3. Simulación del sistema diseñado [30]:
    a. Marco experimental del modelo.
    b. Modelo físico base de simulación.
    c. Modelo agrupado de simulación.
    d. Modelado de pre-actuadores, actuadores y sensores.
    e. Modelado del controlador.
    f. Verificación.

1.3. Herramienta de simulación

En la Tabla 2, se observan las características de diferentes softwares de simulación, y los cuales se analizan para identificar la herramienta adecuada, que permita desarrollar los diferentes entornos gráficos para la simulación del sistema de manufactura, para ello se evaluaron siete (7) características.

Tabla 2. Comparación de software de simulación

	CARACTERISTICAS		Gráficos en 3D	Compatibilidad	Costo	Soporte bibliográfico	Versiones gratuitas	simulación de la dinámica	lenguaje de programación
	SOFTWARE								
1	Arena		√	√	√	MEDIO	√	X	√
2	Automod		√	√	√	BAJO	√	X	√
3	Witness		√	√	√	BAJO	√	√	√
4	Openmodelica		X	√	√	ALTO	√	√	√
5	Factory I/0		√	√	√	MEDIO	√	√	√
6	Stateflow		X	√	√	ALTO	X	√	√
7	Unity		√	√	√	ALTO	√	√	√

Fuente: elaboración propia

Los gráficos en 3D, permiten navegar por la simulación libremente y observar de una manera más real las animaciones elaboradas por el o los desarrolladores de programación y diseño.

La compatibilidad, un factor importante, puesto que este es uno de los pilares de los objetivos propuestos, debido a que este factor es el que admite que el software pueda comunicarse o vincularse con otras plataformas existentes por medio de módulos especiales o simplemente configurando la programación realizada para dicho fin.

El costo del software, como se puede observar todas estas herramientas tienen un costo para las versiones completas o pro, las cuales brindan absolutamente todo el

potencial que las plataformas ofrezcan, para satisfacer a los desarrolladores y diseñadores de programación.

El respaldo bibliográfico, las páginas oficiales ofrecen al cliente manuales, demos, foros y guías para el manejo de la plataforma, en este caso se ha evaluado dicho soporte como bajo, medio y alto, aunque algunos softwares son intuitivos, se debe reconocer que el soporte que brindan algunas plataformas es completo para el usuario, y esto a la larga da un respaldo muy grande para escoger la plataforma más adecuada para proyectos complejos.

En cuanto a las versiones gratuitas se puede mencionar que la mayoría tienen el trial para estudiante, donde se puede visualizar la herramienta en muchos casos limitada, aun así, cabe destacar que, dentro de dichas versiones a pesar de ser de prueba o limitadas en el tiempo de uso y funciones, las plataformas Unity y Open Modélica ofrecen versiones prácticamente completas sin límite de tiempo, lo que se supone un beneficio enorme para el interés del proyecto.

Para finalizar hay dos componentes que van muy ligados y son la simulación del comportamiento dinámico y los tipos de lenguaje de programación que permiten dichos comportamientos, estos van ligados, ya que la programación permite que los elementos o componentes del proceso puedan cambiar su configuración conforme a lo que el proyecto de simulación demande.

En la revisión realizada, algunas herramientas software se especializan en el análisis delos sistemas de gestión y no en los de producción o manufactura en los cuáles la programación es más permisiva; es decir, que se puede manipular o crear elementos que se adecuen a los proyectos que son realizados por los programadores y que a la vez son fáciles de modificar debido a los lenguajes de programación que son compatibles con las plataformas que están siendo comparadas.

De acuerdo a la Tabla 2 se pueden definir como posibles herramientas a Factory I/0 y Unity. Dado que Factory I/O es una herramienta de pago, se elige Unity ya que funcionalmente y por las características que se han definido anteriormente, cumple con el propósito de crear una simulación en un entorno gráfico que permite extraer o llevar a otra plataforma el comportamiento para ser manipulado o monitoreado, a partir del lenguaje de programación C# que es un lenguaje de programación orientado a objetos estandarizado y la posibilidad que tiene Unity como software de videojuegos, que a través de ciertos comandos de programación se pueden extraer las señales, debido a su compatibilidad con otras herramientas, por estos motivos Unity se convierte en el software perfecto para realizar el proyecto de simulación.

Capítulo 2

Modelado de un sistema de manufactura.

2.1. Definición del punto de vista estructural.

2.1.1. Definición del proceso seleccionado.

El proceso que se ha seleccionado como caso de estudio es el ensamble de mosquetones. Los mosquetones, son grilletes de seguridad que se utilizan generalmente para la recreación o la industria. Son diseñados para soportar cargas pesadas y a su vez ser livianos para su fácil manipulación. Un mosquetón utilizado en el deporte del montañismo debe ser capaz de soportar el peso de 2 toneladas y pesar aproximadamente 80 gramos para su fácil transporte y sus dimensiones no exceden los 15 cm. Para este proyecto se realizará el estudio del proceso industrial para el ensamble de mosquetones destinados únicamente para la recreación.

2.1.2. Descripción general del proceso

El proceso consta de seis fases operacionales secuenciales.

Fase 1: Un operario de forma manual ubica las piezas que forman el mosquetón (cuerpo, dedo y eje) y los ajusta en la prensa que contiene la mesa giratoria, una vez

ajustadas las piezas la mesa se encarga de trasladar el mosquetón por las diferentes fases del proceso.

Fase 2: Una remachadora se encarga de ajustar totalmente las piezas del mosquetón.

Fase 3: Proceso de pintura del mosquetón.

Fase 4: Se transporta el mosquetón ya terminado a un banco de prueba con ayuda de un robot manipulador.

Fase 5: Prueba de resistencia, en esta fase se simula la fuerza mínima que debe soportar el mosquetón. Concluida la fase 5, se determina si la pieza cumple con los requerimientos de resistencia establecidos; si la pieza es aprobada, el mosquetón pasa a la fase 6.

Fase 6: Es transportado el mosquetón por un segundo robot manipulador, el cual, se encarga de ubicar el producto terminado en una caja para su embalaje; si la pieza es rechazada, un eyector ubicado en la mesa de prueba expulsa el mosquetón del banco de prueba.

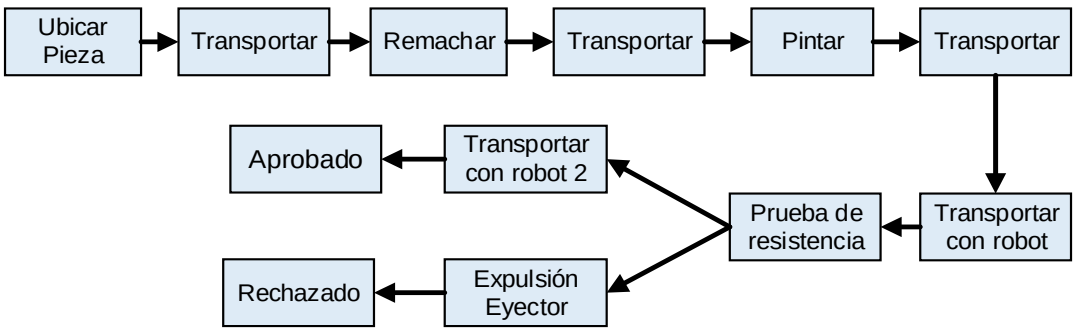


Figura 3. Diagrama de bloques
Fuente: elaboración propia

En la Figura 3 se muestra el proceso para realizar el ensamble de mosquetones, este proceso es automatizado a excepción del ingreso de piezas, donde se necesita un operario que manualmente ubica los materiales para ser ensamblados.

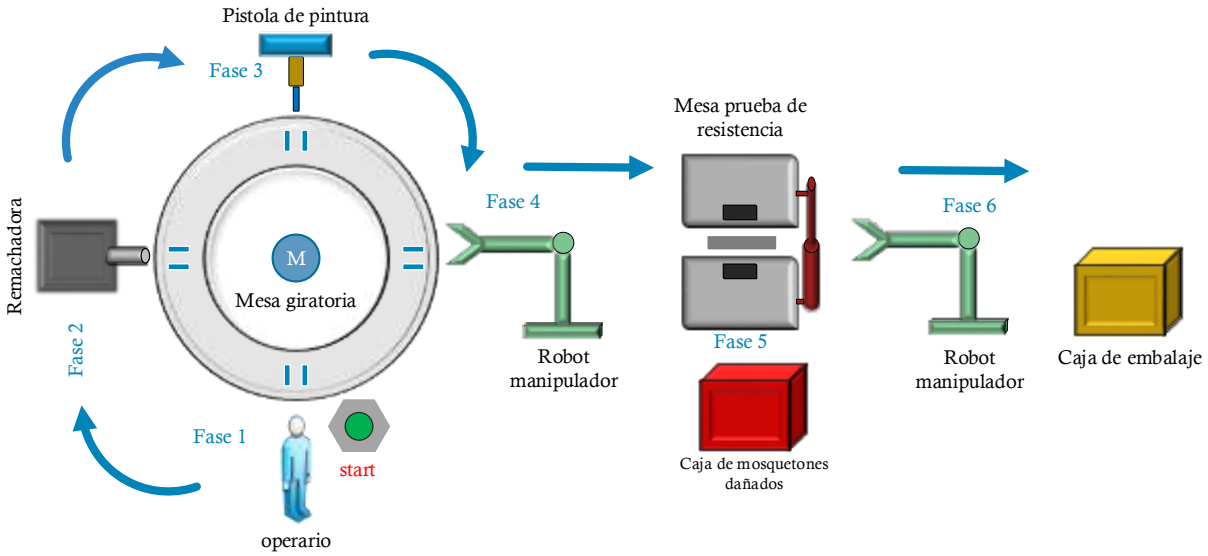


Figura 4. Proceso ensamble de mosquetones.
Fuente: elaboración propia

2.1.3. Definición de entidades.

Tabla 3. Definición de entidades.

Nombre entidad	Tipo de entidad	Atributos
Mesa giratoria	Activa	· Posición de pieza · Orden de activación
Remachadora	Activa	· Posición de pieza · Posición remachadora
Pulverizador de pintura	Activa	· Posición de pieza · Scanner de pintura
Robot manipulador 1	Activa	· Posición de pieza · Posición de robot
Mesa de prueba	Activa	· Disponibilidad · Posición pieza · Posición de cilindro
Robot manipulador 2	Activa	· Posición de pieza · Posición de robot
Área de la planta	Pasiva	Cantidad de área en metros cuadrados

Caja de aprobados	Pasiva	Número de mosquetones
Caja de dañados	Pasiva	Número de mosquetones

Fuente: elaboración propia.

Con lo anterior, se observa que el proceso seleccionado cumple con las características de un sistema de manufactura en lazo cerrado para ser modelado a partir del método de diseño propuesto. Desde el punto de vista estructural es un proceso que puede ser simulado en una herramienta software, ya que su dinámica no muy compleja. Su comportamiento e interacción puede ser modelado como un sistema de eventos discretos y es un sistema que puede ser dividido en subsistemas.

2.2. Alcance del diseño

2.2.1. Comprensión de la situación del problema.

El resultado de esta primera sección es una descripción textual de lo que se requiere con el diseño.

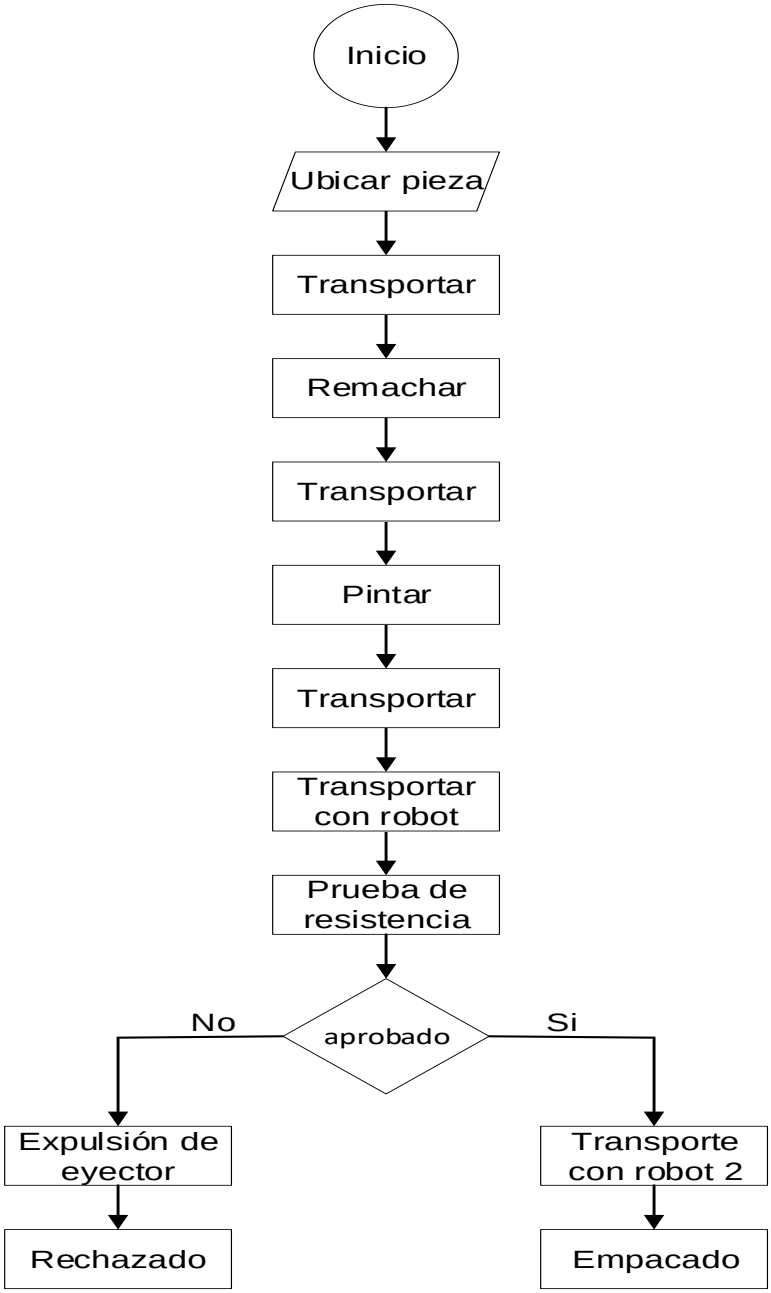


Figura 5. Diagrama de flujo del proceso
Fuente: elaboración propia.

El problema es modelar y simular el proceso para la obtención de mosquetones presentado en el diagrama de flujo de la Figura 5; de tal manera que éste sea cíclico, que cumpla con restricciones de funcionamiento para el cuidado de los equipos, que realice las operaciones de transporte, remachado y pintado de manera automática y la operación ubicación de piezas a la mesa giratoria de forma manual; además que cumpla con las restricciones físicas, por ejemplo, los robots no pueden encontrarse

en la máquina de prueba de resistencia a la misma vez, y que no se dé el accionamiento de la remachadora cuando la mesa está girando. También las restricciones de funcionamiento como: no encender el pulverizador a la misma vez que la mesa, no accionar el eyector cuando la máquina de resistencia está ocupada. Estas restricciones permiten verificar el comportamiento de todo el sistema.

2.2.2. Identificación de objetivos

Tabla 4 Objetivos generales del diseño

Objetivos generales	
Requisitos de simulación	Visualizar en un entorno gráfico el proceso de fabricación de mosquetones en lazo cerrado, de tal manera que responda a señales entrada que generan diferentes modos de funcionamiento y que permita extraer las señales intercambiadas.
Requisitos de reutilización	Visualizar el funcionamiento del proceso acorde a diferentes estrategias de control.
Documentación y usabilidad	Realizar una guía de proceso, en el cual se evidencie el funcionamiento y transmisión de datos del proceso, y la configuración del control externo aplicado al sistema

Fuente: elaboración propia.

Tabla 5. Objetivos de modelado

Objetivos de modelado	
Requisitos actuales	Modelar el proceso de ensamble de mosquetones para obtener una simulación cíclica del sistema.
Requisitos futuros	Adquirir señales de estado del sistema.
Mejora de Políticas	Evaluación de las diferentes políticas de control

Fuente: elaboración propia

2.2.3. Definición de factores de entradas y salidas

Tabla 6. Definición de entradas y salidas del modelo

SALIDAS

Número de Ciclos	Ejecución del proceso de acuerdo a la estrategia de control.
Estados del sistema	Medición de señales que determinan el estado de los procesos.
Número de Piezas fabricadas	Producción del sistema
FACTORES DE ENTRADA	
-Estrategia de control	Modos de funcionamiento
Número de máquinas	Área piso

Fuente: elaboración propia

2.2.4. Contenido del modelo

2.2.4.1. Definición de máquinas e instrumentación

Para la selección de la instrumentación del proceso, se realiza una tabla para cada uno de los elementos o máquinas que se necesitan para llevar a cabo el proceso de ensamble de mosquetones.

Tabla 7. Máquinas seleccionadas.

Máquina	Características	Imagen
FIBROMAT AT.0800	Diámetro de la mesa mm 800 Diámetro del orificio central mm 320 Superestructura máxima diámetro mm 4.500 Carga máxima de transporte kg 10.000 Momento de inercia admisible J kgm² 8.000 Mesa de indexación N 56.000 Peso kg 350	
Remachadory1000	Presión máx. de funcionamiento 12 Kg. Potencia 6 kg. - 2.520 kg. Distancia máx. del porta punzón al contra punzón: 190mm. Distancia mín. del porta punzón al contra punzón: 0mm. Capacidad de remachado: 1 – 10mm Dimensiones: 470 x 52 x 1.640mm. Peso: 155 kg.	
Pistola automática WWA AIR COMBI	Presión máx. de material1.2MPa (12 bar) Temperatura máx. del material100 °C Presión máx. del aire de pulverización0.8MPa (8 bar) Presión mínima del aire de ajuste0.4MPa (4 bar)	

	Presión máx. del aire de ajuste0.8MPa (8 bar) Temperatura máx. del aire50 °	
Robot UR5	Tipo: articulado Número de ejes: 6 ejes Funciones: de manipulación, para ensamblaje, de paletización, pick and place, de mecanizado, de prueba Carga máxima: 5 kg (11.02 lb) Radio de acción: 850 mm Repetitividad: 0.1mm	
LLOYD - EZ20	Desplazamiento Estándar 870mm (34.3 in) Precisión en velocidad <0.2% continua Resolución mínima de carga 0.0001N Pantalla LCD de 40 caracteres x 4 líneas Sistema de Medición de carga ISO 7500:2004 Clase 0.5 ASTM E4 Peso 148 kg (326 lb) Alimentación 112 VCA Y 230 VCA Dimensiones 1.567mm x 2.067mm x 868mm	
Cilindro de aire SC32x50	Tipo de acción: Acción doble Material: Aleación de aluminio Abrir 32mm Carrera 50mm Presión máxima 1MPA Diámetro del hilo de varilla M10x1.25 Diámetro del orificio del aire 1/8 Cantidad 1pc	

Fuente: elaboración propia

Tabla 8. Sensores

MODELO		CARACTERÍSTICAS	IMAGEN
1	**Sensor de proximidad Inductivo LJ12A3 4-Z/BX**	Voltaje de alimentación: 6 a 36 V Tipo de sensor: Inductivo Tipo de salida: NPN Diámetro de la cabeza: 12 mm Distancia de detección: 4 mm Corriente de salida: 300 mA Respuesta en frecuencia: 0.5 Khz Conexión por cable de 3 hilos Marrón: +VCC Negro: Salida (NPN) Azul: GND Objetos de detección: Metálicos Distancia de la detección: 4 mm Detección de objetos: Conductor Estado del interruptor: Normalmente abierto Temperatura de operación: -25 ºC a 55 ºC Dimensiones: 6.2 cm X 2 cm Modelo: LJ12A3- 4-Z/BX	
2	**Sensor de proximidad infrarrojo FBK-INF-CRO-85**	Fuente de alimentación: 5V. Corriente: 80 a 120mA. Rango: 4 cm a 85 cm ajustable. Rojo: +5 V. Amarillo: Señal. Verde: GND (Vss) Longitud del cable: 42cm	
3	**Sensor de proximidad fotoeléctrico reflex E3F-R2N12**	Modelo: E3F-R2N12 Voltaje de Operación: 6 - 36V DC Corriente de trabajo: 300mA Rango de detección: 1 – 200 cm Fuente de luz: led 660nm Modo de detección: Retroreflectivo Dimensiones: D18mm*L75mm Salida: Tipo NPN (NO+NC) Material: plástico Temperatura de trabajo: -25 a 70°C	

Fuente: elaboración propia

Se toman tres tipos de sensores de proximidad diferentes (inductivo, infrarrojo y fotoeléctrico), para detectar la presencia de pieza en cada una de las fases del proceso. Es necesario que las piezas estén perfectamente ubicadas, en la posición correcta para que las máquinas puedan hacer su trabajo; por lo tanto, el censado debe ser con un rango milimétrico.

Tabla 8, evidencia que la opción 1 presenta el rango de medición más pequeño de las tres opciones (4mm), y este rango de medida es lo más relevante del sensor que se va a utilizar. Este sensor presenta una desventaja frente a los otros, ya que solo puede censar materiales metálicos, pero como el mosquetón es construido con aluminio esa desventaja no es problema para el proceso, lo cual lo convierte en el instrumento más indicado.

2.2.4.2. Cálculo del área de la planta

El cálculo del área o superficie de trabajo necesaria para los elementos del proceso, se fundamenta en el método de Guerchet a partir del cálculo de tres superficies: superficie estática (Ss), superficie de gravitación(Sg) y la superficie de evolución (Se); las cuales al sumarse se obtiene la superficie total necesaria para cada máquina.

Superficie estática (Ss): Es la superficie plana correspondiente a las dimensiones de la máquina, y se define en unidades cuadráticas.

Superficie de gravitación (Sg): Es la superficie alrededor de la máquina que se utiliza para manipular el equipo de manera manual por el operador. Se calcula multiplicando la superficie estática de la máquina por el número de caras por las cuales se puede operar.

$$Sg = Ss * N \quad (6)$$

Superficie de evolución (Se): Es el área que se debe reservar de cada máquina para el desplazamiento del personal, evitar accidentes y el mantenimiento de la misma. Se obtiene sumando la superficie estática con la superficie de gravitación y el resultado se multiplica por una constante K.

$$Se = (Ss + Sg) * K \quad (7)$$

$$K = \frac{altura\ promedio\ de\ los\ operarios}{2 * altura\ promedio\ de\ las\ máquinas} \quad (8)$$

$$K = \frac{1.7mts}{2 * 1.7mts}$$

$$K = 0.5$$

Superficie total (St): Es la superficie que se necesita para ubicar las máquinas en un espacio determinado, se obtiene sumando todas las superficies anteriores.

$$St = Ss + Sg + Se \qquad (9)$$

Tabla 9. Área de la planta

Máquina		Superficie estática (Ss) mt^2	Número de caras (N)	Superficie de gravitación (Sg) mt^2	Superficie de evolución (Se) mt^2	Superficie total x máquina mt^2
1	Mesa giratoria	15,2	4	60,8	38	114
2	Remachadora	2	1	2	2	6
3	Robot manipulador 1	2,3	4	9,2	5,75	17,25
4	Robot manipulador 2	2,3	4	9,2	5,75	17,25
5	Banco de prueba	2	2	4	3	7
Total de cada superficie		23,8		85,2	54.5	161,5

Fuente: elaboración propia

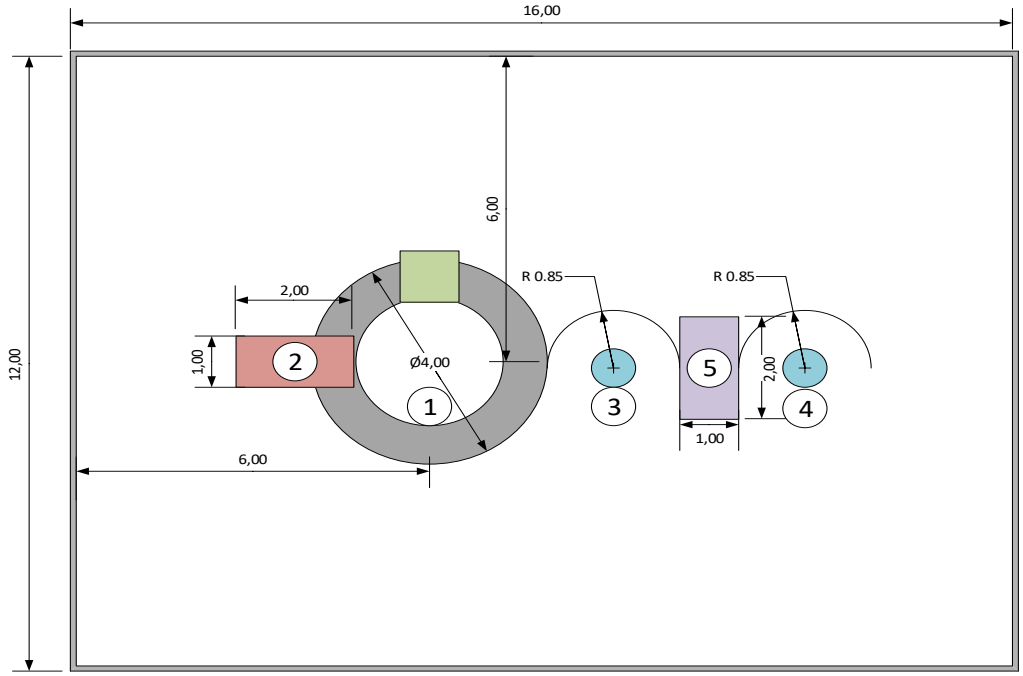


Figura 6. superficie de área de la planta
Fuente: elaboración propia

2.2.5. Modelo del comportamiento individual del sistema

2.2.5.1. Comportamiento individual

2.2.5.1.1 Mesa giratoria

La mesa de transporte o giratoria, tiene como objetivo trasladar las piezas del mosquetón a través de 3 estaciones de operación. La ubicación de las piezas sobre la mesa se hace a través de un operario, quien se encarga de poner el cuerpo, el dedo y el eje del mosquetón; una vez ubicadas las piezas, la mesa puede girar de forma manual utilizando un botón de start (Z) o de forma automática con ayuda del controlador. Para que la mesa gire con ayuda del controlador el sensor de posición (ZE1) debe indicar que hay presencia de pieza para ser transportada. Si el funcionamiento de la mesa es manual, un sensor de posición (ZE2) ubicado en la mesa debe indicar al controlador cuándo debe parar el giro de la mesa. Para que la mesa pueda girar se tiene la condición del que el robot manipulador debe estar en su posición inicial.

El comportamiento de la mesa giratoria se modela a partir de una red de Petri Interpretada (IPN), como se observa en la Figura 7.

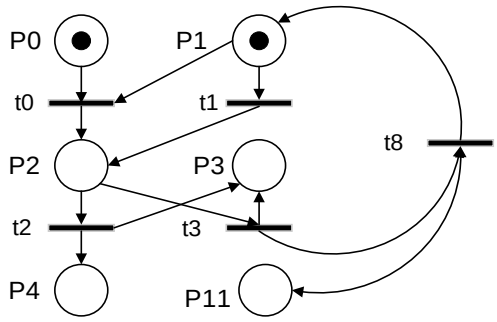


Figura 7. Red de Petri mesa giratoria
Fuente: elaboración propia

Tabla 10. Definición de lugares mesa giratoria

LUGARES	DESCRIPCION
P0	Pieza ubicada para transportar
P1	Mesa disponible
P2	Mesa operando
P3	Mesa apagada
P4	Pieza en espera para ser remachada
P11	Robot manipulador disponible

Tabla 11. Definición de transiciones mesa giratoria

TRANSICIONES	DESCRIPCIÓN

T0	Señal de sensor (ZE1) pieza detectada y señal de control encender mesa giratoria
T1	Señal al presionar el botón de start (Z)
T2	Señal de control apagar mesa giratoria
T3	Señal de Sensor (ZE2) mesa en posición
T8	Señal de control encender mesa giratoria o señal al presionar el botón de start (Z)

Fuente: elaboración propia

2.2.5.1.2 Remachadora

La máquina remachadora cumple la función de ajustar el mosquetón, para que pueda ser activada la remachadora desde el controlador debe existir pieza en la zona de remache, para esto, un sensor de posición (ZE3) indica al controlador la presencia de pieza para ser remachada, de lo contrario el controlador no enciende la máquina. Una vez terminado el proceso de remachado, la pieza queda en un estado de espera para ser transportada a la siguiente estación.

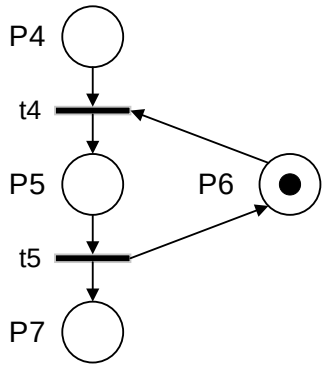


Figura 8. Red de Petri Remachadora
Fuente: elaboración propia

Tabla 12. Definición de lugares Remachadora

LUGARES	DESCRIPCIÓN
P4	Pieza en espera para ser remachada
P5	Remachadora operando
P6	Remachadora disponible
P7	Pieza en espera para ser pintada

Fuente: elaboración propia

Tabla 13. Definición de las transiciones Remachadora

TRANSICIONES	DESCRIPCIÓN

T4	Sensor de posición (ZE3) y señal de control encender remachadora
T5	Señal de control apagar remachadora

Fuente: elaboración propia

2.2.5.1.3 Pulverizador de pintura

El pulverizador de pintura se encarga de pintar el mosquetón. Para que este proceso se lleve a cabo el sensor de posición (ZE4) debe anunciar la llegada de pieza a la zona de pintura, enviado la señal de sensor al controlador para que este pueda encender el pulverizador. El proceso de pintura está programado por tiempo, es decir que el controlador apaga automáticamente el pulverizador al transcurrir un determinado tiempo.

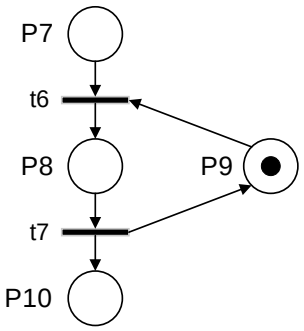


Figura 9. Red de Petri Pulverizador de pintura
Fuente: elaboración propia

Tabla 14. Definición de lugares pulverizador de pintura

LUGARES	DESCRIPCIÓN
P7	Pieza en espera para ser pintada
P8	Pulverizador de pintura operando
P9	Pulverizador de pintura disponible
P10	Pieza en espera para ser transportada

Fuente: elaboración propia

Tabla 15. Definición de transiciones Pulverizador de pintura

TRANSICIONES	DESCRIPCIÓN
T6	Sensor de posición (ZE4) y señal de control encender pulverizador.
T7	Señal de control apagar pulverizador

2.2.5.1.4 Robot manipulador 1

La función que cumple el robot manipulador en el proceso es de transporte, del mosquetón terminado a la estación de prueba de resistencia. Para que se realice esta operación el sensor (ZE5) debe enviar una señal de presencia de mosquetón al controlador, y el robot manipulador debe estar en un estado disponible para trabajar. El estado inicial o disponible del robot es estar ubicado sobre la mesa giratoria, con su brazo arriba y sus pinzas abiertas. La secuencia de operación del robot es: bajar, cerrar pinza, subir, girar a la izquierda, bajar, abrir pinza, subir y girar a la derecha.

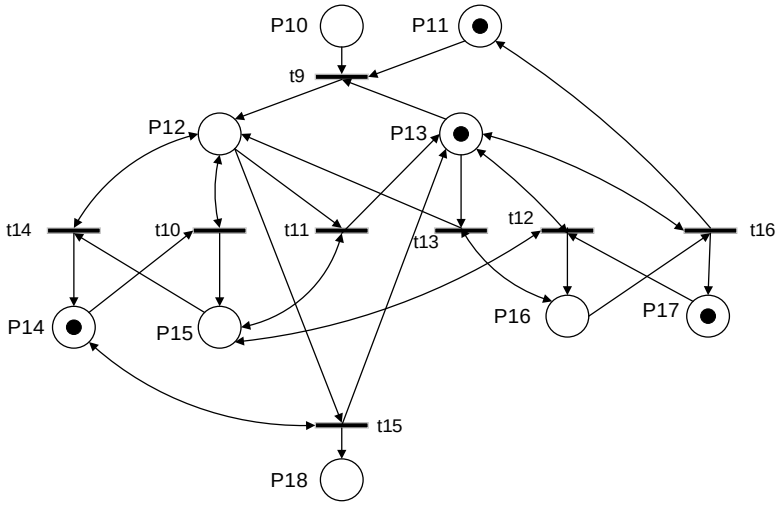


Figura 10. Red de Petri Robot manipulador 1
Fuente: elaboración propia

Tabla 16. Definición de lugares Robot manipulador 1

LUGARES	DESCRIPCIÓN
P10	Pieza en espera para ser transportada
P11	Robot manipulador 1 disponible
P12	Robot manipulador 1 abajo
P13	Robot manipulador 1 arriba
P14	Robot manipulador 1 pinza abierta
P15	Robot manipulador 1 pinza cerrada
P16	Robot manipulador 1 a la izquierda
P17	Robot manipulador 1 a la derecha
P18	Pieza en espera para la prueba de resistencia

Fuente: elaboración propia

Tabla 17. Definición de transiciones Robot manipulador 1

TRANSICIONES	DESCRIPCIÓN
T9	Sensor de posición (ZE5) y señal de control bajar robot manipulador 1.
T10	Señal de control cerrar pinza robot manipulador 1.
T11	Señal de control subir robot manipulador 1.
T12	Señal de control girar a la izquierda robot manipulador 1.
T13	Señal de control bajar robot manipulador 1.
T14	Señal de control abrir pinza robot manipulador 1.
T15	Señal de control subir robot manipulador 1.
T16	Señal de control girar a la derecha robot manipulador 1.

Fuente: elaboración propia

2.2.5.1.5 Máquina de prueba de resistencia

La máquina de prueba de resistencia inicia su funcionamiento cuando el sensor de posición (ZE6) indica la presencia de mosquetón sobre la máquina. La máquina de prueba de resistencia imprime sobre el mosquetón una fuerza de compresión simulada, los estados de la máquina son operando o disponible. El controlador es quien decide si una pieza es aprobada o rechazada; si la pieza es rechazada se activa un eyector el cual expulsa el mosquetón hacia una caja de mosquetones rechazados; si la pieza es aprobada queda en un estado de espera para ser transportado por un segundo robot manipulador hacia la caja de mosquetones aprobados.

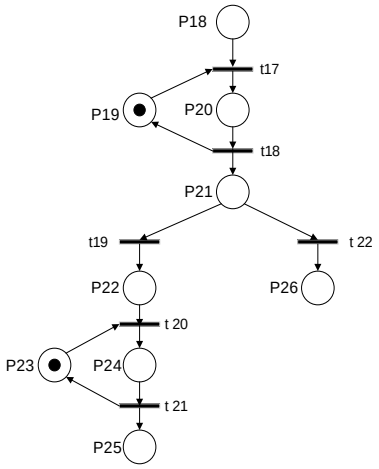


Figura 11. Red de Petri Máquina de prueba de resistencia
Fuente: elaboración propia

Tabla 18. Definición de lugares Máquina de prueba de resistencia

LUGARES	DESCRIPCION
P18	Pieza en espera para la prueba de resistencia
P19	Máquina de prueba de resistencia disponible
P20	Máquina de prueba de resistencia operando
P21	Análisis de aprobación
P22	Mosquetón rechazado
P23	Eyector disponible
P24	Eyector operando
P25	Buffer de mosquetones rechazados
P26	Mosquetón aprobado y en espera de ser transportado

Fuente: elaboración propia

Tabla 19. Definición de transiciones Maquina prueba de resistencia

TRANSICIONES	DESCRIPCIÓN
T17	Sensor de posición (ZE6) y señal de control encender máquina de prueba de resistencia
T18	Señal de control apagar máquina de prueba de resistencia
T19	Señal de control rechazo de pieza
T20	Señal de control encender eyector
T21	Señal de control apagar eyector
T22	Señal de control aprobación de pieza

Fuente: elaboración propia

2.2.5.1.6 Robot manipulador 2

Si la decisión del controlador en la máquina de prueba de resistencia es aprobar el mosquetón, el robot manipulador 2 entra en funcionamiento llevando la pieza terminada a un buffer de mosquetones aprobados y listos para ser comercializados. La secuencia de trabajo del robot manipulador 2 es similar a la secuencia del robot manipular 1.

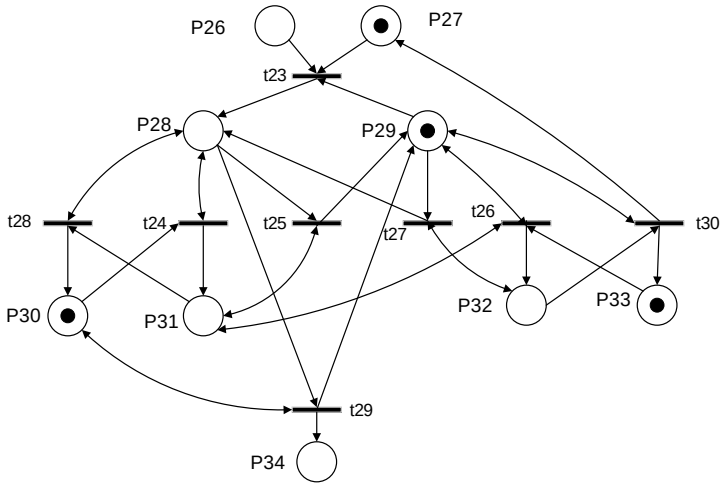


Figura 12. Red de Petri Robot manipulador 2
Fuente: elaboración propia

Tabla 20. Definición de los lugares Robot manipulador 2

LUGARES	DESCRIPCION
P26	Pieza en espera para ser transportada
P27	Robot manipulador 1 disponible
P28	Robot manipulador 1 abajo
P29	Robot manipulador 1 arriba
P30	Robot manipulador 1 pinza abierta
P31	Robot manipulador 1 pinza cerrada
P32	Robot manipulador 1 a la izquierda
P33	Robot manipulador 1 a la derecha
P34	Buffer de piezas aceptadas

Fuente: elaboración propia

Tabla 21. Definición de las transiciones Robot manipulador 2

TRANSICIONES	DESCRIPCIÓN
T23	señal de control bajar robot manipulador 1.
T24	Señal de control cerrar pinza robot manipulador 1.
T25	Señal de control subir robot manipulador 1.
T26	Señal de control girar a la izquierda robot manipulador 1.
T27	Señal de control bajar robot manipulador 1.
T28	Señal de control abrir pinza robot manipulador 1.
T29	Señal de control subir robot manipulador 1.
T30	Señal de control girar a la derecha robot manipulador 1.

Realizando la sincronización de los modelos de los diferentes dispositivos, se obtiene la siguiente red de Petri interpretada que modela el comportamiento de todo el sistema.

2.2.5.2. Comportamiento global.

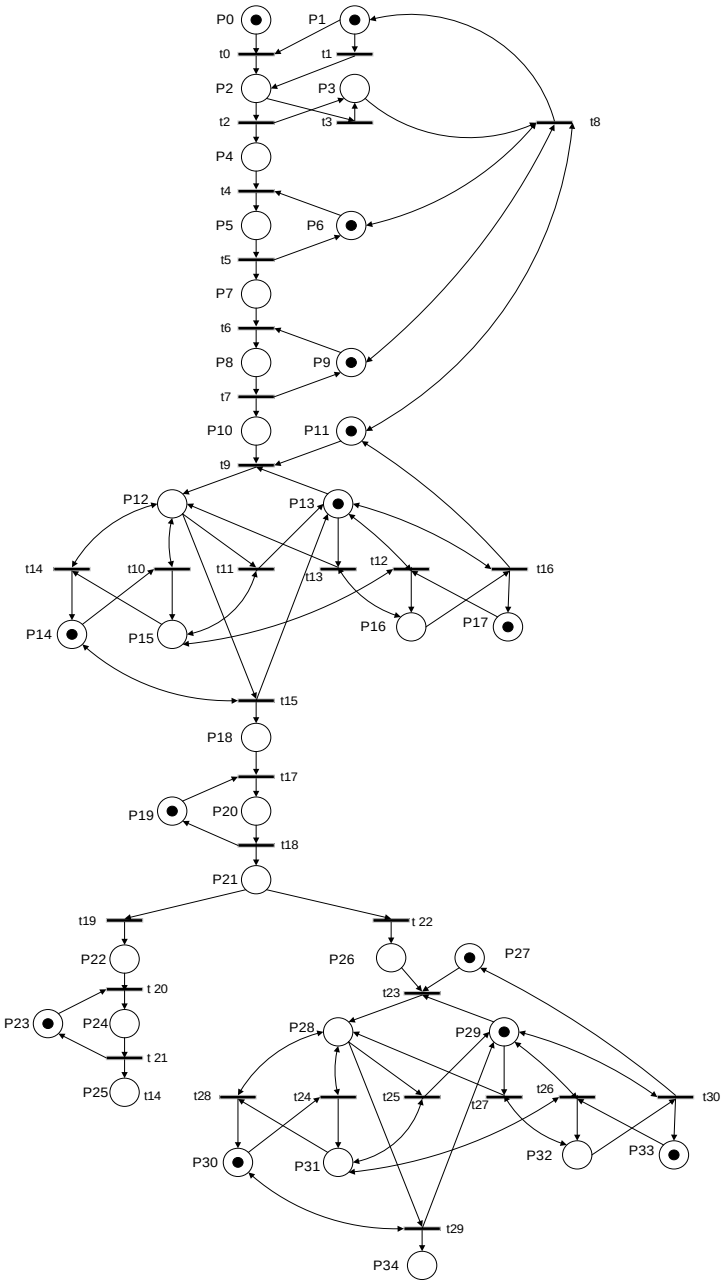


Figura 13. Red de Petri del sistema
Fuente: elaboración propia

Tabla 22. Lugares del sistema completo

LUGARES	DESCRIPCIÓN
P0	Pieza ubicada para transportar
P1	Mesa disponible
P2	Mesa operando
P3	Mesa apagada
P4	Pieza en espera para ser remachada
P5	Remachadora operando
P6	Remachadora disponible
P7	Pieza en espera para ser pintada
P8	Pulverizador de pintura operando
P9	Pulverizador de pintura disponible
P10	Pieza en espera para ser transportada
P11	Robot manipulador 1 disponible
P12	Robot manipulador 1 abajo
P13	Robot manipulador 1 arriba
P14	Robot manipulador 1 pinza abierta
P15	Robot manipulador 1 pinza cerrada
P16	Robot manipulador 1 a la izquierda
P17	Robot manipulador 1 a la derecha
P18	Pieza en espera para la prueba de resistencia
P19	Máquina de prueba de resistencia disponible
P20	Máquina de prueba de resistencia operando
P21	Análisis de aprobación
P22	Mosquetón rechazado
P23	Eyector disponible
P24	Eyector operando
P25	Buffer de mosquetones rechazados
P26	Mosquetón aprobado y en espera de ser transportado
P27	Robot manipulador 1 disponible
P28	Robot manipulador 1 abajo
P29	Robot manipulador 1 arriba
P30	Robot manipulador 1 pinza abierta
P31	Robot manipulador 1 pinza cerrada
P32	Robot manipulador 1 a la izquierda
P33	Robot manipulador 1 a la derecha
P34	Buffer de piezas aceptadas

Fuente: elaboración propia

Tabla 23. Transiciones del sistema completo

TRANSICIONES	DESCRIPCIÓN
T0	Señal de sensor (ZE1) pieza detectada y señal de control encender mesa giratoria
T1	Señal al presionar el botón de start (Z)
T2	Señal de control apagar mesa giratoria
T3	Señal de Sensor (ZE2) mesa en posición
T4	Sensor de posición (ZE3) y señal de control encender remachadora
T5	Señal de control apagar remachadora
T6	Sensor de posición (ZE4) y señal de control encender pulverizador.
T7	Señal de control apagar pulverizador
T8	señal de control encender mesa giratoria o Señal al presionar el botón de start (Z)
T9	Sensor de posición (ZE5) y señal de control bajar robot manipulador 1.
T10	Señal de control cerrar pinza robot manipulador 1.
T11	Señal de control subir robot manipulador 1.
T12	Señal de control girar a la izquierda robot manipulador 1.
T13	Señal de control bajar robot manipulador 1.
T14	Señal de control abrir pinza robot manipulador 1.
T15	Señal de control subir robot manipulador 1.
T16	Señal de control girar a la derecha robot manipulador 1.
T17	Sensor de posición (ZE6) y señal de control encender máquina de prueba de resistencia
T18	Señal de control apagar máquina de prueba de resistencia
T19	Señal de control rechazo de pieza
T20	Señal de control encender eyector
T21	Señal de control apagar eyector
T22	Señal de control aprobación de pieza
T23	señal de control bajar robot manipulador 1.
T24	Señal de control cerrar pinza robot manipulador 1.
T25	Señal de control subir robot manipulador 1.
T26	Señal de control girar a la izquierda robot manipulador 1.
T27	Señal de control bajar robot manipulador 1.
T28	Señal de control abrir pinza robot manipulador 1.
T29	Señal de control subir robot manipulador 1.
T30	Señal de control girar a la derecha robot manipulador 1.

Fuente: elaboración propia

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27	T28	T29	T30
P0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P1	-1	-1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P2	1	1	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P3	0	0	1	1	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P4	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P5	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P6	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P7	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P8	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P9	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P10	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P11	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P12	0	0	0	0	0	0	0	0	0	1	0	-1	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P13	0	0	0	0	0	0	0	0	0	-1	0	1	0	-1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P14	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P15	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P16	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P17	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0
P19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0	0	0	0
P20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0
P21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	-1	0	0	0	0	0	0	0	0
P22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0	0
P23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	0
P24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0	0	0
P25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
P26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0	0	0	0	0	0
P27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	1
P28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	1	0	-1	0
P29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	-1	0	1	0
P30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	2	0	0
P31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1	0	0
P32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	-1
P33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1
P34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figura 14. Matriz de incidencia del sistema
Fuente: elaboración propia

$$M_{p0} = [1; 1; 0; 0; 0; 0; 1; 0; 0; 1; 0; 1; 0; 1; 1; 0; 0; 1; 0; 1; 0; 0; 0; 1; 0; 0; 0; 1; 0; 1; 1; 0; 0; 1; 0]$$

Figura 15. Vector columna del marcado inicial
Fuente: elaboración propia

La Figura 13, muestra la red de la planta, en ella se identifica la cantidad de lugares y transiciones que hacen parte de la red, para el caso de estudio se tiene 35 lugares y 31 transiciones, por lo tanto, la matriz de incidencia del sistema (Figura 14), denotada por $Cp \in Z^{35X31}$. tiene un tamaño de 35 filas por 31 columnas. Debido al tamaño de la red del proceso, generar un árbol de alcanzabilidad es complejo.

2.3. Diseño del controlador

El diseño del control del sistema está basado en [39], donde utilizan las redes de Petri para crear monitores o supervisores que impiden la activación de transiciones, las cuales llevan a estados prohibidos o al bloqueo de la red. El diseño del control se hace mediante el enfoque de supervisor calculado, donde las acciones del control se efectúan sobre la misma red de Petri; para esto, se deben generar ciertas restricciones de funcionamiento de acuerdo al proceso.

2.3.1. Restricciones del sistema

Las restricciones del sistema son aquellos estados o lugares descritos en la red de Petri (Figura 13) que son prohibidos o que no pueden estar activados a la misma vez que otros. Estas restricciones pueden ser físicas que pueden dañar o entorpecer el funcionamiento de las máquinas de operación o de funcionamiento dependiendo del sistema. Las restricciones se redactan a continuación:

La mesa giratoria no puede girar si la remachadora o el pulverizador de pintura están operando. El eyector ubicado en la zona de prueba, no puede activarse si la máquina de prueba de resistencia está operando; del mismo modo, si la mesa de resistencia está operando, el robot manipulador 1 no podrá dirigirse hacia ella, al igual que el robot manipulador 2. Lo anterior, indica cinco restricciones físicas entre estados del sistema. La Tabla 24, contiene los lugares asociados a las restricciones.

Tabla 24. Lugares asociados a las restricciones

Lugares	Descripción
P2	Mesa operando
P5	Remachadora operando
P8	Pulverizador operando
P16	Robot manipulador 1 a la izquierda
P20	Máquina de prueba de resistencia operando
P24	Eyector operando
P28	Robot manipulador 2 abajo

Fuente: elaboración propia

De acuerdo con [39], cada una de las restricciones puede expresarse como una inecuación de la forma $m(p_x) + m(p_y) \leq 1$; donde la suma de las marcas en los lugares Px y Py no pueden ser mayor a uno, lo que quiere decir que una de las dos puede operar. Por lo tanto, cada una de las restricciones del proceso se expresan en forma de inecuación.

- La mesa giratoria y la remachadora no pueden operar al mismo tiempo.

$$m(P2) + m(P5) \leq 1 \quad (10)$$

- La mesa giratoria y el pulverizador no pueden operar al mismo tiempo.

$$m(P2) + m(P8) \leq 1 \quad (11)$$

- La máquina de prueba de resistencia y el eyector no pueden operar simultáneamente.

$$m(P20) + m(P24) \leq 1 \quad (12)$$

- El robot manipulador 1 no puede girar hacia la máquina de prueba de resistencias, si ella está operando.

$$m(P16) + m(P20) \leq 1 \quad (13)$$

- El robot manipulador 2, no puede bajar hacia la máquina de prueba de resistencia, si ella está operando

$$m(P20) + m(P28) \leq 1 \quad (14)$$

Con las restricciones se obtiene la matriz L, donde las filas representan la cantidad de restricciones y las columnas son los lugares del sistema. De igual manera, de obtiene un vector columna b que representa la suma máxima de las marcas asociadas a la restricción.

$$\begin{bmatrix} 0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

L b

Figura 16. Representación matricial de las restricciones
Fuente: elaboración propia

Con la matriz de incidencia del sistema (Figura 14) y la matriz de restricciones L (Figura 16), se obtiene la matriz del controlador utilizando la ecuación:

$$C_c = -L * C'_p \quad (15)$$

Por otra parte, el marcado inicial del controlador esta descrito por la ecuación:

$$M_{c0} = b - L * M_{p0} \qquad (16)$$

$$Cc = \begin{bmatrix}
-1 & -1 & 1 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0
\end{bmatrix}$$

Figura 17. Matriz del controlador
Fuente: elaboración propia

$$Mc0 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Figura 18. Marcado inicial del controlador
Fuente: elaboración propia

Las filas de la matriz Cc (Figura 17) representan los nuevos cinco lugares que van a controlar cada restricción del sistema y las columnas son las transiciones ($C_{C_{5x31}}$). En la matriz Cc, valor de 1 representa las transiciones de entrada y -1 representan las transiciones de salida de los lugares. La matriz de la Figura 18, muestra el marcado inicial que tiene cada monitor. La implementación de los nuevos lugares se evidencia de la Figura 19.

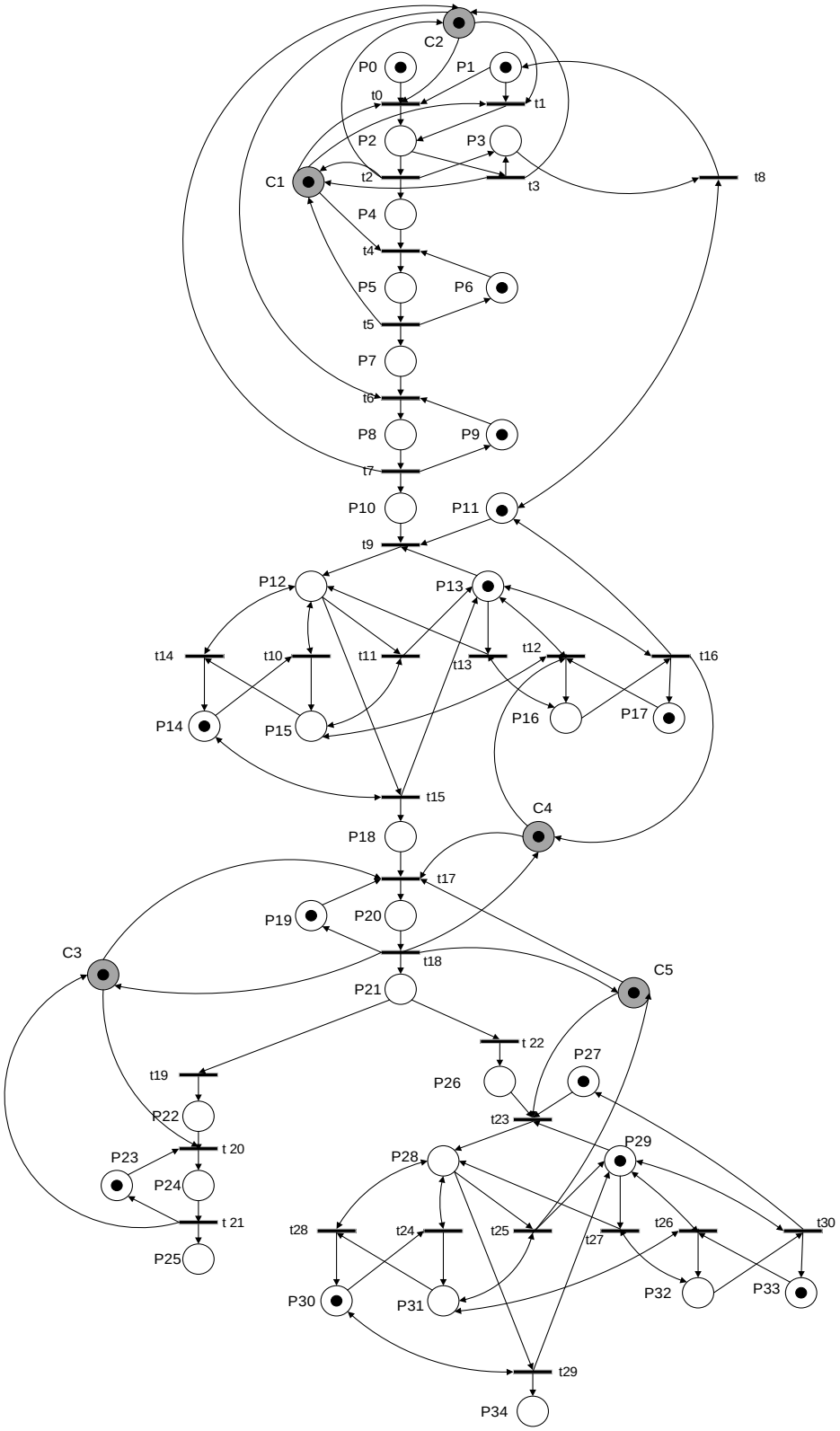


Figura 19. Red de Petri del sistema controlado
Fuente: elaboración propia

Hay que tener en cuenta que el monitor C5 es el que se encarga de impedir que el robot manipulador 2 baje si la máquina de prueba de resistencia está operando, sin embargo, el robot puede bajar a recoger pieza o a soltarla lo que significa que la transición 27 y 29 están asociadas la descarga del robot y no son tenidas en cuenta como entrada y salida del monitor C5.

A continuación, se realiza en análisis de admisibilidad para tener certeza de que los monitores no están relacionados con transiciones no controlables. Las transiciones no controlables son aquellas en las cuales su disparo no puede ser detenido por el controlador, para el sistema las transiciones no controlables son las que tienen asociada al menos un sensor. Con ayuda de la Tabla 23, se analizan las transiciones no controlables y se describen en forma matricial en la Figura 20.

$$Cuc = \begin{bmatrix}
-1 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

Figura 20. Matriz de transiciones no controlables
Fuente: elaboración propia

$$LCuc = \begin{bmatrix} 1 & 1 & -1 & 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 21. Matriz de admisibilidad
Fuente: elaboración propia

Para que L sea admisible, debe cumplirse que $LC_{uc} \leq 0$, o también $LC_{uc} \geq 0$. Como se muestra en la Figura 21, L no es admisible ya que contiene valores de 1 y -1. Por lo tanto, se realizan operaciones entre filas a través de la matriz compuesta descrita en la Figura 22.

$$\begin{bmatrix} C_{uc} & I & 0 \\ LC_{uc} & R1 & R2 \end{bmatrix} =$$

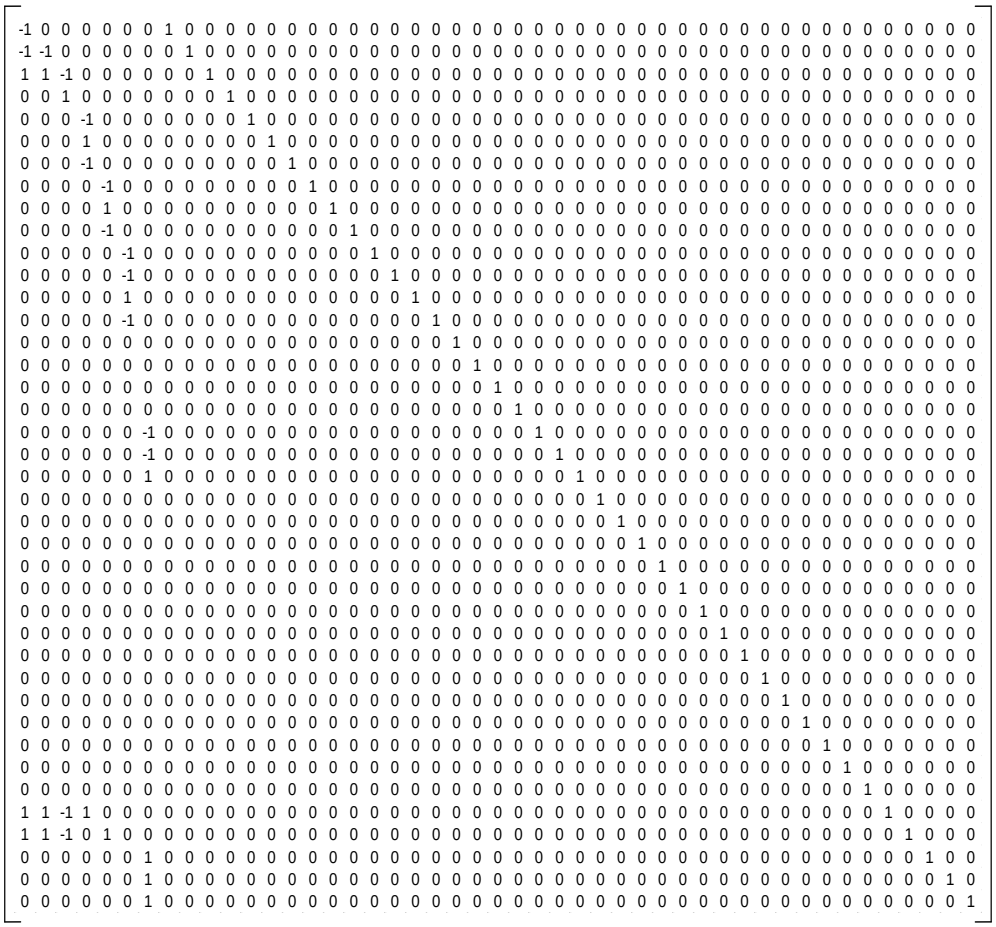


Figura 22. Matriz compuesta para operación por filas
Fuente: elaboración propia

El objetivo de las operaciones entre filas es llevar a la matriz LCuc a valores menores o iguales a cero, las operaciones que se realizan entre filas son las siguientes:

- Primera fila LCuc: fila 2 + fila 5 + fila 36
- Segunda fila LCuc: fila 2 + fila 8 + fila 37
- Tercera fila LCuc: fila 19 + fila 38
- Cuarta fila LCuc: fila 19 + fila 39
- Quinta fila LCuc: fila 19 + fila 40

Realizadas las operaciones entre filas se obtienen la matriz R1 y R2.

$$\begin{bmatrix} C_{uc} & I & 0 \\ LC_{uc} & R1 & R2 \end{bmatrix} =$$

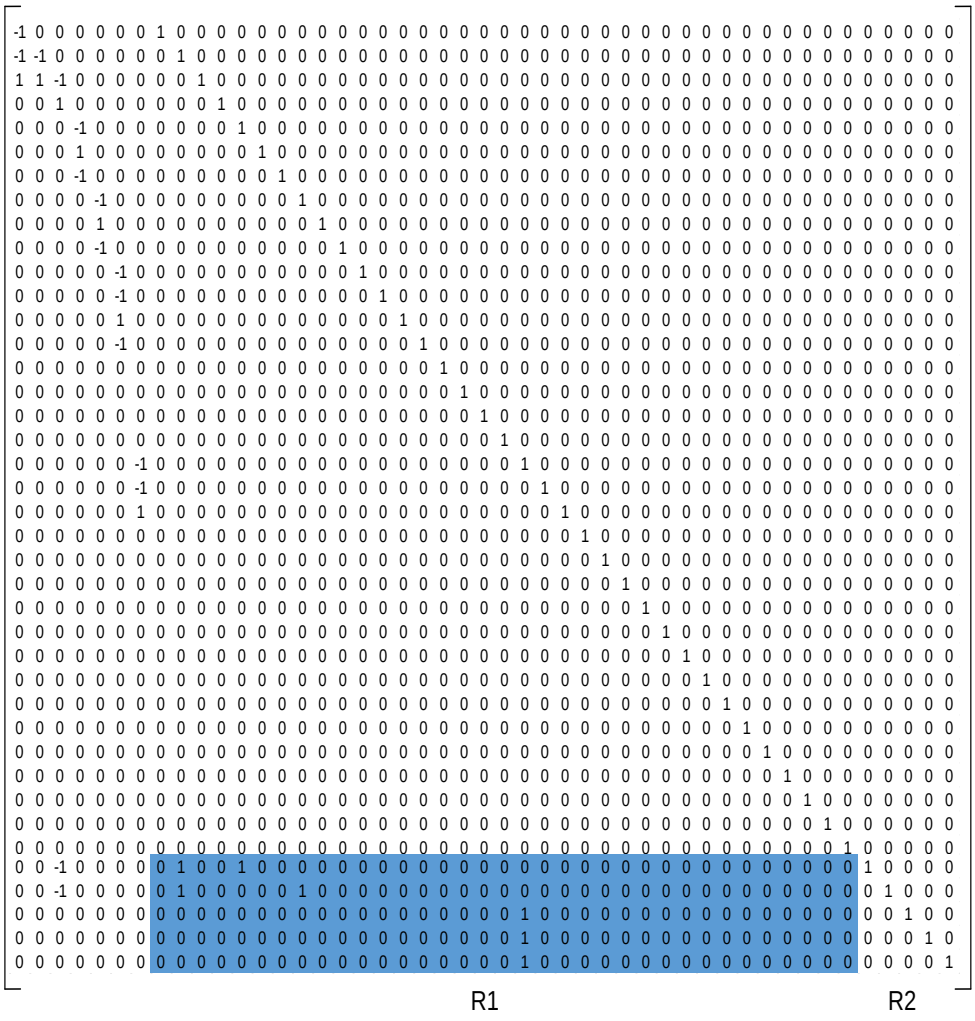


Figura 23. Matriz compuesta con operaciones de filas
Fuente: elaboración propia

Las operaciones entre filas para transformar los valores de LCuc menores o iguales a cero, da como resultado la matriz R1 sombreada en la Figura 23. Se genera el nuevo controlador utilizando la ecuación:

$$C_c = -(R_1 + R_2 L) C_p \quad (17)$$

Su marcado inicial utilizando la ecuación:

$$M_{c0} = R_2(b + \mathbf{1}) - \mathbf{1} - (R_1 + R_2 L) M_{p0} \quad (18)$$

$$Cc = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0
\end{bmatrix}$$

Figura 24. Matriz del controlador admisible
Fuente: elaboración propia

$$Mc0 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Figura 25. Marcado inicial del controlador admisible
Fuente: elaboración propia

Con la matriz del controlador admisible descrita en la Figura 24 y su marcado inicial visto en la Figura 25. Se genera una nueva red de Petri del sistema controlado, donde las salidas de los lugares son dirigidas únicamente a transiciones controlables. La matriz de incidencia del sistema ahora está compuesta por la matriz de incidencia del sistema más la matriz del controlador y el marcaje inicial del sistema de igual manera estará compuesta por el marcaje inicial del sistema y el marcaje inicial del controlador. El sistema total se observa en la Figura 26.

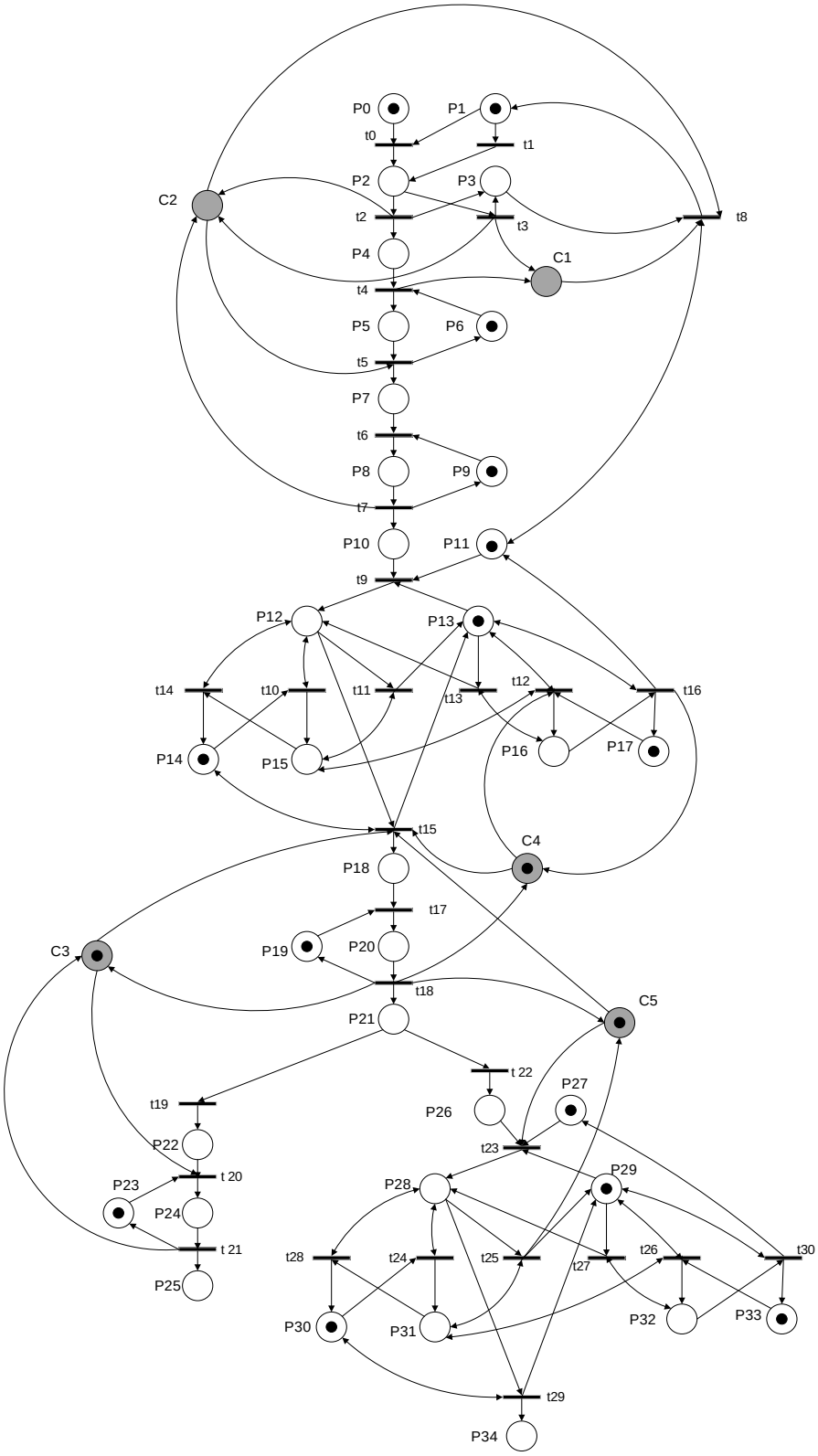


Figura 26. Red de Petri del sistema controlado con admisibilidad
Fuente: elaboración propia

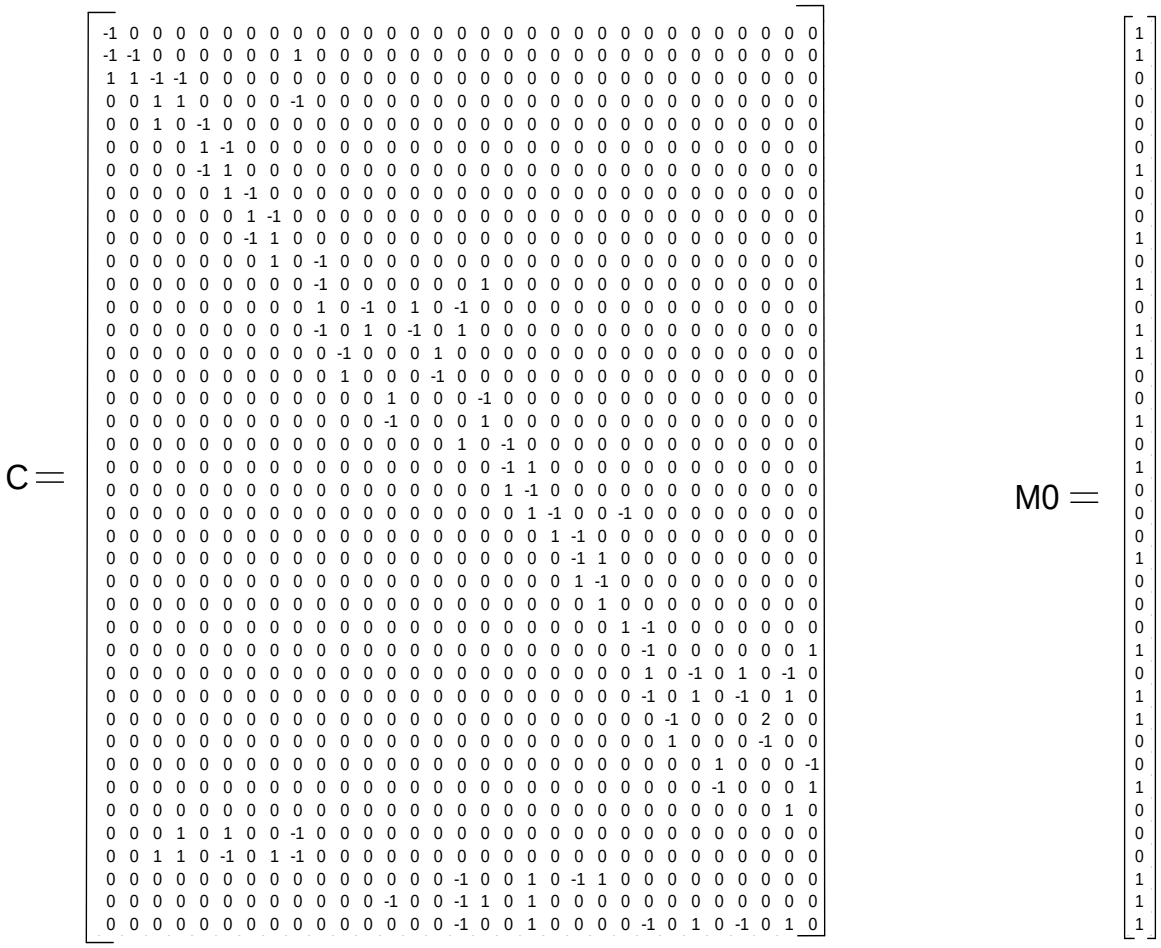


Figura 27. Matriz de incidencia y marcado inicial del sistema
Fuente: elaboración propia

Como resultado del método, se ha logrado obtener el diseño de un sistema de manufactura controlado.

Capítulo 3

Simulación y experimentación

3.1. Características del entorno grafico

Para profundizar de acuerdo a factores ya evaluados en la Tabla 2, se brinda una descripción detallada del software Unity acompañada por gráficos, que representan de una manera clara la interfaz de la herramienta, cuya finalidad es familiarizarse con el entorno gráfico y entender cómo se elabora un proyecto básico en Unity, donde se pueden encontrar diferentes componentes de animación, configuración de fuerzas físicas, figuras geométricas, colores, texturas, editores de programación entre otros parámetros y utilidades que brinda la herramienta, esto con el fin de realizar una simulación que comprende un entorno gráfico avanzado.

Unity es un software desarrollado para videojuegos en esencia, sin embargo se puede evidenciar que este software, se utiliza para fines académicos, puesto que esta herramienta tiene como lenguaje de programación C# que es un lenguaje que se utiliza ampliamente en el mundo de la programación, tiene visualización en 3D y 2D, posee componentes de configuración física para los objetos, se puede también importar gráficos diseñados en otras plataformas de diseño, además tiene una tienda virtual, donde se pueden realizar descargas gratuitas y otras de pago, este software es complemente gratuito para la versión estudiante, lo que brinda una mayor ventaja, ya que cuenta con características, que son significativas para el desarrollo del proyecto de simulación del sistema de manufactura. La descripción grafica del entorno de Unity se puede observar en el Anexo A.

3.2. Marco experimental del modelo.

3.2.1. Entradas:

El entorno de simulación ya cuenta con todo el desarrollo previo de diseño, caracterización de sensores y conectividad a la base de datos, para la toma de registros, sin embargo, se ha optado por añadir variables de usuario al inicio de la simulación, que permiten hacer que el proceso no sea monótono y que se pueda

interactuar con la interfaz gráfica cambiando ciertos parámetros de fabricación y toma de registros. Gráfica 1, muestra la interfaz de parámetros modificables por cada simulación que se realice, ella cuenta con los siguientes parámetros:

Gráfica 1. Parámetros de inicio de simulación del sistema de manufactura.
Fuente: elaboración propia en Unity

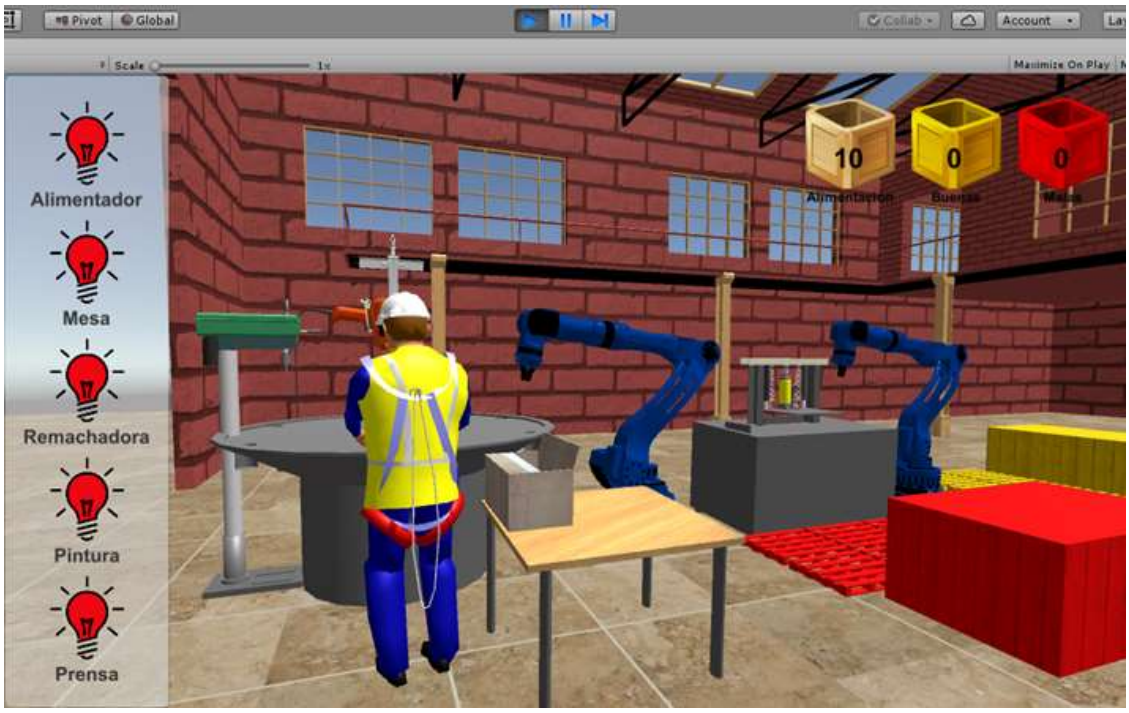
- **Número de productos a fabricar**
  Sección modificable del número de piezas a fabricar (mosquetones), donde el límite máximo permitido es hasta 500 piezas.

- **Porcentaje de daño del producto**
  Parámetro para ingresar un dato numérico que permita tener un porcentaje de daño en las piezas que se van a fabricar.

- **Ingreso de segundos de captura**
  Este parámetro modificable va directamente relacionado con la base de datos, puesto que se puede ingresar un valor numérico en segundos, que permita capturar el estado de los sensores, este valor de captura depende también del número de registros que se desea que muestre la base de datos, el valor permitido va desde 1 segundo hasta 10 segundos, donde cada segundo que se demora la simulación es un registro de base de datos.

- **Borrar la base de datos**
  Permite que al iniciar una simulación nueva se puedan borrar los registros de la base de datos de simulaciones anteriores, si no es el caso, los registros nuevos se sumaran a los registros que se hayan dado en anteriores simulaciones.

Una vez establecidas las configuraciones que se desean en el proceso, se da inicio a la simulación, esto se puede ver en la, donde se observa un entorno grafico completo acompañado del Canvas que es un área dedicada a una interfaz en 2D dentro de la simulación, la cual permite ver información del proceso sin importar donde esté ubicada la cámara de visualización en el proceso, en otras palabras, el Canvas jamás desaparece de la pantalla de visualización.



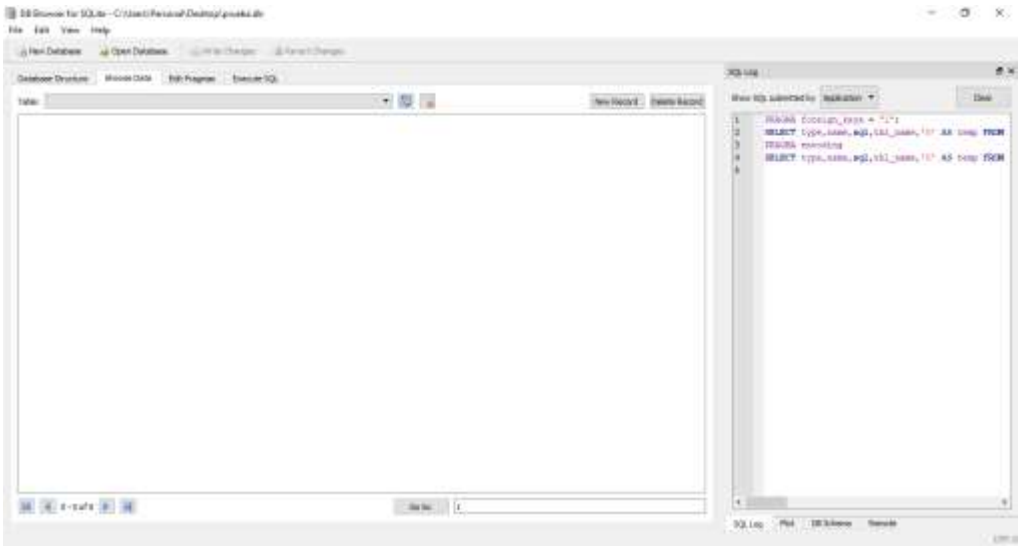
Gráfica 2. Aspecto final del entorno de simulación en Unity.
Fuente: elaboración propia en Unity

En la Gráfica 2, el Canvas está relacionado con dos de los parámetros de modificación de inicio y con la visualización del estado de los sensores de posición de la instrumentación del proceso. Para los parámetros de inicio se relacionan con las cajas que se encuentran en la parte superior derecha del entorno, donde la caja de color café representa el número de piezas a fabricar, la caja amarilla representa el número de piezas en buen estado, y la caja roja representa las piezas defectuosas del proceso. Por otra parte, se encuentran el estado de los sensores de posición del proceso representados con bombillos que encienden rojo para indicar que no hay pieza y verde para indicar que si se encuentra una pieza.

3.2.2. Salidas:

Como parte de la investigación del proyecto, se han planteado alternativas para extraer las señales de los elementos presentes en el proceso, que para el caso particular serían los GameObjects presentes en la escena de simulación, los cuales representan los instrumentos necesarios para la fabricación de mosquetones, estos a su vez gracias al código en C# asociados a ellos, dan parámetros del estado del proceso que se está desarrollando en simulación, debido a la codificación y configuración implementada en el editor de Unity, donde la finalidad es visualizar dichas señales en otro plataforma software, en otras palabras es observar el estado de los sensores del proceso en determinado momento representados como unos y ceros, es así como se pretende hacer ciertas pruebas de visualización de señales llevándolas a plataformas software, que permitan el análisis de su comportamiento en escena; por tanto para validar el diseño se ha propuesto llevar las señales del proceso a un registro en una base de datos, cuyo propósito es plasmar de manera más acertada y visible las señales que emergen de la instrumentación del sistema.

Para comenzar se hace un análisis de la plataforma más conveniente, en cuanto a la bases de datos se refiere, esta debe ser gratuita y de fácil manejo, además de tener una interfaz comprensible al usuario, por este motivo se ha optado por emplear la base de datos conocida como; **SQLite**, la cual provee las características necesarias para la ejecución del proyecto, en la Gráfica 3, se puede observar la interfaz de SQLite, donde en su lado izquierdo se pueden visualizar los registros y en el lado derecho se puede visualizar las líneas de código que se van ejecutando en el trascurso de la toma de datos, además en esta misma se pueden crear y modificar lar tablas e índices entre muchas otras funcionalidades de programación y registro.



Gráfica 3. Interfaz de usuario SQLite.
Fuente: elaboración propia en Unity

3.2.3. Tablas para la base de datos:

Una vez definida la herramienta de bases de datos, se procede al diseño de tablas de funcionamiento y requerimientos para implementarla en la base de datos. Estas tablas contienen el registro con el nombre del instrumento y sensor pertenecientes al sistema de manufactura. La Gráfica 4, muestra la tabla del diseño de los registros de la base de datos que son implementados en SQLite, donde cada variable es asociada a un registro proveniente de la instrumentación de la simulación en Unity.

Gráfica 4. Tabla de diseño de registro de la base de datos.
Fuente: elaboración propia en Unity

La llave que se observa en “id_registro” es la llave primaria de la tabla de registros, es la forma única de identificación de un registro, es decir no puede haber dos registros con la misma identificación. Esta característica sirve especialmente para que, sin importar el número de simulaciones realizadas, se lleve una cuenta de los registros tomados en simulaciones anteriores.

3.2.3.1. Definición del registro de sensores

- **s_alimentador:** Se almacena un valor numérico, para identificar en que momento está activo e inactivo el sensor de alimentación.
- **s_mesa:** Se almacena un valor numérico, para identificar en qué momento se está moviendo la mesa de ensamblaje.
- **s_remachadora:** Se almacena un valor numérico, para identificar en que momento está activo e inactivo el sensor de la remachadora.
- **s_pintura:** Se almacena un valor numérico, para identificar en qué momento se activa la pistola de pintura.

- **s_pinza1:** Se almacena un valor numérico, para identificar en qué momento se abre y cierra la pinza del primer robot en la línea de ensamblaje.

- **s_robotGiro1:** Se almacena un valor numérico, para identificar en que momento está activo e inactivo el giro del primer robot en la línea de ensamblaje.

- **s_robotBrazo1:** Se almacena un valor numérico, para identificar en que momento está activo o inactivo el brazo del primer robot en la línea de ensamblaje.

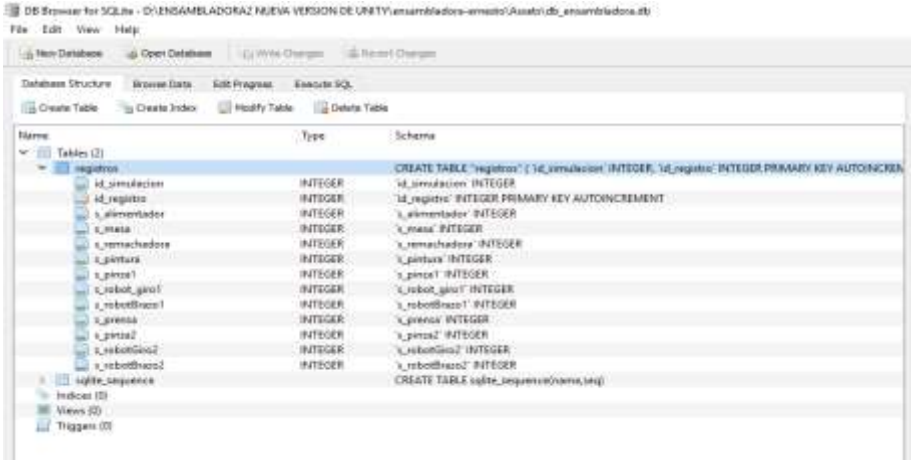
- **s_prensa:** Se almacena un valor numérico, para identificar en que momento está activo e inactivo el sensor de la prensa.
- **s_robotGiro2:** Se almacena un valor numérico, para identificar en qué momento está activo e inactivo el giro del segundo robot en la línea de ensamblaje.

- **s_robotBrazo2:** Se almacena un valor numérico, para identificar en que momento está activo o inactivo el brazo del segundo robot en la línea de ensamblaje

- **s_pinza2**: Se almacena un valor numérico, para identificar en que momento está activo e inactivo el segundo robot de la línea de ensamblaje.

3.2.4. Implementación de las tablas de registro en SQLite

Para visualizar los datos provenientes de la simulación, se implementa el diseño de la Gráfica 5 en SQLite, realizando la estructura con el respectivo nombre y tipo de dato.



Gráfica 5. Creación de la tabla de registros en SQLite.
Fuente: elaboración propia en Unity

En la Gráfica 5, se puede observar que la tabla de registros en la base de datos coincide con el diseño realizado, donde se aprecian los nombres alusivos a la instrumentación del proceso, lo que hace fácil una tarea de reconocimiento de los datos que son objeto de análisis. También se puede apreciar el tipo de dato, que para este caso al ser un sistema de eventos discretos la opción más pertinente es el tipo de dato INTIGER que en su codificación se determina si es un 1 o un 0, que serán plasmados en la data de SQLite.

3.2.5. Extracción de señales

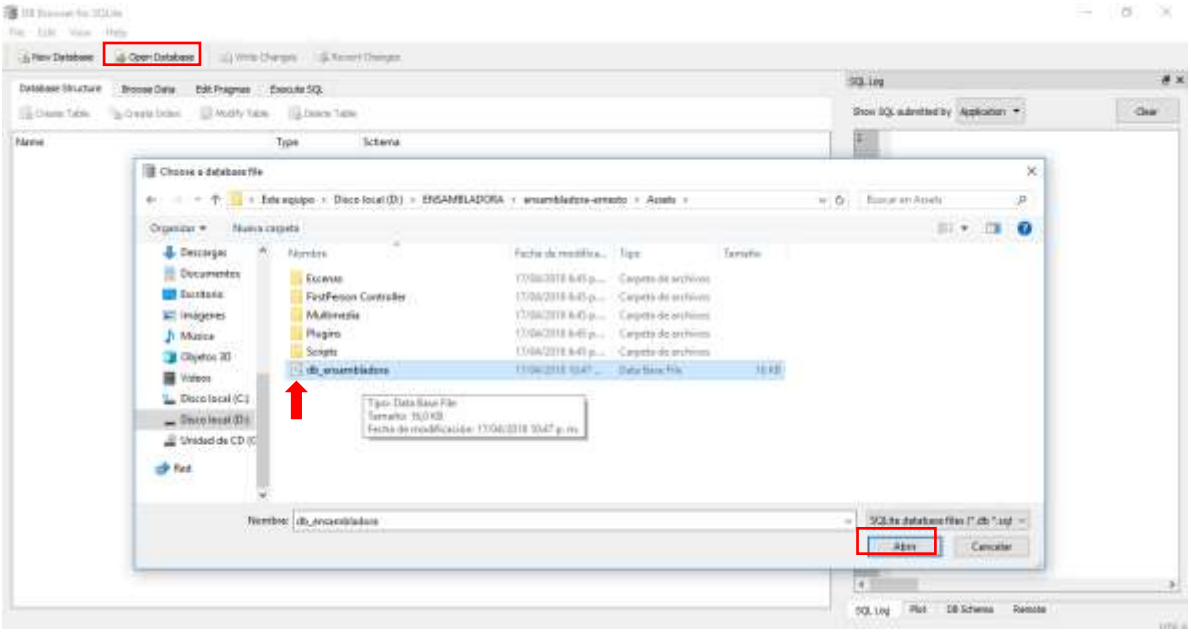
La extracción de datos pertenecientes al sistema de manufactura son enviados mediante un código realizado en Unity de nombre “Base de Datos” que enlaza o permite comunicar las dos plataformas software, esto se da gracias a la librería incluida en el código llamada “Mono.Data.Sqlite”, una vez incluida esta librería esencial se procede a desarrollar el código de recepción enlace y envío de datos.

Para empezar, se hace un listado de variables que están relacionadas con los instrumentos, las cuales proveerán el estado del mismo, activo o inactivo, luego se especifica a qué tipo de dato de Unity pertenecen esas variables, dependiendo el nombre que se le asigne a cada una de ellas en el código, éstas deben aparecer etiquetadas en con el mismo nombre en el registro de la base de datos para que haya la relación en la comunicación. Seguidamente se crean los espacios de la lista de sensores asociados a las variables, con el fin de permitir un registro de cada uno de ellos con respecto al estado de la simulación. Después se elabora la lógica de programación que permite saber el estado, el objeto o instrumento. Luego se desarrolla una función llamada “guardarDatos” que recopila toda la información de los sensores para enviarlos a la función “insertar”, donde se activa el enlace para tener comunicación y después se inserta la información en la base de datos. Por último, se desarrolla la función de consulta del proceso, cuya finalidad es evaluar cada cierto tiempo el estado de las variables cuando la simulación este en curso, esto permite que las funciones de guardar datos e insertar, estén actualizadas en cuanto a los valores del proceso que reciben y envían la base de datos.

Para añadir una variable sensorial, después de que el código ha sido realizado, si el proceso lo demanda, se asocia una variable al objeto que se desee censar, esa variable debe contener un espacio en el script de base de datos, luego se coloca el nombre con el cual se quiere que aparezca en la base de datos y se ingresa en guardar datos, después ese mismo nombre dado en el código se inserta en la creación de la tabla como un registro para su posterior visualización.

Por último, se realiza la conexión a la base de datos, para capturar cualquier registro de los sensores involucrados en el proceso, esto se efectúa de la siguiente manera como se ve en la Gráfica 6.

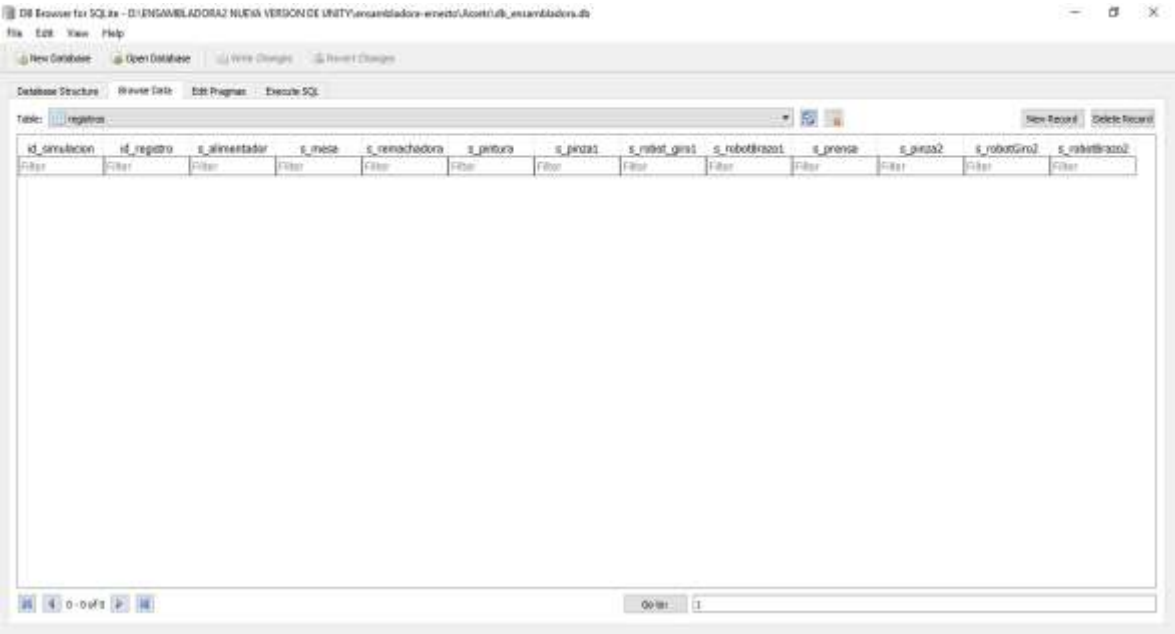
Se abre la herramienta de base de datos, luego se dirige a Open Database en la interfaz de usuario, seguidamente se busca en el explorador la carpeta del proyecto desarrollado en Unity, y se busca el nombre de la base de datos que ha sido previamente programado.



Gráfica 6. Conexión y activación de la base de datos.
Fuente: elaboración propia en Unity

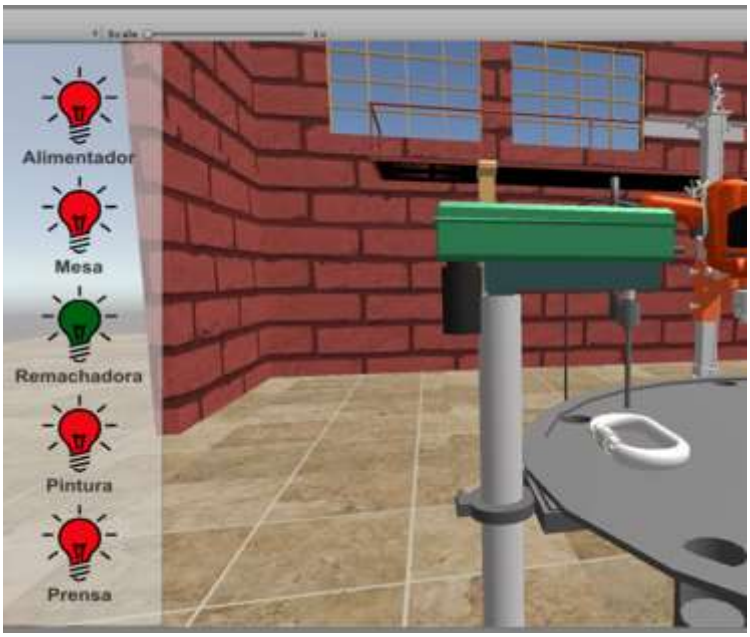
La base de datos que se presenta en la Gráfica 7, muestra los diferentes sensores del proceso prestos a capturar información proveniente de la simulación de Unity, además cuenta con dos capturas más, para un mejor control de los registros, una de ellas es “id_simulacion” que permite visualizar cuantas simulaciones se han efectuado sin importar el número de registros que la simulación contenga. La otra captura de información se presenta en “id_registro” la cual contabiliza cuantas tomas de datos se han realizado desde el inicio hasta el fin de la simulación.

A partir de ahora con todo el despliegue de herramientas software, conexiones y configuraciones ya realizadas, se pueden hacer pruebas de funcionamiento del proyecto de simulación de un sistema de manufactura en lazo cerrado, que permita observar el comportamiento de la instrumentación simulada en Unity ligada a los registros de la base de datos de SQLite, gracias a toda la programación implementada en proyecto.



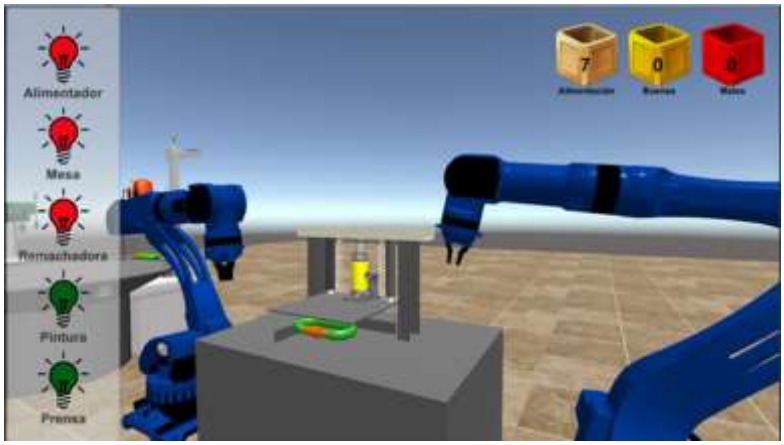
Gráfica 7. Base de datos para la simulación del sistema de manufactura.
Fuente: elaboración propia en Unity

3.3.6 Restricciones físicas del sistema en simulación

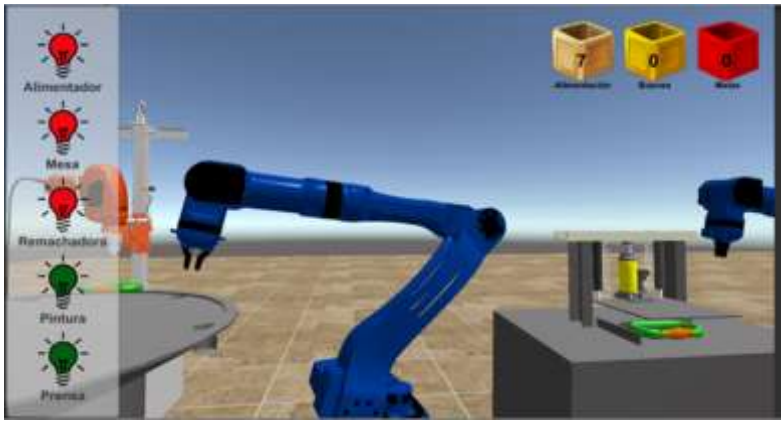


Gráfica 8. Restricción física 1.
Fuente: elaboración propia en Unity

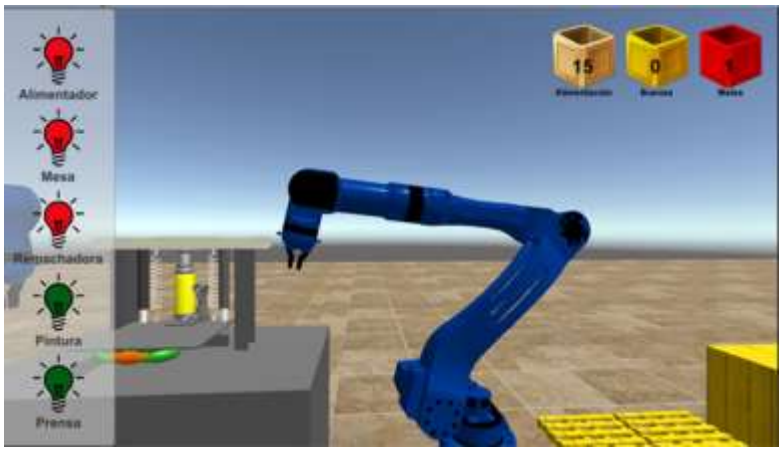
Gráfica 9. Restricción física 2.
Fuente: elaboración propia en Unity



Gráfica 10. Restricción física 3.
Fuente: elaboración propia en Unity



Gráfica 11. Restricción física 4.
Fuente: elaboración propia en Unity



Gráfica 12. Restricción. física 5.
Fuente: elaboración propia en Unity

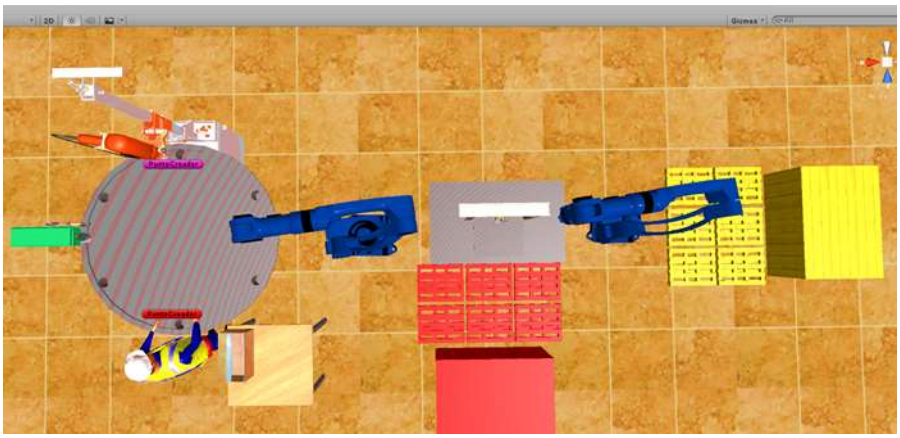
3.3. Modelo físico base de simulación.

La Figura 28, muestra las diferentes etapas que se desarrollan para la simulación de un sistema de manufactura, en total son 7 pasos o etapas por las cuales se desarrolla el proyecto, donde se hace referencia en el inicio al conocimiento del proceso que se quiere crear en un entorno gráfico, después de conocer la instrumentación que hace parte del proceso de manufactura, se hacen los respectivos diseños de piezas en 3D que serán puestas en escena, seguidamente se realiza el ensamblaje y configuración física de las piezas que hacen parte del proceso, luego se prueba el ensamblaje y las configuraciones físicas en el software, teniendo en cuenta los movimientos realistas y la coordinación de los mismos y para finalizar se desarrollan códigos de programación para un mayor control de todo el proceso en general.

Figura 28. Diagrama de bloques de etapas del proceso.
Fuente: elaboración propia en Unity

3.3.1 Instrumentación y objetos del proceso de manufactura en 3D

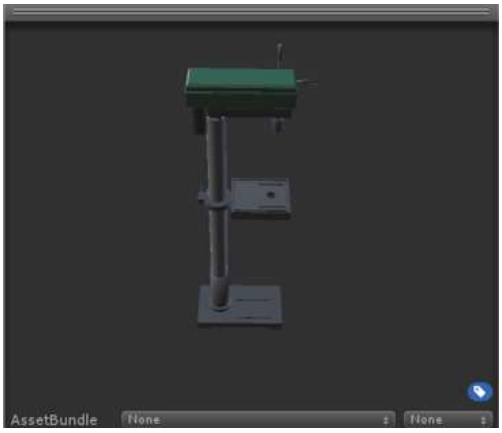
La instrumentación involucrada en el proceso de manufactura de mosquetones, cuenta con 8 diseños que componen la simulación en 3D que se ha desarrollado en Unity, los elementos que hacen parte de la escena son: mesa giratoria, remachadora, pulverizador de pintura, robot manipulador, prensa hidráulica, mosquetón, operario, fábrica, estos GameObjects se realizan teniendo en cuenta el diseño visual y las características propias de cada instrumento u objeto con respecto a su acción en el proceso y los movimientos que este posee.



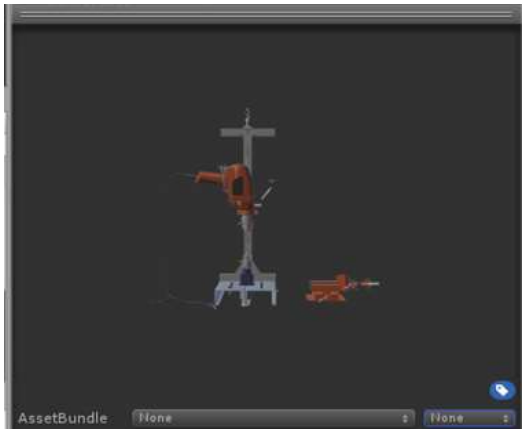
Gráfica 13. Plano de la planta en el entorno de simulación.
Fuente: elaboración propia en Unity

En la Gráfica 13, se muestra el proceso visto desde la parte superior en 2D para hacer alusión al proceso de manufactura que se encuentra en la Figura 6, solo que esta vez el proceso de manufactura ya se encuentra implementado en la herramienta de software Unity, además se ha etiquetado numéricamente, para abordar cada componente por separado.

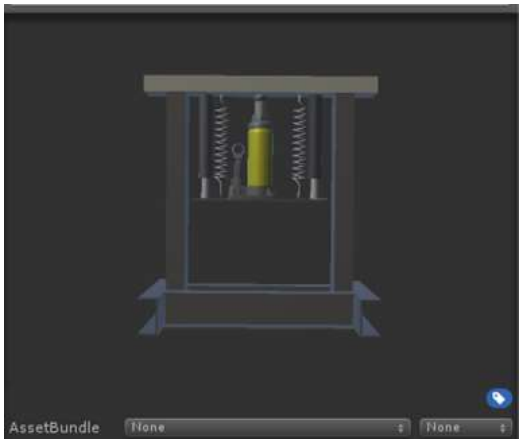
Gráfica 14. Mesa giratoria
Fuente: 3D Warehouse



Gráfica 15. Remachadora
Fuente: 3D Warehouse

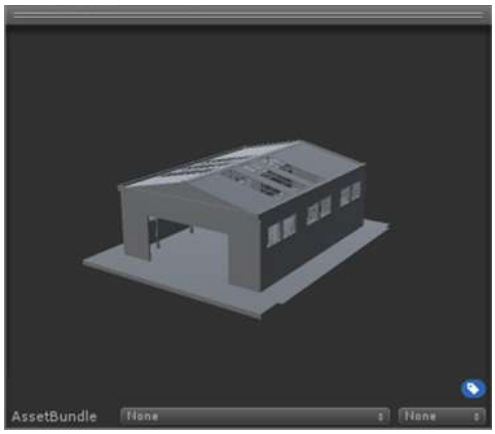


Gráfica 16. Pulverizador de pintura
Fuente: 3D Warehouse



Gráfica 17. Máquina de prueba de resistencia
Fuente: 3D Warehouse

Gráfica 18. Robot manipulador
Fuente: 3D Warehouse



Gráfica 19. Fábrica
Fuente: 3D Warehouse

Gráfica 20. Mosquetón
Fuente: 3D Warehouse

Gráfica 21. Operario
Fuente: 3D Warehouse

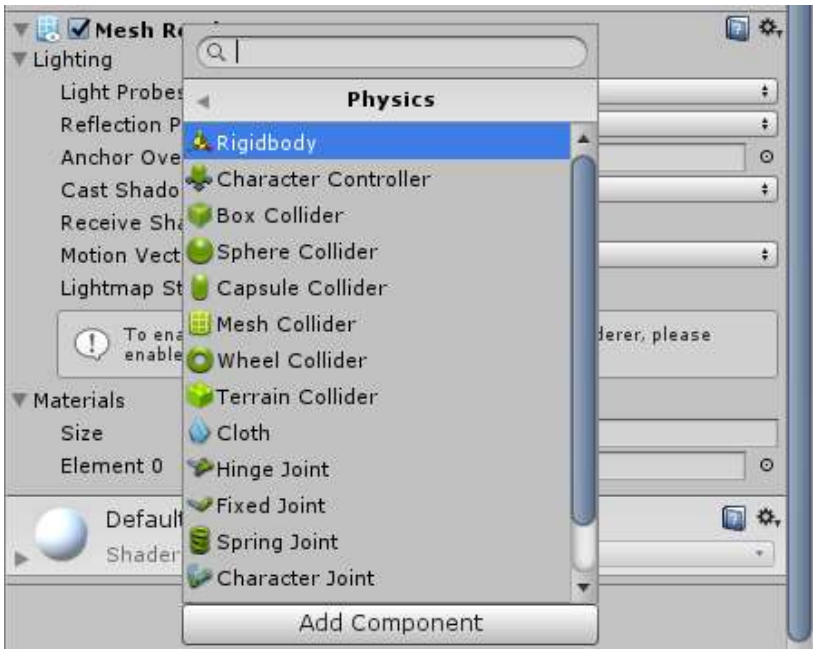
La simulación elaborada en Unity para el proceso de manufactura, cuenta con otros componentes auxiliares gráficos, con el fin de dar un mejor aspecto visual a la simulación y crear un ambiente industrial para el proceso que se está desarrollando, para ello se cuenta con los componentes gráficos en 3D como lo son; edificación de ambiente industrial (Gráfica 19), mosquetón (Gráfica 20) y un obrero Gráfica 21.

Cabe anotar que las imágenes anteriores, son tomadas del visualizador en 3D que provee el software Unity (por eso su color), estas han sido agregadas al proyecto en su mayoría de otras fuentes y bibliotecas de figuras en 3D, que han sido un gran soporte para la simulación del proceso, también cabe aclarar que algunas figuras en 3D han sido modificadas desde su origen, es decir que se toma como base un diseño CAD original y este es rediseñado en Unity, en cuanto su estructura, color, tamaño, movimiento entre otras características, con el fin de que dichas figuras se adecuen al proyecto que se está elaborando.

3.3.2 Configuración física para la instrumentación y objetos del proceso

Un aspecto a tener en cuenta para la simulación del proceso, es la física como característica de un objeto u objetos definidos como GameObjects presentes en la escena, puesto que es conveniente determinar este tipo de característica, para que los objetos sean afectados por colisiones, gravedad y otras fuerzas, que son controladas por medio scripts, para darle cierta dinámica a los objetos caracterizados con este componente. Esta física es proporcionada por el software Unity, en la barra de opciones, que se encuentra en la parte superior con el nombre de Component , donde se pueden encontrar las diferentes alternativas para anexar la física que la simulación requiera, otra forma de agregar los componentes o características físicas a los objetos, es por medio del Inspector, que en su parte final permite la opción de agregar los componentes tan solo seleccionando el objeto al que se desee incorporar ciertas características, esto se puede observar en Gráfica 22, donde la ventana Physics muestra las diferentes posibilidades.

Para el caso del proyecto, se describen los aspectos y componentes físicos más relevantes que hacen parte de la simulación, con el fin de dar a conocer como ciertos elementos de las escenas tienen movimientos convincentes o como ellos interactúan en el medio en los que han sido sumergidos, cabe anotar, que si bien Unity permite grandes posibilidades en este sentido, la versión que se utiliza para este proyecto, es una versión gratuita o de estudiante y por lo tanto se pueden perder posibilidades en cuanto a la física que es relacionada a los objetos.



Gráfica 22. Ventana de componentes físicos
Fuente: Unity

A continuación, se presentan las descripciones más importantes del componente físico que brinda Unity.

- **Rigidbody**

Es el que permite agregar a los GameObjects gravedad, colisiones, torque y fuerzas físicas para la interacción con otros objetos presentes en la escena.

- **Carácter Controller**

Se utiliza principalmente para la configuración del control de un personaje en tercera persona, es decir controla los movimientos que se pueden realizar en la escena desde el punto de vista de visualización.

- **Box Collider**

Es un colisionador en forma de cubo, se utiliza principalmente para objetos que tengan esta forma.

- **Mesh Collider**

Colisionador de precisión para GameObjects con el componente físico de Rigidbody

- **Terrain Collider**

Superficie para colisiones con la misma figura del objeto adjunto a dicha superficie

- **Cloth**

Proporciona la física para la simulación de telas o ropa de los personajes de una escena como puede ser la rigidez, aceleración, movimiento constante fricciones y colisiones.

- **Hinge Joint**

Sirve para agrupar los objetos que contengan el componente físico Rigidbody, restringiendo los movimientos para que se muevan como si estuvieran conectados por una bisagra.

- **Spring Joint**

Junta dos objetos con el componente físico Rigidbody, donde se permite que estén distanciados pero conectados con un resorte.

- **Constant Forcé**

Fuerza constante que se añade a los objetos con el componente Rigidbody, esta fuerza tiene componentes vectoriales que pueden ser configurados en los ejes x, y, z.

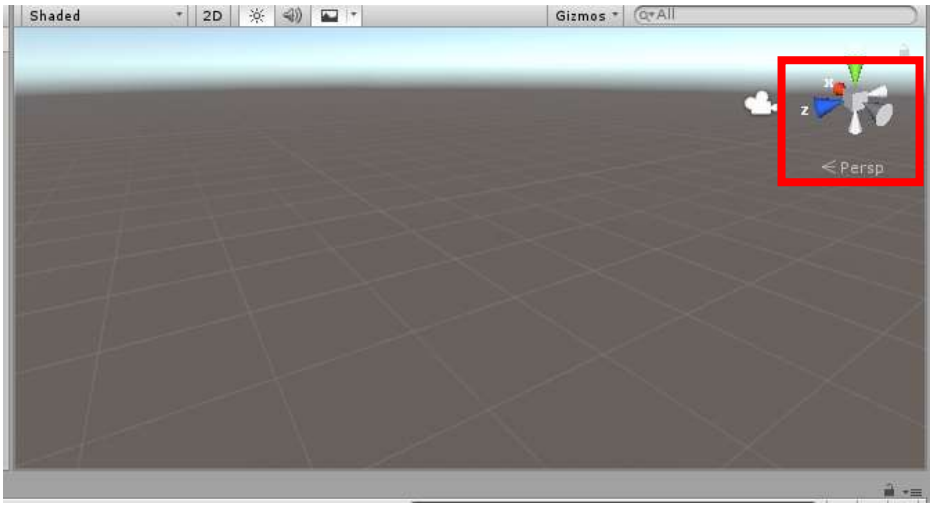
- **Carácter Joint**

Articulación extendida de la rótula, que permite limitar las articulaciones en cada eje en cuanto a giros oscilaciones y fuerzas aplicadas a la articulación.

3.4. Modelo agrupado de simulación.

Teniendo un conocimiento de las partes o piezas en 3D que componen la simulación tanto en el proceso de manufactura como el de las figuras que hacen parte del entorno para la visualización, se procede entonces a ensamblar los componentes del proceso en el ambiente propuesto por Unity (Scene), el cual cuenta con un ambiente espacial, donde se irán agregando las piezas diseñadas en 3D en sus respectivas coordenadas, ya que se pretende garantizar que haya cohesión y fluidez para la simulación.

Teniendo claro lo que se pretende desarrollar en cuanto al orden y el proceso de manufactura, se inicia manipulando los ejes coordenados x, y, z, con el fin de tener una excelente ubicación espacial, que en el transcurso del trabajo será de gran utilidad para el ensamble, diseño y programación, este eje de coordenadas se puede observar en la parte superior derecha de la pantalla, así como se ve en la **Gráfica 23**.



Gráfica 23. Entorno visual para las escenas
Fuente: Unity

Como nota se puede mencionar que los componentes gráficos también pueden ser manipulados desde el Boque 3, el cual ya se ha hecho mención y que en él se encuentra el Inspector, con la componente Transform que también da la posibilidad de hacer respectivas ubicaciones ingresando numéricamente las coordenadas pertinentes en los ejes coordenados.

Luego de tener presente la ubicación espacial, se procede a la importación de diseños en 3D, esto en el caso de ser necesario para la simulación, ya que de otra forma es utilizar los diseños que han sido elaborados o editados en Unity, estos diseños son guardados en la carpeta de Assets que se aloja en el bloque 4 y que a su vez crea una escena de la visualización del proyecto, que para este caso tan solo es una, debido a que no se necesita cambiar de locación en la simulación, todo se desarrolla bajo un mismo escenario, con la posibilidad de recorrerlo y tener varias vistas de la misma escena o mejor dicho del proceso de manufactura.

Gráfica 24. Visualización de componentes 3D
Fuente: Unity

La Gráfica 24, muestra el abanico de piezas o graficas en 3D disponibles y ya cargadas en el software Unity, como se observa en dicha gráfica, la selección que se ha realizado en los Assets o en las carpetas donde se aloja la figura, es la del robot manipulador, la cual se representa en el visualizador 3D de Unity, donde se hace una manipulación con respecto al ángulo de visión. Para que la figura haga parte del entorno de simulación solo es necesario arrastrarla hacia el espacio de trabajo de Unity o escenario, que se representa en la Gráfica 23, es aquí donde la pieza o componente gráfico adquiere ciertas características por defecto y que se pueden observar en el inspector de dicha figura, estas características son las más básicas, como puede ser la rotación , la escala , el posicionamiento , el color entre muchas otras, también es importante nombrar que a medida que se vaya necesitando más características de funcionamiento estas pueden ser agregadas y modificadas de acuerdo a la necesidad.

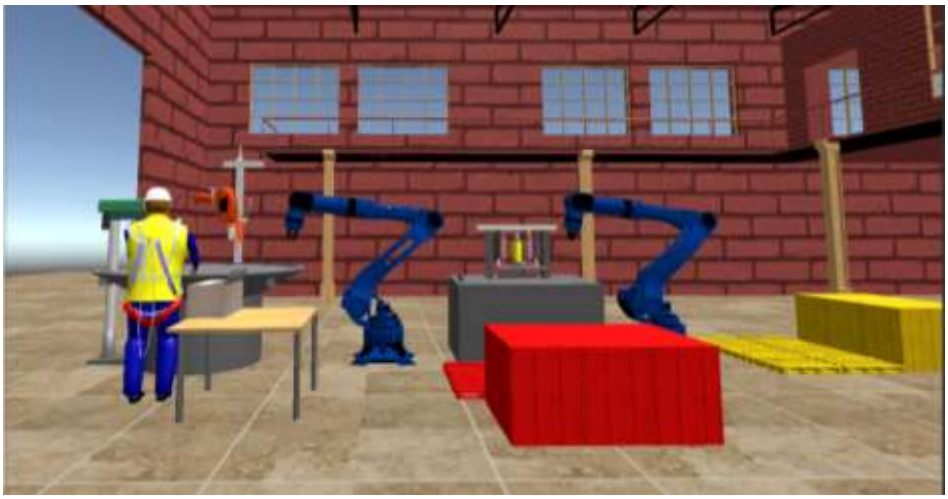
Ahora se procede a colocar todos y cada uno de los diseños y realizados, en el escenario de Unity, no sin antes decir que para que todas las piezas estén a un mismo nivel, lo más conveniente es crear un plano en 3D, donde en su superficie se podrán ir colocando los diseños y figuras necesarias para la simulación, este plano se puede encontrar en los GameObjects. Hay que tener en cuenta dos aspectos importantes a la hora de colocar una pieza, el primero es que las gráficas de los elementos en 3D, tengan un tamaño a escala real y que tenga las proporciones adecuadas y segundo que se ubiquen espacialmente teniendo un punto de referencia, como puede ser un eje de coordenadas.

3.4.1. Vista del entorno gráfico del sistema de manufactura en 3D

Gráfica 25. Vista interior y exterior del entorno de la simulación
Fuente: elaboración propia en Unity

En la Gráfica 25, se observa el plano ya realizado con una textura realiza en Unity que hace las veces de piso de la fábrica y también se observa la edificación tanto externa como internamente, a la cual se le han asignado colores y texturas para la ambientación visual de la simulación, no sobra decir que piso fue realizado con las herramientas que proporciona Unity, en el apartado GameObject se selecciona 3D Object y posteriormente se selecciona Plane, se hace énfasis en este sentido, puesto que Unity permite realizar este tipo de figuras geométricas, así como también la asignación de texturas y demás, en pocas palabras en ocasiones no se necesitan herramientas externas de diseño para poder lograr figuras geométricas básicas.

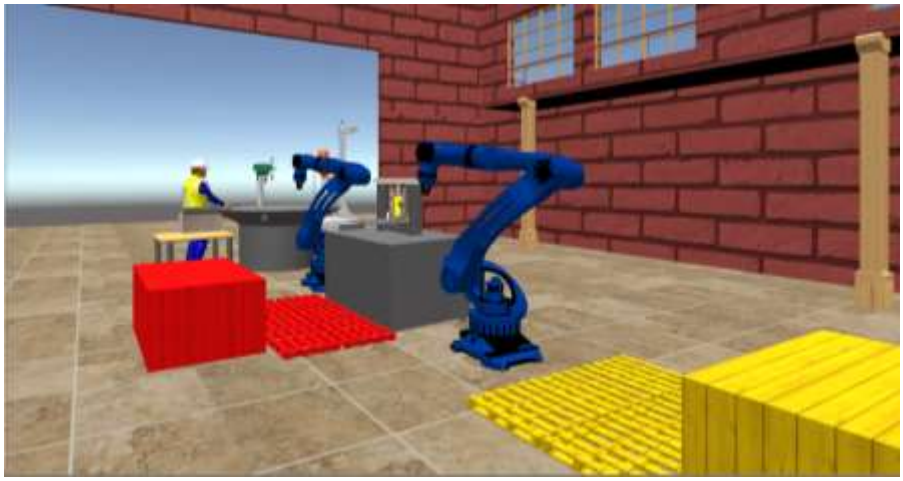
En las siguientes gráficas se observa el posicionamiento adecuado de los diseños realizados en 3D de acuerdo a los subprocesos en la manufactura y que han sido previamente descritos en capítulos anteriores, estos diseños en 3D han sido modificados en muchas de sus características para adecuarlos al proyecto, por mencionar algunas el color y las texturas, muchas de ellas escogidas de la paleta de colores que ofrece Unity y otras importadas de sitios especializados que ofrecen gran cantidad de texturas y colores, otra característica el tamaño de las piezas, puesto que la mayoría de ellas se sobredimensionan, debido a que algunas son importadas de otros programas con enfoque al diseño y que muchas veces al colocar las piezas directamente en el entorno de Unity, estas sobrepasan la escala que ya se viene manejando con otro tipo de piezas, estas dimensiones se pueden modificar desde el Inspector de la pieza que se esté evaluando, por último, las características del robot y del operario se jerarquizaron las piezas que forman parte de los diseños, esto con el fin de obtener los movimientos que requiere la simulación; ya que, estos son demasiado específicos y se necesitaba este tipo de organización; para ser más explícito Unity contempla un concepto llamado Parenting, que permite vincular las piezas con la finalidad de dar movimientos mucho más naturales y que las piezas que forman parte tanto del robot como del operario se muevan coordinadamente a la hora de hacer los respectivos códigos de programación.



Gráfica 26. Vista del proceso #1
Fuente: elaboración propia en Unity

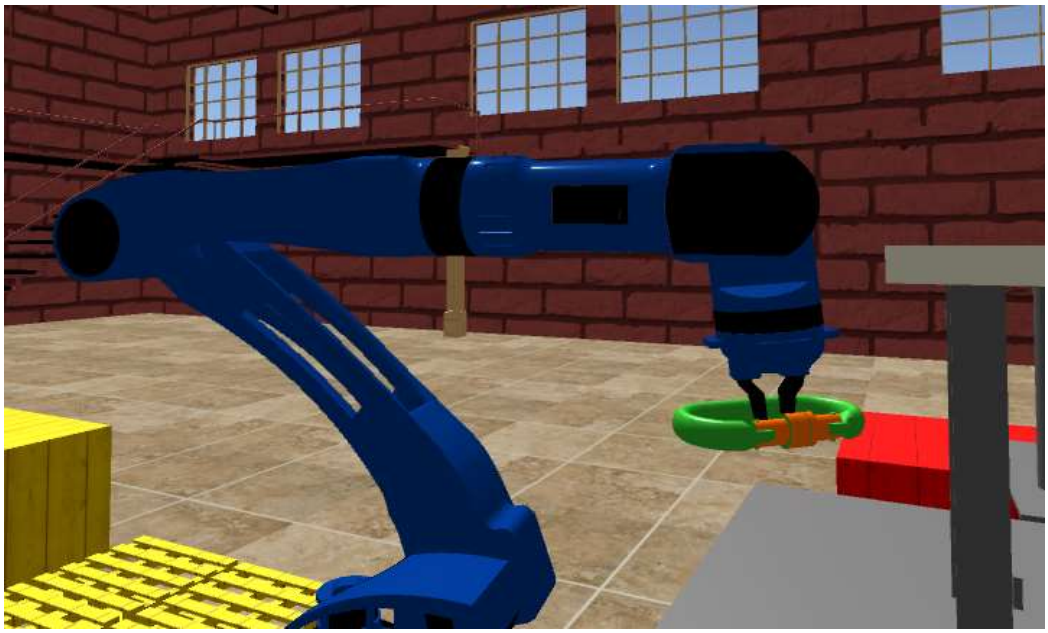
Gráfica 27. Vista del proceso #2
Fuente: elaboración propia en Unity

Gráfica 28. Vista del proceso #3
Fuente: elaboración propia en Unity



Gráfica 29. Vista del proceso #4
Fuente: elaboración propia en Unity

En el aspecto visual de la simulación del sistema de manufactura ya ensamblado se puede observar que hay diferentes tipos de ángulos de visión del proceso lo que permite ver cualquier estado del proceso de simulación y no solo cuando se esté diseñando sino también en modo de funcionamiento. Una muestra de ello se observa en la Gráfica 30, donde por medio de los comandos del teclado se hace un acercamiento al robot manipulador haciendo una captura de un mosquetón, también se pueden realizar alejamientos incluso fuera de la edificación para observar el proceso en el ángulo que se desee.



Gráfica 30. Acercamiento a un robot manipulador en la simulación
Fuente: elaboración propia en Unity

3.4.2. Simulación del proceso de manufactura sin control

Una vez realizado el respectivo análisis de las configuraciones físicas involucradas en el proceso de manufactura y las asignaciones de las características correspondientes a cada GameObject, se procede a hacer la primera prueba de simulación, no sin antes nombrar que para dicha simulación se han realizado scripts básicos para programar la dinámica y el funcionamiento de todo el proceso de manufactura, en donde los objetos presentes en la escena interactúan en el medio y se evidencia el tipo de movimiento que ellos realizan sin ningún tipo de control, con respecto a restricciones y funcionamiento en general, es por este motivo que por el momento se han realizado scripts básicos, ya que estos no contienen en sí, reglas de control o de funcionamiento, pero si tienen comandos en el código para determinar ciertos rangos de movimiento o trayectorias presentes en la simulación.

Para dar soporte a esta primera prueba de simulación en Unity, se cuenta con un **video** del proyecto para apoyar descripciones de esta primera etapa de simulación, en el cual se pueden observar los movimientos de todos objetos que hacen parte del proceso, obviamente algunos no tienen animación, puesto que son objetos pasivos en la escena y no tienen un código de programación asociados a ellos.

3.4.3. Descripción del proceso simulado sin control

El proceso de manufactura simulado está en su etapa de prueba, es decir en esta parte de la simulación se verifica que todo el ensamblaje de piezas en 3D esté debidamente programado para realizar diferentes tipos acciones, como por ejemplo rangos de movimiento, trayectorias de todos los componentes de escena, movimientos realistas, coherencia de etapas de los subprocesos, velocidades de los GameObjects, verificación de colisiones y efectos de la fuerza gravitacional, estas acciones pueden ser analizadas tanto visualmente como en la pestaña Console del bloque 4, que permite observar errores de funcionamiento del sistema.

Al iniciar la simulación del proceso se observa que todos los GameObjects y componentes de escena, tienen un movimiento característico el cual ha sido programado para que realice desplazamientos coherentes en el proceso, lo que diferencia esta etapa de las demás, es que no tiene características de restricción de acciones de control, que se presentan en la simulación, que por supuesto hace que la instrumentación del sistema de manufactura simulado trabaje sin que haya productos a fabricar, aunque los movimientos son coherentes con el instrumento, las repeticiones del mismo no son las adecuadas, puesto que realiza funciones aun sin haber dado órdenes o sin que pase un mosquetón por la zona, donde dichos instrumentos deben realizar su respectiva función de proceso.

3.5. Modelo de pre-actuadores, actuadores y sensores.

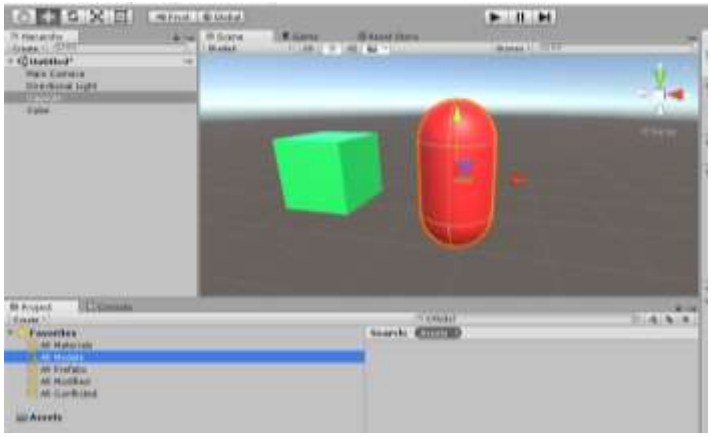
Una vez realizada la simulación en términos generales, lo que se pretende es ir más allá de las características o componentes físicos asignados a cada objeto, es decir permitir implementar un código en lenguaje C# (lenguaje nativo de Unity) que sea capaz de proporcionar un cambio en las propiedades de los GameObjets, aunque estos son versátiles, la relación del objeto a un código de programación lo hará más controlable además de ampliar su performance en la o las escenas de cualquier proyecto.

Para ilustrar mejor como se puede implementar un código y asignarlo a un GameObject de manera general, se hace necesario realizar una descripción detallada de los pasos a seguir en la herramienta Unity, para explicar el alcance de esta implementación.

3.5.1. Implementación y edición de un script a un GameObject

Paso 1:

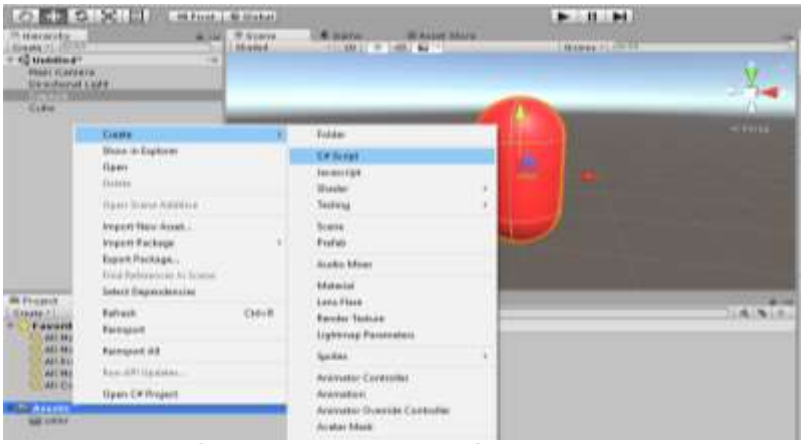
El paso número uno consiste en simplemente, agregar GameObjects a la escena Gráfica 31, en este caso se presentan un cubo verde y una capsula roja, que representan de manera general los objetos que hacen parte de un desarrollo de simulación en la parte visual.



Gráfica 31. Entorno virtual de dos GameObjects
Fuente: Unity

Paso 2:

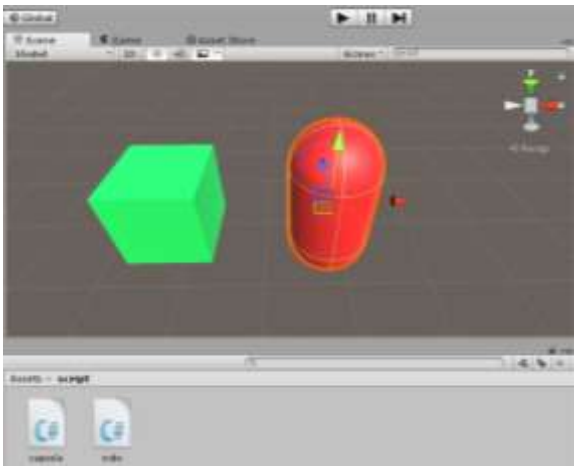
Una vez colocados los GameObjects necesarios para la simulación, se dirige al bloque 4 de la de la interfaz de Unity, para crear el script, tan solo haciendo clic derecho en Assets y buscar C# script, como lo muestra la Gráfica 32.



Gráfica 32. Modo de creación de un script
Fuente: Unity

Paso 3:

Como se observa en la Gráfica 33, se han creado dos scripts, puesto que existen dos GameObjects en la escena, y los cuales tienen los respectivos nombres de cada una de las figuras geométricas presentes, ahora lo único que se hace es arrastrar cada script y soltarlo en la figura que se desee, para que posteriormente sea blanco de la programación en el lenguaje C#.



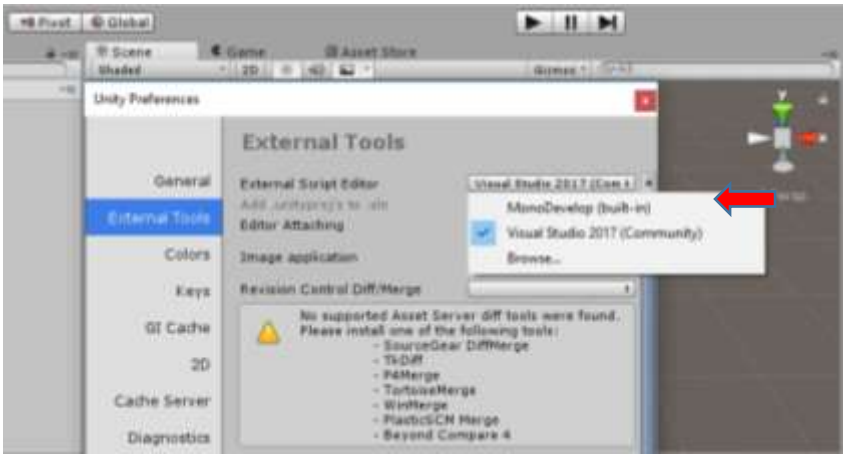
Gráfica 33. Asignación de script a los GameObjects
Fuente: elaboración propia en Unity

Paso 4:

Posteriormente a la asignación del script al GameObject, se verifica en el Inspector, que dicho script esté relacionado a la figura geométrica a la cual se hace referencia Gráfica 34, una vez verificada la existencia, se dirige al script correspondiente en Assets y se le da doble clic, con la finalidad de que se abra o se despliegue la herramienta de desarrollo para programar los GameObjects.

Gráfica 34. Verificación de scripts
Fuente: Unity

Nota: para elegir la herramienta de desarrollo de programación, para los scripts que más se adecuen a las necesidades, Unity tiene por defecto dos plataformas que son; MonoDevelop y Visual Studio 7, la forma de poder hacer esta elección se muestra en la Gráfica 35.

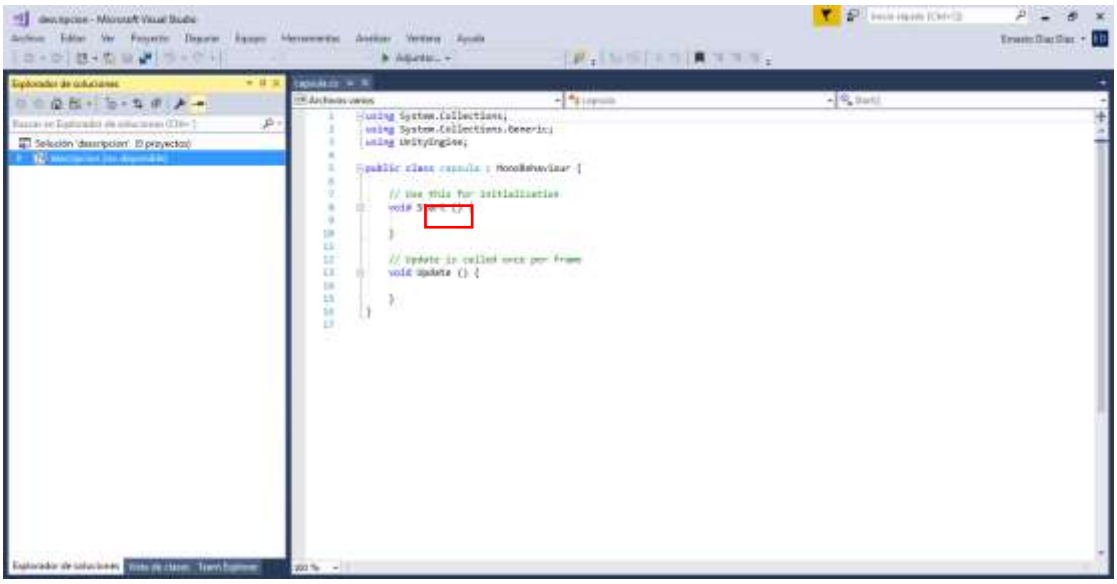


Gráfica 35. Selección del editor para programación
Fuente: elaboración propia en Unity

En la parte superior izquierda de la interfaz de Unity, se encuentra Edit la cual desplegará una ventana y después se sigue la siguiente secuencia Preferenses, External Tools, en esta última y con apoyo de la Gráfica 35 se puede visualizar que el Editor para este caso es Visual Studio 2017, herramienta con la cual se trabajara totalmente en el desarrollo del proyecto de simulación.

Paso 5:

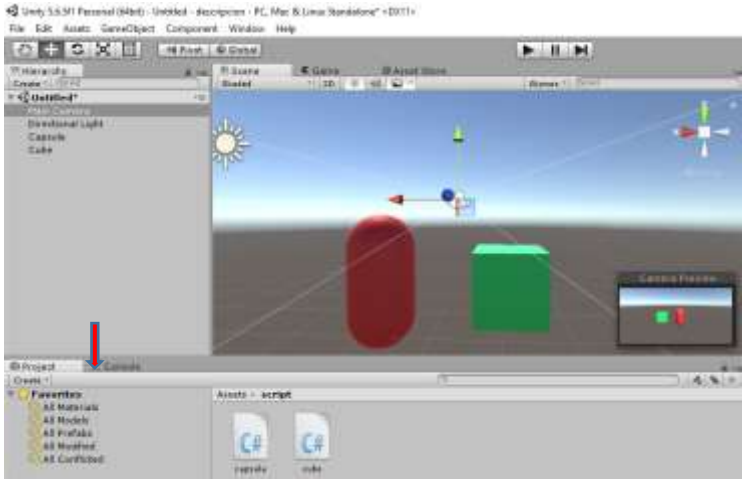
En este paso se elaboran los códigos en lenguaje C#, en el editor de preferencia, este editor como ya se mostró hace parte de las herramientas de Unity, y no es necesario hacer una descarga previa o instalación.



Gráfica 36. Interfaz de Visual Studio 2017
Fuente: Visual Studio 2017

En la Gráfica 36, se muestra la interfaz de usuario de Visual Studio 2017, en la cual se realizan los códigos correspondientes a la simulación de ensamblaje de mosquetones, es de anotar que cuando se realiza la apertura de cualquier script en el editor, éste ya trae consigo una anatomía de texto para programar, donde vienen las librerías básicas por defecto, el nombre que se ha dado al código en este caso es capsula, como se puede observar en la Gráfica 36, y por último es la estructura del texto se encuentra el Start y Update, que sirven para hacer un llamado de funciones a Unity antes de que comience la simulación y para la actualización de código de los GameObjects frame por frame, respectivamente.

Paso 6:



Gráfica 37. Posicionamiento de la cámara principal
Fuente: Unity

Para finalizar una vez hecho el código deseado se procede a verificar si el código desarrollado está bien elaborado, eso se puede observar en la pestaña Console, donde se verifican los posibles errores de programación entre otras cosas, ya realizada esta verificación simplemente se decide qué tipo de vista se quiere para el inicio de la simulación, configurando la cámara principal en la posición deseada, así como se observa en la Gráfica 37, por último se pincha el botón de Play de la interfaz de Unity para observar los movimientos o funciones asignadas a cada GameObject en el código de programación.

Para concluir estos 6 pasos básicos, cabe decir que se realizan con la finalidad de hacer amigable la simulación del proceso, por ello se escogen figuras geométricas sencillas para ilustrar de la manera más fácil y rápida, la forma en que se realizan funciones y configuraciones, que la herramienta Unity permite.

3.5.2. Descripción de códigos del proceso de manufactura en lenguaje C#

Para la codificación de la simulación se asigna un script a cada elemento y GameObject del proceso excepto para la comunicación a la base de datos, esto con el fin de manipular las características tanto físicas como de animación de cada uno. En total se encuentran 12 scripts relacionados al entorno simulado, los cuales se relacionan así:

Tabla 25. Tabla de scripts

6	Control	Instrumentación del proceso
1	Control	Comunicación para la base de datos
1	Simulación	Control de movimientos del operario
2	Simulación	Determinación de producto para buenos y malos
1	Simulación	Registro de datos de productos a fabricar
1	Simulación	Contabilizador de mosquetones

Fuente: elaboración propia.

Los scripts mostrados en la Tabla 25, son los que determinan todo el funcionamiento del proceso de manufactura, donde los códigos que pertenecen a la instrumentación y a la comunicación son parte del controlador de todo el sistema, puesto que se han relacionado variables en el código de programación, que relacionan los movimientos o acciones de los instrumentos u objetos, que determinan las características funcionales de cada uno de ellos, en otras palabras cuando un elemento presente en la simulación presenta algún movimiento en la escena, al estar relacionado a una variable codificada, tiene un cambio en dicha variable, y son estos cambios en la animación los que permiten detectar si un instrumento tiene un estado de encendido o apagado, dichos cambios que se tienen en los instrumentos o en algunos casos los objetos, son los que se envían a la base de datos por medio del código de comunicación para su análisis.

Por otra parte, existen otros códigos pertenecientes a la simulación, cuya finalidad es que el proceso tenga un registro contable, un registro de numeración de productos a fabricar, un registro de contabilidad de productos buenos y malos, los cuales en la simulación al inicio tendrán la facultad de ser variables públicas, donde la característica principal es que es modificable por cualquier usuario que manipule el sistema simulado en Unity.

3.6. Modelado del controlador.

Para controlar el sistema de manufactura se diseña un controlador en lenguaje C# con todas las características restrictivas y de funcionamiento, que se han considerado del proceso, estas características son implementadas en el código perteneciente a cada instrumento o más bien dicho a cada GameObject, el cual tiene una función específica en la simulación, el motivo por el cual se implementan las características restrictivas y de funcionamiento en los diferentes códigos, es debido a que el software Unity no permite que se asocie un solo código de funcionamiento para controlar o manipular diferentes elementos presentes en la simulación de la escena.

Para representar el controlador del proceso se ha optado por realizar un gráfico de **Diagrama de clases** en lenguaje UML, que permita de describir de manera clara la estructura del sistema con respecto a las clases que la componen, donde se muestre las relaciones entre los objetos, las operaciones y los atributos pertenecientes a cada una de ellas.

3.6.1. Diagrama de clases del controlador

La Figura 29, representa el controlador del proceso de manufactura en un diagrama estructurado UML, en el cual existen siete clases pertenecientes al controlador del proceso y una clase llamada **Mono Behaviour** perteneciente por defecto al software Unity. De esta clase las demás clases realizadas heredan las funcionalidades y prestaciones de la herramienta como son: sus atributos y propiedades, en cuanto a código de programación se refiere, en otras palabras, no importa el proyecto que se realice o la clase que se cree en la plataforma, siempre va a estar ligada a **Mono Behaviour** por su posición jerárquica en la herramienta. Las otras clases pertenecientes al diagrama del controlador, hacen parte del proceso de manufactura, donde cada clase asociada a un instrumento tiene sus propios atributos y funciones. Cada bloque del diagrama cuenta con el nombre de la clase, seguido de los atributos y luego de las funciones, cada uno de estos rasgos característicos posee un operador positivo que significa que es un atributo público en el código y el operador negativo quiere decir que es un atributo privado en código, estos datos o esta información que provee el diagrama de clase en UML tiene como funcionalidad modificar el controlador en partes especificas del proceso, puesto que el nombre que se le ha dado a la clase, corresponde a un instrumento simulado y cada bloque contiene las variables y funciones que comprenden la funcionalidad de dicho instrumento en el proceso de manufactura.

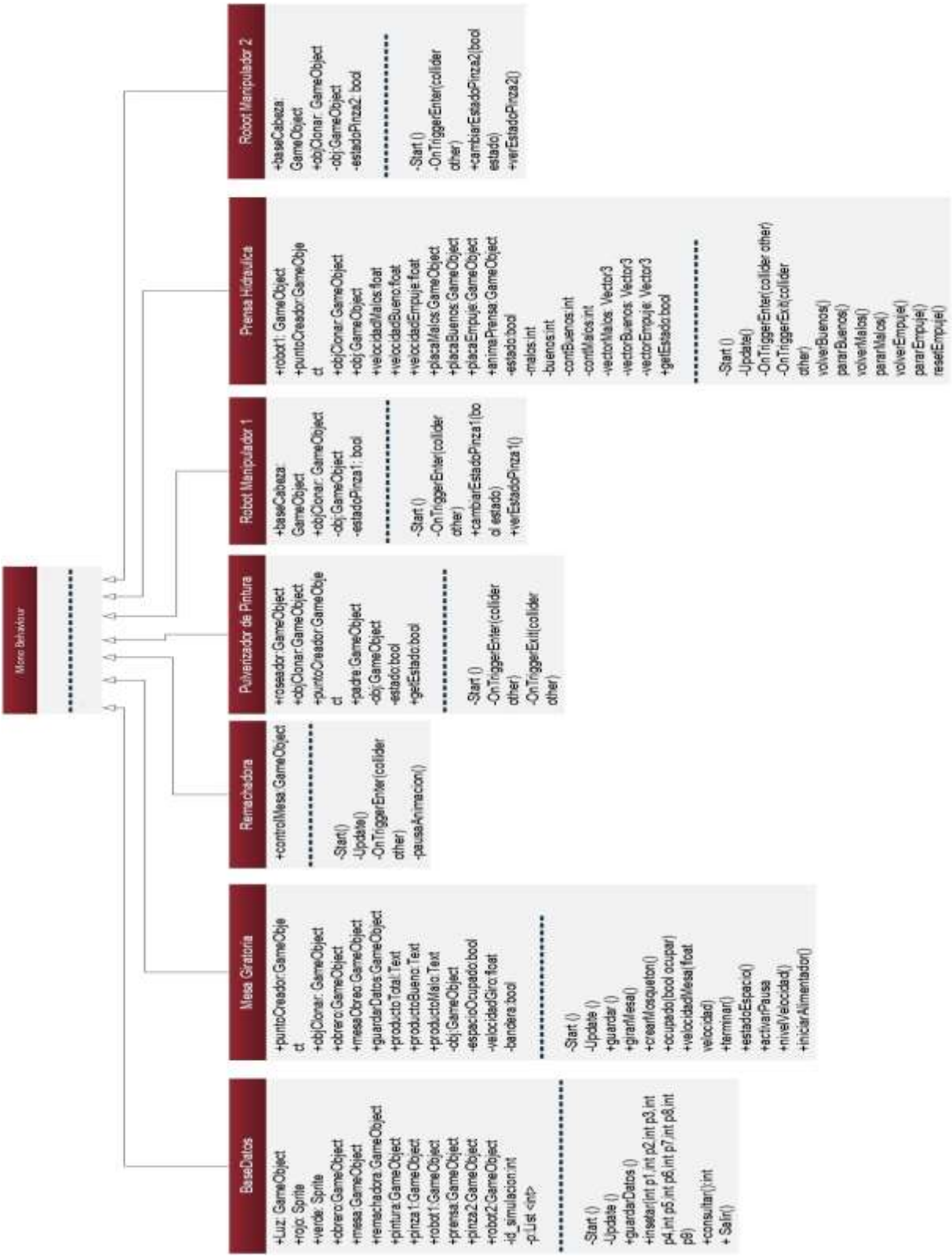


Figura 29. Diagrama de clases del controlador
Fuente: elaboración propia

Capítulo 4

Análisis de resultados

4.1. Introducción

En el presente capítulo se presenta la funcionalidad total del proyecto de simulación del sistema de manufactura en lazo cerrado, donde se exponen dos modos de funcionamiento del sistema, haciendo diferentes cambios en la codificación, más exactamente en el diagrama de clases representado en la Figura 29. También se ven retratados los modos de funcionamiento en cuanto a su comunicación con la base de datos, donde se verifica la relación del estado del proceso, con las señales sensoriales de los instrumentos, que son enviadas con su respectiva nomenclatura, es decir que haya una coincidencia de lo que se muestra en el proceso de simulación con la información que se presenta en interfaz de la base de datos. Cabe anotar que para observar el desempeño del proyecto tanto Unity como SQLite deben estar visualmente activos, para que en la simulación no se detenga en el proceso y los datos se obtengan en tiempo real. Por último, se hace el análisis se las señales obtenidas, gracias a un algoritmo de identificación.

4.2. Cambios en la codificación para los modos de funcionamiento

Se han escogido dos modos de funcionamiento del proceso de manufactura, para ser objeto de comparación y análisis, los cambios que se realizan permiten hacer varias pruebas de flexibilidad, en cuanto a la codificación del sistema ligada al funcionamiento de la instrumentación en la escena de Unity, esto debido a que el proceso puede ser cambiante en el tiempo, donde se le puede dar un funcionamiento o aspecto diferente al que se tiene, esto implica un cambio notorio en la visualización del proceso y las señales que en conjunto proporciona el sistema.

4.2.1. Configuración del controlador para los modos de funcionamiento

Para desarrollar distintos modos de funcionamiento es necesario la manipulación de los códigos de programación, lo que implica una lógica de funcionamiento diferente para poder visualizar el performance del proceso, el diagrama de clases, es una representación que ayuda a la labor de hacer cambios de lógica rápidamente, yendo al instrumento en el cual se desea hacer una renovación de funcionalidad.

La forma de realizar la diversificación del funcionamiento del controlador, requiere del conocimiento pleno de las variables asociadas a los instrumentos y a las funciones que estos desempeñan en la simulación, esta información se presenta en los atributos y en las operaciones de la clase asociada al instrumento, donde se encuentra la lógica de funcionamiento de cada uno de ellos y que es susceptible a modificaciones de la misma, con la finalidad de que cada instrumento presente comportamientos diferentes.

Figura 30. Diagrama de clases de la mesa giratoria
Fuente: elaboración propia en Unity

En la Figura 30, se muestra el diagrama de clase de la mesa giratoria, donde los atributos se encuentran en la parte superior, estos contienen las variables del código y el tipo de dato, las variables pueden tomar el nombre que se desee, eso dependerá del programador, por ejemplo se observa un atributo llamado VelocidadGiro con tipo de variable flotante, esta variable es la que va a interactuar en las funciones del

proceso y puede ser invocada en la lógica de la clase a la cual pertenece, otra posibilidad de esta variable, es la información que se pueda alojar en ella para posibles registros en el estado del proceso. En la parte inferior se observan las operaciones, que representan la lógica de funcionamiento del instrumento, es aquí donde en realidad se cambia el funcionamiento del proceso, en este caso de la mesa giratoria. Las operaciones también llamadas funciones hacen un llamado a los atributos para realizar operaciones matemáticas, analizar estados de las variables, cambios de parámetros, funcionalidades etc. Por ejemplo, para realizar un cambio en la velocidad de animación de la mesa es necesario ir a la operación velocidadMesa (float velocidad), también se encuentra una función de nombre ocupado () que en su lógica contiene que, la mesa que cuando la mesa este ocupada el operario no podrá colocar ningún mosquetón.

Para realizar los cambios de funcionalidad de pueden agregar tanto atributos u operaciones o también cabe la posibilidad de eliminar cualquiera de ellos, si así se requiere, como se evidencia para realizar cambios en el modo de funcionamiento, se requiere un vasto conocimiento del proceso y de programación para poder implementar renovaciones a la hora de simular procesos en una herramienta software.

4.2.2. Comparación de cambios realizados en los diagramas UML

A continuación, se presenta la comparación de los dos modos de funcionamiento del proceso, representados con diagramas de clases, Figura 31 y Figura 32.

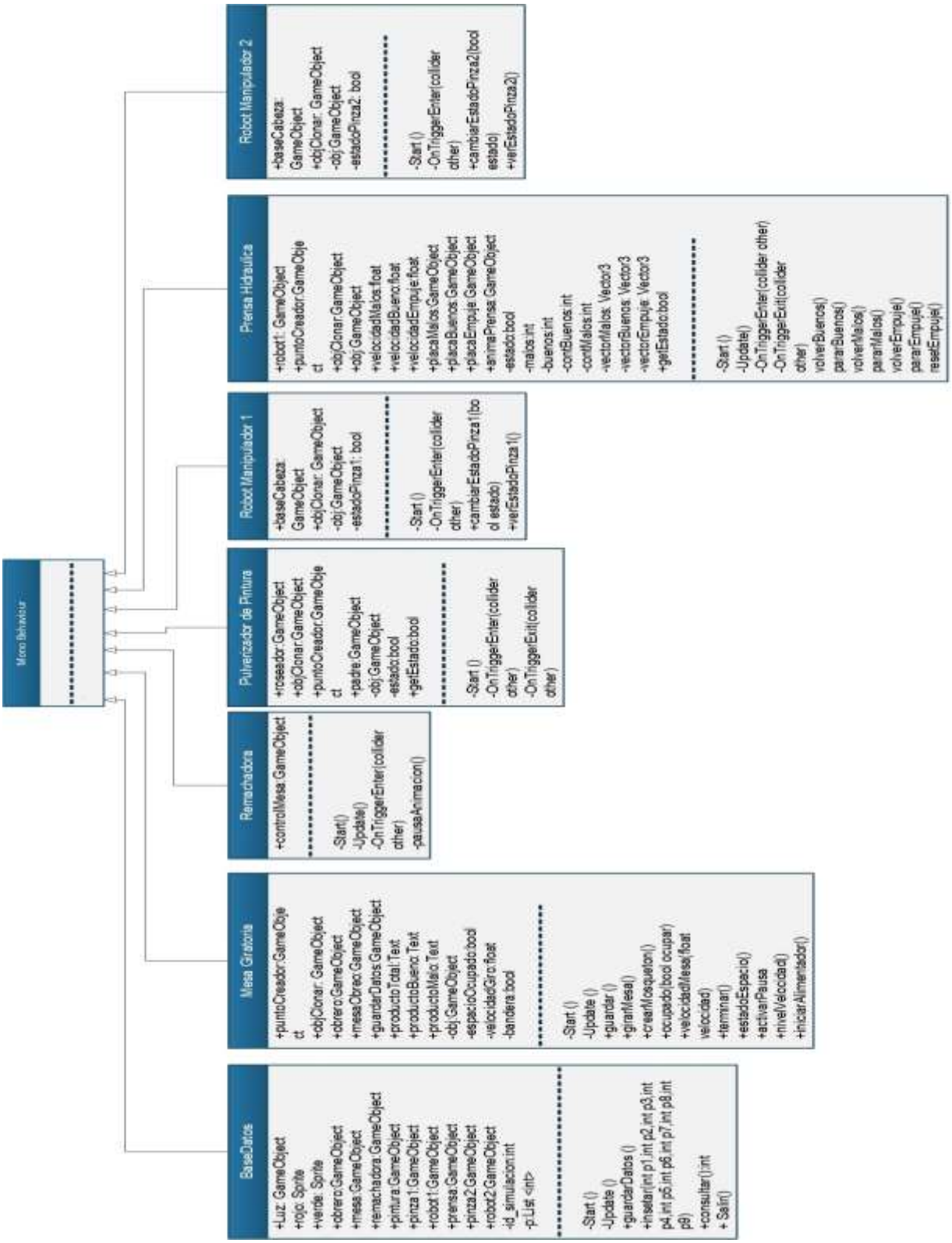


Figura 31. Diagrama de clases para el primer modo de funcionamiento

Fuente: elaboración propia.

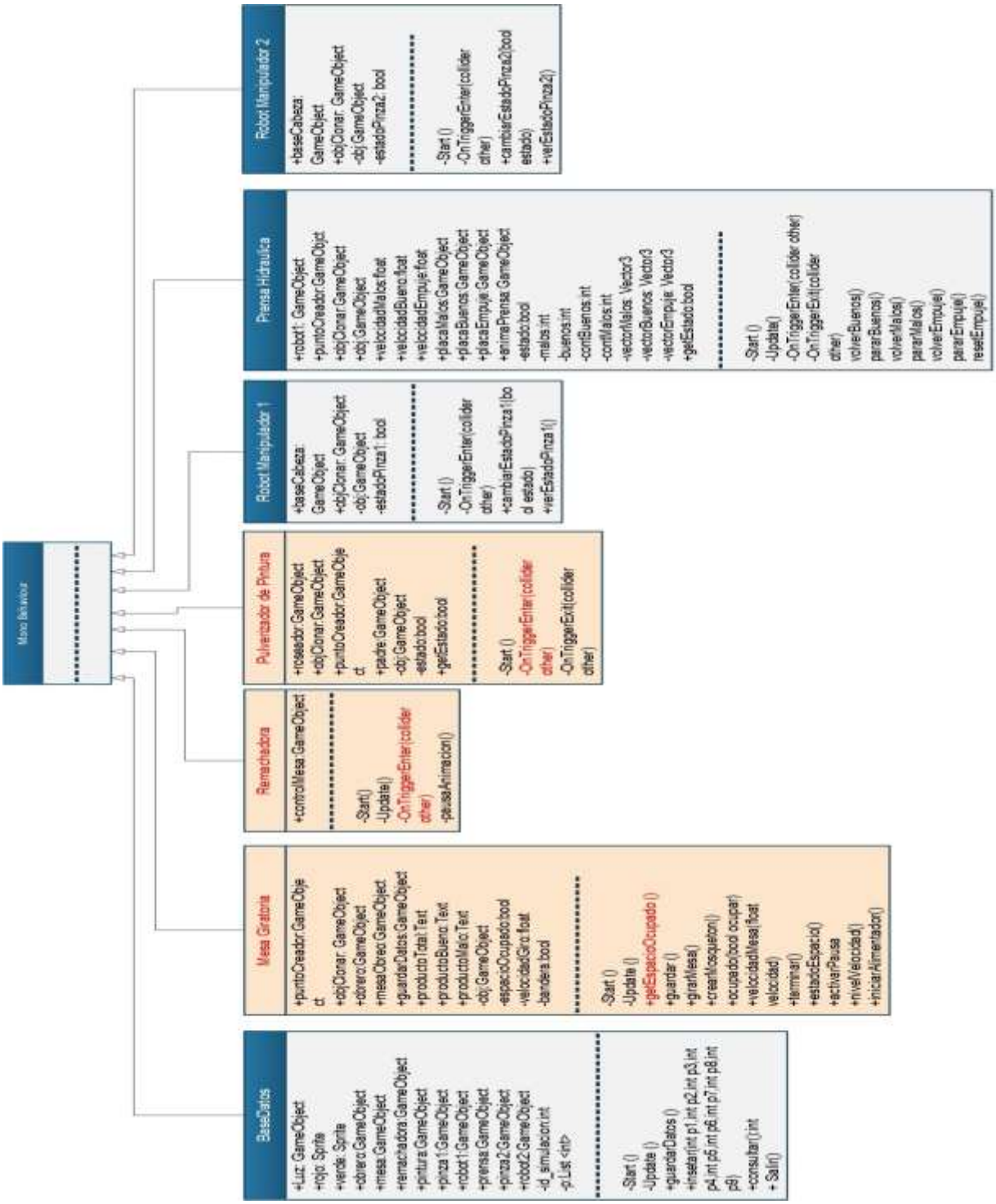


Figura 32. Diagrama de clases para el segundo modo de funcionamiento

Fuente: elaboración propia.

Para indicar de forma clara los cambios que se presentan en la codificación para cambiar el modo de funcionamiento, se tiene que las clases que no han sido alteradas por dichos cambios se representan en color azul, y el caso contrario las que han sido modificadas en su codificación se han representado en color rojo.

Como se puede apreciar en la Figura 31 se muestra el primer modo de funcionamiento del sistema simulado, este modo consta de siete clases de proceso, donde se han programado funcionalidades del mismo. Este diagrama se utiliza como base para generar diferentes comportamientos, por ende, su color azul, y de aquí parten los modos de funcionamiento que se quieran dar al sistema en escena del entorno simulado.

Ahora se examina la Figura 32, donde se observan clases de tonalidad roja, lo cual indica que han sido alteradas, ya sea en los atributos o en las operaciones de la clase que lo presenta, en este caso han sido alteradas las clases; Mesa Giratoria, Remachadora, Pulverizador de pintura, también se puede anotar que solo han sufrido modificaciones las operaciones de las clases ya mencionadas, estas se subrayan para darle mayor realce y que puedan ser identificadas fácilmente.

4.3. Descripción de los modos de funcionamiento

Los cambios en la funcionalidad que se realizan en el controlador codificado (diagrama de clases), proveen los diferentes modos de funcionamiento, para ello se describen los modos resultantes de dichos diagramas.

Modo de funcionamiento 1:

El modo de funcionamiento 1 implica que en el proceso de manufactura fabrique varios mosquetones en la mesa giratoria, lo que conlleva a que varias funciones se estén ejecutando al mismo tiempo en dicha mesa. El operario tendrá también la posibilidad de ingresar al sistema más mosquetones en un menor tiempo, puesto que con la finalización o ejecución de la primera estación que es el remachado, el operador ya puede ingresar al proceso otro mosquetón.

Gráfica 38. Simulación del primer modo de funcionamiento
Fuente: elaboración propia en Unity

En la Gráfica 38, se aprecian tres mosquetes en la mesa giratoria, el operario no pondrá otro más hasta que el mosquete C haya pasado por la remachadora, también se aprecia que el mosquetón A sale de la zona de pintura con un color verde, y el mosquete B ya ha sido remachado y se dirige al pulverizador de pintura.

Modo de funcionamiento 2:

El modo de funcionamiento 2 comprende que en el proceso de manufactura solo haya un mosquetón en la mesa giratoria, es decir que cuando el robot manipulador_1 recoja el mosquetón presente en la mesa, el operario ya podrá ingresar al proceso otro mosquete en la mesa giratoria

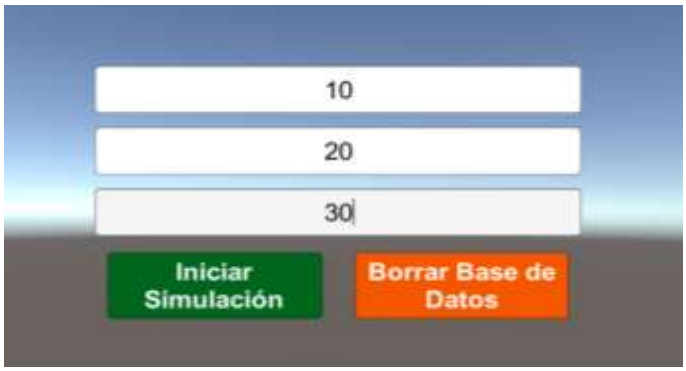
En la Gráfica 39, se observa que a pesar de que hay mosquetones en disponibilidad para la alimentación del sistema, el operario no puede ingresar otro hasta que el robot ejecute su función de recoger el mosquetón. Por este motivo, se encuentra solo un mosquete en la mesa giratoria.

Gráfica 39. Simulación del segundo modo de funcionamiento
Fuente: elaboración propia en Unity

4.4. Toma de registros y comparación de resultados

Conociendo los dos modos de funcionamiento, el siguiente paso es capturar las señales sensoriales de los instrumentos del proceso, con la finalidad de realizar una comparación de los estados en diferentes tiempos de simulación, con las mismas características y parámetros de entrada para el inicio del proceso en el entorno, para ello se realiza un registro temporal de cada simulación, observando cada cierto tiempo el estado total del proceso en la base de datos correspondiente a cada uno.

En la Gráfica 40, se observa los parámetros de entrada de inicio de simulación para cada modo de funcionamiento, donde se aprecia que, el número de productos a fabricar es de 10 mosquetes, el porcentaje de daño de mosquetones es del 20% y 30 segundos para la toma de registros sensoriales en la base de datos.



Gráfica 40. Parámetros de inicio
Fuente: elaboración propia en Unity

Los parámetros de inicio arrojan los siguientes resultados de simulación para cada modo de funcionamiento, donde se realiza un conteo en segundos desde el inicio de la simulación hasta el final del recorrido del ultimo mosquetón del proceso, esto se aprecia en la Tabla 26, donde también se muestra la diferencia de tiempo de simulación para la misma configuración de entrada.

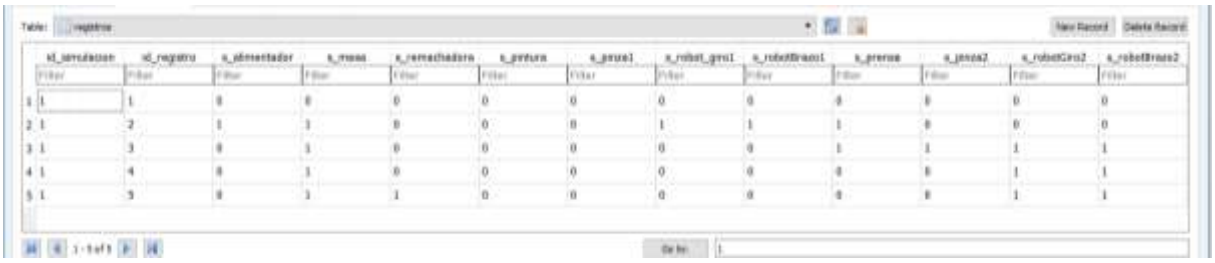
Tabla 26. Tiempos de simulación para cada modo de funcionamiento

Modo	# Productos	Daño (%)	Captura de registros (s)	Tiempo (min)
1	10	20	30	2.44
2	10	20	30	5.26
				2.42

Fuente: elaboración propia

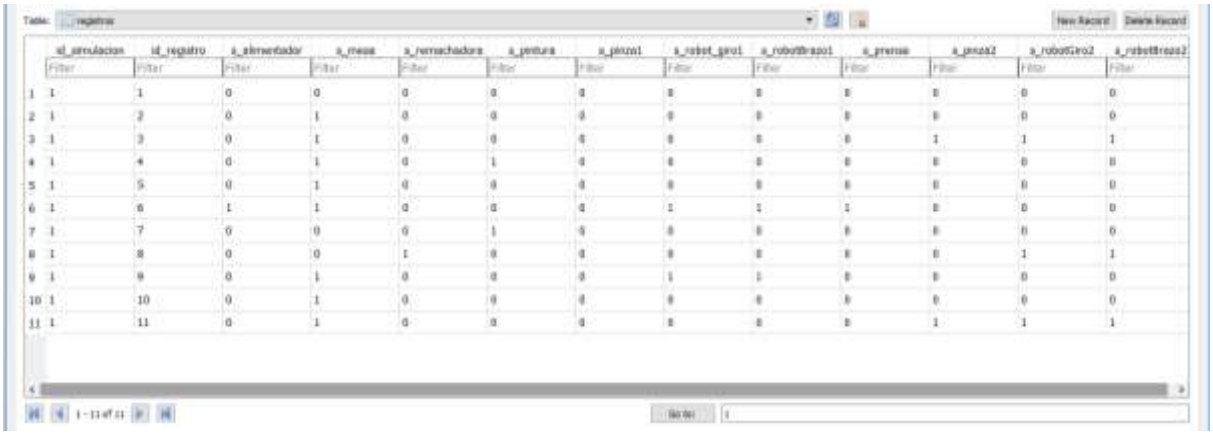
La diferencia de los modos de funcionamiento es de 2.42 minutos, lo que indica que para los mismos parámetros de entrada hay una diferencia bastante notable, que se traduce en casi el doble de tiempo en simulación para el proceso, el análisis también conlleva a escoger un modo de funcionamiento adecuado, más rápido que permita realizar más piezas en el menor tiempo posible, esto se corrobora con la Tabla 27, donde se observa 3 comparaciones para las mismas configuraciones de entrada y se determina la diferencia de tiempos entre cada una de ellas (Anexo B).

Analizando los tiempos del entorno simulado por cada modo de funcionamiento, se realizan ahora las capturas de las señales de la instrumentación del proceso, estas capturas se realizan cada 30 segundos hasta la finalización de la simulación.



Gráfica 41. Registros del primer modo de funcionamiento
Fuente: elaboración propia en Visual Studio

En la Gráfica 41, se observa el registro de estados para el primer modo de funcionamiento, que consta de 5 de ellos, puesto que la duración de la simulación es mucho más corta y el número de registros en menor.

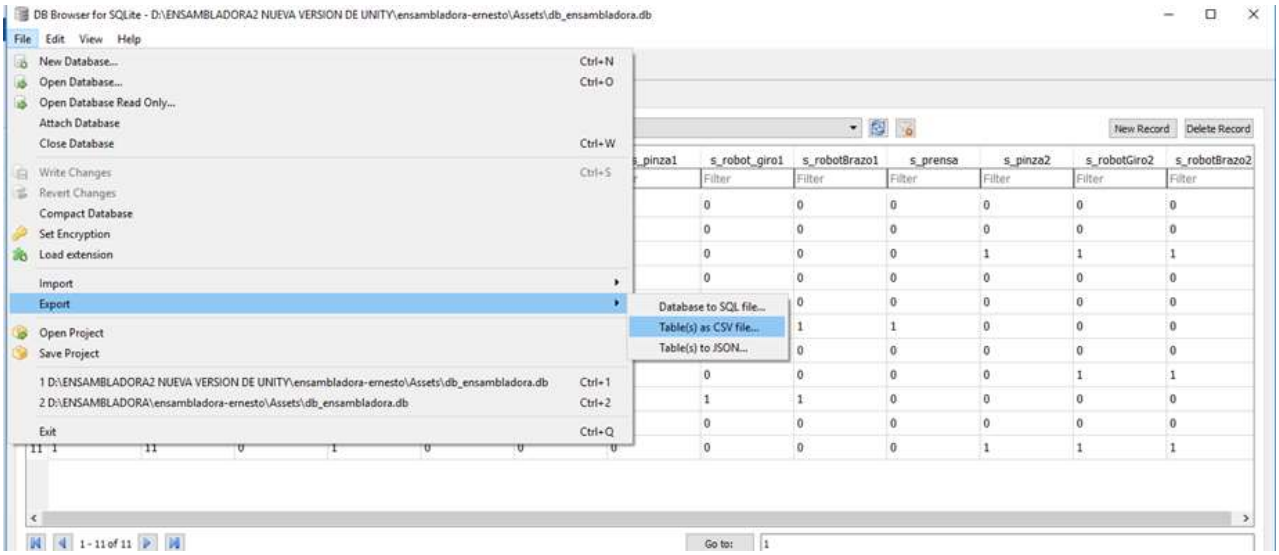


Gráfica 42. Registros del segundo modo de funcionamiento
Fuente: elaboración propia en Visual Studio

En la Gráfica 42, se aprecia los estados del proceso para el segundo modo de funcionamiento, el cual consta de 11 registros, donde se observan las nomenclaturas de los sensores y el estado de cada uno de ellos en el preciso momento en donde se realiza la captura de la simulación.

4.4.1. Datos exportados a Excel

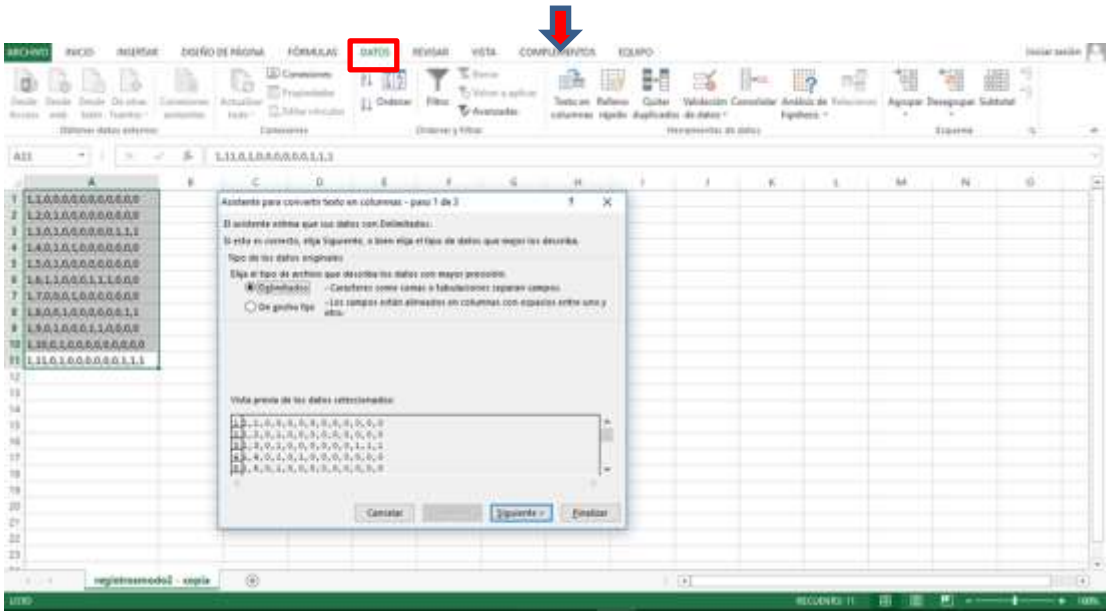
Con la finalidad de manipular la información obtenida de la base de datos SQLite, se quiere enviar o exportar los datos obtenidos a la plataforma Excel, para realizar análisis con un algoritmo de identificación, la forma de exportar los datos se presenta a continuación.



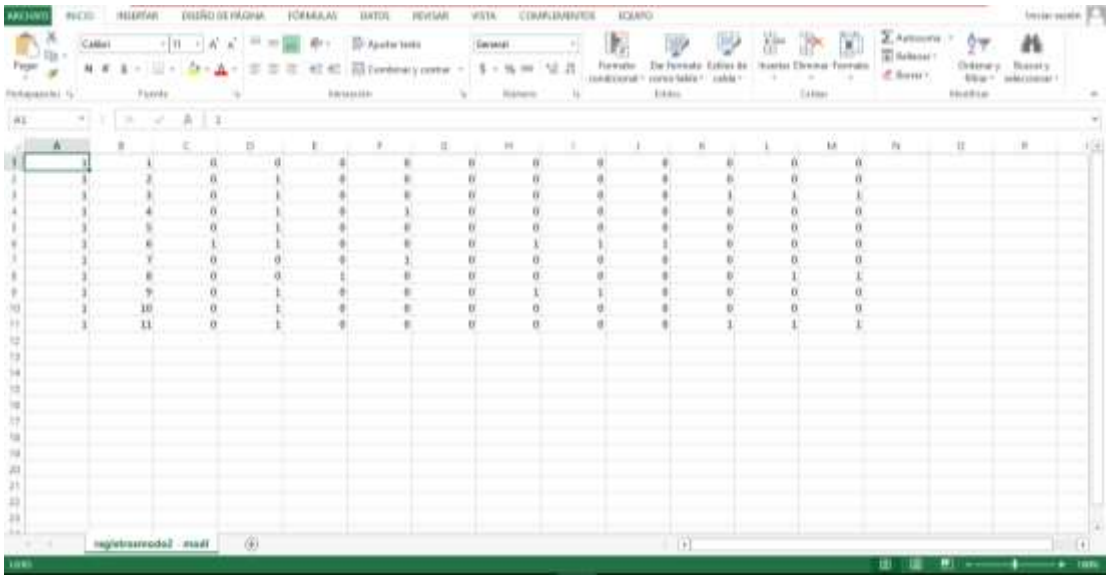
Gráfica 43. Envió de información a Excel
Fuente: elaboración propia en Visual Studio.

En la Gráfica 43, se presenta la interfaz de SQLite, donde están alojados los registros de la simulación, en esa misma interfaz se exporta con la opción de Table(s) as CSV file para que se pueda guardar el registro de la información.

Una vez se realiza la exportación de la información, al desplegar el documento, este se muestra como en la Gráfica 44, por tanto se hace necesario realizar un cambio, para que los datos sean distribuidos correctamente en cada celda de Excel, para ello se dirige a la pestaña de Datos, se selecciona la información que se desee y posteriormente se le da la opción de Texto en columnas de esa pestaña, esa operación da como resultado una visualización organizada de la información de las señales obtenidas en simulación, esto se observa en la Gráfica 45.



Gráfica 44. Vista de la información en Excel
Fuente: elaboración propia en Excel



Gráfica 45. Visualización de los datos de simulación en Excel
Fuente: elaboración propia en Excel

Conclusiones y Trabajos Futuros

Tomando como base el diseño del sistema de manufactura, se logra implementar dicho diseño en un entorno gráfico simulado, teniendo en cuenta todas las características de funcionamiento de los instrumentos y restricciones del proceso.

Para controlar el sistema de manufactura se lleva a cabo una codificación de todo el sistema, que permite realizar modificaciones de funcionamiento a partir del código relacionado a cualquier instrumento del proceso, cuya finalidad es tener alternativas de funcionalidad y que el sistema en general sea flexible a cualquier cambio.

Se consigue extraer las señales o estado del proceso por medio de la comunicación entre Unity y la base de datos, lo cual proporciona una visualización de los sensores de los instrumentos, para corregir posibles errores o llevar esa información a otras plataformas que puedan analizar dicha información.

Se pretende como trabajo futuro realizar una interfaz que analice las señales en tiempo real, con el fin de realizar identificación de la planta o diseño de control para prácticas de laboratorio en plantas virtuales.

Bibliografía

[1] B. Esmaeilian, S. Behdad, and B. Wang, “The evolution and future of manufacturing: A review,” *J. Manuf. Syst.*, vol. 39, pp. 79–100, 2016.

[2] P. Gu, H. a. Rao, and M. M. Tseng, “Systematic Design of Manufacturing Systems Based on Axiomatic Design Approach,” *CIRP Ann. - Manuf. Technol.*, vol. 50, no. 1, pp. 299–304, 2001.

[3] K. Salahshoor, M. S. Khoshro, and M. Kordestani, “Fault detection and diagnosis of an industrial steam turbine using a distributed configuration of adaptive neuro-fuzzy inference systems,” *Simul. Model. Pract. Theory*, vol. 19, no. 5, pp. 1280–1293, 2011.

[4] A. . Anglani, A. . Grieco, M. . Pacella, and T. . Tolio, “Object-oriented modeling and simulation of flexible manufacturing systems: A rule-based procedure,” *Simul. Model. Pract. Theory*, vol. 10, no. 3–4, pp. 209–234, 2002.

[5] V. Sanz, A. Urquia, F. E. Cellier, and S. Dormido, “System modeling using the Parallel DEVS formalism and the Modelica language,” *Simul. Model. Pract. Theory*, vol. 18, no. 7, pp. 998–1018, 2010.

[6] C. Gertosio, N. Mebarki, and A. Dussauchoy, “Modeling and simulation of the control framework on a flexible manufacturing system,” *Int. J. Prod. Econ.*, vol. 64, no. 1, pp. 285–293, 2000.

[7] S. Nandagawe and S. P. Sarmah, “Development and application of a simulation model for throughput improvement in the melting shop of a steel plant,” vol. 6, no. 2, pp. 267–281, 2009.

[8] M. Jagstam, V. C. Corporation, and M. Simulation, “Proceedings of the 2002 Winter Simularion Conference,” 2002.

[9] S. F. Owens and R. R. Levary, “Evaluating Design Alternatives of an Extruded Food Production Line Using Simulation,” 2015.

[10] G. Wang, Y. Yan, X. Zhang, J. Shangguan, and Y. Xiao, “A Simulation Optimization Approach for Facility Layout Problem,” pp. 734–738, 2008.

[11] B. P. Zeigler, T. Discrete, and E. System, “S . d. a. f.,” pp. 716–722, 1994.

[12] M. Jahangirian, T. Eldabi, A. Naseer, L. K. Stergioulas, and T. Young, “Simulation in manufacturing and business: A review,” *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 1–13, 2010.

[13] G. Popovics and L. Monostori, “An approach to Determine Simulation Model

Complexity," in *Procedia CIRP*, 2016, vol. 52, pp. 257–261.

[14] A. Negahban and J. S. Smith, "Simulation for manufacturing system design and operation: Literature review and analysis," *J. Manuf. Syst.*, vol. 33, no. 2, pp. 241–261, 2014.

[15] H. El Haouzi, A. Thomas, and J. F. Pétin, "Contribution to reusability and modularity of manufacturing systems simulation models: Application to distributed control simulation within DFT context," *Int. J. Prod. Econ.*, vol. 112, no. 1, pp. 48–61, Mar. 2008.

[16] M. S. Engineering and H. Kong, "A comprehensive survey and future trend of simulation study on FMS scheduling," no. 1985, 2004.

[17] Y. Qamsane, A. Tajer, and A. Philippot, "Distributed Supervisory Control Synthesis For Discrete Manufacturing Systems," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 396–401, 2016.

[18] G. Di Orio, G. Cândido, and J. Barata, "The Adapter module: A building block for Self-Learning Production Systems," *Robot. Comput. Integr. Manuf.*, vol. 36, pp. 25–35, 2015.

[19] R. Shah and P. T. Ward, "Lean manufacturing: Context, practice bundles, and performance," *J. Oper. Manag.*, vol. 21, no. 2, pp. 129–149, 2003.

[20] N. Benkamoun, W. ElMaraghy, A.-L. Huyet, and K. Kouiss, "Architecture Framework for Manufacturing System Design," *Procedia CIRP*, vol. 17, pp. 88–93, 2014.

[21] D. Wu, D. W. Rosen, L. Wang, and D. Schaefer, "Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation," *CAD Comput. Aided Des.*, vol. 59, pp. 1–14, 2015.

[22] W. B. G. Pahl, "Engineering design," *Des. Counc.*, vol. 984, 1984.

[23] S. Robinson, *Conceptual modelling for simulation Part II: a framework for conceptual modelling*. 2008.

[24] G. A. L.G. Birta, *Modelling and Simulation: Exploring Dynamic System Behaviour*. .

[25] S. Durieux and H. Pierreval, "Regression metamodeling for the design of automated manufacturing system composed of parallel machines sharing a material handling resource," *Int. J. Prod. Econ.*, vol. 89, no. 1, pp. 21–30, 2004.

[26] K. F. Nielsen J, "A resource capability model to support product family analysis," *JSME Int. J.*, vol. Series C, pp. 568–575, 2006.

[27] R. R. L. S.F. Owens, "Evaluating design alternatives of an extruded food

production line using simulation," *Simulation*, pp. 626–632, 2002.

[28] B. P. Zeigler, "Hierarchical, modular discrete-event modelling in an object-oriented environment," pp. 219–230, 1987.

[29] A. Philippot, M. Sayed-Mouchaweh, and V. Carré-Ménétrier, *Modelling of a discrete manufacturing system by parts of plant*, vol. 13, no. PART 1. IFAC, 2009.

[30] B. P. Zeigler, "Hierarchical , modular discrete-event modelling in an object-oriented environment *," pp. 219–230, 1986.

[31] M. L.Brodie, "Axiomatic definitions for data model semantics," *Inf. Syst.*, vol. Volume 7, no. 2, pp. 183–197, 1982.

[32] R. Wang, A. B. Nellippallil, G. Wang, Y. Yan, J. K. Allen, and F. Mistree, "Systematic design space exploration using a template-based ontological method," *Adv. Eng. Informatics*, vol. 36, no. October 2017, pp. 163–177, 2018.

[33] S. Cassandras, Christos G., Lafortune, *Introduction to Discrete Event Systems*. 1999.

[34] R. Chandra, V. and Kumar, "A Discrete Event Systems Modeling Formalism Based one Event Occurrences Rules and Precedence," *IEEE Trans.*, 2001.

[35] J. M. Rohée, B., Riera, B., Carré-Ménétrier, V. and Roussel, "Outil d'aide à l'élaboration de modèles hybrides de simulation pour les systèmes manufacturiers.," *2nd journées Dr. MACS, Reims, Fr.*, 2007.

[36] J. Machado, "Influence de la prise en compte d'un modèle de processus en vérification formelle des systèmes à événements discrets.," *Thèse l'Ecole Norm. Supérieure Cachan, Cachan, Fr.*, 2006.

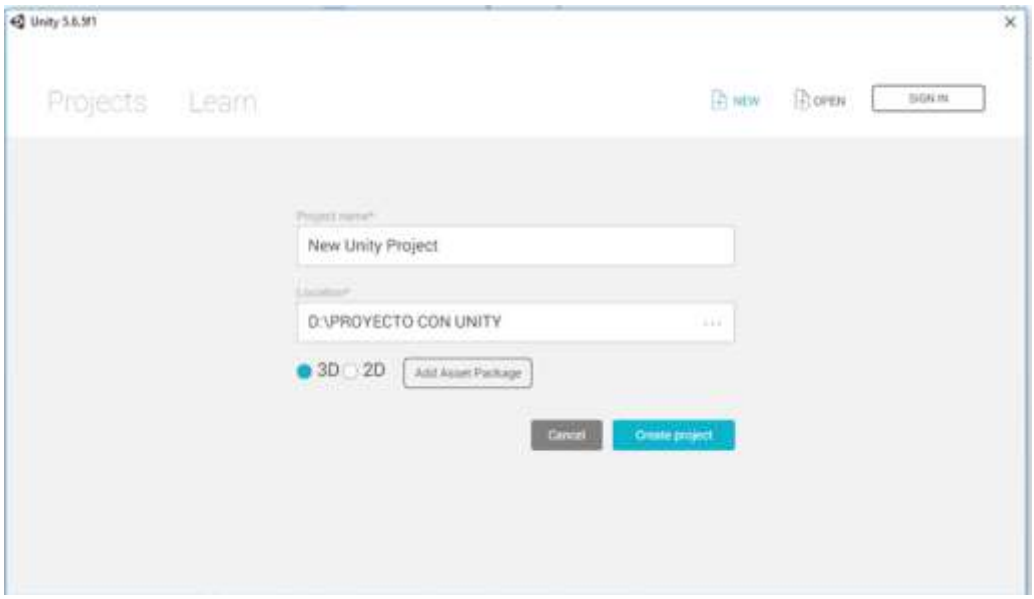
[37] L. Tang, Yih, "A framework for part type selection and scheduling in FMS environments," *Jour- nal Comput. Integr. Manuf.*, vol. 8, pp. 102–115, 1995.

[38] N. Furian, M. O'Sullivan, C. Walker, S. Vössner, and D. Neubacher, "A conceptual modeling framework for discrete event simulation using hierarchical control structures," *Simul. Model. Pract. Theory*, vol. 56, pp. 82–96, 2015.

[39] J. O. Moody and P. J. Antsaklis, "Petri net supervisors for DES with uncontrollable and unobservable transitions," *IEEE Trans. Automat. Contr.*, vol. 45, pp. 462–476, 2000.

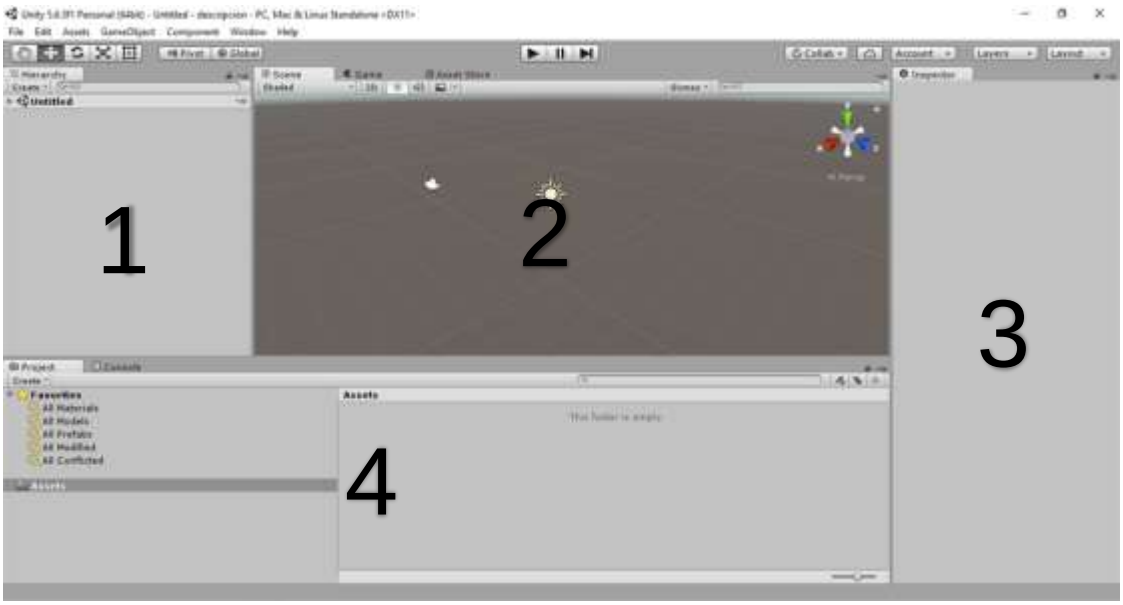
Anexo A. Descripción de la interfaz gráfica de Unity

El entorno gráfico de Unity (Gráfica 46) en su inicio da la posibilidad de creación de un proyecto nuevo, donde se puede escoger entre la visualización en 3D Y 2D para este caso se va a utilizar el componente en 3D, después se nombra el proyecto y se escoge el disco en donde estará alojado para su posterior revisión, por último, se crea el proyecto.



Gráfica 46. Ventana de inicio del software Unity

Al realizar los pasos anteriores, se espera unos pocos segundos a que cargue la herramienta con las respectivas carpetas involucradas en el proyecto, de manera que cuando cargue el mismo, aparecerá una pantalla como el de la Gráfica 47.



Gráfica 47. Interfaz de Usuario

Para referenciar la interfaz de usuario del inicio para proyectos en Unity, se hace la descripción por bloques para dejar claro la mayoría de herramientas presentes en cada uno de ellos, para esto se realiza un etiquetado numérico a cada uno de los bloques presentes en el entorno gráfico de la interfaz de inicio, que provee el software de simulación Unity.

Bloque 1:

Gráfica 48. Bloque 1

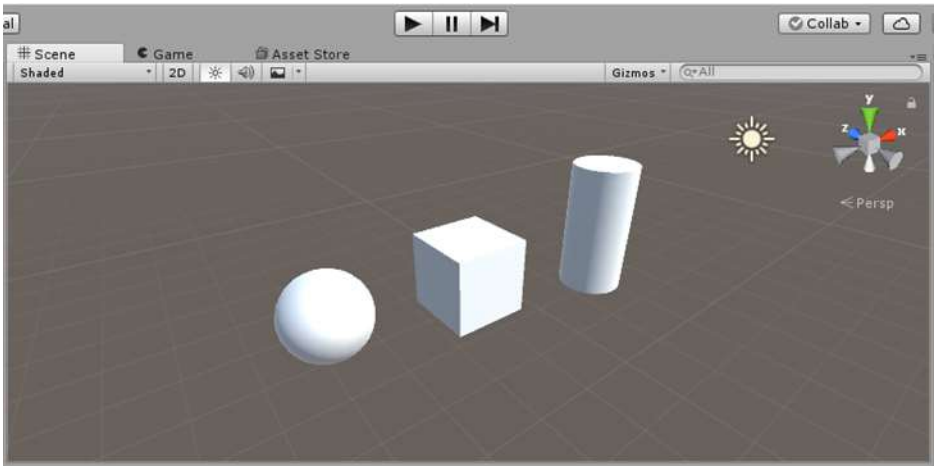
En este bloque 1 de la interfaz (Gráfica 48), se puede identificar principalmente las opciones para la vista general de un proyecto, en principio con el Untitled, que representa el nombre de la escena. Main Camera permite al usuario situarse en determinadas posiciones en el proyecto haciendo un enfoque particular de lo que se quiere mostrar al iniciar una simulación, esta cámara permite encuadrar escenas las veces que sea necesario para realzar una situación. Otra opción es el Directional light, que hace posible redirigir la luz del sol en un ángulo o posición deseada, con el fin de crear sombras o iluminar partes de las escenas donde sea conveniente dicha iluminación. En Create Unity permite agregar los elementos necesarios para las escenas, los cuales tienen por nombre GameObjects, estos son figuras geométricas básicas que pueden ser objeto de diversos cambios, las figuras o GameObjets como se llamaran de ahora en adelante están disponibles en 3D y 2D. También se pueden agregar en este bloque, luces para iluminar escenas, audios, videos, partículas en el ambiente y añadir otra cámara para la visualización si así se requiere. Cada vez que se agregue un elemento de los cuales se ha hecho mención, estos irán apareciendo en este bloque, seguido de Main camera y Directional Light que son las opciones que vienen por defecto a la hora de la creación de un proyecto. Por último, se observan los iconos que aparecen en la parte superior y que se aprecian en la Gráfica 49.

Gráfica 49. Icono de desplazamiento y dimensión de los objetos

Estos iconos son los que permiten el desplazamiento del usuario en la escena, el dimensionamiento de los GameObjects presentes en ella y los de giros o rotaciones en posiciones adecuadas para el entorno gráfico de simulación.

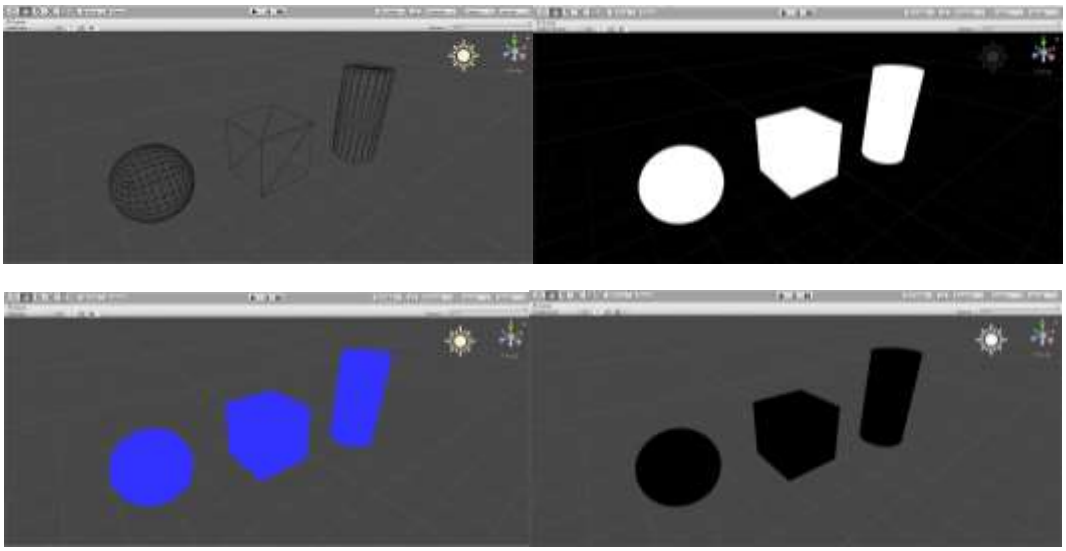
Bloque 2:

El bloque 2 es el que permite la visualización del entorno grafico que se está desarrollando, en él se encuentran las opciones de Game y Scene, siendo esta ultima la que se está mostrando en la Gráfica 50 y donde se realizan los cambios y configuraciones pertinentes para el desarrollo del producto final de simulación. En Game es el que permite ver el proyecto con todos los detalles de animación configurados, solo pulsando el botón de play en la parte superior o se pinchando la pestaña de mismo nombre, cabe anotar que en las dos visualizaciones se pueden hacer recorridos como personaje principal con el fin de observar el funcionamiento del desarrollo tanto en sus movimientos como en detalles de diseño, en este bloque si se requiere, se puede visualizar las escenas en 2D y volver a 3D si se cree conveniente para tener perspectiva del entorno gráfico.



Gráfica 50. Visualizador de Scene de Unity

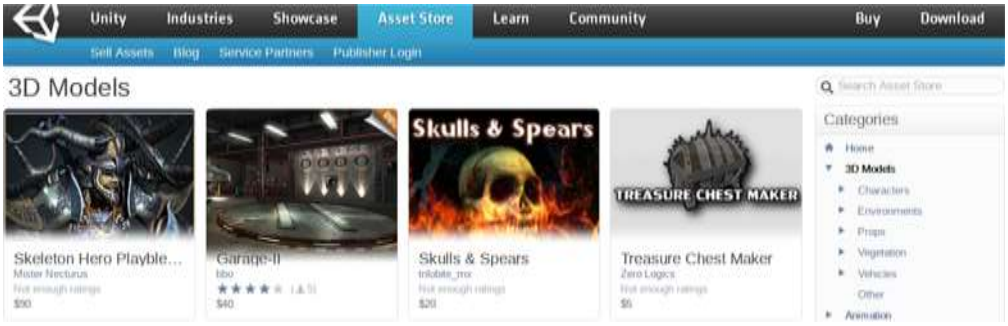
En las pestañas de este bloque 2 también se puede encontrar opciones para la iluminación y la vista en escena como; Shaded que es la opción que permite esta configuración, donde las tres figuras geométricas y el plano presentes se pueden ver con diferentes contrastes y colores, la Gráfica 51 muestra este tipo de herramienta de visualización para una misma escena.



Gráfica 51. Iluminaciones de escena

Un aspecto para nombrar en la visualización del bloque 2 son los mismos, que se utilizan para ayudar a la configuración de visualización de escenas, es decir, habilitar o inhabilitar componentes de animación, gráficos, audios, luces, fricciones, cámaras etc.

Para terminar se hace mención de una de uno de los ítems interesantes que posee el software Unity, el cual es el acceso a una tienda virtual llamada Asset Store, donde se puede encontrar gran variedad de modelos en 3D, para el desarrollo de un proyecto, donde algunos de ellos son gratuitos y otros de paga, en caso de no encontrar el recurso necesario en la tienda, el software Unity tiene la ventaja de poder importar elementos o figuras diseñadas en 3D y 2D de otros sitos o plataformas software especializados en este tipo de recursos, solo hay que tener en cuenta que las extensiones de lo que se desee importar tienen que tener la terminación .obj o .fbx , que son formatos reconocidos por el software Unity para las animaciones de escena.

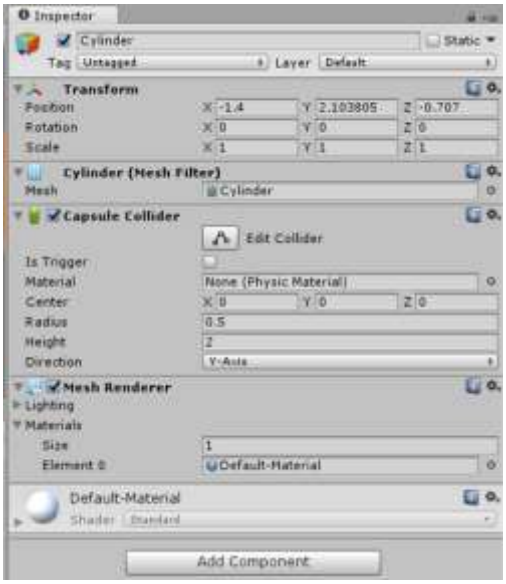


Gráfica 52. Vista de la tienda virtual de Unity

Bloque 3:

Este bloque es uno de los que más características posee, sin embargo se hace un análisis general de este bloque, puesto que algunas características o configuraciones que pueden ser añadidas a lo largo del proyecto no son parte del mismo, por tanto se realiza una descripción detallada del contenido y propiedades presentes en los GameObjects que hacen parte de la simulación del sistema de manufactura en las escenas, estas propiedades ya dependen de la particularidad de cada objeto y que función realiza en la escena, como se puede observar en la figura 20, el Inspector muestra características de un GameObject que ha sido seleccionado, en este caso se seleccionó un cilindro, para observar que características básicas posee, se dice básicas por el simple hecho de que las propiedades que puede contener un GameObject son muchas, y por ende cada objeto tiene unas determinadas características básicas por defecto como por ejemplo en el componente Transform, este contiene la posición, rotación y escala en cada uno de los ejes x, y, z, en el cilindro, lo que permite posicionar un GameObject en prácticamente cualquier posición de la escena. El Capsule Collider, es una característica de como la cápsula que envuelve el

objeto seleccionado interactúa con los demás objetos, es decir hay una cápsula imaginaria que envuelve al objeto y la cual puede ser modificada en su radio en su centro y en su altura, para que cuando colisione con un objeto de la escena dependiendo de las dimensiones que se hayan seleccionado, este cambie su comportamiento, puede ser un cambio de color o la desintegración del GameObject que esta colisionando con él, entre otros características. Siguiendo con la descripción Mesh Renderer sirve para para realizar opciones de iluminación y sombras del objeto en cuestión con bastante detalle. Ya para finalizar el último aspecto tiene que ver con los materiales y colores usados en los GameObjects es el Default-Material, los cuales se pueden seleccionar, crear o importar o simplemente escoger los que ofrece la herramienta Unity en ese sentido, los GameObjects de por si vienen con un color blanco-grisáceo, que es el color por defecto que incluye dicho objeto y al cual se le pueden realizar cambios de color de gracias a la paleta de colores presentes en este ítem.



Gráfica 53. Inspector

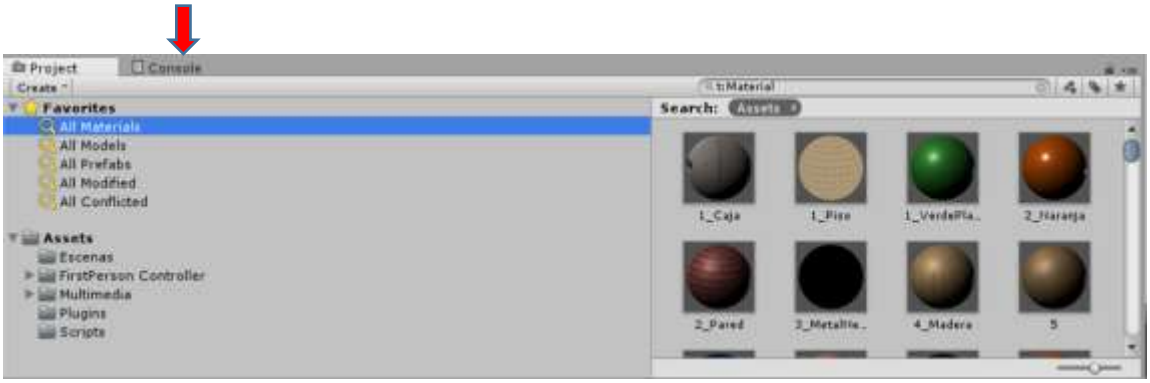
Hay que dejar en claro que las descripciones son básicas, debido a que las características más complejas se verán posteriormente en este capítulo, además hay que decir que las descripciones que se elaboraron fueron las de una figura geométrica común y corriente y las características que vienen por defecto con ella.

Bloque 4:

Este bloque representa la gran variedad de elementos y características añadidos o creados en Unity para cualquier proyecto desarrollado, en donde se puede encontrar la lista de materiales creados o texturas importadas anexadas previamente a los

GameObjects, o que pueden ser creadas para un uso posterior de las escenas, así como se puede apreciar en la Gráfica 54, donde el lado izquierdo pertenece a la búsqueda y a las carpetas relacionadas a todo el material presente en la simulación, como; modelos gráficos, plugins, escenas creadas, control de visualización, scripts y multimedia. En el lado derecho se puede observar el resultado y visualización de la búsqueda de elementos que se realiza en las opciones del lado izquierdo. Para este caso se ha realizado la búsqueda de materiales de todos los componentes gráficos presentes en el proyecto, dando como resultado una cantidad considerable de colores y texturas representadas como esferas.

Por último, la pestaña en la parte superior izquierda de nombre Console, es la encargada de mostrar el estado de la simulación en cuanto a los errores de programación y errores de funcionamiento del software, esto con el fin de proporcionar detalles del desarrollo del proyecto y de las funcionalidades y configuraciones que lo componen.



Gráfica 54. Visualizador de componentes de proyecto

Anexo B. Tiempos de simulación para los modos de funcionamiento

# Prueba	Modo	# Productos	Daño (%)	Captura Registros (s)	Tiempo (min)
1	1	10	20	30	2.44
	2	10	20	30	5.26
					2.42
2	1	20	20	30	5.20
	2	20	20	30	9.50
					4.30
3	1	30	20	30	7.22
	2	30	20	30	15.03
					7.41

Tabla 27. Comparación de tiempos de simulación

En la Tabla 27, se muestran 3 pruebas de simulación, la cual contiene diferentes parámetros de entrada, estos parámetros son comunes para los dos modos de funcionamiento de cada prueba, se observa además la diferencia de tiempos en los modos de funcionamiento con la misma configuración de inicio.

Anexo C. Descripciones de software de simulación

ARENA

ARENA es un software de simulación por computadora, que ofrece muchas herramientas para entender de manera fácil y detallada las cualidades del sistema que se está plasmando en dicha plataforma, ya que además de representar el sistema, efectúa automáticamente diferentes análisis del comportamiento de este, con el fin de encontrar las mejores alternativas de uso o diferentes caminos, para llegar a una mejor solución. Arena también facilita la disponibilidad del software, formado por módulos de lenguaje siman (lenguaje de simulación) el cual es utilizado para modelamiento de sistemas complejos de manufactura. Este programa permite también combinar las ventajas de los simuladores de alto nivel con la flexibilidad de lenguajes generales

Arena incluye además animaciones dinámicas en el mismo ambiente del trabajo que se esté desarrollando y provee apoyo integrado, incluyendo gráficas para los diseños estadísticos analizando los aspectos más importantes que son parte del estudio del sistema simulado en esta herramienta.

AUTOMOD

AutoMod es un software de simulación tridimensional, en el cual se modelan los sistemas de distribución, manufactura y manipulación de materiales más complejos, gracias a que combina las características de facilidad de uso de un lenguaje de simulación. La experiencia de **AutoMod** en el modelado de manejo de materiales y operaciones logísticas lo ha convertido en el producto de elección en el sector de automoción, alta tecnología, aeropuertos, puertos, correos, almacenamiento, manejo de materiales y muchas otras industrias.

La animación en 3D, la simulación eficiente y numerosas opciones de análisis son fáciles de usar debido a que es una herramienta detallada en cuanto a la interfaz y las palabras clave que se presentan en la misma, lo que da una muy buena percepción para el manejo del software **AutoMod**.

AutoMod ofrece una variedad de aplicaciones posibles debido a su núcleo de simulación muy eficiente. Éstos se extienden por ejemplo para modelar procesos de fabricación con simulación del almacén y simulaciones de la cadena de fuente a la conexión es decir una emulación en línea.

Estructura / módulos

- AS / RS (tecnología de almacén)
- Transportador (tecnología transportadora)
- Cinemática (robots)
- Pathmover (transportadores de suelo)
- Puente grúa (puente de grúas)
- Energía y libre (transportadores del carro)
- Tanques y Tuberías (líquidos, etc.)
- AutoTruck (vehículos de transporte)

WITNESS

Es una herramienta de simulación poderosa de procesos logísticos y de fabricación, donde se modelan entornos de trabajo, donde se simulan las implicaciones de las diferentes decisiones de cualquier proceso sin importar su complejidad, debido a esto este software provee resultados de simulación, en los cuales se analiza el mejor de ellos para crear estrategias de negocios empresariales y de ahí tomar el mejor camino para saber si se aborda o no una inversión o simplemente se realizan los cambios respectivos en el sistema simulado.

El enfoque del software **Witness** se basa en que crea representaciones de los sistemas en cuanto a la visualización de la vida real de los mismos, a través de modelos dinámicos que contiene la herramienta, proporcionando a su vez datos en medidas de producción que posteriormente el software transforma para mejorar los procesos, evaluando las alternativas y en donde además se pueden realizar mejoras continuas, para fomento la creatividad y del trabajo en equipo, gracias a su interfaz gráfica interactiva y que permite comprender en gran medida el proceso que se lleva a cabo

Características

- Facilidad de uso con implementación estándar en Windows
- Herramienta muy interactiva
- Potente conjunto de opciones de control y lógica
- Diseño sencillo y potente a través de bloques

- Estructura modular y jerárquica
- Dibujo del proceso
- Técnicas y métodos de optimización.
- Visualización en 3D.
- Análisis de minería de datos.
- Predicciones, planes y scheduling.

OPENMODELICA

OPENMODELICA es el entorno de simulación y modelado de Modélica, el cual es un lenguaje de código abierto basado en el lenguaje orientado a objetos, dicho lenguaje es fundamentado en la ecuación, para modelar sistemas físicos de gran complejidad y en los cuales se ven inmersos componentes térmicos, hidráulicos, electrónicos, mecánicos, eléctricos y control orientado a procesos. Este software está destinado a uso industrial y académico donde su desarrollo está respaldado por Open Source Modelica Consortium (OSMC) que es una organización no gubernamental sin fines de lucro, que fomenta el desarrollo del código abierto del lenguaje informático Modelica.

Esta instrumento software provee a los usuarios documentación, libros, códigos, tutoriales, ejemplos, librerías, scripts, además de la descarga gratuita del paquete de OpenModelica al cual se le pueden ir añadiendo las diferentes herramientas, dependiendo la necesidad del usuario o simplemente actualizar los módulos convenientes que están disponibles en la página oficial de OpenModelica, así como los otros componentes ya mencionados, a continuación se mencionan las diferentes herramientas que posee OpenModelica, además de hacer una breve descripción de cada uno de ellos.

STATEFLOW

Stateflow es un software de diseño gráfico interactiva para modelar y simular lógica de decisión combinatoria y secuencial basado en máquinas de estado y diagramas de flujo, esta herramienta software es a menudo utilizada para modelar lógica de control dirigido por estados o eventos, donde se controla dinámicamente los dispositivos físicos, ya que por medio de las señales que estos emiten, se puede modelar una forma en la cual el sistema pueda reaccionar ante los eventos o cambios que se puedan generar en los dispositivos.

Entre las prestaciones que encontramos de Stateflow de matlab se pueden nombrar.

Características

- Visualización de diagramas de estado, así como también detección de errores en tiempo de ejecución.
- Semántica de ejecución determinista con jerarquía, paralelismo, operadores temporales y eventos.
- Representaciones de MATLAB y Simulink de algoritmos periódicos y continuos en diagramas de Stateflow, tablas de verdad y diagramas de flujo
- Entorno de modelado, componentes gráficos y motor de simulación para modelizar y simular lógica compleja.
- Diagramas de estado, tablas de transición de estado y matrices de transición de estado que representan máquinas de estado finito.

FACTORY I/O

La herramienta Factory I/0 es un software que permite construir y simular fábricas en 3D rápidamente, para aprender tecnologías de automatización, para ello este software cuenta con pieza y escenarios inspiradas en aplicaciones de la industria los cuales tienen diferente nivel de dificultad a la hora de un diseño o de detección de fallas de escenarios ya elaborados , este software está diseñado en gran medida para la conexión a un PLC que es el controlador más común en la industria, sin embargo también puede conectarse a otras plataformas.

Características

- **Drivers I/0**: El software cuenta con la posibilidad de interactuar con muchas tecnologías como son por ejemplo: Modbus, PLC, SoftPLC entre otras, además cada edición de la plataforma incluye drivers para una tecnología en específico.

- **Componentes industriales:** Gracias a la biblioteca de componentes presente en el software se puede crear una planta virtual, ya que esta cuenta con más de 80 componentes industriales que incluye sensores, estaciones, ascensores entre otros.

- **Personalización de escenarios:** En Factory I/0 se encuentran unas herramientas para la edición inteligente en cuanto a la construcción de escenarios 3D, esto con el fin de que al utilizar la herramienta se tenga una experiencia cómoda y natural al crear escenarios propios.

- **I/0 digital y analógico:** La mayoría de los componentes de la plataforma disponen de I/0 digitales y analógicas, las cuales se pueden usar como mejor convenga a la hora de crear un escenario y en donde el ingenio hace parte del criterio para seleccionar el mejor dispositivo que se pueda integrar un buen diseño.

- **Modo de fallos:** Es un modo más bien enfocado a la parte académica en donde hay sistemas parcialmente desarrollados, con el único objetivo de que los alumnos puedan encontrar los fallos a esos sistemas y poder desarrollar su capacidad de síntesis.

- **Integración sencilla:** A través de Factory I/0 se pueden integrar el software al equipo de aprendizaje entre ellos están: Siemens Y Allan Bradley PLC (Ethernet) cabe anotar que para otras marcas se hace necesario un cableado especial o se necesita tarjetas de interfaz.

- **Prácticas de control:** para practicar tareas de control Factory I/0 cuenta con más de 20 situaciones inspiradas en aplicaciones industriales típicas que son realistas para el desarrollo del control en los diferentes escenarios.

UNITY

Unity es una de las más completas plataformas que existen para el desarrollo de videojuegos, la cual permite la creación de estos para múltiples plataformas a partir de un único desarrollo, en donde se pueden encontrar desarrollo para las consolas como PlayStation, Xbox y Wii en escritorio Linux, PC y Mac en móviles y tabletas como iOS, Android, Windows Phone y BlackBerry, esto en un inicio indica lo poderosa y versátil que es la herramienta Unity para diferentes propósitos.

Es una tecnología que posiblemente es la de mayor crecimiento, gracias al lenguaje de programación utilizado, la compatibilidad con otras plataformas, y los entornos gráficos que maneja la herramienta entre otros aspectos.

Debido a la gran extensión en los componentes de unity, con respecto a servicios funciones y plataformas compatibles, se hará énfasis en los módulos que competen para el desarrollo del proyecto en esta plataforma.

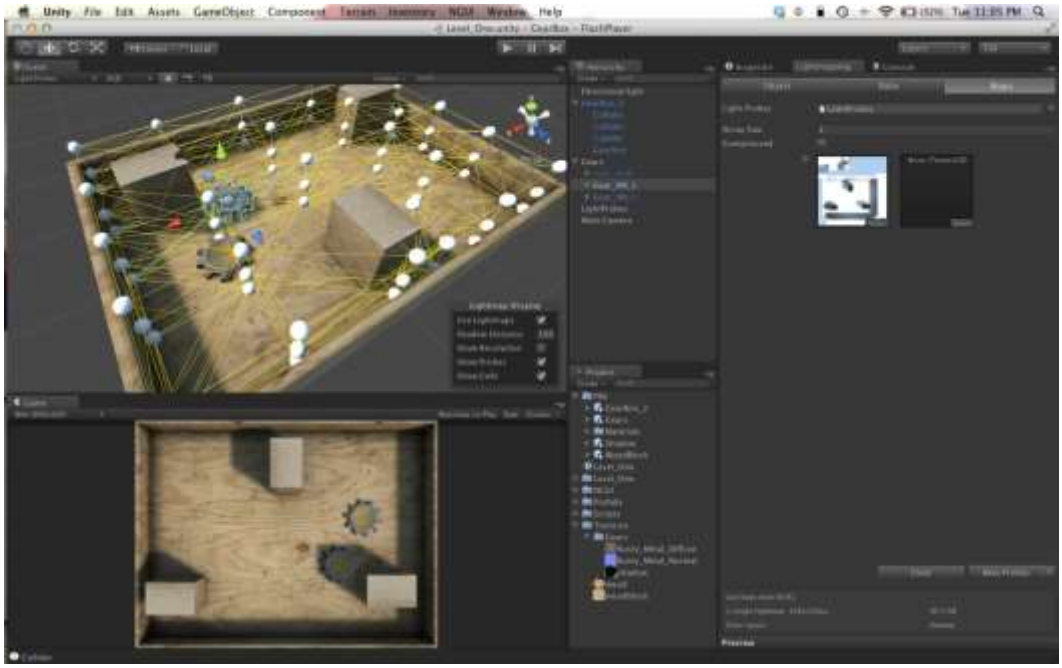
Entorno de Unity3D

1. **Explorador**: Lista todos los elementos (o activos) de tus proyectos. Permite ordenar de forma sencilla tu aplicación. En esta vista se encuentran tus imágenes, escenas, scripts, audios, prefabs, texturas, atlas y todos los elementos que se usaran en un juego o aplicación.
2. **Inspector**: Muestra y define las propiedades de los elementos de tu proyecto. Modifica valores de forma rápida, cambia texturas arrastrando ficheros desde el Explorador, añade scripts, guarda prefabs entre otros.

3. **Jerarquía**: Lista jerárquica de los elementos de una escena.
4. **Escena**: Diseño y maqueta de un juego completo o una pantalla o sección de éste. Cada escena representa un nivel o sección diferente del juego (portada, nivel 1, nivel 2, etc.) Simplemente se arrastra activos desde el Explorador y se editan las variables desde el Inspector.
5. **Juego**: Visualiza el juego a distintas resoluciones.

   Cabe resaltar que unity también tiene tecnología para desarrollo en 2D donde tiene casi las mismas prestaciones que la tecnología 3D descrita anteriormente.

En la **Gráfica 55**, se puede apreciar el entorno en 3D de Unity, donde se está realizando una estación, la cual tiene en su parte superior la escena que es donde se realizan los diseños y cambios pertinentes, que incluyen cambio de color, objetos en movimiento, luminosidad, dimensionamiento entre otros, y la parte inferior muestra la escena configurada ya en modo videojuego, que es como el cliente ya observa en definitiva el desarrollo del proyecto, en el lado derecho simplemente se observa la jerarquía de carpetas figuras u objetos que hacen parte del diseño en el software.



Gráfica 55. Entorno grafico de Unity3D

Unity también posee otras características y funciones que ayudan y potencian la realización de videojuegos y otros proyectos en la plataforma, tales características son:

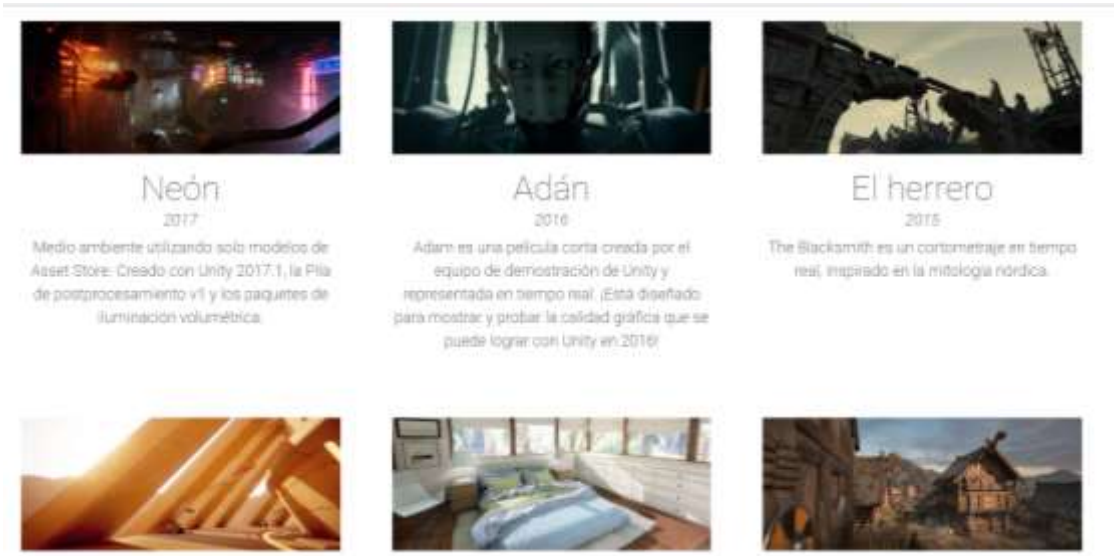
- Renderizado de gráficos
- Plataformas
- Editor ampliable
- Herramientas de arte y diseño
- Multijugador
- Realidad virtual y aumentada
- Rendimiento del motor
- Tienda de activos unity
- Colaboración entre equipos

Recursos bibliográficos

En la página oficial de Unity se puede encontrar una gran cantidad de soporte bibliográfico como tutoriales y documentación, que permiten realizar trabajos de mucha complejidad, gracias a que estos incluyen proyectos de gran calidad en modo tutorial para ir desarrollando paso a paso en diferentes niveles de dificultad, además cuenta con temas como la física de los objetos, interfaces, programación en C++ , animaciones, como desarrollar gráficos en 2D 3D , informes de rendimiento entre muchos otros como complemento de todas las funciones que se pueden desarrollar en Unity, adicionalmente a esto en la página oficial de Unity se pueden descargar manuales de como aprender a manejar Unity desde lo más básico a lo más complejo, pasando por los objetos que se diseñan en la plataforma hasta la programación de los mismos indicando comandos en C++ para la movilidad o animación de las escenas creadas en este software.

Demos

Los demos que presenta la página de Unity son proyectos realizados por gente del común con el fin de que los nuevos desarrolladores puedan ver el alcance del software y se puedan dar idea de las cosas que se pueden crear, estos demos van acompañados de videos y pequeñas descripciones de cómo se lograron algunos efectos grafica 13.



Gráfica 56. Demos desarrollados en Unity

Comunidad

En la comunidad de Unity, se pueden analizar soluciones y experiencias de desarrolladores que han querido compartir el conocimiento con los usuarios de la plataforma, donde se puede retroalimentar de la información proporcionada y recibir respuestas de la comunidad, además de poder entrar a foros de temas importantes dependiendo el proyecto al cual se desea desarrollar.

La comunidad también provee un issue tracker que es un rastreador de problemas para obtener información del estado de los errores presentados en algún desarrollo.

Certificaciones

Por si fuera poco, Unity ofrece la posibilidad de certificarse como desarrollador en 4 especialidades, esto con el fin de validar los conocimientos y las habilidades de programación.

- Certificado de desarrollador
- Certificado de programador experto en juegos
- Certificado de artista técnico experto en rigging y animación
- Certificado de artista técnico experto en sombreado y efectos

Estos certificados tienen un valor que fija la plataforma, no está por demás resaltar que para lograr estos certificados hay que presentar unos exámenes en línea para adquirir el logro.

Navegadores

Unity 3D también permite desarrollos para navegadores. Todos los navegadores modernos permiten reproducir estos juegos desarrollados en Unity, como por ejemplo

- Google Chrome
- Firefox
- Internet Explorer
- Safari.

Gráfica 57. Juegos realizados en Unity3D para diferentes plataformas

Tarifas

Para los desarrolladores es una barrera por el costo total de la plataforma, aunque se puede empezar por la licencia gratuita, pero tiene ciertos límites, esta versión trae consigo el logotipo de Unity en la carga inicial del juego o proyecto desarrollado y solo puede usarse si la facturación total de la empresa no supera los 100.000 anuales.

El precio de la licencia de Unity Pro es de 1.500 dólares por persona más los impuestos. Permite el uso de todas las prestaciones de Unity Pro en hasta 2 ordenadores (de la misma persona). Las principales mejoras se encuentran en efectos, texturas y rendimiento 3D.

Anexo D. Scripts

Basado en la Tabla 25. Se realiza una descripción de los scripts que componen la simulación del proceso, para ello se dividen en dos clases, una de ellas es los códigos de control que se resaltan en franja azul y la otra clase son los de color rojizo que son códigos auxiliares para el funcionamiento del proceso.

Scripts control:

Los scripts pertenecientes al control son los de la instrumentación del proceso y el código para comunicación entre el entorno gráfico y la base de datos

Scripts instrumentación del proceso:

- **Remachadora:**

  La codificación de la remachadora se realiza, iniciando el objeto en 3D como una variable de Unity llamada ***GameObject*** , posteriormente se inicia el disparador de evento ***OnTriggerEnter*** que es el disparador que se ejecuta cuando hay un mosquetón para remachar, para activar el disparo en la remachadora se recurre a la función de programación ***Collider*** que es la que indica internamente si hay producto o no, esta función permite configurar el sensor de posición de la remachadora, después de finalizado el remachado, se recurre a una variable de programación para que indique que la remachadora a terminado su función y que la mesa giratoria ya puede encenderse.

- **Mesa giratoria:**

  La mesa giratoria es uno de los códigos de mayor complejidad, ya que en ella se realizan varias funciones que son las siguientes:

  bandera false o true: (retorna el valor del estado de la mesa giratoria por cada movimiento los valores son 1 encendido y 0 para apagado.)

  PlayerPrefs: (permite que a partir de la figura del mosquetón en 3D se puedan realizar las operaciones de transporte, es decir que el mosquetón se mueva con la mesa)

  transform.eulerAngles: (Función compuesta que es la encargada de realizar los cambios de posición de la mesa giratorio en cuanto a los ángulos de giro que se requieren en el proceso, a esta función se le pasan los parámetros de entrada que son los grados que se quiere que la mesa del giro)

  WaitForSeconds: (función que recibe como argumento un tiempo determinado en caso de que se quiera que la mesa haga una pausa aun después de haber sido disparado un evento que indique moverse)

  La lógica de programación para la mesa giratoria incluye la invocación de las variables ***collider*** de la remachadora el pulverizador y el robot manipulador 1 que son los que desencadenan el disparo de la mesa, esta interrelación se ejecuta gracias a la invocación de las variables de otros objetos en la lógica de la mesa giratoria, esto se realiza colocando el nombre de la variable en funciones como ***girarmesa***, ***velocidadmesa*** entre otros, con el objetivo de realizar operaciones con dichas variables.

- **Pulverizador:**

El código de programación del pulverizador de pintura cuenta con dos operaciones el primero es el OnTriggerEnter para indicar el disparo para que se ejecute la función del pulverizador y la segunda operación es el ***OnTriggerExit*** que es la función de programación que indica cuando el disparo inicial está terminado y el objeto queda listo para otro disparo de evento, en esta función, también se puede codificar el aspecto final del producto, en este caso el color con que sale el mosquetón.

- **Robot 1**

  El robot 1 cuenta con un código de programación con variables para las pinzas, brazo y giro del robot manipulador 1, estas variables permiten ser incluidas en una animación de movimiento del robot, para que se vea más real, estas animaciones son permitidas en el motor de Unity que se puede encontrar en el ***inspector*** donde se puede realizar cambios de tiempo de ejecución, velocidad del robot, en un editor especial que pertenece a la plataforma.

  La lógica se realiza con las sentencias ***transform.position + Variable, es*** decir a partir de la información de la variable asociada al objeto, se realiza una transformación de posición en la simulación.

- **Banco de prueba**

  El script del banco de prueba tiene como principal característica, que es aquí donde se invocan la mayor cantidad de variable y operaciones de la simulación ,puesto que es aquí donde se ejecuta la prueba de que si un mosquetón es bueno o malo, esta invocación se realiza con finalidad de que el mosquetón sea recogido por el robot manipulador 2 si es bueno o sea expulsado por el eyector si es malo, además en este código está el funcionamiento del banco de prueba para que a través del ***collider*** realice el desplazamiento de la placa.

  La codificación se realiza con las sentencias ***OnTriggerEnter y OnTriggerExit*** para la funcionalidad del banco, las otras funciones se realizan involucrando códigos de la simulacion cuya descripción se encuentra posteriormente.

- **Robot 2**

  La programación del robot 2 se realiza exactamente igual a la programación del robot 1 con las mismas funciones, operaciones y variables.

Para todos y cada uno de los códigos de programación de la instrumentación del proceso, se tienen en cuenta las variables **guardardatos()** que sirve para guardar el estado del proceso y **getstado()** que es para capturar las señales del proceso

Script para comunicación con la base de datos:

- **Base de Datos:**

El código que pertenece a la comunicación de la base de datos, es el código cuya función es recopilar todos los eventos del proceso por cada instrumento, con la función ***p.Add(0)*** que permite crear un espacio en memoria para que posteriormente sea llenado con un dato de proceso, el cero entre paréntesis ira cambiando dependiendo el número de instrumentos que hayan en el proceso. Seguidamente se recurre a la función ***GetComponent*** para incluir la lógica del instrumento y obtener su estado, después se implementa el ***Debug.Log*** que es el que Recopila la información de los sensores codificados y le envía esta información a la función de ***insertar ()*** , una vez se encuentra la información en esta función se procede a la conexión con la base de datos con la sentencia ***IDbConnection dbconn;*** la cual tiene como objetivo enviar toda la información del proceso a SQLite.

Scripts simulación:

Los códigos de programación clasificados como scripts de simulación son los encargados de realizar funciones lógicas a cada elemento asociado a él, son auxiliares en el sentido de que pertenecen a la simulacion, pero no pertenecen a la instrumentación que conlleva el proceso

Script control de movimientos del operario:

A partir de las funciones propias de un ser humano y de los movimientos del mismo, se realiza una configuración jerárquica de las piezas que componen la figura en 3D del operario, esto se hace con el fin de que la figura mencionada no sufra alteraciones en los movimientos, que han sido codificados para dicha pieza, es decir se realiza un orden jerárquico de las piezas con el fin de darle movimiento natural a la figura, por ejemplo una mano es inferior jerárquicamente que el antebrazo, ya que si se mueve el antebrazo que es superior ,por ende se debe mover la mano del operario.

Los movimientos naturales del operario se realizan gracias al componente de Unity llamado Rigidbody, que es el componente físico de la plataforma, la codificación del operario se realiza mediante los comandos, que se encuentran en el Rigidbody asociado a cada gráfico en 3D

Programación:

El operario se anima o se configura para los movimientos naturales con el componente ***Transform*** de Unity, el cual permite mover la pieza en 3D en los ejes x, y, z configurando cada posición en la que se desea cada vez que en el proceso incluya un mosquetón en la alimentación, seguidamente se realiza una operación llamada ***OnTriggerEnter*** que es la encargada de indicar si el operario está haciendo su labor o está en espera, dando como retorno un ***true*** o ***false*** dependiendo del estado en proceso del operario.

Scripts determinación para buenos y malos:

La lógica para esta determinación, es un código basado en la sentencia **if** el cual permite categorizar los mosquetones, en buenos y malos, gracias al parámetro de inicio de configuración del porcentaje de daño del producto, donde este valor es el argumento de entrada en el código para que se ejecuten las líneas programadas con el fin de separar los mosquetones en el banco de prueba.

Programación:

Este código de programación cuenta con dos operaciones la primera es el ***OnTriggerEnter*** en la cual se codifica mediante un vector llamado ***Vector3*** que es una sentencia de Unity para guardar posiciones en un vector y de ahí, tener la posibilidad de hacer operaciones, en esta operación se codifican los mosquetones buenos con la función ***Ramdom.Range*** que es la encargada de escoger un mosquetón aleatoriamente para que se pueda definir como bueno o malo. Por último, se utiliza la función de Unity ***OnTriggerExit*** para finalizar el código y que este por medio de una bandera con retorno de entero, muestre el estado de los productos buenos y malos en una variable.

Script registro de datos de productos a fabricar:

Este registro tiene como finalidad, la visualización de los mosquetones que se quieran elaborar en el proceso, en el Canvas de la simulación en Unity se aprecia en la parte superior el icono de alimentación, el cual contiene los mosquetones, que en la configuración de inicio de indicaron, estos irán disminuyendo conforme la simulacion del proceso vaya corriendo.

Programación:

Este código a través de una lógica operacional de obtener un valor de usuario y guardarlo en una variable X, permite realizar la resta de mosquetones, mediante un disparador de evento llamado ***Trigger***, donde dependiendo el modo de funcionamiento del proceso y de la configuración del operario, este código restara el valor de usuario mientras que el operario alimente el proceso.

Script contabilizador de mosquetones

Para el proceso es fundamental que se realicen contabilizaciones de los mosquetones en la simulación, es decir hacer registros de contabilizador de alimentación, buenos y malos en cualquier parte de la escena.

Programación:

La contabilización de mosquetones del proceso se realiza a partir de creación de variables en el código, las cuales interactúan en todo el proceso, puesto que son llamadas cada vez que finalice una función de un instrumento, para ello se usan funciones de suma y resta para que estas operaciones se puedan visualizar en el Canvas del proceso simulado. La función de contabilizador se realiza con la sentencia

Tag**,** donde cada mosquetón que sale de la alimentación es marcado automáticamente por esta sentencia, lo que permite que a pesar de que sea el mismo objeto o figura tenga una identificación interna dentro del proceso, con la finalidad de realizar operaciones o enviar información del estado del proceso, por ejemplo, cuantos mosquetones permanecen en la mesa giratoria en un instante de tiempo. La sentencia u operación se encuentra en el ***inspector*** de Unity.

Para tener la facultad de poder programar en Unity, es necesario tener nociones o base de programación en C++ , para entender las sentencias y funciones que provee Unity, estas sentencias se encuentran definidas de una manera completa en la sección de documentación de Unity, donde además se encuentran ejemplos tanto básicos como complejos de todas y cada una de las funcionalidades y parámetros de cada sentencia, edemas de poder implementarla, puesto que también se muestra la estructura o la sintaxis de cada función para ser implementada en el editor de código.

Unity cuenta además se tutoriales en video para complementar los conocimientos adquiridos donde se enseñan a hacer videojuegos desde el inicio para principiantes, esto da la posibilidad de ir creciendo como programador en C#.

Anexo E. Plataformas compatibles con Unity.

- IOS
- Android
- Windows
- Universal Windows platform
- Mac
- Linux
- WebGL
- PS4
- PSVITA
- XBOXONE
- 3DS
- Oculus Rift
- Google cardboard
- Steam VR PC
- Playstation VR
- Gear VR
- Windows mixed reality
- Daydream
- Android TV
- Samsung Smart
- TvOS

- NINTENDO
- Fire OS
- Facebook Gameroom
- Google ARcore
- Vuforia
- Apple ARkit