Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Simulación de entornos de producción mediante LEGO® MINDSTORMS NXT

Autor: José Enrique Sánchez López

Tutor: José Manuel Framiñán Torres

Dpto. Organización y Gestión de Empresas I
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Simulación de entornos de producción mediante LEGO® MINDSTORMS NXT

Autor:

José Enrique Sánchez López

Tutor:

José Manuel Framiñán Torres

Catedrático de Universidad

Dpto. de Organización y Gestión de Empresas I

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado: Simulación de entornos de producción mediante LEGO® MINDSTORMS NXT

Autor: José Enrique Sánchez López

Tutor: José Manuel Framiñán Torres

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2013

El secretario del Tribunal

Agradecimientos

A mis Padres, por todo.
(y para que respiréis tranquilos)

A José Manuel, por su paciencia y dedicación.

Resumen

La necesidad de poner en práctica métodos de enseñanza más atractivos para los alumnos universitarios es cada vez más clara. En todas las disciplinas técnicas, las clases prácticas donde el alumno debe experimentar y aplicar los conocimientos teóricos adquiridos van ganando terreno frente a otros métodos más tradicionales. La docencia en el ámbito de la Organización y la Producción industrial, tradicionalmente teórica, no debe ser una excepción. Por esta razón, este TFG tiene por objetivo desarrollar un simulador físico, a pequeña escala, de un proceso industrial, cuyo fin sea ser utilizado en asignaturas de la intensificación de organización y producción del Grado en Ingeniería en Tecnologías Industriales. Para construir el simulador se ha utilizado el producto electrónico LEGO® Mindstorms y su funcionamiento ha sido programado en MATLAB a través de una librería llamada RWTH Toolbox, desarrollada en la Universidad Ténica de Aquisgrán.

Índice

Resumen **vii**

Índice **viii**

Índice de Figuras y Tablas **x**

1. Objeto del Proyecto **1**

1.1 Justificación del Proyecto. *1*

1.2 Sumario de los capítulos. *2*

2. Descripción de la Problemática **4**

2.1 Estudio de contenidos. *4*

2.1.1 Contenidos generales. 4

2.1.2 Industria 4.0: Smart factory. 7

2.2 Análisis de Requisitos. *9*

2.2.1 Requisitos generales de un simulador. 9

2.2.2 Requisitos específicos del prototipo construido. 9

3. Selección de Elementos Físicos y Software **11**

3.1 Alternativas Consideradas. *11*

3.2 LEGO Mindstorms NXT. *13*

3.2.1 Contenido del producto. 13

3.2.2 Software. 14

3.2.3 Conexión Bluetooth con NXT. 14

3.3 RWTH Toolbox. *16*

3.3.1 Instalación 16

3.3.2 Conexión a través de MATLAB. 16

3.3.3 MotorControl. 18

3.3.4 Utilización de varios NXT simultáneamente. 19

3.4 Utilización RWTH Toolbox. *20*

3.4.1 Elementos básicos. 20

3.4.2 Sensores. 22

3.4.3 Utilización Motorcontrol. 22

4. Diseño, Construcción y Programación **25**

4.1 Diseño del prototipo *25*

4.1.1 Modo de funcionamiento 26

4.2 Construcción del simulador. *28*

4.2.1 Brazo robótico (Load/unload station). 28

4.2.2 Cinta transportadora. 29

4.2.3 Máquinas 30

4.2.4 Tecnología de código de barras. 31

5. Validación del Prototipo **34**

6. Conclusiones y Propuestas **38**

6.1 Inversión: Lego Mindstorms EV3. *39*

6.2 Sistemas control producción y niveles de inventario. *40*

6.3 Aleatoriedad y fallos. *40*

6.4 Industria 4.0 *41*

7. Anexos **42**

7.1 Código comentado. *42*

7.2 Resultados simulaciones. *46*

Bibliografía **48**

ÍNDICE DE FIGURAS Y TABLAS

Figura 2.1: SCP – Matrix [Fuente: Rohde *et al*., 2000] 4

Figura 2.2: Esquema estación de producción [Fuente: Framiñán, 2017] 5

Figura 2.3: Industria 4.0 [Fuente: Stich, 2017] 7

Figura 3.1: Logos [Fuente: Internet] 11

Figura 3.2: Menú NXT [Elaboración Propia] 14

Figura 3.3: LEGO MINDSTORMS Edu NXT (I) [Elaboración propia**]** 15

Figura 3.4: LEGO MINDSTORMS Edu NXT (II) [Elaboración propia] 15

Figura 3.5: Bluetooth.ini [Elaboración propia] 17

Figura 3.6: Motorcontrol, NeXT Explorer. [Elaboración Propia] 19

Figura 3.7: NXT Motor [Elaboración propia] 23

Figura 4.1: Esquema del proyecto. [Elaboración propia] 25

Figura 4.2: Esquema del proyecto (II) [Elaboración propia] 27

Figura 4.3: Fotografía del prototipo (Alzado) [Elaboración propia] 28

Figura 4.4: Brazo Robótico [Elaboración porpia] 28

Figura 4.5: Fotografía del prototipo (Planta) [Elaboración propia] 29

Figura 4.6: Link tread wide with two pin holes (nº 57518) [Fuente: Jezek, 2000] 29

Figura 4.7 Sensor Ultrasónico [Fuente: NXT User Guide] 31

Figura 4.8 Lector de códigos de barras [Fuente: HONEYWELL] 31

Figura 4.9: Generación web de códigos de barras [Elaboración Propia] 32

Figura 4.10: Códigos de Barras utilizados [Elaboración propia] 33

Figura 5.1: Tiempos de llegada. Máquina 1 [Elaboración propia] 35

Figura 5.2: Tiempos de llegada. Máquina 2 [Elaboración propia] 35

Figura 5.3: Tabla de resultados de simulaciones [Elaboración Propia] 36

Figura 5.4: Tiempos de llegada. Histograma [Elaboración propia]..... 36

Figura 6.1: LEGO Mindstorms EV3 [Fuente: Internet] 39

Figura 6.2: Flujo de información. Aplicación Industria 4.0 [Elaboración propia] 41

Figura 7.1: Tabla de simulaciones [Elaboración propia] 47

1. OBJETO DEL PROYECTO

"Anything is easy if you can assimilate it to your collection of models"

- Seymour Papert –

Este Trabajo Final de Grado (TFG) tiene por objetivo diseñar, construir y programar un simulador físico, a pequeña escala, de un proceso productivo, mediante el producto electrónico LEGO Mindstorms NXT. Con ese objetivo en mente, se pretenden seguir unos pasos concretos.

En primer lugar, se quiere analizar los requisitos que debe cumplir un simulador de este tipo para que sirva a su cometido, así como su potencial desde un punto de vista didáctico. En segundo lugar, se quiere realizar una evaluación y posterior descripción de los distintos medios físicos y software, a nuestro alcance, para la elaboración de los simuladores. Adicionalmente, se aspira a solucionar o aclarar todas las dificultades técnicas que puedan presentar los medios físicos y software utilizados. Esto permitiría que futuros trabajos puedan centrarse en el desarrollo de los simuladores, sin tener que dedicar esfuerzos adicionales a la instalación y utilización de las herramientas seleccionadas.

1.1 Justificación del Proyecto.

Las ventajas de incluir actividades prácticas en la docencia, especialmente en la de índole científica, han sido ampliamente difundidas y son conocidas por todos. Las tradicionales clases magistrales están quedándose cada vez más obsoletas y nuevos métodos de aprendizaje se están abriendo paso. Seymour Papert (1928-2016), doctor en matemáticas y profesor del Massachusetts Institute of Tecnology (MIT), fue uno de los pioneros en apostar por el acercamiento de las nuevas tecnologías a las escuelas y desarrolló una teoría educativa llamada construccionismo. Dicha teoría, basada en el constructivismo del psicólogo Jean Piaget (1896-1980), aboga por una enseñanza basada en la experimentación y donde el alumno se enfrente a problemas reales de forma autónoma, potenciando así su creatividad. Esto se puede resumir con el concepto *"Learning-by-making"* (Papert, 1991), o "Aprender haciendo" en castellano, a través del que se introducen todas las bondades de una participación constante del estudiante en la instrucción, a la que Papert prefiere referirse como 'construcción'.

Frente a otras ramas de la ingeniería industrial (mecánica, química, etc.), donde es más habitual realizar actividades prácticas en un laboratorio, en los estudios de la rama de Organización y Producción no resulta tan fácil. Para ello sería necesario que los alumnos trabajasen en un proceso industrial real,

sobre el terreno, estudiándolo, recogiendo datos, introduciendo cambios y analizando como afectan dichos cambios al proceso en estudio. Por razones obvias, todo esto se antoja imposible desde un punto de vista económico. Así que, o bien limitamos el construccionismo a una herramienta dialéctica, tal y como se hace en las ciencias sociales, o bien exploramos otras opciones.

El primer objetivo de este trabajo final de grado es, precisamente, sentar las bases para la puesta en marcha de clases prácticas en el ámbito de la enseñanza de la Organización y Producción industrial. La intención es mejorar la docencia introduciendo tareas que incrementen la motivación y la implicación del alumnado en asignaturas del ámbito mencionado. Para ello se propone la construcción de simuladores físicos, a pequeña escala, de distintos procesos productivos. De esta forma, los alumnos podrían realizar en el laboratorio un trabajo de campo que se asemeje al que se pueda realizar en torno a un proceso productivo real, pudiendo aplicar los conocimientos adquiridos en las clases teóricas.

1.2 Sumario de los capítulos.

En este primer capítulo se han marcado los objetivos del TFG y se ha justificado su utilidad y las fuentes principales de las que bebe. A continuación, se incluye un breve resumen sobre cada uno de los capítulos que componen este documento.

- **Capítulo 2. Descripción de la Problemática:** En este capítulo se hace un análisis sobre el potencial de esta estrategia didáctica. Se exponen contenidos, relacionados con la organización y la producción industrial, que se pueden ser impartidos usando simuladores. Posteriormente, se hace un análisis de requisitos de los simuladores para que cumplan con las expectativas.

- **Capítulo 3. Selección de Elementos Físicos y Software:** Aquí se abordan las herramientas elegidas para la construcción y la programación del simulador. Se valoran distintas alternativas y se justifica la elección. Finalmente, se explica detalladamente la utilización y la instalación de las herramientas escogidas (LEGO Mindstorms y RWTH Toolbox).

- **Capítulo 4. Diseño, Construcción y Programación:** En el capítulo 4, se desgrana el proceso de construcción y programación del prototipo creado como ejemplo, dejando claro el diseño y la estructura del simulador, así como su comportamiento y las características particulares del funcionamiento.

- **Capítulo 5. Simulaciones del Prototipo:** Aquí se explica como se han ejecutado las simulaciones del prototipo, se exponen los datos recabados y se extraen conclusiones sobre ellos. Hace las veces de un capítulo de validación del simulador.

- **Capítulo 6: Conclusiones y Propuestas:** Se esbozan lo que en, opinión del autor, deberían ser las futuras líneas de trabajo en este área, incluyendo mejoras aplicables directamente al prototipo presentado.

- **Anexos:** El final del documento consta de dos anexos. Por un lado, se adjunta la programación realizada en MATLAB para el comportamiento del prototipo. Por otro lado, se muestran los resultados completos de las simulaciones realizadas, para que puedan ser consultadas exhaustivamente si el lector lo requiere.

2. DESCRIPCIÓN DE LA PROBLEMÁTICA

Una de las mayores dificultades encontradas a la hora de desarrollar este Trabajo Final de Grado ha sido definir unos objetivos a alcanzar por el simulador, puesto que hay pocos precedentes que sirvan de guía. De esta forma, este capítulo tiene como objetivo evaluar distintos contenidos teóricos que puedan ser impartidos y posteriormente reflejarlos en una lista de requisitos que deban cumplir los simuladores para satisfacer sus objetivos docentes.

2.1 Estudio de contenidos.

En este apartado se detallan contenidos que pueden ser impartidos a través de las enseñanzas prácticas que se proponen en este documento. Desde los contenidos básicos de asignaturas del Grado en Ingeniería en Tecnologías Industriales, hasta conceptos innovadores como puede ser la Industria 4.0.

2.1.1 Contenidos generales.

Lo ideal sería que tratásemos de abarcar la mayor cantidad de contenido posible, y por lo tanto, también de asignaturas. Así, el alumno podría trabajar en todos los aspectos relacionados con la producción dentro de la cadena de suministro, que aparecen dentro del recuadro en rojo de la *Figura 2.1: SCP – Matrix.*

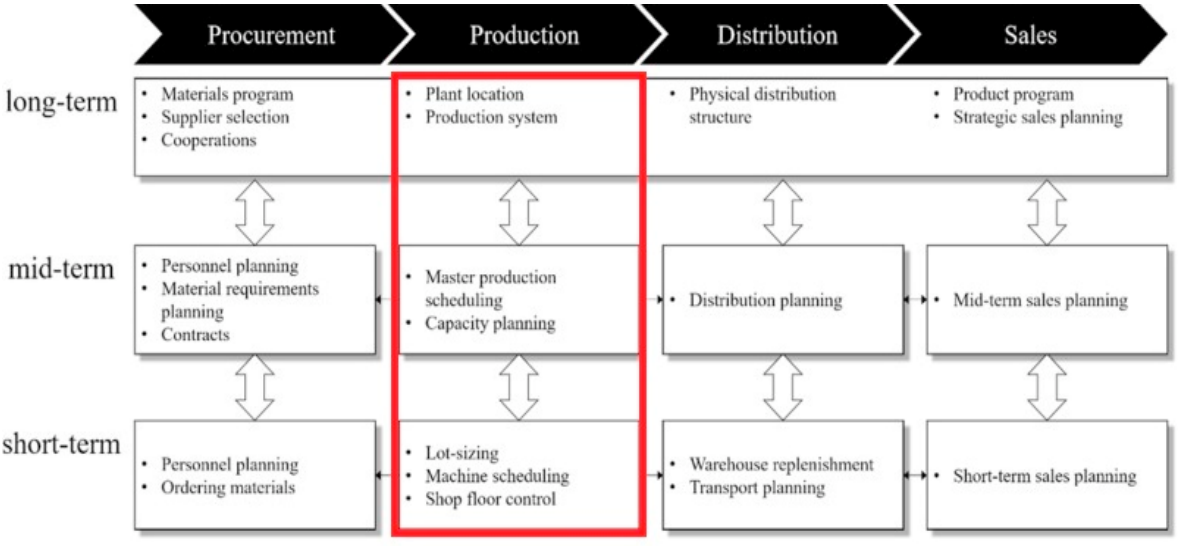

Figura 2.1: SCP – Matrix [Fuente: Rohde *et al.*, 2000]

La localización de la planta (Plant location) y el plan maestro de producción (Master Production scheduling o MPS) son quizás los aspectos más difíciles de trasladar a este tipo de enseñanzas prácticas. En el caso de la localización, porque habría que incluir elementos de transporte. En el caso del MPS, porque requeriría un tiempo del que no se dispone habitualmente en una asignatura universitaria. Sin embargo, tanto la elección del sistema de producción (Production system) y la distribución en planta de la fábrica, en el largo plazo, como la lotificación (Lot-sizing), la secuenciación (Scheduling) y el control de la producción (Shop floor control) en el corto plazo, parece que no plantean estos problemas. De esta forma, se presentan dos tareas posibles que proponer al alumnado:

- **Tarea 1: Diseño, construcción y automatización** de un simulador físico de un proceso industrial determinado, que trate de cumplir de forma óptima unas especificaciones dadas.
- **Tarea 2: Programación y control de la producción** de un simulador físico ya dado, donde los alumnos tendrán que entender y optimizar su funcionamiento.

Centrándonos en el contenido a impartir, se pretende que los alumnos se familiaricen y trabajen de forma práctica con conceptos vistos en asignaturas de 4º curso de GITI de la intensificación de organización y producción, como pueden ser "Diseño de Productos y Procesos" o "Programación de Operaciones". Algunos de estos conceptos están reflejados en la *Figura 2.2: Esquema estación de producción* y su manejo es fundamental para enfrentarse a las tareas propuestas anteriormente.

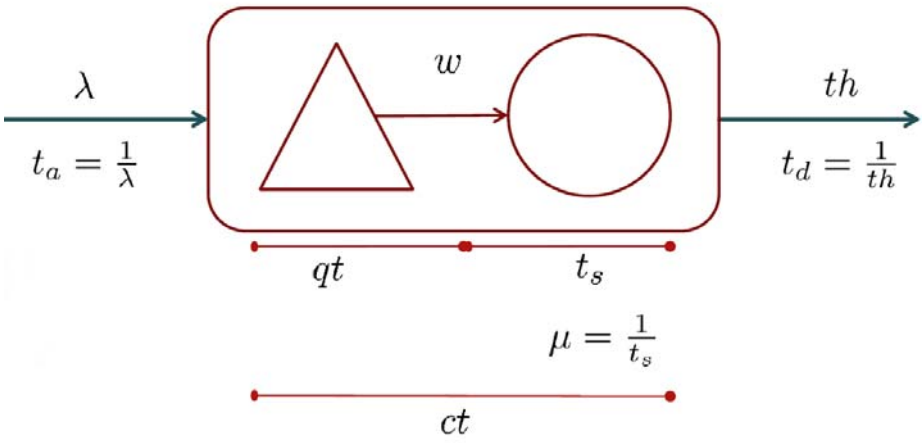

Figura 2.2: Esquema estación de producción [Fuente: Framiñán, 2017]

Indicadores:

- **λ**: Tasa media de llegada de trabajos.
- **t_a**: Tiempo medio entre la llegada de un trabajo y el siguiente. *(Time of arrival)*
- **th**: Tasa media de salida de trabajos. *(Throughput)*
- **t_d**: Tiempo medio entre la salida de un trabajo y el siguiente. *(Time of departure)*
- **w**: número de trabajos medio en la estación. *(WIP: Work in Progress)*
- **μ**: Tasa media de proceso de trabajos.
- **t_s**: Tiempo medio de procesado de trabajos. *(Time of departure)*
- **qt**: Tiempo medio de espera a ser procesado. *(Queue time)*
- **ct**: Tiempo medio de ciclo de un trabajo. *(Cycle time)*

Todos estos indicadores deben ser fácilmente medibles y quedar recogidos, para que el estudiante pueda comprobar la exactitud de sus aproximaciones o cómo se ven alterados al introducir cambios.

Entornos de proceso:

Los alumnos tendrán que familiarizarse con los distintos entornos de producción (Flowshop, Openshop, Jobshop, etc), tanto para el diseño del proceso, a la hora de decidir que *layout* es más interesante para cumplir las especificaciones dadas, como para la secuenciación de la producción, puesto que la optimización de dicha secuenciación depende en gran medida del entorno de estudio.

Sistemas de control push/pull:

En la segunda tarea, será indispensable establecer un sistema de control de la producción que garantice que el sistema se ajuste a los valores de diseño. Incorporar estos sistemas a nuestros proyectos, permitirá al alumno ahondar en sistemas más complejos como el Kanban o en sistemas híbridos que, en mi opinión personal, se tratan poco en nuestros estudios. Concretamente se estudia Kanban en profundidad en la asignatura 'Sistemas Integrados de Producción', porque es una parte fundamental del *Lean Management*, pero el resto de sistemas de control quedan bastante en el olvido.

- Sistemas push: Material Requirement Planning (MRP)
- Sistemas pull: CONWIP, Kanban
- Sistemas híbridos.

Una actividad muy interesante podría consistir en enfrentar al simulador a distintos tipos de demanda y que los alumnos tengan que comprobar qué tipo de producción es más eficiente ante cada tipo de demanda.

2.1.2 Industria 4.0: Smart factory.

Una propuesta más ambiciosa que la expuesta en el apartado anterior consistiría en aplicar los principios de la industria 4.0, fundamentalmente aquellos relacionados con el internet de las cosas, al simulador. Existe un vacío importante en nuestros estudios respecto a esta materia y puede ser una oportunidad excepcional para introducirla, al menos brevemente.

El concepto de *Smart Factory* como una fábrica cuyos distintos elementos están intercomunicados y son capaces de interactuar entre ellos, así como una recogida exhaustiva de una gran cantidad de datos que permita monitorizar la fábrica y ayude así en la toma de decisiones, puede ser implementado en el funcionamiento de algún simulador.

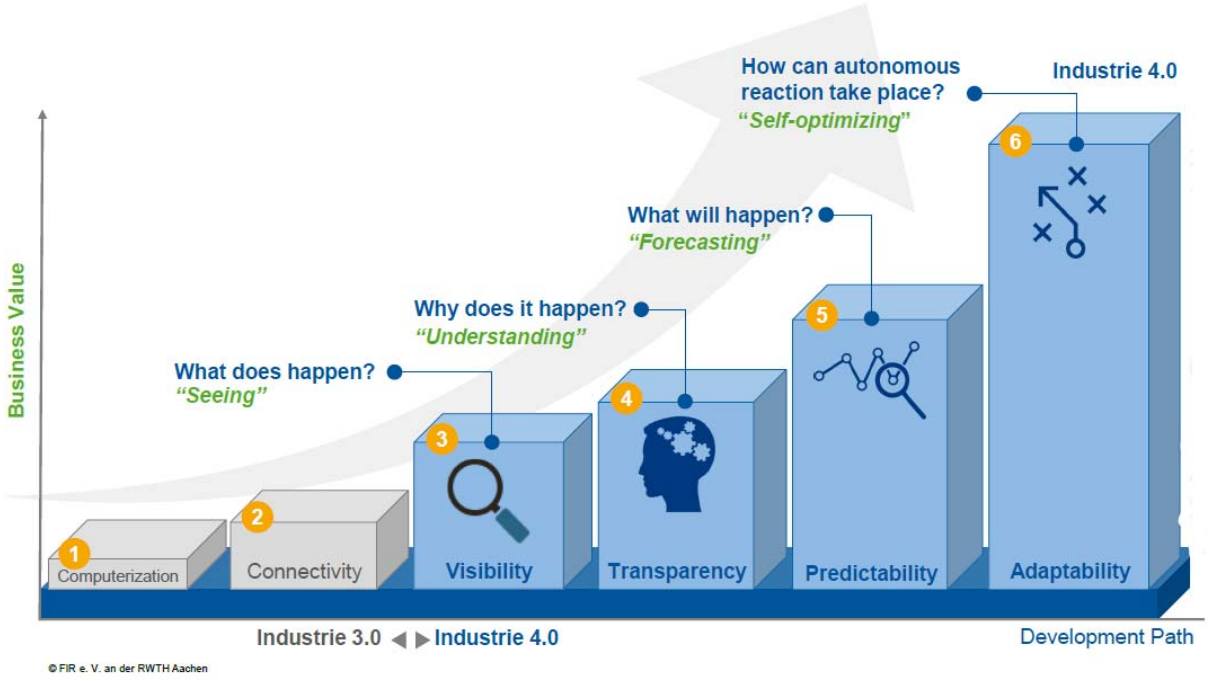

Figura 2.3: Industria 4.0 [Fuente: Stich, 2017]

En la *Figura 2.3: Industria 4.0*, se describe de forma sencilla cuáles son los cuatro pasos fundamentales en el camino hacia la Industria 4.0. En el eje de abscisas del gráfico encontramos el grado de implementación de la industria 4.0 frente al valor empresarial en el eje de coordenadas y representa un crecimiento exponencial del segundo respecto del primero. Los pasos son los siguientes:

- **Ver**: En primer lugar, hace falta una gran cantidad de sensores de todo tipo para obtener una buena base de datos sobre la que poder trabajar.

- **Comprender**: A continuación, hemos de hacer un análisis de los datos recogidos e interpretarlos para saber qué está ocurriendo en todo momento en nuestro proceso productivo. Cuando estos procesos adquieren gran envergadura y hay una gran cantidad de datos hablamos de *Big Data Analysis*. Unas de las herramientas más útiles en este estadio de la implementación son los análisis de correlación, a través de los cuales se buscan relaciones entre unos datos y otros, tratando de buscar justificaciones para determinados comportamientos del proceso.

- **Pronosticar**: Como extensión del punto anterior, aquí ya no solo se trata de saber qué elementos repercuten en otros y entender por qué, sino tratar de buscar indicadores que nos sirvan para poder predecir resultados o comportamientos.

- **Autonomía**: El último paso consiste en conseguir cierto grado de autonomía para el proceso productivo, de tal forma que pueda hacer frente a eventos inesperados por sí mismo o sea capaz de optimizar su funcionamiento en distintas situaciones.

Como veremos en el apartado *3.2.1 Contenido del producto*, disponemos de un amplio abanico de sensores LEGO que pueden facilitar el primero de los pasos que acabamos de comentar. Los dos pasos centrales (comprender y pronosticar) corresponden a un análisis de los datos recogidos, para el que tenemos distintas herramientas a nuestra disposición como, por ejemplo, hojas de cálculo o *software* como *R*. Gracias a las conclusiones obtenidas a través de dicho análisis, se pueden programar distintos comportamientos para el simulador, de tal forma que pueda modificar su funcionamiento de forma autónoma en función de distintos parámetros. Probablemente, este último paso plantee mayor dificultad, puesto que la programación podría llegar a ser muy compleja y no se trata del objetivo central de este TFG.

2.2 Análisis de Requisitos.

Además de los medios físicos y de software que utilicemos, hay una serie de factores importantes a tener en cuenta a la hora de diseñar nuestro simulador. En primer lugar, vamos a desgranar aquellos requisitos que deben ser comunes a cualquier simulador de este tipo, para, a continuación, detallar aquellos específicos del simulador elaborado en este TFG.

2.2.1 Requisitos generales de un simulador.

- **Tamaño del simulador:** Para poder simular un proceso industrial real tendremos que poner en marcha al menos una unidad mínima de producción, una estación, que está compuesta por un *buffer* y una máquina. Sin embargo, parece evidente que un sistema más complejo brindaría más oportunidades desde un punto de vista didáctico, puesto que seríamos capaces de mostrar un abanico de situaciones más amplio.

- **Automatización sencilla:** Nuestro objetivo fundamental es mejorar la calidad de la enseñanza de las asignaturas del ámbito de la organización y la producción, por lo que no debemos centrarnos en la elaboración de complejas soluciones para automatizar el comportamiento de nuestro simulador. De hecho, pueden llegar a ser contraproducentes si los estudiantes acaban dedicando muchos esfuerzos a ese respecto, en vez de centrarse en los conceptos de nuestro ámbito.

- **Recabar datos:** Para que los alumnos puedan poner en prácticas las herramientas teóricas aprendidas en clase, es necesario que puedan obtener la mayor cantidad de datos posibles del simulador. Especialmente importantes son los tiempos y para recogerlos se pueden usar distintos tipos de sensores, como los de luz o ultrasonido.

- **Almacenamiento de datos:** Como ampliación del punto anterior, los datos, una vez recabados, han de ser almacenados para proceder a su posterior tratamiento y análisis. La utilización de hojas de cálculo puede servir tanto para el almacenamiento como para dicho análisis.

2.2.2 Requisitos específicos del prototipo construido.

- **Procesamiento de varios tipos de trabajos:** Es interesante emplear al menos dos tipos de trabajos diferentes, con la posibilidad de que tengan características distintas, por ejemplo, distintos tiempos de servicio. Esto abriría un importante abanico de posibilidades y de situaciones en nuestro entorno de producción.

- **Trazabilidad de los trabajos:** Procesar más de un tipo de trabajo requiere de algún método de identificación para cada uno de ellos, a lo largo de las distintas etapas del proceso productivo, para poder cumplir con los requisitos generales de recabar y almacenar datos. Existen diversas posibilidades como pueden ser la tecnología RFID o la tecnología de código de barras.

- **Combinación de operaciones automatizadas y manuales:** La presencia combinada de operaciones automatizadas y manuales, nos permitiría observar cómo influye cada tipo al proceso, así como comparar sus efectos.

- **Más de una máquina:** Añadir más de una máquina al prototipo nos daría la posibilidad de tratar la disyuntiva paralelización/serialización. Si lo combinamos con el hecho de tener más de un tipo de trabajo, permitiría que el simulador adquiera comportamientos de entornos de producción de tipo *Jobshop*, *Flowshop* u *Openshop*. Abriéndonos la puerta al estudio de conceptos como la ruta de trabajos y la aproximación de distintos problemas de optimización en el campo de la programación de operaciones.

3. SELECCIÓN DE ELEMENTOS FÍSICOS Y SOFTWARE

Para llevar a cabo un simulador tenemos que construirlo y posteriormente, programarlo. Por lo tanto, necesitamos elegir tanto un soporte físico como un lenguaje de programación. En este capítulo, en primer lugar evaluaremos los distintos elementos a nuestro alcance para la construcción del simulador y justificaremos la elección final. Una vez hecho esto, procederemos a detallar la instalación y la utilización de dichos elementos para cumplir con uno de los objetivos marcados en este TFG.

3.1 Alternativas Consideradas.

Para la construcción del prototipo adjunto a este TFG, se ha utilizado, como ya se ha comentado, la generación NXT de la línea de productos Mindstorms de la empresa danesa LEGO. La principal razón estriba en que es el material del que disponía el tutor del TFG, sin embargo parece interesante evaluar otras posibilidades y compararlas de cara a posibles inversiones futuras.

En primera instancia, parecería interesante utilizar productos de las marcas Arduino o Raspberry Pi. Por un lado, Arduino ofrece una amplia gama de microcontroladores con distintas aplicaciones enfocadas fundamentalmente a la electrónica. Por otro lado, Raspberry Pi es un producto en sí mismo, consistente en un pequeño procesador con un sistema operativo propio y considerablemente más potente que los microcontroladores Arduino.

Figura 3.1: Logos [Fuente: Internet]

Si bien los procesadores y microcontroladores mencionados poseen muchas entradas que servirían para conectarlos con los sensores y actuadores del simulador, tanto una opción como otra carecen de

materiales tales como motores o piezas para los mecanismos y estructuras del simulador. Si quisiéramos utilizar estas líneas de productos, tendríamos que buscar por separado todos los actuadores, sensores y materiales de construcción que necesitáramos, lo que complicaría el proceso ostensiblemente. Otro aspecto a considerar es la programación, ya que Arduino utiliza una variante de C++ y el lenguaje más extendido en Raspberry Pi es Python. Los alumnos del Grado reciben poca formación sobre estos lenguajes, lo que podría entrañar problemas.

Al contrario que Arduino y Raspberry Pi, la línea de productos de LEGO Mindstorms proporciona *kits* de trabajo que incluyen todo aquello que necesitamos, desde procesadores hasta motores y sensores pasando por las piezas para construir estructuras y mecanismo. Además la programación se realizaría mediante el programa informático MATLAB, a cuyo lenguaje los alumnos están más que acostumbrados. Cabe destacar que hay experiencias previas de la utilización de LEGO Mindstorms en la docencia universitaria. El doctor Young Jae Jang, profesor del Korean Advanced Institute of Science and Technology (KAIST) ha construido una pequeña línea de producción para que sus alumnos experimenten sin necesidad de que el KAIST haya tenido que construir su propia fábrica.

Otro ejemplo de la utilización de Lego Mindstorms en la universidad es la Rheinisch Westfälische Technische Hochschule Aachen (RWTH Aachen), que ofrece un proyecto a sus alumnos de primer semestre de grados relacionados con la electrónica y la informática. En dicho proyecto, llamado *MATLAB meets LEGO Mindstorms* (Behrens *et al*., 2010). los estudiantes tienen que familiarizarse con la utilización de las herramientas de Lego y con la programación en Matlab, para después acabar desarrollando un robot propio que lleve a cabo alguna tarea de forma autónoma. Para que esto fuera posible, investigadores de dicha universidad desarrollaron la *RWTH Toolbox* (Kopcazka *et al*., 2012), un paquete de Matlab para poder programar fácilmente la generación NXT de Mindstorms. A este proyecto de la universidad alemana le siguieron, entre muchos otros, proyectos de la Universidad de Cambridge o de la norteamericana Universidad de Clemson.

En general, podemos decir que la utilización de estas herramientas está bastante extendida en facultades de electrónica y de informática, pero en el campo de la organización y la producción, o *Industrial Engineering* como se conoce internacionalmente, el profesor Jang es un pionero. En el documento **"***Statement of Teaching"* (Jang, 2016), se afirma que la respuesta de los estudiantes ha sido muy buena, sobre todo desde el punto de vista de la motivación y de la implicación de sus alumnos. Se asegura, además, que las notas no han dejado de mejorar desde que se implantó el programa.

3.2 LEGO Mindstorms NXT.

NXT es la segunda generación de la línea de productos Mindstorms de la empresa danesa LEGO®. Se puso a la venta en el año 2006 y sustituyó a la primera generación, llamada RCX. En 2013 fue lanzada la versión más reciente, la EV3, que es la única que sigue comercializándose en estos momentos.

A pesar de la existencia de varias versiones, todas ellas comparten una serie de características fundamentales que no han variado con el tiempo. La más importante es la existencia de un procesador con forma de ladrillo, al que nos referiremos habitualmente como '*brick* programable' o directamente 'NXT'. La función de este procesador es activar o parar los motores y recibir información de los sensores, que varían según la versión que tengamos, conectados al *brick* a través de un cable de ethernet.

La programación ha de hacerse mediante un PC, ya sea utilizando el software de LEGO o bien otro lenguaje de programación. Para la transmisión del programa desde el ordenador hasta el *brick* NXT se puede hacer uso de un USB o de la tecnología *Bluetooth*.

3.2.1 Contenido del producto.

Para la elaboración de este proyecto, nos hemos servido de material de la generación NXT del que ya se disponía:

Concretamente de 3 unidades del producto **Mindstorms Education NXT Base Set 9797**, que incluye:

- *Brick* programable (Incluye batería, cargadores y cable USB)
- Motores: 3 servomotores.
- Sensores: 2 sensores de contacto, 1 sensor de sonido, 1 sensor de ultrasonido y 1 sensor de luz.
- Material informático en forma de CD que incluye el NXT Software v2.1 y una guía de usuario para la utilización de Mindstorms.
- Piezas tradicionales de LEGO para la construcción de estructuras y mecanismos.

Además, contábamos con una unidad del producto **Mindstorms Education Resource Set 9648** que incluía piezas tradicionales habituales como complemento de las que vienen incluídas en el Set 9797. Para más información sobre los contenidos exactos de estos productos es interesante remitirse a la guía de usuario del producto (NXT User Guide v2.1, 2009).

3.2.2 Software.

Tal y como hemos anunciado en el apartado 3.2, la programación del comportamiento de los prototipos fabricados con Mindstorms puede hacerse de diversas formas. La primera de ellas es utilizar el software adjunto al Set 9797, llamado LEGO MINDSTORMS Edu NXT, que incluye un programa para la utilización de un sencillo lenguaje de programación gráfico que se asemeja mucho al LabVIEW. Este lenguaje, asocia cada acción a un tipo de bloque, de tal forma que estos pueden ser colocados uno detrás de otro creando una secuencia a la que se le pueden añadir bifurcaciones y realimentaciones. Como se puede comprobar, es un lenguaje muy sencillo y visual diseñado para niños, por lo que su uso en enseñanzas superiores universitarias resultaría poco interesante. La posibilidad de utilizar otros lenguajes de programación, útiles más allá de esta aplicación, para que el alumnado adquiera experiencia en su manejo, parece mucho más interesante.

3.2.3 Conexión Bluetooth con NXT.

Hemos elegido la conexión mediante *Bluetooth* entre el PC y nuestro NXT por su sencillez y por la comodidad que implica reducir el número de cables del prototipo. El proceso de conexión que se explica en este apartado solo sería suficiente si se fuese a programar con el software proporcionado por LEGO. Para utilizar otros lenguajes o programas, el siguiente procedimiento solo es el primer paso:

1. **Emparejamiento:** En primer lugar, debemos emparejar el NXT y el dispositivo desde donde vayamos a programarlo. Para ello accedemos al submenú Bluetooth del brick NXT desde el menú principial del mismo, como se puede ver en la *Figura 3.2: Menú NXT*. Dentro de ese submenú debemos activar el bluetooth y asegurarnos de que nuestro Brick se encuentra visible para que otros lo puedan detectar. Una vez hecho esto, buscamos el dispositivo que queremos conectar con el Brick a través de la opción 'Search' de este submenú. Éste nos sugerirá una clave para hacer más segura la conexión, la aceptaremos e introduciremos finalmente dicha contraseña en nuestro ordenador.

Figura 3.2: Menú NXT [Elaboración Propia]

2. **Comprobación de la conexión a través del Software de LEGO:**

Si hemos instalado el software, deberíamos poder iniciar una aplicación llamada *NXT 2.1 Programming.* Una vez abierta, iniciamos un nuevo programa como se indica en la *Figura 3.3: Software LEGO (I)*. Lo único que nos interesa es comprobar que el emparejamiento ha funcionado correctamente y que el software es capaz de detectar el Brick NXT.

Figura 3.3: LEGO MINDSTORMS Edu NXT (I) [Elaboración propia]

A continuación, llegaremos a otra pantalla donde podremos hacer las comprobaciones pertinentes. Después de hacer click en el botón indicado mediante el círculo rojo, tal y como se indica en la *Figura 3.4: Software LEGO (II)*, se nos desplegará el siguiente menú que debería tener este aspecto.

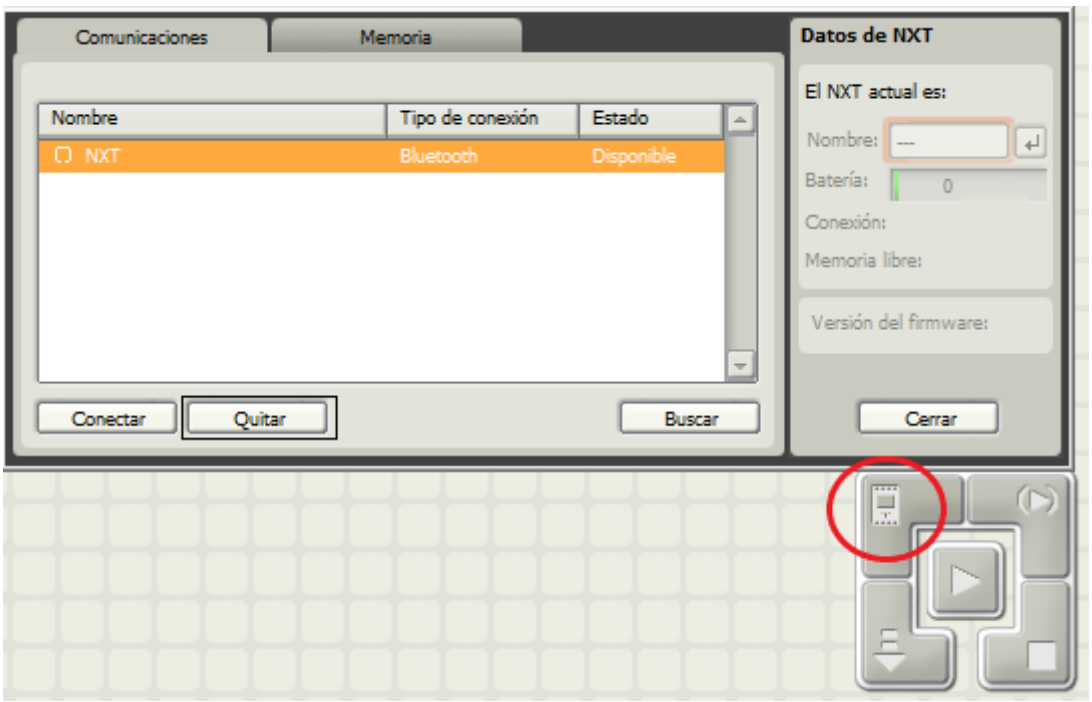

Figura 3.4: LEGO MINDSTORMS Edu NXT (II) [Elaboración propia]

Si todo ha ido bien, solo restaría terminar la conexión a través de MATLAB.

3.3 RWTH Toolbox.

Tal y como mencionamos en la introducción, la RWTH Toolbox es una extensión para MATLAB, desarrollada por investigadores de la universidad de Aachen, con el objetivo de permitir a sus alumnos programar la generación NXT de LEGO Mindstorms. Se ha utilizado la versión de MATLAB r2015b (Mathworks, 2015), si bien la Toolbox es compatible desde la versión 2008b. En este apartado comentaremos los aspectos más importantes de dicha extensión, fundamentales para acometer el proyecto.

3.3.1 Instalación

1. Descargamos la RWTH Toolbox del apartado de descargas de la página web del proyecto, *http://www.mindstorms.rwth-aachen.de* (Kopcazka *et al*., 2012). Hemos seleccionado la versión 4.07, la más reciente que ha recibido la categoría de 'estable'.

2. El formato del archivo descargado es .zip, asi que tendremos que extraerlo en una carpeta y colocarlo en el directorio deseado.

3. Hay que agregar ese directorio a la ruta de archivos de MATLAB, para que el programa tenga acceso a la Toolbox, este procedimiento puede variar según la versión de MATLAB. En la R2015b, la utilizada en este proyecto, hay un apartado en la pestaña de *Home*, llamada *Set Path* que sirve para este propósito. Hacemos click en *Add with Subfolder* y se nos abrirá un explorador de archivos, buscamos la carpeta donde hayamos extraído el contenido descargado, la seleccionamos y aceptamos.

4. Finalmente, comprobamos que hemos completado la instalación con éxito. Para ello utilizamos el comando 'ver' de MATLAB, que nos mostrará una lista de todas las extensiones instaladas. Si en dicha lista aparece 'RWTH - Mindsotrms NXT Toolbox', significa que todo ha ido correctamente.

3.3.2 Conexión a través de MATLAB.

Se ha descartado la utilización de la conexión a través de USB por ser mucho más compleja en el caso de utilizar un PC con un sistema operativo Windows 64 bits, que la conexión a través de Bluetooth.

Si hemos seguido correctamente los pasos del apartado *3.2.3 Conexión Bluetooth con NXT* no deberíamos tener problema en finalizar la conexión siguiendo estos pasos:

Bluetooth.ini: En primer lugar, debemos crear el archivo Bluetooth.ini en la carpeta donde estén todos los archivos de la RWTH Toolbox, siguiendo el ejemplo de un archivo de ejemplo proporcionado: *bluetooth-example-windows64.ini* o bien *bluetooth-example-windows32.ini*, dependiendo de nuestro sistema operativo. En la versión de 64 bits, la utilizada en este proyecto, únicamente debemos fijarnos en que el nombre del *brick* esté correctamente escrito. Esto es el primer campo del archivo donde podemos leer 'NXT-NAME=NXT'. En el caso de que sólo trabajemos con un *brick*, este paso es inmediato, pero en el caso de que estemos trabajando con varios Brick simultáneamente debemos prestar atención a este paso, como se explica en el apartado *3.3.4 Utilización de varios NXT simultáneamente.*

```
[Bluetooth]

NXT-Name=NXT
NXT-MAC=00165302F0DD
Channel=1

BaudRate=9600
DataBits=8

SendSendPause=5
SendReceivePause=25

Timeout=2
```

Figura 3.5: Bluetooth.ini [Elaboración propia]

***Handle*:** Para terminar la conexión, no tenemos más que llamar a la función COM_OpenNXT pasándole como argumento el archivo creado anteriormente 'bluetooth.ini'. Sin embargo hay que prestar atención a que esta función nos devolverá un elemento que pasaremos a llamar '*handle*' ('manejador' en castellano). Esto nos servirá como referencia para que, al utilizar el resto de funciones de la Toolbox, podamos pasar el *handle* como uno de los argumentos de la función y así especificar inequívocamente qué NXT brick queremos que reciba dicha función. Como se puede entender, esto es fundamental cuando trabajemos con varios NXT simultáneamente.

```
h=COM_OpenNXT('bluetooth.ini');
```

En el caso de que estuviésemos trabajando con un único brick, podemos configurar el único *handle* que vamos a utilizar como predeterminado y así no tenemos que introducirlo constantemente en el resto de funciones que utilicemos. Para ello utilizamos la función COM_SetDefaultNXT() con el *handle* como argumento.

```
COM_SetDefaultNXT(h);
```

Para la desconexión, disponemos también de otra función. Podemos cerrar una conexión concreta incluyendo su *handle* en el argumento o escribir 'all', con lo que cerraremos todas las conexiones abiertas

```
COM_CloseNXT('all');
```

3.3.3 MotorControl.

Si queremos controlar los motores de forma precisa necesitaremos utilizar una extensión de la RWTH Toolbox llamada MotorControl. Dicha extensión ya viene incorporada en la carpeta de la Toolbox, concretamente en el directorio */tools/motorcontrol*. Para poder utilizarla, debe estar ejecutándose un programa en el NXT llamado 'Motorcontrol22.rxe'. Este programa recibirá la información sobre el funcionamiento deseado de los motores y controlará su movimiento mientras el código en MATLAB sigue ejecutándose.

Por lo tanto, antes de poder usar la extensión, debemos introducir el programa en el *brick* y para ello no hay más que seguir los siguientes pasos:

1. En primer lugar, hay que descargarse la aplicación **'NeXTExplorer.exe'** desde el sitio web de BRICXCC (s.f.). Esta aplicación nos permitirá explorar los archivos del *brick* programable y pasarle el 'Motorcontrol22.rxe' del que hablábamos anteriormente.

2. Con el *brick* conectado a nuestro ordenador, preferiblemente a través del USB, ejecutamos NeXTExplorer. Como se puede comprobar en la *Figura 3.6,* la ventana emergente estará divida en dos, la parte de la derecha es un navegador de archivos en nuestro ordenador, así que buscamos ahí el Motorcontrol22.rxe, en la carpeta especificada anteriormente, y lo arrastramos a la parte izquierda de la pantalla, que representa a nuestro *brick*.

3. Finalmente, tenemos que ejecutar el archivo **TransferMotorControlBinaryToNXT.bat**, que se encuentra también en */tools/motorcontrol*, y seguir las instrucciones que se nos indican. Este archivo contiene un conjunto de instrucciones MS-DOS que finalizará el proceso. Lo único que faltaría por hacer es reiniciar los *bricks* programables y reconectarlos con nuestro ordenador.

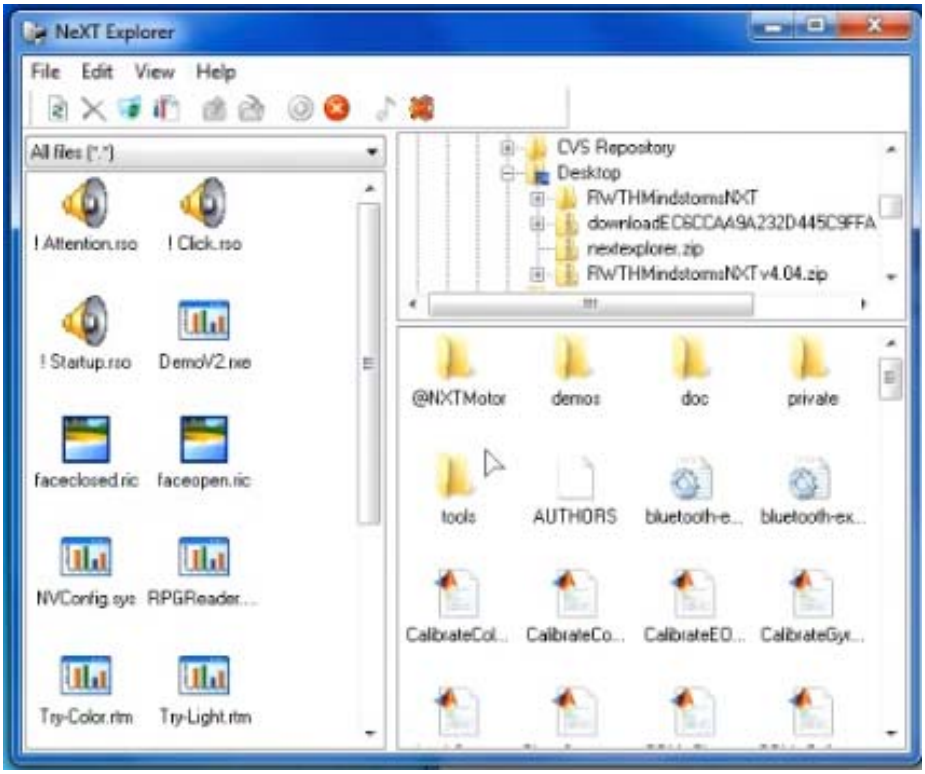

Figura 3.6: Motorcontrol, NeXT Explorer. [Elaboración Propia]

Cuando hayamos restablecido la conexión, si todo ha ido correctamente, en la pantalla de nuestro *brick* programable aparecerá un mensaje indicando que se está ejecutando 'Motorcontrol22.rxe'. Tenemos la oportunidad de pulsar el botón naranja para abortar la ejecución, de tal forma que todas las órdenes recibidas por el NXT, que incluyan comandos incluidos en MotorControl, no se ejecutarán. Esto puede ser muy útil para detener el movimiento en el caso de que la programación del mismo haya fallado y no estemos obteniendo el resultado deseado.

3.3.4 Utilización de varios NXT simultáneamente.

1. Conectamos mediante USB y abrimos la aplicación software de LEGO *NXT 2.1 Programming*, tal y como hicimos en el punto 2 del apartado *3.2.3 Conexión Bluetooth con NXT*

2. Cambiamos el nombre del *brick* programable. Todos vienen con un nombre predefinido 'NXT' y en la *Figura 3.4: Software de Lego (II)* podemos ver en el margen derecho de la imagen la opción que nos permite cambiar el nombre.

3. Tendremos que generar un nuevo archivo 'bluetooth.ini' por cada nuevo *brick* programable que conectemos. Lo recomendable es copiar y pegar el primero que

hayamos creado, modificarle el nombre (Ejemplo: 'Bluetooth2.ini') e introducir el nuevo nombre del NXT tal y como se explica en el apartado *3.3.2 Conexión a través de MATLAB.*

En este proyecto en concreto, como se han utilizado dos *bricks* programables, el primero ha conservado su nombre 'NXT' y su archivo *.ini* asociado se ha mantenido como 'bluetooth.ini'. Al segundo *brick* programable se le ha llamado 'NXT2' y a su archivo *.ini* 'bluetooth2.ini'

Nombre = NXT → bluetooth.ini Nombre = NXT2 → bluetooth2.ini

Alternativamente, el cambio de nombre también puede hacerse desde MATLAB a través de la función de la Toolbox:

```
NXT_SetBrickName(nombre, handle);
```

3.4 Utilización RWTH Toolbox.

Esta herramienta nos proporciona una serie de funciones que nos permitirán programar cómodamente una vez hayamos aprendido a manejarlas. Con la Toolbox instalada, siempre podremos utilizar el comando *help* de MATLAB seguido del nombre de la función deseada, para obtener una guía detallada sobre los objetivos de la función en cuestión y sobre su sintaxis.

3.4.1 Elementos básicos.

- **Puertos**:

Para facilitarnos el trabajo, la Toolbox incorpora variables cuyo valor es el número que identifica a cada puerto del *brick* programable, tanto para los motores como para los sensores. De esta forma, no tendremos que introducir un número en el argumento de las funciones que requieran que se especifique el puerto, si no el propio nombre del puerto. Para los motores: MOTOR_A, MOTOR_B y MOTOR_C. Mientras que para los sensores: SENSOR_1, SENSOR_2, SENSOR_3 y SENSOR_4.

- **Manejo básico de motores:**

Existe una función básica para controlar los motores sin mucha precisión:

```
DirectMotorCommand (port, power, angle, speedRegulation, syncedToMotor,
turnRatio, rampMode);
```

Es una forma sencilla y rápida de poner los motores en funcionamiento a la velocidad deseada, que se puede regular mediante el argumento *power*. En dicho argumento debemos introducir un entero entre 0 y 100, donde 100 es la máxima potencia, o entre 0 y -100 si queremos que gire en el sentido contrario. En cuanto al resto de argumentos son comportamientos especiales que no son utilizados en este proyecto y que quedan un tanto obsoletos frente al uso de Motorcontrol, como veremos más adelante. Para no tenerlos en cuenta, escribimos 'off' en su lugar, como vemos en el ejemplo a continuación. Información adicional sobre su su uso se puede consultar en la documentación (Kopcazka *et al*., 2012).

```
DirectMotorCommand (MOTOR_A, 30, 0, 'off', 'off', 'off', 'off');
```

Esta función plantea también otro problema, y es que no hay forma de indicar el *handle* y por tanto solo puede actuar sobre el NXT predeterminado en la conexión. Por otro lado, para parar el motor, tenemos dos opciones, o utilizar *DirectMotorCommand* poniendo un 0 en el argumento *power*, o bien utilizamos la siguiente función:

```
StopMotor (port, mode, handle);
```

- **Tiempos de espera: WaitFor( )**

Otro factor que debemos tener muy en cuenta a la hora de programar es el tiempo de espera. Cada vez que se ejecuta una función en MATLAB, es enviada inmediatamente al brick NXT para que haga la tarea encomendada. Por lo tanto, si enviamos dos comandos, de tal forma que se envíe el segundo antes de que se hayan terminado las tareas relacionadas con el primero, el *brick* programable rechazará el segundo comando y no lo llevará a cabo. Para solucionar este problema, podemos hacer uso de la función *WaitFor( )*. Es especialmente útil en combinación con la extensión MotorControl, como se explica en el apartado *3.3.3 MotorControl*.

Sin embargo, esta función puede llegar a generar problemas, ya que al utilizarla estaremos frenando la ejecución del código y esto puede dificultar la utilización de dos *bricks* programables en paralelo. Hablaremos de ellos y propondremos alguna solución también en el apartado *3.3.3 MotorControl*.

3.4.2 Sensores.

El procedimiento para la utilización de los sensores es bastante sencillo e igual para todos ellos, así que vamos a poner un ejemplo con el sensor de presión (sensor *switch*). En primer lugar, hay que abrir el sensor mediante la función:

```
OpenSwitch(port,handle);
```

A continuación, para obtener los valores indicados por el sensor, utilizaremos:

```
GetSwitch(port,handle);
```

Esta función devuelve un entero, así que tendremos que asignarla a una variable creada en MATLAB:

```
Valor_switch=GetSwitch(port,handle) ;
```

Finalmente, incorporaremos al final de nuestro código una función que cierre el sensor, para que no pueda interferir con lo que hagamos con posterioridad.:

```
CloseSensor(port,handle);
```

Nos podemos servir de este ejemplo para hacer uso de todos los sensores, puesto que el nombre de las funciones sigue también la misma sintaxis:

```
Open/get/close+tipo de sensor(port, handle);
```

Éste es un ejemplo práctico en el que se utilizan los sensores de distancia de ultrasonido:

```
  OpenUltrasonic(SENSOR_1, h);
distance=GetUltrasonic(SENSOR_1, h);
  CloseSensor(SENSOR_1, h);
```

Si quisiéramos, podríamos repetir el mismo procedimiento con otro tipo de sensores que incorpora el set de NXT. Como el de sonido (sound) o el de color (colour).

3.4.3 Utilización Motorcontrol.

Como ya se ha explicado, la extensión MotorControl nos ayudará a controlar de forma mucho más precisa el movimiento de los motores. La columna vertebral de dicha extensión es la función:

```
NXTMotor (port)
```

Pasando como argumento el motor que se vaya a utilizar (MOTOR_A, MOTOR_B o MOTOR_C), la función nos devuelve un objeto de la clase NXTMotor, cuyos campos o propiedades por defecto para el caso del motor B quedan reflejados en la *Figura 3.7: NXT Motor,* presente a continuación.

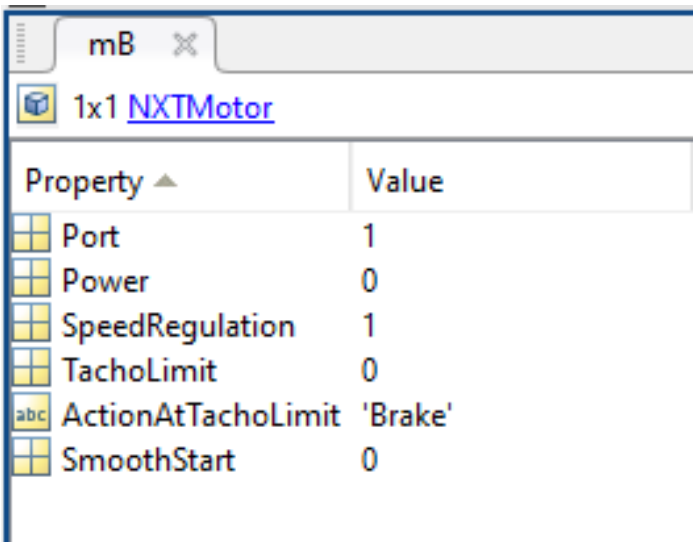

Figura 3.7: NXT Motor [Elaboración propia]

Las propiedades de especial interés para nosotros son el puerto (port) que queda definida en la creación del objeto a través del argumento de la función NXTMotor(port). La potencia, que funciona de la misma manera que en DirectMotorCommand, un entero entre 0 y 100 donde 100 es la potencia máxima y se añade un signo negativo si se quiere que el motor rote en sentido contrario. Por último, *Tacholimit*, que especifica el ángulo en grados que debe girar el motor en cuestión, puede tomar valores entre 0 y 999.999, donde el 0 indica un movimiento ininterrumpido.

El procedimiento, por tanto, es el siguiente:

1. Primero hemos de crear el objeto como se acaba de explicar:

```
mA=NXTMotor(MOTOR_A);
```

2. En segundo lugar, se han de modificar los campos del objeto creado en función del movimiento que queramos obtener. Por ejemplo, si quisiéramos que el motor diera una vuelta completa a media potencia en el sentido de las agujas del reloj, deberíamos indicar lo siguiente:

```
mA.Power=50;
mA.Tacholimit=360;
```

Aquí tenemos que tener en cuenta los tiempos de espera de los que hablamos en el punto *3.4.1 Elementos básicos*. Si queremos que el código pare de ejecutarse hasta que los motores terminen el movimiento que estamos programando, con el objetivo de evitar una posible pérdida de información, hacemos uso de la función OBJ.waitfor(timeout, handle) que introdujimos también en ese punto:

```
mA.waitfor(60, h);
```

El argumento *timeout* es un límite de tiempo (en segundos) para el tiempo de espera, es decir que, si transcurridos los segundos indicados los motores no hubieran terminado, el código continuaría ejecutándose. Esto resulta muy útil para que la ejecución no se quede estancada en el caso de que ocurriera algún tipo de error.

Sin embargo, esa función plantea una desventaja importante que habrá que sortear en la programación. Al dejar de ejecutarse el código, no se estará enviando información a ningún NXT, en el caso de que estemos utilizando más de uno. Esto puede provocar la existencia de intervalos de tiempo en los que un NXT no esté ejecutando una orden, cuando realmente está libre para hacerlo. La solución es reordenar el código cuidadosamente para enviar las órdenes a los distintos NXT en los momentos óptimos para el aprovechamiento del tiempo.

3. Finalmente, hemos de enviar el objeto que hemos creado y modificado al *brick* programable para que este ponga en marcha los motores. Para ello, disponemos también de una función:

```
mA:SendToNXT (h);
```

4. DISEÑO, CONSTRUCCIÓN Y PROGRAMACIÓN

Después de haber desgranado, en el capítulo *3. Selección de Elementos Físicos y Software,* todas las herramientas que vamos a utilizar y haber explicado minuciosamente todo aquello que pudiese generar problemas técnicos, en este capítulo vamos a detallar el procedimiento seguido para la construcción de nuestro propio simulador.

4.1 Diseño del prototipo

Nuestro simulador sigue el esquema de la *Figura 4.1: Esquema del proyecto.* El flujo de material comienza con la llegada de los trabajos al *Buffer*, desde donde son cargados a la cinta transportadora desde una estación de carga y descarga (*Load/unload station* en inglés).

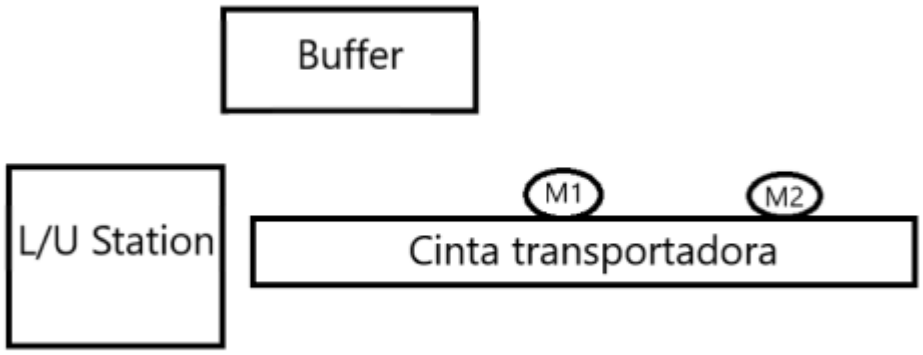

Figura 4.1: Esquema del proyecto. [Elaboración propia]

Para la automatización de la estación de carga y descarga se utilizará un brazo robótico dotado de unas pinzas capaces de cargar los dos tipos de trabajos que va a procesar el prototipo. Estos dos tipos de trabajos consistirán en bolas de mediano tamaño, incluidas en los sets de LEGO de los que se dispone. A simple vista se podrá diferenciar un tipo de trabajo de otro, porque las bolas serán de dos colores distintos (rojo y azul). Sin embargo, para llevar un registro exhaustivo, a cada trabajo se le ha adherido un código de barras que, como se explicará en el apartado *4.1.4 Tecnología de Código de barras*, resultará muy útil para diferenciar las características de los trabajos que se cargan en el sistema.

Finalmente, cabe mencionar que se han empleado dos *bricks* programables distintos, uno se encarga de controlar el brazo robótico y el siguiente controla la cinta transportadora y las máquinas M1 y M2.

4.1.1 Modo de funcionamiento

En este apartado se detalla el proceso que llevará a cabo el simulador y se explican algunas cuestiones importantes de la programación. La totalidad de la programación se puede consultar en el anexo *7.1 Código comentado*, concretamente en la función *final.m*. Esta función puede ser ejecutada después de *conectar.m* y *conectar2.m*, también presentes en dicho anexo, que se encargan de establecer la conexión entre los dos *bricks* programables en uso y el ordenador utilizado.

Los trabajos van llegando al buffer manualmente y a partir de ahí el proceso está automatizado con la excepción de la lectura de los códigos de barras, que ha de hacerse también manualmente. Al inicio del programa, han de indicarse el número de trabajos que se van a procesar de una vez, como si de un lote se tratase.

1. Posición inicial: El sistema parte de una posición inicial en la que el brazo róbotico se encuentra dispuesto para tomar un nuevo trabajo del buffer (orientado hacia el buffer) y el soporte de la cinta transportadora se encuentra al principio de la misma.

2. Carga del trabajo: En primer lugar, el brazo robótico se encarga de cargar el trabajo en la cinta transportadora, colocándolo encima del soporte, mediante un movimiento constante en el tiempo. Dicho movimiento comienza con la lectura inicial del código de barras. Mediante esa lectura, el sistema sabe que el trabajo está dispuesto y puede proceder a cargarlo en la cinta y a procesarlo.

3. Cinta transportadora: Una vez que el trabajo se encuentra sobre el soporte, la cinta transportadora se activa y hace avanzar el trabajo en cuestión hasta alcanzar la máquina 1, donde se parará los segundos que vengan estipulados en la variable *tiempo_servicio1*. Cuando haya transcurrido ese tiempo, la cinta transportadora volverá a activarse y se volverá a detener, el tiempo que indique la variable *tiempo_servicio2*, al alcanzar la máquina 2. Finalmente se pondrá en marcha unos segundos, determinados experimentalmente, necesarios para alcanzar el final de la cinta transportadora.

4. Salida del trabajo: Con el soporte situado en el final de la cinta transportadora, se puede proceder a retirar el trabajo. Esto ha de hacerse manualmente y hay que volver a leer el código

de barras para que quede registrado. Tras la mencionada lectura, se inicia automáticamente el último paso.

5. Retorno a la posición inicial: Se activa la cinta transportadora, pero en sentido opuesto al utilizado hasta ahora para poder devolver el soporte a la posición inicial. Con el objetivo de agilizar el proceso, el brazo robótico ya volvió a su posición inicial mientras los trabajos estaban siendo procesados en las máquinas (punto 3).

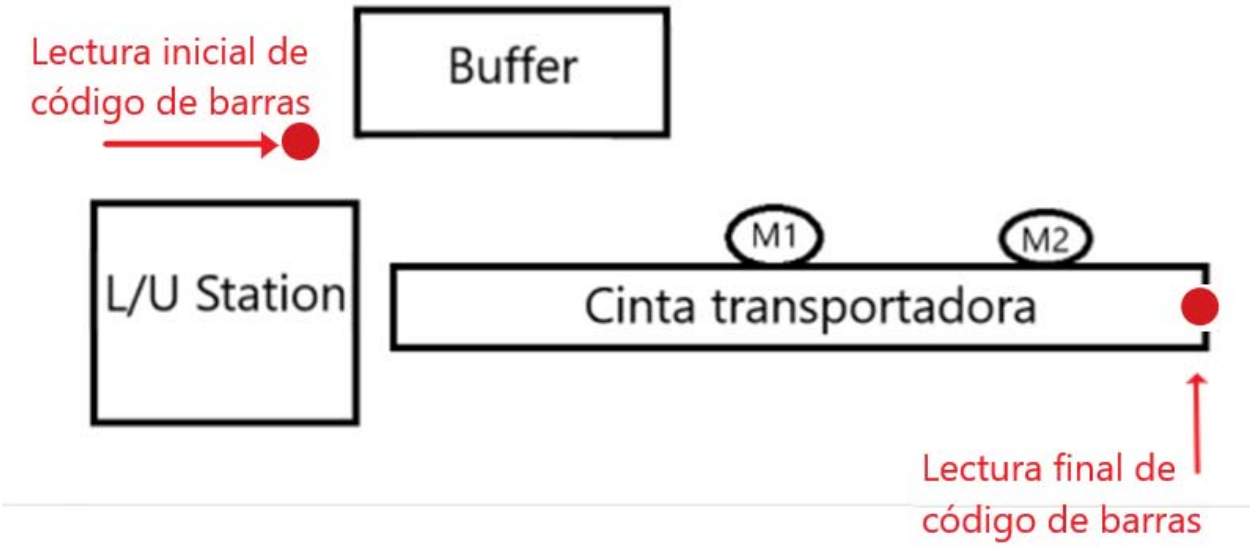

Figura 4.2: Esquema del proyecto (II) [Elaboración propia]

4.2 Construcción del simulador.

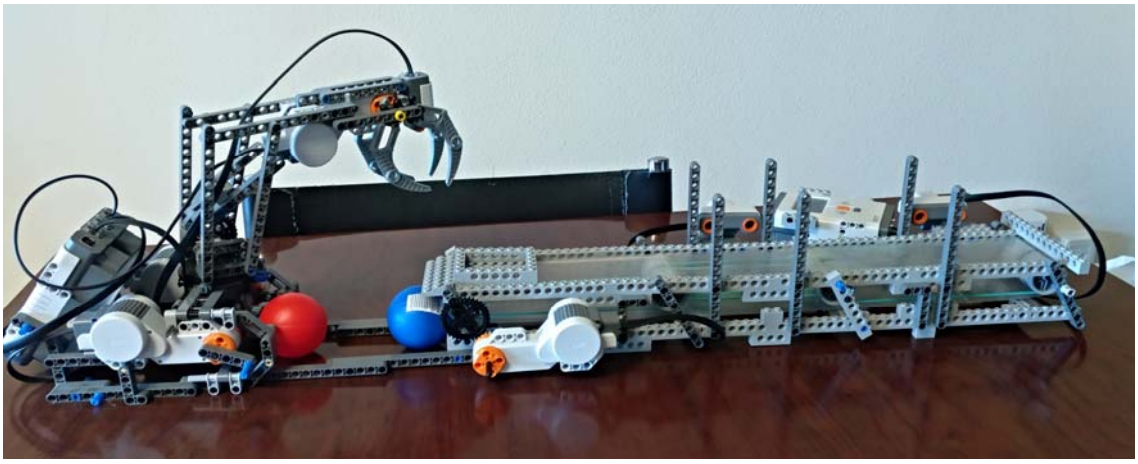

Figura 4.3: Fotografía del prototipo (Alzado) [Elaboración propia]

4.2.1 Brazo robótico (Load/unload station).

El brazo robótico se ha construido siguiendo las instrucciones de un modelo expuesto en la guía para usuarios (NXT User Guide v2.1, 2009). Consta de tres motores: El primero hace rotar el brazo sobre sí mismo, el segundo se encarga de extender y recoger el brazo y el tercero acciona las pinzas que se encuentran en el extremo del brazo.

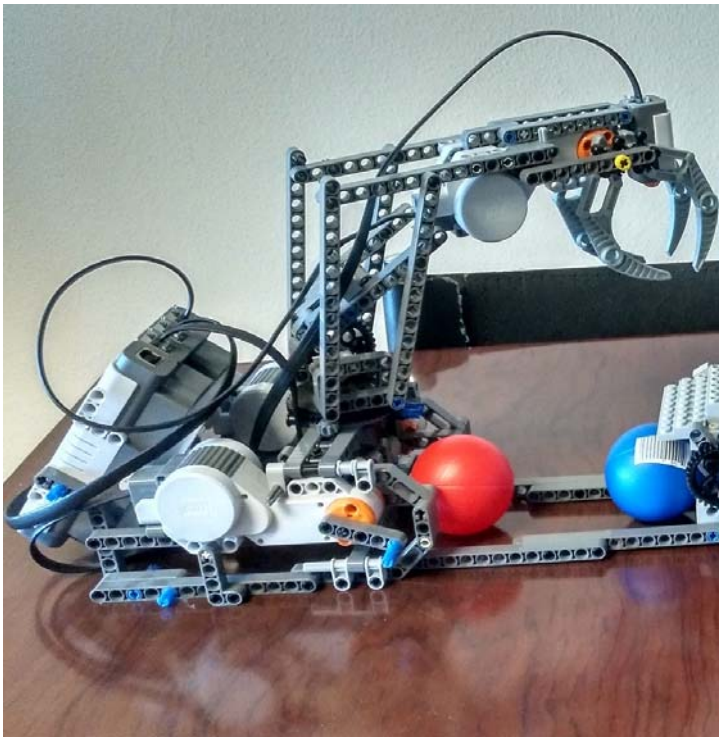

Figura 4.4: Brazo Robótico [Elaboración propia]

4.2.2 Cinta transportadora.

A diferencia del brazo robótico, la cinta transportadora ha sido construida con un diseño propio que vamos a explicar someramente. Para poder identificar fácilmente las piezas utilizadas, se recomienda utilizar la lista de piezas de la guía (NXT User Guide v2.1, 2009)., donde aparecen los números identificativos utilizados en esta descripción

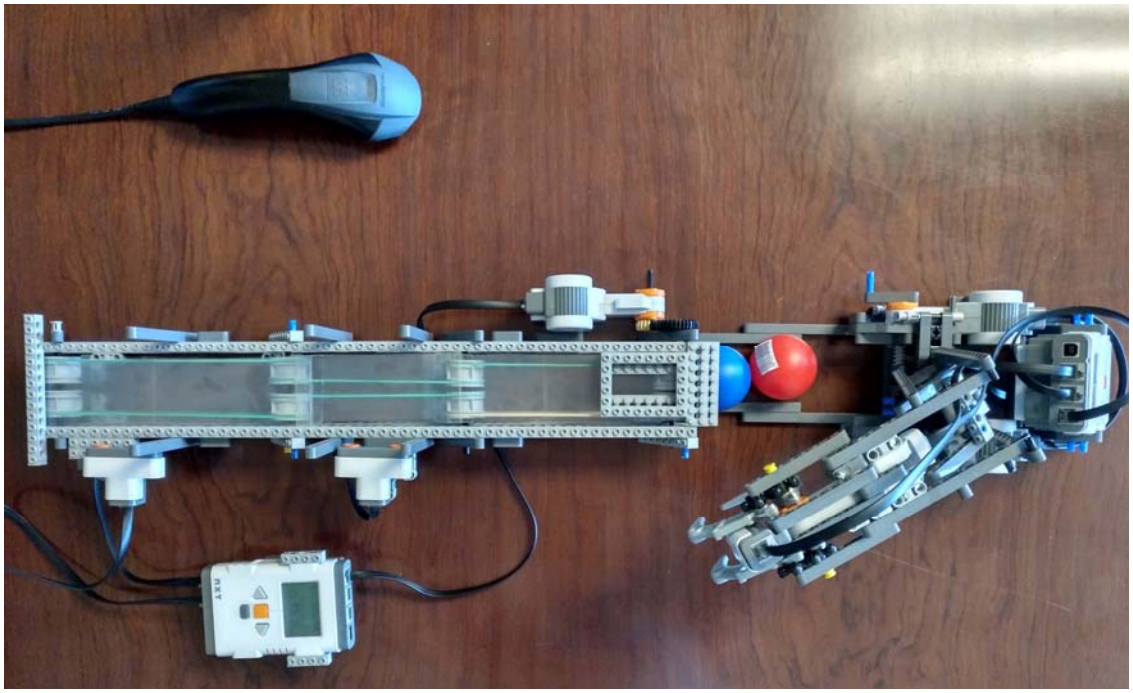

Figura 4.5: Fotografía del prototipo (Planta) [Elaboración propia]

Revisando la bibliografía, la mayor parte de las cintas transportadoras empleadas en simuladores de LEGO utilizan la pieza número 57518, que se puede ver en la *Figura 4.6*.

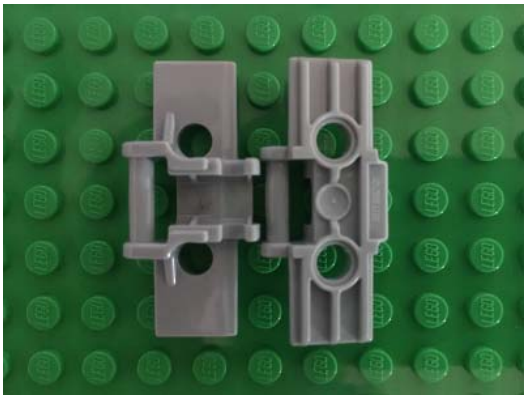

Figura 4.6: Link tread wide with two pin holes (nº 57518) [Fuente: Jezek, 2000]

Nosotros no disponíamos de dicha pieza, así que tuvimos que buscar una alternativa, como se explica a continuación. En primer lugar, la estructura está hecha con las vigas con pernos 1x16 (nº 4211443),

unidas longitudinalmente entre si gracias a las mismas piezas de menor tamaño 4211442 y 4211466, para formar cuatro piezas de gran longitud.

Estas piezas se disponen en dos niveles de altura, dos abajo y dos arriba. Los dos niveles son interconectados por cuatro vigas (módulo de 9, gris oscuro, 4210757) por cada lado. Dichas piezas no son colocadas de forma totalmente vertical, sino ligeramente inclinadas, por el exterior de la estructura, equidistantes y simétricas a ambos lados.

Gracias a esa simetría, vamos a poder introducir unos ejes (370726) que queden apoyados en cada pareja de piezas 4210757 y por cada eje vamos a incluir dos piezas centrales de rueda (4297210). Uno de esos ejes tiene que ir conectado al motor y para comunicar el movimiento al resto de ejes se han utilizado 8 gomillas elásticas. Estas gomillas están situadas alrededor de las ruedas y se han utilizado 2 para conectar cada par de ejes. Las ruedas 4297210 se han situado lo más cerca de los laterales posibles, para reducir la tensión en los ejes debida a la tirantez de las gomillas.

Finalmente, para construir la cinta por donde discurrirán los trabajos, se ha utilizado una banda de un plástico semi-rígido, que hemos colocado envolviendo los 4 ejes con sus respectivas ruedas y que hemos unido por su extremo, con tensión suficiente para que el movimiento de las ruedas haga deslizar también la banda de plástico.

Adicionalmente, se ha tenido que añadir un soporte rectangular para que los trabajos se mantengan en su posición una vez estén en la cinta, ya que son esferas y podrían salir rodando.

4.2.3 Máquinas

El objetivo de este TFG no es desarrollar una automatización compleja y vistosa para el simulador, sino que sea útil para los alumnos de organización y producción. De tal forma que, prácticamente lo único que nos puede llegar a interesar de las máquinas son sus tiempos de servicio y todo lo que pueda estar relacionado con ellos. Teniendo eso en cuenta, se han colocado sensores ultrasónicos, que son parte del set de LEGO Mindstorms, en la posición de las máquinas para poder programar la parada de los trabajos y se ha programado también el tiempo que deben estar paradas simulando un tiempo de servicio real.

Figura 4.7 Sensor Ultrasónico [Fuente: NXT User Guide]

4.2.4 Tecnología de código de barras.

Hemos colocado un código de barras a cada uno de los trabajos para poder identificarlos inequívocamente y ser capaces de monitorizar las entradas y salidas de material de nuestro proceso productivo. Esto tiene muchas posibles aplicaciones desde el punto de vista de la gestión de inventario o del control de la producción, pero sobre todo nos permite recoger una importante cantidad de datos que nos facilitarán una compresión profunda del comportamiento del proceso productivo en estudio. Para servirnos de la tecnología de código de barras en nuestro prototipo necesitamos tanto de un lector, como la capacidad de generar códigos.

- **Lector de código de barras:**

El lector de código de barras utilizado es un Honeywell Voyager (1200g) del que disponía el tutor del TFG. Se trata de un aparato capaz de leer los códigos de barras más frecuentes y cuya utilización es muy sencilla, puesto que solo nos hará falta conectarlo al ordenador a través de un cable USB.

Figura 4.8 Lector de códigos de barras [Fuente: HONEYWELL]

- **Generación del código de barras**: Para obtener los distintos códigos de barras que hemos utilizado, nos hemos servido de un sitio web (Barcode Generator, s.f.). Es uno de los muchos que existen y que nos permiten codificar un número o una cadena de caracteres a casi cualquier tipo de código de barras de los muchos existentes. Nosotros hemos utilizado el UPC-A [1], puesto que era uno de los tipos compatibles con el lector del que disponemos. Este tipo de códigos es habitualmente utilizado en la venta al por menor en el mundo anglosajón, su equivalente europeo sería el EAN-13.

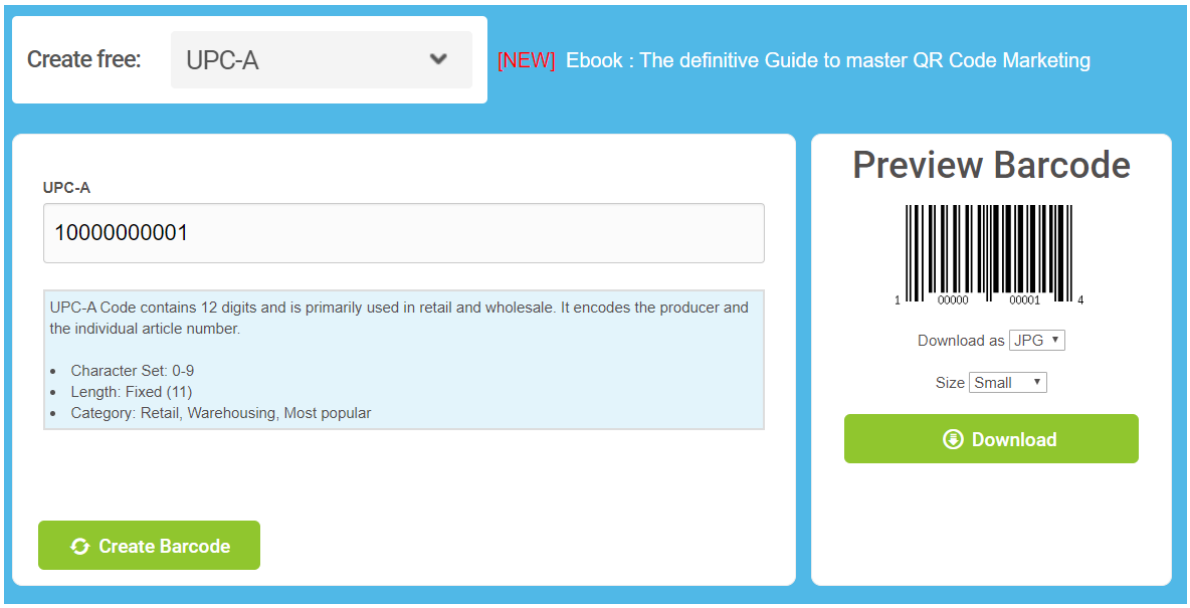

Figura 4.9: Generación web de códigos de barras [Elaboración Propia]

- **Sintaxis:** El UPC-A es capaz de codificar 11 cifras numéricas, aunque finalmente presenta 12, puesto que el proceso de codificación añade una adicional. Habitualmente se codifica el emisor y el receptor del producto, pero en este caso nos hemos servido de él, en primer lugar, para saber qué tipo de producto es de los dos que manejamos. Para ello hemos puesto, en la primera cifra significativa, o bien un 1 para indicar que es un trabajo de tipo 1 (bola azul), o bien un 2 para indicar que es un trabajo de tipo 2 (bola roja). En segundo lugar, nos hemos servido de la última cifra significativa para identificar a los distintos trabajos de un mismo tipo. Así que, los hemos numerado del 1 al 3, ya que solo teníamos tres unidades de cada tipo. En la *Figura 4.10: Códigos de*

[1] UPC: Universal Product Code.

barras utilizados, están presente los seis códigos utilizados, que siguen la sintaxis que acabamos de explicar.

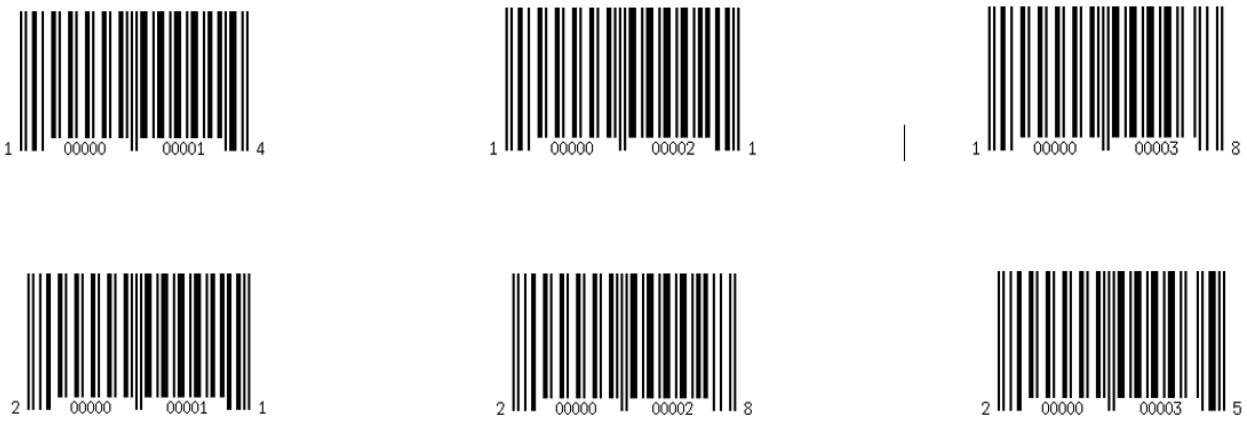

Figura 4.10: Códigos de Barras utilizados [Elaboración propia]

5. VALIDACIÓN DEL PROTOTIPO

Con la construcción y la programación del simulador terminadas, realizamos una serie de 50 simulaciones para validar su comportamiento. Todos los datos de las simulaciones se pueden consultar en el anexo 7.2 Resultados Simulaciones. La mayoría de los datos que se han decidido guardar son tiempos de llegada/salida de cada trabajo. Debemos saber, que cada uno de estos tiempos es medido respecto a su momento inicial, que corresponde a la primera lectura del código de barras de cada trabajo. Los datos recogidos son los siguientes:

- **Lectura inicial del código de barras:** Guarda tanto el tipo de producto como el código en sí. Nos sirve también como punto de partida para evaluar el resto de los tiempos.
- **Tiempo de llegada a la máquina 1, $T_a(1)$:** Cuando el sensor ultrasónico, que representa la máquina 1, detecta la presencia del trabajo, se guardan los segundos transcurridos desde la lectura inicial del código de barras.
- **Tiempo de llegada a la máquina 2, $T_a(2)$:** Mismo procedimiento que para los tiempos de llegada a la máquina 1, pero utilizando el segundo sensor.
- **Lectura final del código de barras:** Vuelve a guardar el código como método de comprobación de que todo ha transcurrido correctamente. En este caso, también se guarda el tiempo en segundos, que consideramos como el instante en que el trabajo abandona la línea de producción (T_d).

Todos estos datos son incluidos directamente desde MATLAB a una hoja de cálculo del programa Microsoft Excel. Para ello utilizamos la función:

```
xlswrite(FILE,ARRAY,SHEET,RANGE)
```

En el anexo 7.*1 Código comentado* se puede encontrar cómo se ha utilizado dicha función. Además, se puede consultar su uso en el propio MATLAB a través del comando: `help xlswrite`.

Con todos los datos introducidos en la hoja de cálculo hemos elaborado gráficos que nos permiten confirmar ciertas características del comportamiento.

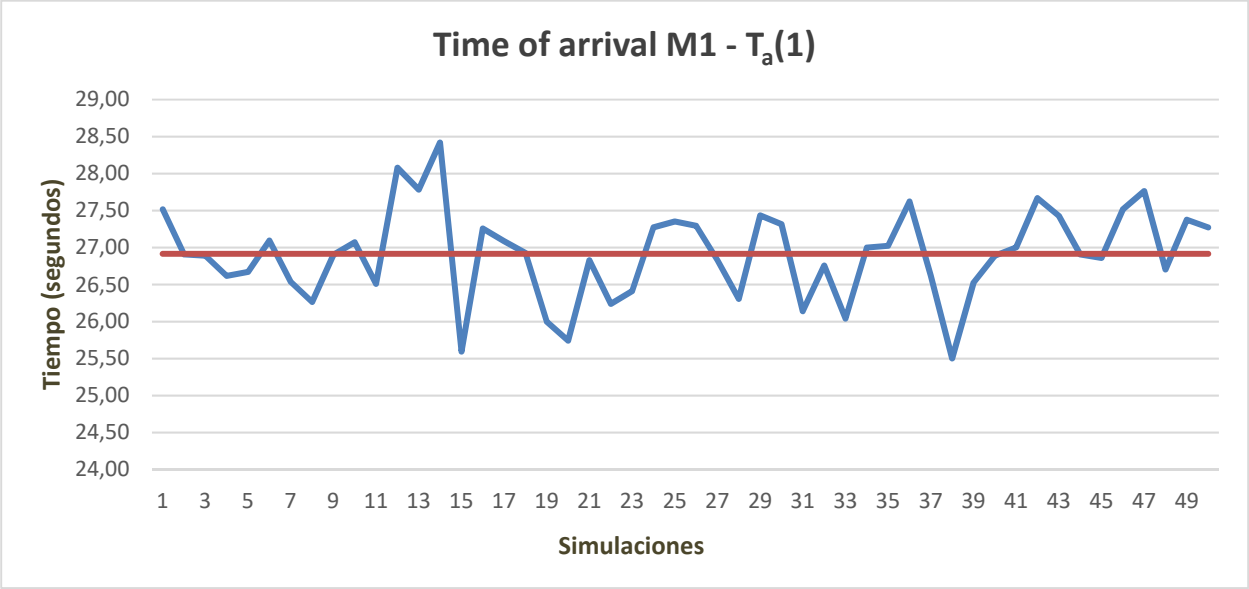

Figura 5.1: Tiempos de llegada. Máquina 1 [Elaboración propia]

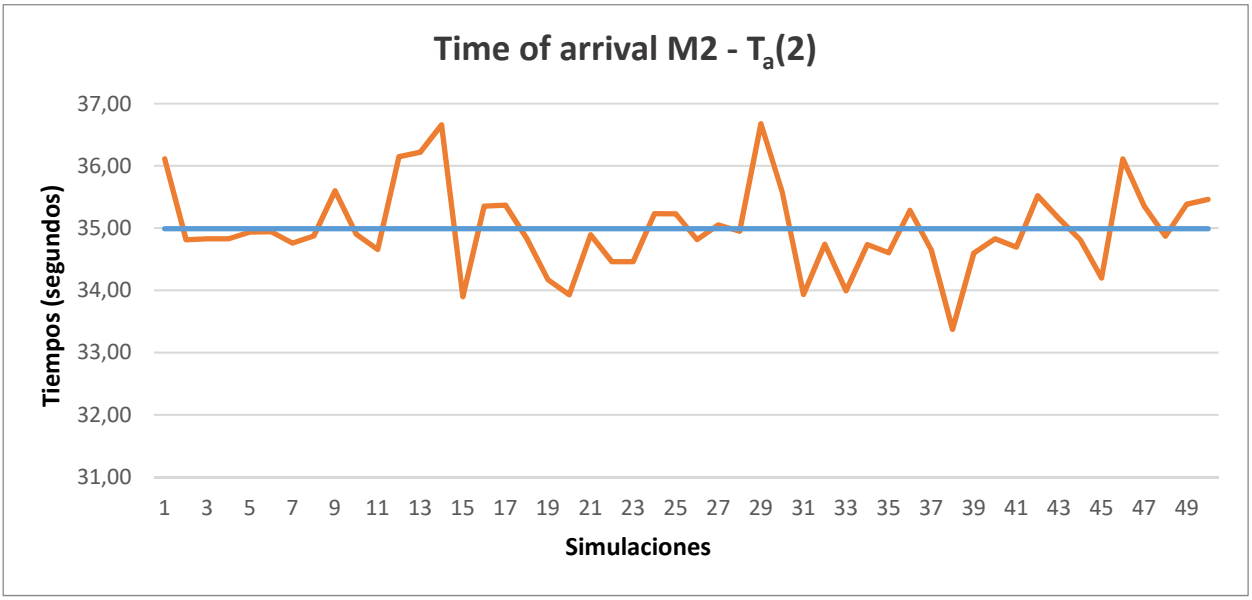

Figura 5.2: Tiempos de llegada. Máquina 2 [Elaboración propia]

Tanto en la *Figura 5.1* como en la *Figura 5.2*, podemos comprobar que los tiempos de llegada a la máquina 1, $T_a(1)$, y la máquina 2, $T_a(2)$, distan de ser constantes en todas las simulaciones, aunque oscilen claramente entorno a una media. Esto puede contradecir lo que se podría pensar en un principio, ya que el tramo del proceso que determina estos tiempos de llegada está completamente automatizado. Habitualmente decimos que cuando un proceso está totalmente automatizado su variabilidad es cero, pero aquí podemos comprobar que, pueden presentar variabilidad, si bien esta es pequeña. *En la Figura*

5.3: Tabla de resultados de simulaciones, vemos claramente que las desviaciones típicas de los tiempos de llegada a las máquinas son muy pequeñas en comparación con sus respectivas medias.

El coeficiente de variación, según la fórmula $c_a = \frac{\sigma_a}{t_a}$ → $C_a(1) = 0{,}0227$; $C_a(2) = 0{,}0197$

	T_d	$T_a[1]$	$T_a[2]$
Media	**40.96**	**26.92**	**34.99**
Varianza	**2.806086**	**0.372386**	**0.468721**
Desv. Típica: σ_a	**1.675138**	**0.610234**	**0.684632**

Figura 5.3: Tabla de resultados de simulaciones [Elaboración Propia]

Es interesante comprobar como estos tiempos de llegada siguen una distribución normal, tal y como se puede empezar a vislumbrar en la *Figura 5.4: Tiempos de llegada. Histograma*.

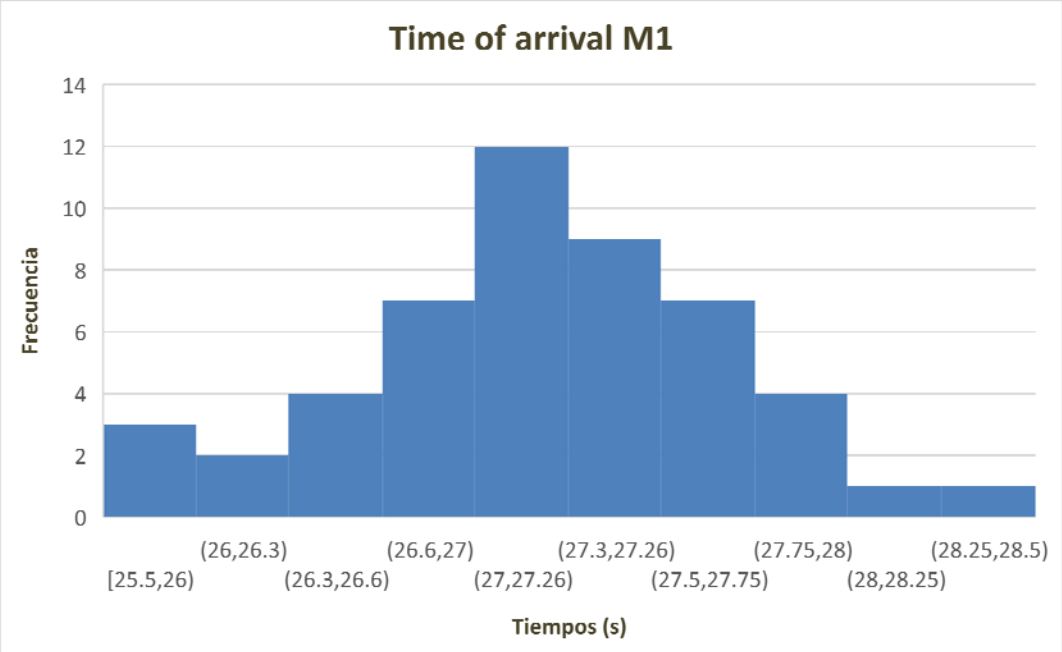

Figura 5.4: Tiempos de llegada. Histograma [Elaboración propia]

La distribución no se adecúa completamente a una distribución normal, pero esto es causa del número de simulaciones, que no ha sido suficientemente alto por culpa del largo tiempo que conlleva hacerlas.

Conforme se iba incrementando el número de simulaciones se apreciaba a simple vista cómo la distribución iba asemejándose cada vez más a una normal.

Conclusión de la validación:

En primer lugar, hemos demostrado que el objetivo fundamental, la trazabilidad de los productos ha sido cumplido. Hemos comprobado que el sistema es capaz de diferenciar a lo largo de todo el proceso unos trabajos de otros, lo que le permitiría procesarlos de distinta forma. Además, se recaban y se almacenan los datos requeridos, desde la identificación del trabajo, hasta los tiempos asociados a su producción.

Otra conclusión interesante, relativa a las diferencias entre tareas manuales y automatizadas, se puede extraer del importante incremento que sufre la desviación típica en la última medida de tiempo tomada, T_d. Como se explicó en el apartado *4.1.1 Modo de funcionamiento*, esta medida se toma con la lectura final del código de barras, algo que tiene que hacer manualmente la persona que esté operando el simulador. Es destacable que, con esa pequeña intervención de un operario humano, la desviación típica presente un incremento respecto de T_a (1) y T_a (2) del 274% y del 245%, respectivamente, quedando el coeficiente de variación → $C_d = 0{,}0401$.

6. CONCLUSIONES Y PROPUESTAS

Como ya se ha mencionado alguna vez a lo largo del presente documento, este proyecto nace con la intención de ser una primera toma de contacto con esta filosofía didáctica y pretende abrir el camino a una profundización posterior por parte de otros alumnos o investigadores. En este capítulo, se hace un balance del trabajo realizado y se plantean las distintas ideas que han ido surgiendo y que se han quedado en el tintero, con el objetivo de que pueda servir de inspiración a futuros interesados en investigar en este tema.

Una de las tareas fundamentales a acometer para incrementar el interés del simulador es aumentar el número de elementos que lo componen respecto al del modesto prototipo presentado aquí. Si se dispone de varias estaciones, así como de sistemas de transporte entre ellas, las posibilidades de innovar se multiplican. En cualquier caso, hay tres cosas fundamentales que deberían ser las primeras tareas a realizar como continuación de este proyecto:

- Suministrador de trabajos: Ser capaces de automatizar un mecanismo que vaya suministrando los trabajos a la línea y no tener que hacerlo manualmente nos permitiría trabajar con tasas medias de llegada de trabajos (λ) exactas.

- Buffers intermedios: Introduciendo el suministrador de trabajos automático ya tendríamos un buffer inicial perfectamente definido entre el propio suministrador y la máquina 1. Añadiendo más máquinas con sistemas de transporte independientes nos permitirá analizar procesos más complejos. Esto lo podríamos conseguir con una pequeña variación respecto al prototipo presentado, que consistiría en dividir la cinta transportadora en tramos que sean independientes, las propias cintas transportadoras pueden actuar como *buffers* finitos.

- Trasladar la lectura del código de barras: Una alternativa a automatizar el suministro de trabajos. Tal y como ha quedado el simulador, la lectura inicial del código de barras se realiza justo antes de ser cargada a la cinta transportadora, pero si dicha lectura se produjese cuando los trabajos ingresan al buffer, seríamos capaces de determinar la tasa de llegada de trabajos. Sin embargo, no seríamos capaces de imponer una, mientras que con el suministrador automático si podríamos.

6.1 Inversión: Lego Mindstorms EV3.

Este proyecto ha sido realizado mediante Lego Mindstorms NXT, la segunda generación de esta línea de juguetes electrónicos de la empresa danesa. Sin embargo, en 2013 ya se empezó a comercializar la tercera generación, llamada Lego Mindstorms EV3. La principal ventaja que proporcionaría la tercera generación a un proyecto como este es, que recientes versiones de MATLAB, concretamente desde la versión R2014b, incluyen un paquete de soporte para EV3. Esto superaría a la RWTH Toolbox y facilitaría enormemente la instalación y la puesta en marcha de todo el proceso, que ha ocupado una parte importante de este trabajo final de grado.

Figura 6.1: LEGO Mindstorms EV3 [Fuente: Internet]

Nos permitiría también utilizar la herramienta Simulink, a la que se le puede sacar mucho partido desde un punto de vista gráfico, por ejemplo. Otro aspecto positivo es que el material de la generación NXT del que dispone el tutor de este TFG no quedaría inservible, puesto que tanto los motores como los sensores se pueden utilizar también con EV3. Lo único indispensable serían los *bricks* programables de la nueva generación que, por desgracia, suponen la mayor parte del elevado coste del producto total.

En conclusión, una posible incorporación de estos métodos a la actividad docente del departamento requeriría una inversión para adquirir algunos productos de la última generación, si se le quiere sacar el máximo partido. Tanto la instalación como la utilización de todo lo relacionado con EV3 están bastante más depuradas y los alumnos ahorraría mucho tiempo ante posibles problemas técnicos, ya que tanto NXT como la RWTH Toolbox han dejado de desarrollarse y la compatibilidad con sistemas operativos o versiones de MATLAB es cada vez más difícil.

6.2 Sistemas control producción y niveles de inventario.

Un sistema para la identificación inequívoca del producto fabricado ofrece muchas posibilidades desde el punto de vista logístico de la empresa y son fácilmente implementables en los entornos simulados mediante Lego. En el caso del prototipo creado en este TFG, se han utilizado códigos de barras, pero existen también otras posibilidades como la tecnología RFID[1], muy presente en la industria actual. La nueva generación EV3 ya incorpora una sencilla compatibilidad con esta tecnología y sería algo muy interesante a tener en cuenta.

En cualquier caso, uno de estos sistemas abre la puerta a la implementación de un sistema de control de la producción, como ya se pronosticaba en el punto *2.1.1 Contenidos Generales*, como es el caso del e-Kanban (Visbal Pérez, 2016). Este sistema está basado en el tradicional Kanban, pero con la utilización de alguno de los sistemas de identificación que hemos mencionado en el párrafo anterior. Dicha combinación permite eliminar los elementos físicos transmisores de la información del Kanban tradicional y centraliza toda la información. Esto facilita la monitorización de inventarios y de los flujos de material, aumentando así el control sobre el proceso productivo.

De esta forma, que el alumno deba escoger o diseñar un sistema de control de la producción, así como decidir los niveles de inventarios ante distintos tipos de demanda puede ayudar a obtener una honda comprensión de los sistemas de control más conocidos, así como abrir la puerta a la investigación de interesantes problemas de optimización.

6.3 Aleatoriedad y fallos.

Tal y como se ha explicado en el punto *4.1.1 Modo de Funcionamiento*, las máquinas del prototipo tienen un tiempo de servicio determinista de 4 segundos y no se ha incluido la posibilidad de que fallen. Obviamente, esto es un caso muy concreto, por lo que sería interesante, y sencillo, incorporar cierta aleatoriedad a los tiempos de servicio para simular un proceso que no esté automatizado por completo. Otro aspecto interesante es introducir el mantenimiento en el prototipo, contemplando la posibilidad de que las máquinas o incluso la cinta transportadora se puedan averiar. Así se podrían introducir muchos conceptos interesantes del mantenimiento que se han visto en la asignatura de 3º de GITI, Simulación de Procesos Productivos, por ejemplo, el MTBF (Mean Time between Failure). Se abriría

[1] RFID: Radio Frecuency Identification.

también la posibilidad de que los alumnos tuvieran que escoger estrategias de mantenimiento óptimas, considerando mantenimientos preventivos y correctivos.

6.4 Industria 4.0

Si bien no se ha implementado ningún sistema propio de la industria 4.0 en el prototipo, durante el desarrollo del proyecto se ha reflexionado al respecto y se ha llegado a la siguiente conclusión: Sería conveniente subdividir el proceso productivo en módulos con el objetivo de poner en práctica conceptos de la Industria 4.0. Cada uno de estos módulos estaría regido por un *brick* programable que remitiría todos los datos a un ordenador central. De esta forma, se establecería una comunicación indirecta entre cada uno de los módulos, tal y como se puede intuir en la *Figura 6.2: Flujo de información. Industria 4.0*. Como ya avanzábamos en el estudio de contenidos, concretamente en el apartado *2.1.2 Industria 4.0: Smart Factory*, esta comunicación entre módulos es fundamental para cumplir las tesis de la Industria 4.0. Cada uno de los módulos podrá ajustar su comportamiento a la información aportada por el resto. Para ello sería fundamental que estuvieran programados por separado, al contrario de como se ha hecho para el prototipo que se ha realizado para este TFG, puesto que el volumen de código será bastante mayor y resultaría poco práctico incluirlo todo en el mismo algoritmo.

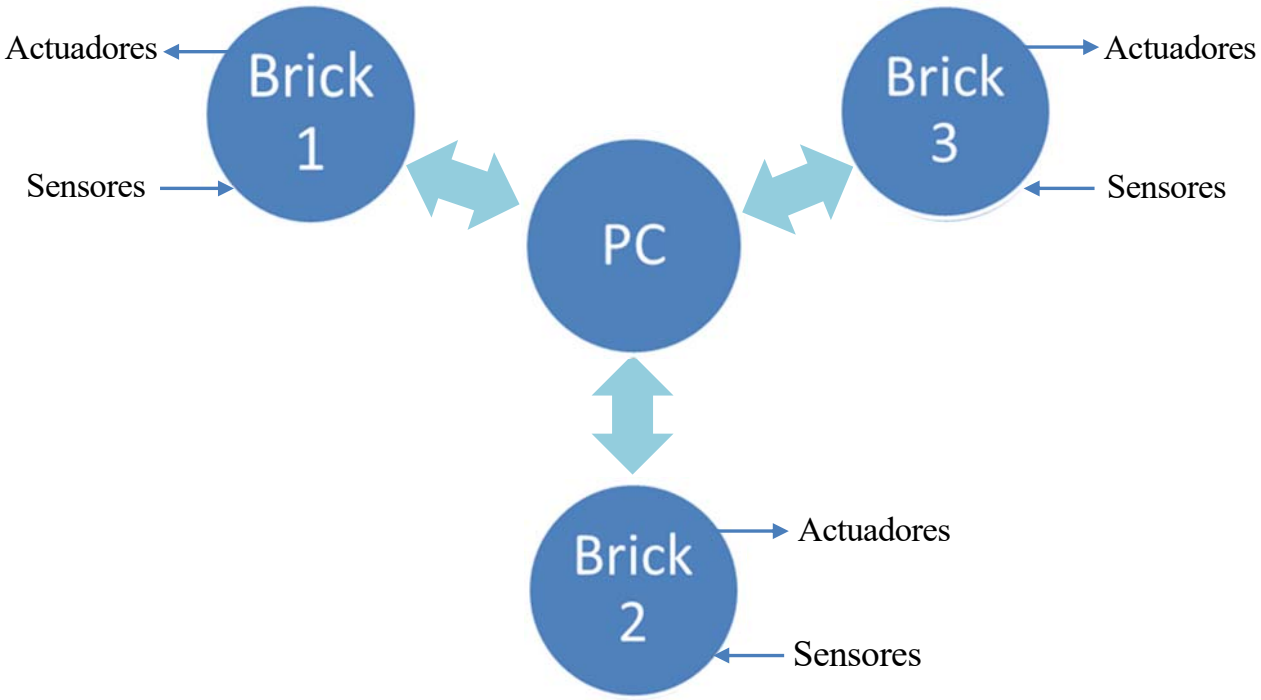

Figura 6.2: Flujo de información. Aplicación Industria 4.0 [Elaboración propia]

7. ANEXOS

7.1 Código comentado.

CONECTAR.m

```
tic
COM_CloseNXT('all');
h=COM_OpenNXT('bluetooth.ini');
COM_SetDefaultNXT(h);
NXT_PlayTone(440, 500, h);
Toc
```

CONECTAR2.m

```
tic
h2=COM_OpenNXT('bluetooth2.ini');
NXT_PlayTone(440, 500, h2);
Toc
```

DESCONECTAR.m

```
COM_CloseNXT('all');
```

FINAL.m

```
%indicamos el número de repeticiones que vamos a realizar (tamaño del
lote)
i=2;
k=2;
num_repeticiones=input('Indique el tamaño del lote: ');
indice_repeticiones=0;
lote=zeros(num_repeticiones,3);
t_arrival_2=zeros(num_repeticiones,1);
t_arrival_1=zeros(num_repeticiones,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%
while indice_repeticiones<num_repeticiones
%MOTOR A: AGARRAR/SOLTAR
%MOTOR B: EXTENDER/RETRAER EL BRAZO
%MOTOR C: ROTACIÓN DEL BRAZO SOBRE SI MISMO
    mB=NXTMotor(MOTOR_B);
    mC=NXTMotor(MOTOR_C);
    mA=NXTMotor(MOTOR_A);
%SENSOR DE PRESIÓN
    OpenSwitch(SENSOR_1,h2);
    Switchvalue=0;
%LEEMOS CÓDIGO DE BARRAS
   j=1;
    barcode=input('Lea el codigo de barras: ');
    tic
    %CASO PARA PRODUCTO 1
    if barcode>200000000000
        lote(i,j)=1;
```

```
        lote(i,j+1)=barcode;
        lote(i,j+2)=toc;
    end
    %CASO PARA PRODUCTO 2
    if barcode<200000000000
        lote(i,j)=2;
        lote(i,j+1)=barcode;
        lote(i,j+2)=toc;
    end
    xlswrite('codigobarra.xlsx',lote);
    tic
%BAJAMOS EL BRAZO
    mB.Power=-50;
    mB.TachoLimit=360*4;
    mB.SendToNXT(h2);
    mB.WaitFor(45,h2);
%CERRAMOS PINZAS
    mA.Power=30;                 %positivo para cerrar.
%mA.TachoLimit=40;
    mA.SendToNXT(h2);
    while 1
    Switchvalue=GetSwitch(SENSOR_1,h2);
     if Switchvalue==1
        break;
     end
    end
mA.Power=0;
mA.SendToNXT(h2);
%mA.WaitFor(2,h2);
%FUNCIÓN DE DESCARGA.            (descarga_trabajo)
%GIRAMOS EL BRAZO
    mC.Power=-75;
    mC.TachoLimit=360*17;
    mC.SendToNXT(h2);
    mC.WaitFor(45,h2);
%BAJAMOS BRAZO
    mB.Power=-50;
    mB.TachoLimit=360*2;
    mB.SendToNXT(h2);
    mB.WaitFor(45,h2);
%ABRIMOS PINZAS
    mA.Power=-30;                %positivo para cerrar.
    mA.TachoLimit=30;
    mA.SendToNXT(h2);
    mA.WaitFor(45,h2);
%LEVANTAMOS EL BRAZO
    mB.Power=50;
    mB.TachoLimit=360*6;
    mB.SendToNXT(h2);
    mB.WaitFor(45,h2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%CONTROL DE LA CINTA TRANSPORTE PRINCIPAL
%ABRIMOS LOS DOS SENSORES DE ULTRASONIDO
    OpenUltrasonic(SENSOR_1);
    OpenUltrasonic(SENSOR_2);
    distance1=GetUltrasonic(SENSOR_1);
    distance2=GetUltrasonic(SENSOR_2);
```

```
%DEFINIMOS TIEMPOS SERVICIO DE CADA MÁQUINA.
    Tiempo_servicio1=4;
    Tiempo_servicio2=4;
        while distance1>=10
         DirectMotorCommand(MOTOR_A, 40, 0, 'off', 'off', 'off', 'off');
         distance1=GetUltrasonic(SENSOR_1);
        end

%MÁQUINA 2
    pause(0.4); %para que se pare justo en el sitio indicado.
    StopMotor(MOTOR_A,'off');

    %t_arrival_2(i,9)=toc;       %medimos el tiempo de llegada
    lote(i,8)=toc;

    pause(Tiempo_servicio1);
        while distance2>=10
         DirectMotorCommand(MOTOR_A, 40, 0, 'off', 'off', 'off', 'off');
         distance2=GetUltrasonic(SENSOR_2);
        end
%GIRAMOS EL BRAZO (DEVOLVEMOS EL BRAZO A SU POSICIÓN INICIAL PARA
AGILIZAR)
    mC.Power=75;
    mC.TachoLimit=360*17;
    mC.SendToNXT(h2);
%MÁQUINA 1
    pause(0.5);
    StopMotor(MOTOR_A,'off');
    lote(i,9)=toc; %guardamos el tiempo de llegada.
    pause(Tiempo_servicio2);
    DirectMotorCommand(MOTOR_A, 40, 0, 'off', 'off', 'off', 'off'); %para
llegar hasta el final
    pause(0.5);
    StopMotor(MOTOR_A,'off');
%LEEMOS CÓDIGO DE BARRAS
   j=5;
    barcode=input('Lea el codigo de barras: ');

    %CASO PARA PRODUCTO 1
    if barcode>200000000000
        lote(i,j)=1;
        lote(i,j+1)=barcode;
        lote(i,j+2)=toc;
    end
    %CASO PARA PRODUCTO 2
    if barcode<200000000000
        lote(i,j)=2;
        lote(i,j+1)=barcode;
        lote(i,j+2)=toc;
    end
    i=i+1;             %incrementamos i para escribir en la siguiente
fila
                           del excel
%ESCRIBIMOS EN LA HOJA DE CÁLCULO
```

```
    xlswrite('codigobarra.xlsx',lote,1,'A1');
%VUELTA POSICIÓN INICIAL
    DirectMotorCommand(MOTOR_A, -40, 0, 'off', 'off', 'off', 'off');
%para llegar hasta el final
    pause(8.5);
    StopMotor(MOTOR_A,'off');
%ACTUALIZAMOS INDICE REPETICIONES
    indice_repeticiones=indice_repeticiones+1;

end %aquí acaba el bucle while

%PARA ACABAR, CERRAMOS LOS SENSORES
    CloseSensor(SENSOR_1,h2);
    CloseSensor(SENSOR_1);
    CloseSensor(SENSOR_2);
```

7.2 Resultados simulaciones.

- Hemos hecho un total de 50 simulaciones en las que se hayan recogido datos.
- La columna de la izquierda responde a los datos recogidos en la lectura inicial del código de barras. El apartado del tiempo (T_i) es siempre cero puesto que se ha considerado esta medición como la referencia para medir los tiempos. Aún así se registran como medida de control para evitar errores.
- La columna de la derecha refleja datos obtenidos a través de la lectura final del código de barras, así como los tiempos de llegada a la máquina 1, $T_a(1)$, y a la máquina 2, $T_a(2)$.

Lectura inicial

clase	Código barras	T_i
2	100000000038	0
2	100000000014	0
2	100000000038	0
1	200000000035	0
1	200000000028	0
2	100000000014	0
2	100000000038	0
2	100000000021	0
2	100000000014	0
1	200000000028	0
2	100000000038	0
1	200000000035	0
2	100000000021	0
2	100000000014	0
2	100000000038	0
1	200000000028	0
1	200000000011	0
1	200000000035	0
1	200000000028	0
2	100000000038	0
2	100000000014	0
2	100000000021	0
1	200000000011	0
2	100000000014	0
1	200000000011	0
2	100000000038	0
1	200000000028	0

Lectura final

clase	Código barras	T_d	$T_a(1)$	$T_a(2)$
2	100000000038	43.13	27.52	36.11
2	100000000014	40.50	26.91	34.81
2	100000000038	40.17	26.89	34.83
1	200000000035	43.48	26.62	34.83
1	200000000028	40.92	26.67	34.94
2	100000000014	40.46	27.10	34.94
2	100000000038	40.73	26.54	34.76
2	100000000021	40.33	26.27	34.87
2	100000000014	41.43	26.90	35.60
1	200000000028	40.54	27.07	34.90
2	100000000038	39.99	26.51	34.65
1	200000000035	41.31	28.08	36.15
2	100000000021	42.30	27.78	36.22
2	100000000014	41.91	28.42	36.66
2	100000000038	39.62	25.59	33.90
1	200000000028	40.68	27.26	35.35
1	200000000011	40.82	27.09	35.37
1	200000000035	40.22	26.93	34.83
1	200000000028	39.61	26.00	34.17
2	100000000038	39.26	25.74	33.93
2	100000000014	40.16	26.83	34.89
2	100000000021	39.61	26.24	34.46
1	200000000011	39.98	26.41	34.46
2	100000000014	41.77	27.27	35.23
1	200000000011	40.56	27.35	35.23
2	100000000038	40.32	27.29	34.82
1	200000000028	40.36	26.83	35.05

2	100000000021	0		2	100000000021	40.38	26.31	34.95
1	200000000035	0		1	200000000035	42.38	27.43	36.68
1	200000000035	0		1	200000000035	41.84	27.32	35.58
2	100000000021	0		2	100000000021	39.50	26.14	33.93
1	200000000028	0		1	200000000028	45.04	26.76	34.74
2	100000000038	0		2	100000000038	39.05	26.04	33.99
1	200000000035	0		1	200000000035	40.36	27.00	34.74
2	100000000014	0		2	100000000014	40.19	27.02	34.60
2	100000000021	0		2	100000000021	40.91	27.62	35.29
2	100000000014	0		2	100000000014	39.95	26.62	34.65
1	200000000028	0		1	200000000028	39.56	25.50	33.37
1	200000000011	0		1	200000000011	39.85	26.53	34.60
2	100000000038	0		2	100000000038	40.17	26.89	34.83
1	200000000035	0		1	200000000035	40.56	27.00	34.69
1	200000000011	0		1	200000000011	41.14	27.67	35.52
1	200000000028	0		1	200000000028	49.32	27.43	35.15
2	100000000014	0		2	100000000014	40.50	26.91	34.81
2	100000000038	0		2	100000000038	39.83	26.86	34.20
2	100000000038	0		2	100000000038	43.13	27.52	36.11
2	100000000014	0		2	100000000014	41.22	27.76	35.35
1	200000000011	0		1	200000000011	40.41	26.70	34.87
1	200000000035	0		2	100000000035	41.66	27.38	35.38
2	100000000021	0		2	100000000021	40.91	27.27	35.46

Figura 7.1: Tabla de simulaciones [Elaboración propia]

BIBLIOGRAFÍA

[1] Behrens A., Atorf L., Schwann R., Neumann B., Schnitzler R., Ballé J., Herold T., Telle A., Noll T.G., Hameyer K., & Aach T. (2010). "*MATLAB Meets LEGO Mindstorms - A Freshman Introduction Course Into Practical Engineering*", RWTH Aachen University.

[2] BRICXCC (BRICX Command Center), (s.f.). SOURCEFORGE. Programmable Brick Utilities. *http://bricxcc.sourceforge.net/utilities.html* (Última visita: septiembre 2018)

[3] Barcode Generator, (s.f.). Página web gratuita para la generación de códigos de barras. *http://www.barcode-generator.org* (Última visita: septiembre 2018)

[4] Framiñán, J. M. (2017). "*Apuntes de la asignatura Diseño de Productos y Procesos*". Departamento de Organización y Gestión de Empresas I, Universidad de Sevilla.

[5] Harel, I. & Papert, S. (1991). "*Constructionism"*. Ablex Publishing. Westport, CT, US.

[6] HONEYWELL. Official Website. Morris Plains, Nueva Jersey, Estados Unidos © 2018 Honeywell International Inc. *www.aidc.honeywell.com* (Última Visita: Septiembre 2018)

[7] Jang, YJ. (2016). "*Statement of Teaching*". Industrial and Systems Engineering, KAIST, Yuseong, Daejeon, South Korea.

[8] Jang, YJ. & Yosephine, VS. (2016). "*LEGO Robotics Based Project for Industrial Engineering Education*," International Journal of Engineering Education, v.32, no.3, pp.1268 - 1278

[9] Jezek, D. (2000) BrickLink: Catálogo de sets, piezas y productos de LEGO. © 2018 BrickLink Limited *https://www.bricklink.com/v2/main.page*

[10] Kopcazka, M. *et al*. (2012) *RWTH Toolbox v4.07*. Lehrstuhl für Bildverarbeitung, RWTH Aachen University. *http://www.mindstorms.rwth-aachen.de* (Última visita: septiembre 2018)

[11] Mathworks, (2012). MATLAB support for LEGO Mindstorms: (Última visita: agosto de 2018) https://es.mathworks.com/hardware-support/lego-mindstorms-matlab.html

[12] Mathworks, (2015). MATLAB & SIMULINK version r2015b: (Última visita: julio de 2018) https://es.mathworks.com/products/new_products/release2015a.html MATLAB is a registered trademark of The MathWorks, Inc.

[13] National Instruments (Official Website), Contenido para LEGO Mindstorms (LABVIEW) *https://www.ni.com/academic/mindstorms/* (Última visita: Julio de 2018)

[14] NXT User Guide v2.1 (2009). Mindstorms Education NXT Base Set 9797. LEGO® and MINDSTORMS® are Registered Trademarks.

[15] Papert, S. (1980) "*Mindstorms: children, computers and powerful ideas* " Basic Books, Inc. New York, NY, USA.

[16] Rohde, J; Meyr H. & Wagner M. (2000). "*Die Supply Chain Planning Matrix*" Darmstadt Technical University, Department of BWL, Institute for Business Studies.

[17] Sriram T., Vishwanatha Rao K., Biswas S. & Ahmed B. (1996). "*Applications of barcode technology in automated storage and retrieval systems"* Corp. Res. & Dev. Div., Bharat Heavy Electr. Ltd., Hyderabad, India.

[18] Stich, V. (2017) Apuntes de la asignatura Logística Industrial "*Vorlesung 11. Industrielle Logistik (Sommersemester 2017)*". Forschungsinstitut für Rationalisierung (FIR), RWTH Aachen University.

[19] Visbal Pérez, ET. (2016). *"Herramientas tecnológicas aplicables al Kanban para la optimización de los procesos en la empresa"* Visión gerencial ISSN 1317-8822 Año 15 Nº 1 enero - junio 2016 Pg: 82 – 104.