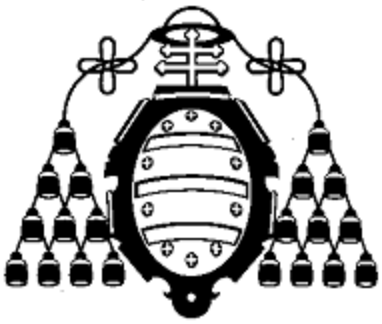

UNIVERSIDAD DE OVIEDO

CENTRO INTERNACIONAL DE POSTGRADO

MASTER EN INGENIERÍA MECATRÓNICA

TRABAJO FIN DE MÁSTER

DISEÑO, DESARROLLO Y SIMULACIÓN DE UN SISTEMA MECATRÓNICO DE BAJO COSTE PARA PRÁCTICAS

Julio 2014

Adrián Álvarez Cuervo **Ignacio Álvarez García**

[Firma] **[Firma]**

AGRADECIMIENTOS

RESUMEN

El objetivo del presente proyecto es desarrollar un software de simulación de un sistema mecatrónico de bajo coste, formado por diversos componentes (mecánicos y electrónicos), cuyo diseño permite modificar fácilmente la configuración del mismo.

El prototipo a simular dispone de dos grados de libertad, una rotación y una traslación, y diferentes sensores para adquirir datos de posición y velocidad. El simulador ha de tener la versatilidad suficiente como para poder implementar cada uno de los subsistemas que componen el prototipo, de forma tal que se puedan configurar para ver el funcionamiento bajo diferentes parámetros de entrada.

Este proyecto se divide en las siguientes partes:

1- Simulador mecánico: pretende simular de forma básica y genérica un sistema mecánico compuesto por: motor, reductor, guía, tuerca y husillo. Las variables más representativas que afectan a la dinámica del sistema han de ser parametrizables.

2- Simulador electrónico: pretende simular la electrónica necesaria para la adquisición de magnitudes físicas mediante la implementación de diferentes sensores y circuitos de adaptación de señal. También dispondrá de parámetros configurables como, resolución de los sensores, ruidos etc.

3- Librería de usuario: librería que permite al usuario interactuar tanto con el simulador como con el sistema real, adquiriendo los datos de los diferentes sensores para cerrar el lazo de control.

Para el desarrollo del simulador se utilizarán librerías y bibliotecas concretas de código abierto y multiplataforma que evite al usuario tener instalar programas de pago para la utilización del sistema.

PALABRAS CLAVE

Simulador - Multiplataforma - Software libre - Prácticas

ÍNDICE GENERAL

1. INTRODUCIÓN

El presente proyecto trata sobre el diseño, construcción y simulación de un sistema mecatrónico que sirva como complemento de apoyo, tanto a profesores como a alumnos, en las clases prácticas y/o teóricas de las diferentes ramas de una ingeniería. Será por lo tanto una herramienta que pretende hacer las clases más didácticas favoreciendo la comprensión de los diferentes temas que se puedan impartir.

Este sistema pretende ser un banco de ensayo cuyo abanico de posibilidades y de casos de uso sea tan amplio como sea posible, por lo que se su diseño estará enfocado a tratar de añadir alternativas de uso en los distintos campos de una ingeniería. Principalmente estará enfocado en las ramas de mecánica, electrónica y control:

- Mecánica: estudio de diferentes soluciones de accionamiento y transmisión, y su influencia en el comportamiento del sistema.

- Electrónica: interfaces de E/S para sensores y accionadores, y tarjetas electrónicas de control.

- Control: modelización del sistema, realización de algoritmos de control y estudio del comportamiento en lazo abierto y cerrado.

Para ofrecer esta gama de alternativas se necesitará diseñar el equipo de forma que sus subsistemas (mecánica, sensores, accionadores) sean configurables, lo que permitirá estudiar su comportamiento en diversas circunstancias y ante diferentes requerimientos de control. Además ha de disponer de sistemas de sensorización con los que extraer datos para su uso tanto en tiempo real como para un estudio posterior. En la medida de lo posible se realizará de tal forma que sus componentes sean intercambiables para dotar al sistema de más versatilidad.

Tratándose de un tema didáctico, y con el fin de sacarle el máximo provecho, se desarrollará un software con el que simular las características de las diferentes versiones del sistema, sin tener que disponer de él físicamente, lo que mejora y aumenta sus condiciones de uso.

Por tanto, el proyecto queda claramente divido en dos líneas distintas de trabajo pero directamente relacionadas entre sí desde el comienzo, siendo estas el banco de ensayos físico y el simulador de éste.

1.1. Descripción básica del prototipo

En estos primeros apartados describir el funcionamiento y características del sistema real implica también definir cómo será el simulador puesto que ha de contener los mismos elementos básicos que el prototipo. Más adelante se concretarán que mecanismos se utilizarán para la realización de los diferentes movimientos y de los sensores que los medirán, así como sus rangos de uso y otros parámetros de interés. En este punto en concreto se detallará a groso modo las características más básicas tanto del sistema físico como del sistema simulado.

Independientemente de su forma y tamaño, el sistema contará con dos tipos de movimiento: uno lineal y otro rotacional. De esta manera, puede estudiarse, desde diferentes puntos de vista (mecánicamente, control, etc.), la influencia que tiene el movimiento rotacional en el movimiento lineal, y viceversa (el movimiento de la masa no tendrá las mismas condiciones dependiendo de la inclinación, al igual que el par de rotación, según la posición a lo largo del desplazamiento lineal). La figura 1.1 muestra los dos tipos de movimiento con los que necesariamente contará el sistema.

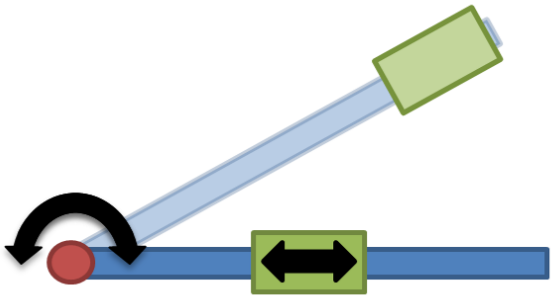

Figura 1.1. Representación de movimientos del sistema

Las restricciones iniciales no imponen más que seguir este esquema, y los métodos para lograrlo no están establecidos, pero serán aquellos que maximicen las posibilidades del sistema cumpliendo los requisitos.

El sistema de la figura anterior tiene dos grados de libertad, para enfocarlo a asignaturas de regulación y control siempre será útil la implementación de sensores que permitan medir el movimiento lineal y rotacional. Con esto, poco a poco se enfocan los elementos básico que ha de tener el banco de ensayos y por lo tanto el simulador. Además se van perfilando el tipo de de asignaturas en las que podría utilizarse el sistema, aun sin haber definido detalladamente los elementos que lo componen ni el tipo de estos. El cuadro 1.1 muestra algunos de los posibles casos de uso.

ASIGNATURA	POSIBLES PRÁCTICAS O TEMAS
Física	Estudio en el comportamiento de una masa en función de la pendiente
Mecánica	Estudio de los pares de torsión y fuerzas resultantes en el eje de giro
Electrónica	Adquisición e interpretación de datos ofrecidos por los sensores
Regulación	Realización de control de velocidad y/o posición de una masa fija

Tabla 1.1. Casos de uso

1.2. Esquema general del proyecto

Pata entender el grueso del proyecto y encaminar el resto de decisiones que se tomarán a partir de ahora con respecto a cómo se llevara a cabo su desarrollo, es necesario explicar más detalladamente los bloques de los que está compuesto y el nexo entre ellos. La figura 1.2 muestra las dos líneas de trabajo a seguir.

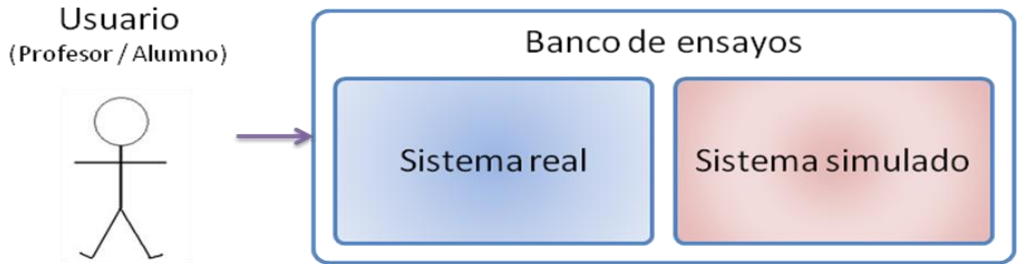

Figura 1.2. Líneas de desarrollo

Al usuario, sea profesor o alumno, se le proporcionara el banco de ensayos para trabajar. El banco de ensayos, ya sea con su sistema real o con el simulado, ha de ser capaz de ofrecer los mismos resultados con el fin de permitir al usuario llegar a las mismas conclusiones independientemente del camino seguido.

Para la permitir la interacción entre usuario y maquina (entendiendo maquina a ambos sistemas real y simulado) es necesario un interfaz que ofrezca datos del sistema. La figura 1.3 muestra de forma muy genérica el funcionamiento global del sistema.

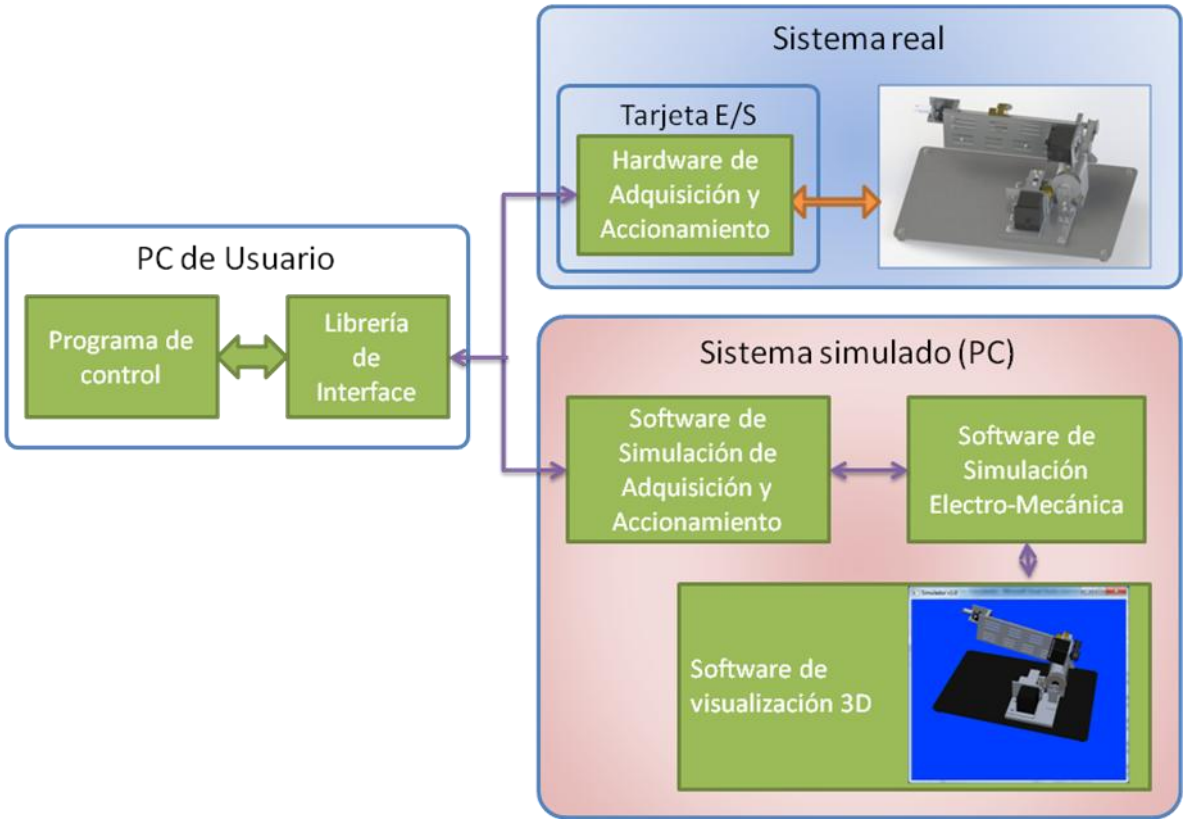

Figura 1.3. Esquema general del proyecto

Hasta ahora solo se sabía del sistema que es un dispositivo que se utilizará como elemento de apoyo en las clases teórico-practicas y que consta de dos grados de libertad (una rotación y una traslación) de los cuales se adquirirán datos mediante sensores. Ahora ya se está en disposición de detallar el sistema algo más. Cabe aclarar que el esquema anterior muestra la situación en la cual entran mas factores en cuenta, es decir, una situación donde el usuario desea realizar un programa de control ya sea para el sistema real o para el simulado. Evidentemente no todos los usuarios querrán hacer tal control y simplemente querrán observar el funcionamiento de los mecanismos bajo unas condiciones.

De la figura 1.3 se deduce por tanto que el sistema real no tendrá solo un par de sensores sino que ha de disponer de una tarjeta de entrada y salida de datos que controlen los diferentes mecanismos. En cuanto al simulador, éste ha de disponer de al menos tres bloques. Al igual que en el dispositivo real, es necesario incluir la simulación de una tarjeta de entrada y salida de datos que en este caso servirá para controlar el bloque encargado de la simulación electromecánica. Además se representaran los movimientos del mecanismo en un modelo 3D. Para gobernar el banco de ensayos será necesario implementar una librería con funciones básicas con las que se podrá extraer o enviar datos al sistema.

El presente proyecto trata sobre la realización de la parte de simulación, de todas formas se explicarán los rasgos más importantes del sistema real puesto que depende de forma directa de él. Además, entender el funcionamiento del prototipo ayudará también a la comprensión del simulador y de parte de su desarrollo.

2. REQUISITOS

En este capítulo se establecen los requisitos para la realización del banco de ensayos. Resulta evidente que para realizar la simulación de algo es necesario saber cómo funciona y que características tiene ese "algo". Por esta razón se explicaran los requisitos establecidos para la realización del sistema real ya que en algunos aspectos serán muy parecidos a los del sistema simulado.

2.1. Requisitos del sistema mecánico

Los requisitos, en su mayoría, suelen venir impuestos de manera externa por lo que hay que clarificarlos y ordenarlos. En esta ocasión, los requisitos impuestos inicialmente son los siguientes:

- Sistema adaptable con posibilidad de variar las configuraciones. No se establece de manera clara una convergencia hacia una solución, si no que se estudiará la viabilidad de ofrecer el mayor abanico posible.

- Utilizable para prácticas didácticas en asignaturas ingenieriles.

- Bajo coste: es interesante que el producto fuera lo más económico posible, para realizar, si el resultado es satisfactorio, más de una unidad en un futuro.

- Robusto: el producto va a estar expuesto a acciones y manipulaciones que en muchos casos pueden ser erróneas, por lo que tiene que estar preparado para soportarlas y evitar riesgos, tanto para la máquina como para la persona.

- Seguridad: las máquinas deben cumplir normativas de seguridad, pero se hará hincapié buscando trabajar activamente en la misma.

- Desplazamiento lineal y desplazamiento rotacional: como se ha explicado anteriormente, seguir este esquema sin ningún tipo de restricción más al respecto.

Además de esto, se establecen una serie de requisitos propios que se consideran fundamentales para lograr un buen resultado:

- Adaptación del diseño a la producción en el taller del departamento de fabricación de la Universidad de Oviedo: con esto se obtendrá un menor coste y se facilitarán las pequeñas modificaciones posteriores.

- Sencillez: ofrecer un suficiente grado de complejidad en las posibilidades, pero sin dejar de lado la sencillez en operaciones de ajuste, cambios de configuración, etc. Se considera fundamental el diseño de un sistema lo suficientemente abierto como para desarrollar nuevas soluciones en el futuro. En caso contrario, su uso quedará mucho más restringido y será menos atractivo desde el punto de vista didáctico.

- Tamaño manejable: a menor tamaño, el producto será más cómodo y atractivo para su empleo.

- Masa adaptable: es interesante poder variar la masa a desplazar, con los beneficios didácticos que eso supone.

- Accionamientos eléctricos: aunque existen otro tipo de accionamientos que podrían emplearse, como por ejemplo, hidráulicos o neumáticos, la complejidad del sistema crecería enormemente, como también el precio, y el aumento de casos de uso no quedaría compensado con estas. Es por tanto, que los accionamientos se limitan a aquellos de tipo eléctrico.

2.2. Solución adoptada

La solución adoptada tiene que ser válida tanto para la simulación a desarrollar en el presente proyecto, como para la implementación física a desarrollar en el proyecto complementario. Puesto que la 2ª tiene más restricciones que la 1ª (precio, fabricabilidad, montabilidad, etc.), la mayoría de decisiones de implementación se han tomado en el proyecto complementario y asumido por consenso en el presente. A continuación se muestran las decisiones de implementación más relevantes.

2.2.1. *SOLUCIÓN MECÁNICA*

El sistema mecánico seleccionado consta de dos grados de libertas como ya se ha comentado con anterioridad. El movimiento de rotación se realiza mediante un motor que transmite su potencia a trabes de una reductora diseñada de tal forma que se puede variar la relación entre la entrada y la salida. Las relaciones de reducción que dispone el prototipo son 1:13, 1:110 y 1000. La figura 2.1 muestra el diseño del bloque reductor en sus distintas configuraciones.

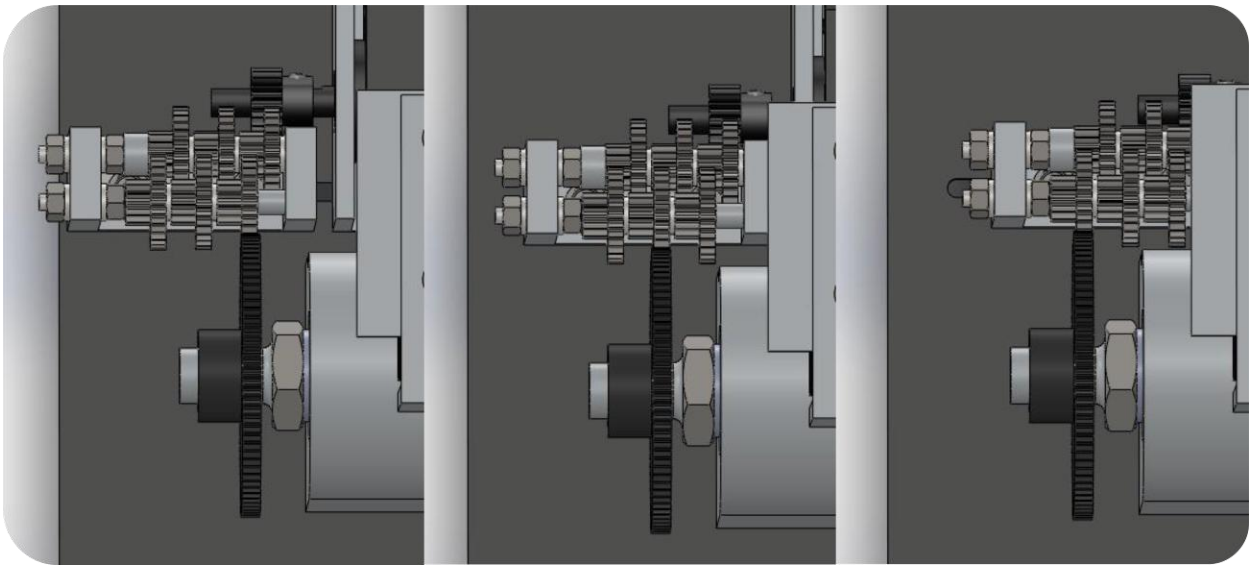

Figura 2.1. Configuraciones del bloque reductor

El movimiento lineal se realiza convirtiendo el movimiento de rotación de un motor a movimiento lineal mediante un husillo, para esto se ha implementado además un sistema de guiado con una guía prismática y un carril. El motor en este caso transmite su potencia mediante un mecanismo de poleas y correa dentadas. En la imagen de la figura 2.2 se muestran los diferentes componentes que forman el bloque para la realización del movimiento lineal.

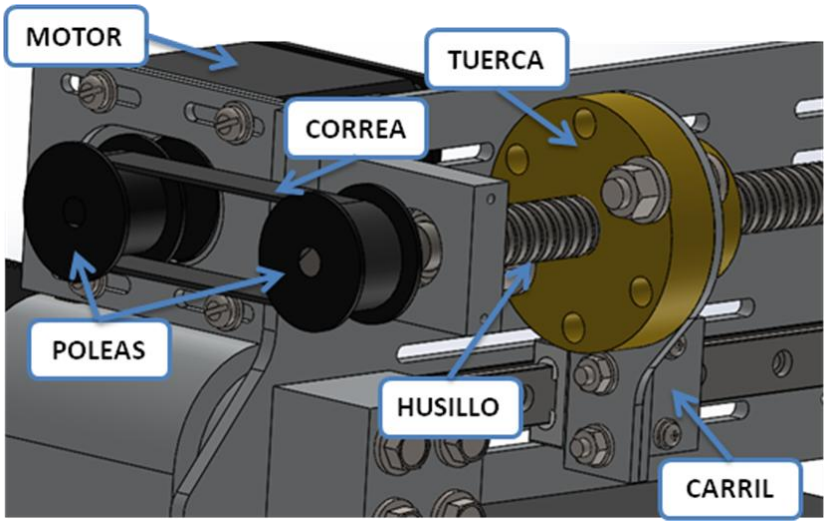

Figura 2.2. Sistema de desplazamiento lineal

Cumpliendo con los requisitos propuestos, el sistema ha sido diseñado por módulos intercambiables por lo que tanto los motores, como el husillo o la guía pueden ser sustituidos. Este hecho ha de ser tenido en cuenta a la hora de realizar el simulador del sistema. En la figura 2.3 se muestra la solución mecánica final, así como el tipo y localización de algunos sensores.

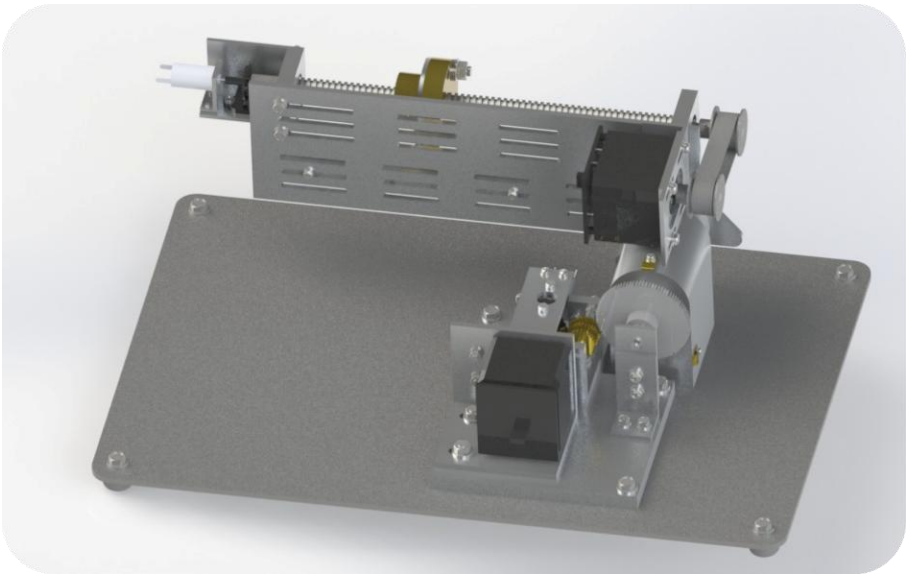

Figura 2.3. Solución mecánica adoptada

En la tabla 2.1 se muestra un resumen de las características a tener en cuenta de cara a la realización del simulador.

FUENTE DE POTENCIA	MOVIMIENTO	TRANSMISIÓN	RELACIÓN DE TRASMISIÓN	DESTINO DE POTENCIA
Motor eléctrico	Rotación	Engranajes	1:1	Sistema de movimiento lineal
Motor eléctrico	Lineal	Correa dentada	1:1 - 1:110 - 1:1000	Bloque husillo tuerca

Tabla 2.1. Tabla resumen del sistema mecánico

2.2.2. *SOLUCIÓN ELECTRÓNICA*

Este apartado se puede decir que se compone de dos bloques, el correspondiente con la selección del tipo de sensores y de su localización y el correspondiente con las placas de entrada y salida de datos, conexionados y distribución de estas.

Los sensores seleccionados para la medida de los parámetros de velocidad y posición son:

- Tacómetro: usado para convertir la velocidad angular en un valor analógico de tensión equivalente.

- Potenciómetro: usado para convertir la posición angular de uno de los ejes a un valor analógico.

- Codificador rotatorio o encoder: usado para convertir la posición angular de los ejes a un código digital.

En el movimiento de rotación se han utilizado los tres tipos de sensores, donde el potenciómetro por cuestiones mecánicas ha sido ubicado en el eje de salida de la reductora. En cambio tanto el encoder como el tacómetro están acoplados directamente en el eje del motor.

El movimiento lineal tiene acoplados al husillo un tacómetro y un encoder, en este caso las restricciones de vida útil del potenciómetro no permiten su implementación en el sistema. Nótese que la medida de desplazamiento lineal de la tuerca no es posible realizarla con ninguno de los sensores de los que se disponen, este aspecto resulta interesante para la realización de prácticas de control aunque también ha de tenerse en cuenta a la hora de realizar el código de simulación que más adelante se detallará. La figura 2.4 representa la ubicación física de cada sensor en el prototipo real

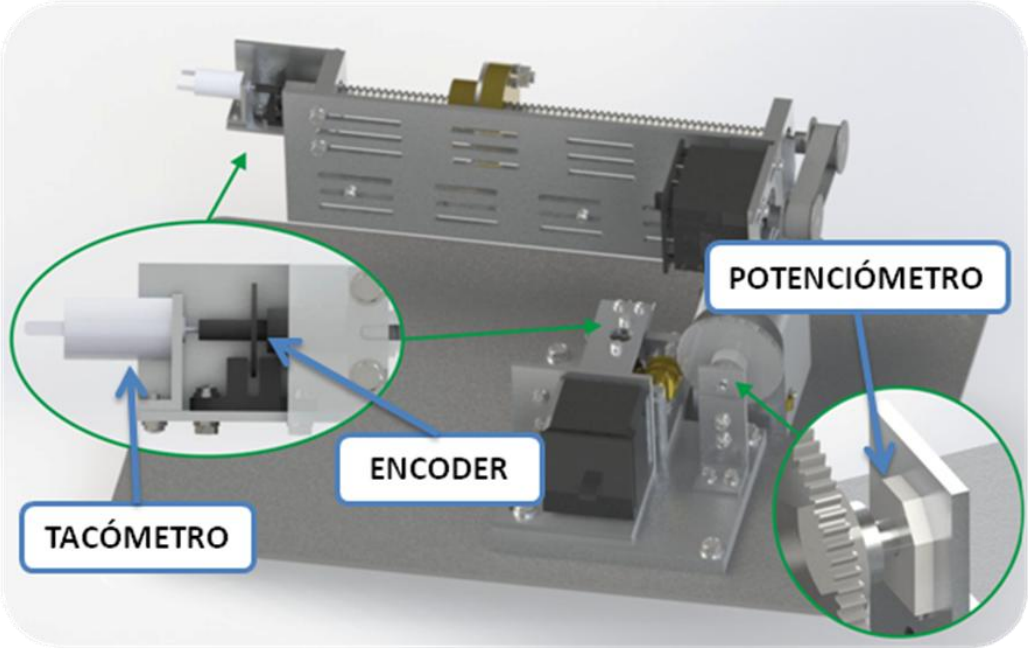

Figura 2.4. Sensores y posicionamiento

El simulador ha de tener por lo tanto los sensores mencionados, la tabla 2.2 representa un resumen de alguno de los aspectos a tener en cuenta. Aunque en ella no se especifican rangos de trabajo ni resolución de los distintos sensores son también datos a tener presentes. Adicionalmente cuenta con varios finales de carrera que podrán ser utilizados para saber que el sistema ha llegado a su tope mecánico

BLOQUE	SENSOR	LOCALIZACIÓN	VARIABLE A MEDIR
Movimiento de rotación	Tacómetro	Eje del motor	Velocidad angular del eje del motor
	Encoder	Eje del motor	Posición angular del eje del motor
	Potenciómetro	Eje de salida del reductor	Posición angular del eje del motor (1)
Movimiento lineal	Tacómetro	Husillo / Eje polea conducida	Velocidad angular del eje husillo
	Encoder	Husillo / Eje polea conducida	Velocidad angular del eje husillo

(1) Aplicar relación de transmisión seleccionada

Tabla 2.2. Tabla resumen de la disposición de los sensores

Los sensores mencionados han de estar disponibles tanto para el usuario como para el propio sistema. El sistema necesita las señales para mostrar la información en un interfaz y al usuario se le ofrecen estas señales para que pueda realizar placas de control con las que interactuar con el sistema. El diagrama de bloques de la figura 2.5 muestra, como su propio nombre indica, los bloques que componen el sistema electrónico implementado en el prototipo. De cara a la realización del simulador será totalmente indiferente como estén distribuidos o localizados los diferentes componentes, por el contrario si es de interés ver que bloques contiene y su relación entre ellos.

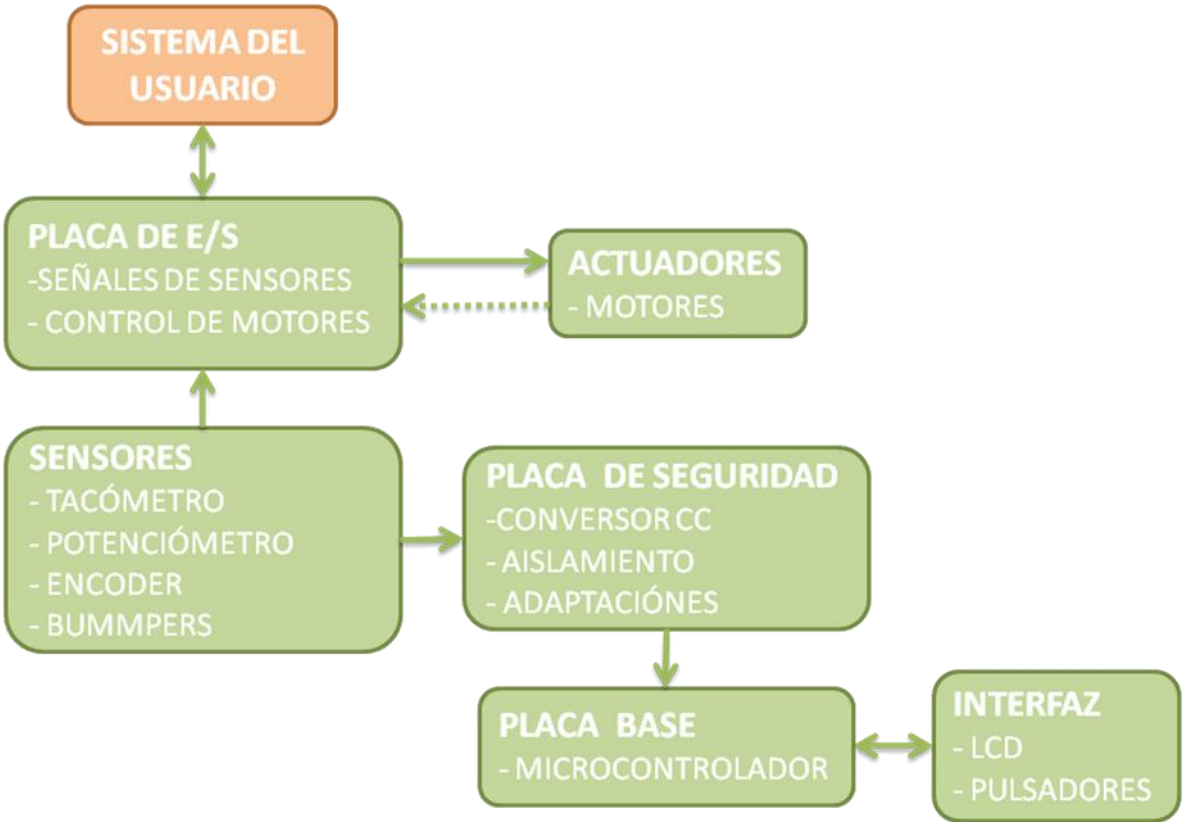

Figura 2.5. Diagrama de bloques del sistema electrónico

2.3. Requisitos del simulador

Tras la descripción básica de los elementos mecánicos y electrónicos que conforma el prototipo ya se está en disposición de especificar una parte de los requisitos. En el apartado anterior se vieron varios componentes que se encuentran en el sistema y que por lo tanto habrán de ser simulados necesariamente. Los componentes que han de simularse son:

- Tacómetro
- Potenciómetro
- Encoder
- Bummper
- Husillo trapezoidal
- Guía prismática
- Reductora
- Motor

Dado como se ha comentado varias veces el sistema ha de ser lo más abierto posible y está sujeto a modificaciones se ha de cumplir:

- Componentes genéricos: esto quiere decir que ha de ser posible la modificación de parámetros como resolución tensión de salida etc. en el caso de los sensores. En los elementos mecánicos estos parámetros pasarían a ser rozamientos, pares u otras variables concretas de cada elemento en sí.

- Sensores posicionables: aunque en el sistema ofrecido los sensores tienen posiciones fijas conocidas nada evita que el usuario coloque algún tipo de sensor en otro eje diferente al que está puesto. Esta circunstancia ha de estar prevista para poder simular tal situación.

Hasta ahora los requisitos vienen impuestos por los componentes que constituyen el sistema real, evidentemente cuantas más funcionalidades se le incluyan al simulador mas didáctico será. Por esta razón en este proyecto se han propuesto requisitos para mejorar el sistema real tanto como sea posible, para ello se ha impuesto cumplir las siguientes especificaciones:

- Simulación básica de una placa de control: en ocasiones el usuario querrá realizar su propio sistema de control, para ello necesitará utilizar algunas de las señales de los sensores disponibles. Estas señales tienen que ser adaptadas al rango de entrada del dispositivo de control utilizado y podrán tener perturbaciones de ruido u offset. Se realizara por tanto la simulación básica de un sistema de adaptación de señales.

- Librería de interface orientada a microcontrolador pic: la figura 1.3 muestra el esquema general del proyecto donde se incluye esta librería. Dado que en la Universidad de Oviedo es bastante común el uso de microcontroladores pic se implementarán las instrucciones de tal forma que se asemejen al lenguaje usado por éstos.

- Implementación de pantalla LCD alfanumérica: aunque no es indispensable si resulta interesante la simulación de una pantalla LCD para permitir la visualización de los parámetros que desee el usuario. Las funciones para el control de la pantalla se incluirán en la librería mencionada en el punto anterior.

- Datos gráficos de variables del motor: los motores necesitaran más o menos par para mover la carga en función de la masa de esta o de su posición, variaciones en algunos de estos parámetros implicaran variaciones en su consumo de corriente. Implementar graficas en tiempo real de corriente y par ayudará a la comprensión de fundamentos físicos básicos así como del funcionamiento de un motor.

- Simulación mecánica independiente: realizar un sistema de simulación independiente de la parte mecánica implica no necesitar un software de control para mover es mecanismo. Se pretende por tanto poder modificar no solo la masa sino el ángulo del husillo y la tensión del motor que lo gobierna. Gracias a esto podrá observarse visualmente la dinámica de la masa sin necesidad de implementar un control previo.

Además de todos los requisitos anteriores se desea cumplir con la siguiente especificación, no siendo ésta estrictamente obligatoria:

- Simulador multiplataforma: con fin de facilitar al usuario su instalación, resulta de interés realizar una aplicación multiplataforma ejecutables desde cualquier tipo de sistema operático, esto implica no en algunos casos no necesitar instalar una máquina virtual únicamente para ejecutar el simulador.

Los requisitos y deseos expuestos en este bloque tratan de cómo se verá el simulador de cara al usuario que lo utilice, más adelante se completará la lista de requisitos teniendo en cuenta otros aspectos.

3. PLANTEAMIENTO

En este capítulo se define qué se pretende realizar y de qué forma. Se recuerda por tanto, respondiendo a la primera pregunta, que es lo que se desea realizar, ahora ya de forma más detallada puesto que ya se han impuesto unos requisitos previos que indicar el camino a seguir. El esquema de la figura 3.1 representa el desglose fundamental de los apartados en los que se divide el proyecto. Tener este diagrama claro es fundamental para empezar a tomar decisiones sobre cómo enfocar la forma de trabajo o de abordar cada bloque.

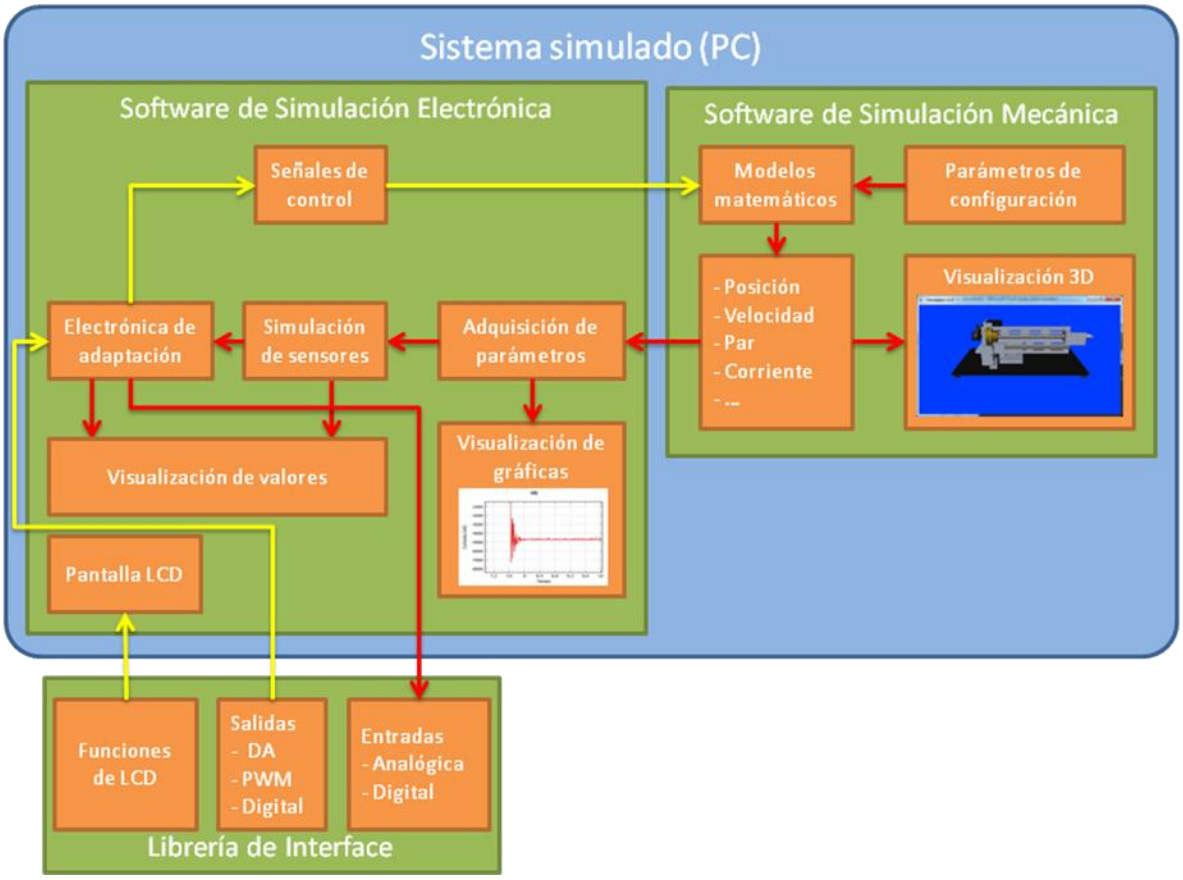

Figura 3.1. Desglose del simulador

La figura anterior divide el problema en tres apartados diferentes que ya se podían intuir en los requisitos siendo estos:

- *Software de simulación mecánica*: Sera el encargado de simular los mecanismos del sistema real, para ello se necesitarán los modelos matemáticos de los componentes mecánicos que formen el sistema. Dado que pueden ser modificados es necesario poder modificar los parámetros que afecten al funcionamiento del sistema, de esta forma se podrán simular varios configuraciones. Gracias a los modelos matemáticos se podrán obtener, entre otras cosas, los parámetros de velocidad y posición que permitirán, mediante visualización en un modelo 3D, representar el funcionamiento del sistema. Para cumplir con las especificaciones del apartado anterior se han de calcular las corrientes y pares de los motores que se usen, estos datos han de enviarse al simulador electrónico para su procesamiento.

- *Software de simulación electrónica*: Una de las funciones de este bloque será procesar los datos de los diferentes motores para su visualización por pantalla en modo de graficas a tiempo real. Ambos simuladores han de estar comunicados entre sí de alguna forma para permitir enviar y recibir la trama e datos correspondientes. Los valores de posición y velocidad serán procesados por los sensores correspondientes transformándolos en variables eléctricas cuyo valor habrá que representar numéricamente. Posteriormente será necesario aplicar una adaptación de señal para cada una de ellas, de esta forma quedaran adecuadas al rango de trabajo que se desee.

- *Librería de interfaz*: Esta librería a de disponer de funciones básicas de entrada y salida de datos que permitan cerrar el lazo de control para la regulación del sistema en caso de ser necesaria. Además como complemento adicional se le han de incluir funciones de escritura para una pantalla LCD alfanumérica

Cada uno de los elementos necesitará uno o varios de los siguientes apartados:

- Visualización 3D:
    - Programa de modelización 3D para generación de los modelos de los componentes mecánicos.
    - API (librería software) de renderizado 3D para la visualización del sistema simulado.

- Interfaz de usuario:
    - API para interfaz de usuario.

- Programación de la simulación en sí misma:
    - Conjunto de elementos físicos que interaccionan → programación orientada a objetos (C++) para encapsular los comportamientos de cada elemento, y facilitar la reusabilidad en caso de adiciones/modificaciones.

    - Comportamiento dinámico de cada elemento en función de las ecuaciones diferenciales que lo definen.

3.1. Selección de la implementación

Anteriormente ya se ha resumido qué se va a realizar con lo cual queda responder a la pregunta de cómo se va a implementar. Para responder a esta pregunta hay que hacer un análisis previo de los diferentes caminos que se pueden seguir y hacer un baremo entre ventajas e inconvenientes de cada uno de ellos. Principalmente existen dos caminos, estos se basan en la utilización de software propietario (licencias de pago) o software libre (entendiendo libre en este caso como: que se puede usar, escribir, modificar y redistribuir gratuitamente). A continuación se detalla brevemente los problemas y facilidades que supone seguir una u otra dinámica de trabajo.

3.1.1. *SOFTWARE PROPIETARIO*

Se tomará como referencia Matlab, Simulink y Labview para realización de interfaz grafico de usuario y programación e implementación de modelos matemáticos, y Solidworks para representación del modelo 3D. Aun existiendo muchos más softwares que podrían permitir el desarrollo del simulador, se hace mencionan esos cuatro programas en concreto por ser los más comunes y conocidos en el mundo de la ingeniería.

El uso de software propietario presenta como principal desventaja la necesidad de una licencia de pago. En muchos casos se puede paliar este problema solicitando a la empresa correspondiente una licencia para estudiantes. Otra de las desventajas es el espacio que ocupan en el disco duro y el gran uso de memoria ram que necesitan estos programas, por lo que ordenadores de poca potencia no podrían ejecutar el simulador en ellos. Siguiendo de lado de los inconvenientes, muchos de estos programas solo tienen distribución para sistemas operativos Windows, con lo cual, impediría que el simulador pudiese ser multiplataforma.

Las ventajas más destacadas por los softwares mencionados anteriormente es la gran cantidad de funcionalidades disponibles. Labview está orientado, entre a otras muchas cosas, al tratamiento y representación de datos, e incluso tiene sus propios dispositivos con los que adquirirlos. Resulta evidente que, a priori, es una buena opción para la realización del presente proyecto. Otra de las ventajas de Labview, es la posibilidad de comunicarse con Solidworks, con lo que se podría aprovechar esta circunstancia para simular los movimientos correspondientes sobre el modelo 3D de forma muy sencilla y directa.

Además, tanto Labview como Matlab permiten, de forma muy simple, la creación de interfaces gráficos de usuario. Matlab también permite, gracias al módulo de Simulink, poder realizar modelos matemáticos de forma fácil, algo más que interesante para la realización del simulador que se pretende crear.

Analizando las ventajas y los inconvenientes se puede observar que con este tipo de programas y combinándolos entre si de alguna u otra forma se podría crear un simulador de manera relativamente sencilla. Por otra parte en caso de haber complicaciones suele existir una gran documentación al respecto, ya sea en manuales de la empresa o en foros.

3.1.2. *SOFTWARE LIBRE*

Si se sigue este camino el principal problema que se encuentra es que no existen programas del estilo a Labview, Matlab etc. por lo que el desarrollo del simulador será, a priori, bastante más arduo ya que se tiene que realizar desde cero programando cada una de sus funcionalidades. Además, el software libre por lo general, no dispone de tanta documentación e información como en el caso del software de pago. Estas dos circunstancias implican que el desarrollador necesite más tiempo para la realización del simulador.

Por otra parte, la principal ventaja del uso de software libre es (además de ser gratuito) que en la gran mayoría de los casos tienen distribución para diferentes sistemas operativos, lo que permitiría al usuario realizar un simulador multiplataforma. De cara al usuario este aspecto es el más importante ya que no se tendrá que preocupar de adquirir licencias ni de instalar una maquina virtual por no tener un sistema operativo compatible con el simulador. Además, el hecho de que el desarrollador tenga que realizar el simulador desde cero implica que, el software desarrollado será específico para la simulación del prototipo en cuestión. Como consecuencia no se necesitará la instalación de múltiples y pesados softwares adicionales en el ordenador.

Si antes se comentaba la posibilidad del uso de Solidworks para representar el modelo 3D y sus movimientos, ahora es necesaria la utilización de algún tipo de motor de juegos que permita la representación del mismo, en combinación con un código específico encargado de realizar los cálculos de los modelos matemáticos. Para realizar el interfaz gráfico se podría utilizar el propio motor de juegos aunque también existen otras opciones que se detallaran en apartados posteriores.

Al igual que en el apartado anterior se puede ver que es posible realizar un simulador aunque, en este caso, con alguna complicación mas.

3.1.3. *CAMINO A SEGUIR*

Después del planteamiento inicial ya se está en disposición de tomar una decisión sobre el camino a seguir para el desarrollo del simulador.

Uno de los requisitos (aunque no indispensable) es la posibilidad de desarrollar un simulador valido para cualquier tipo de sistema operativo, es por esto que, no se utilizará ningún tipo de software que no sea multiplataforma. Esta circunstancia determina por lo tanto, el camino a seguir a partir de ahora.

En apartados posteriores de detallará mas exhaustivamente los softwares, librerías, motores de videojuegos y programas de modelado que se utilizarán para el desarrollo del simulador del banco de ensayos, así como sus características, funcionalidad y problemáticas que presentan.

3.1.4. *CONCLUSIONES*

Independientemente del camino escogido existe la posibilidad de realizar el simulador del banco de ensayos. Dado que en el planteamiento inicial existen tres partes diferenciadas e independientes, los futuros desarrolladores pueden mejorar escoger qué camino seguir para realizar y/o mejorar cada uno de los diferentes bloques del proyecto.

3.2. Selección de herramientas

Tras haber concretado el modo de trabajo queda decidir que herramientas se utilizaran para el desarrollo del simulador, entendiendo por herramientas todo aquel software adicional que se vaya usar como por ejemplo motores de juegos, programas de modelado, librerías etc.

3.2.1. *MOTOR DE JUEGOS*

En primer lugar se necesita seleccionar el motor de juegos con el que se va a interactuar para la representación del modelo 3D. Antes de seleccionarlo es necesario explicar brevemente que es un motor de juegos.

Un motor de juegos es un conjunto de sistemas de software extensibles y que facilita el desarrollo de un videojuego sin una gran modificación de código. La arquitectura del motor de juegos permite la portabilidad en varias plataformas, incluyendo sistemas operativos de computadoras de escritorios, consolas de videojuegos y dispositivos móviles. La ventaja de un motor de juegos es que con un comando se realizan rutinas que normalmente llevarían varios comandos en una librería gráfica como OpenGL o DirectX.

La funcionalidad básica de un motor es proveer al videojuego de un motor de renderizado para los gráficos 2D y 3D, motor físico o detector de colisiones, sonidos, scripting, animación, inteligencia artificial, redes administración de memoria y un escenario gráfico.

Cabe aclarar que, un motor de juegos por sí solo no resulta útil sino que se necesitaran modelos 3D (ya sean animados o no) por lo que será necesario, además, poseer de algún tipo de software de modelado 3D

En la aplicación que aquí se aborda no es necesario tener grandes conocimientos sobre el amplio mundo de la programación de juegos, pero si resulta interesante tener una pequeña base con el fin de familiarizarse con que es, cómo funcionan y cómo se utiliza un motor de juegos. Esta información puede ser especialmente interesante para futuros desarrolladores y/o usuarios que deseen mejorar el aspecto del simulador mecánico realizado en el presente proyecto. Para ello pueden consultarse los documentos de carpeta *01 - Motores de juegos* [1, 2] adjunta en los anexos.

Para la selección del motor de juegos a utilizar se han tenido en cuenta algunos requisitos mencionados con anterioridad y además se han especificado algunos nuevos para el desarrollo del sistema simulado:

- Licencia gratuita
- Sistema multiplataforma
- Orientado a modelos 3D (nuevo)
- Lenguaje de programación c o C++ (nuevo)
- Documentación suficientemente extensa para principiantes (nuevo)
- Códigos de ejemplo (nuevo)

Existen múltiples motores de juegos a disposición de los usuarios con diferencias y también similitudes. Resulta especialmente difícil decantarse por alguno de ellos si no se tiene experiencia previa en la programación de videojuegos ya que no se sabe, a priori, cual es más fácil de manejar para la aplicación concreta que se aborda. Esto hace que la elección tenga que ser, en cierto modo, a ciegas. Para minimizar este hándicap se han consultado los documentos adjuntos en los anexos *01 - Motores de juegos*. En ellos se detallan las características específicas de cada motor de juegos, lo que ayuda a tomar una decisión respecto a cual utilizar.

La selección final para abordar el proyecto es Panda3D al cumplir todos los requisitos mencionados con anterioridad. Aunque a priori también se podría usar Irrlitch, la documentación de Panda3D resulta más interesante al ser más extensa y estar perfectamente dividido en apartados, lo que lo hace más atractivo para principiantes.

3.2.2. *PROGRAMA DE MODELADO*

Hasta el momento se sabe que herramienta utilizar para renderizar el modelo 3D pero no se sabe con cual se realizará tal modelo. Dado que ya se disponía de un modelo 3D realizado en la el proyecto complementario, este apartado tratará de guiar al lector sobre los aspectos que habrían de tenerse en cuenta en caso de no disponer de tal modelo.

En este caso no se tendrá en cuenta si el software de modelado es multiplataforma, gratuito etc. ya que el usuario no necesitará tenerlo instalado para utilizar el simulador. La selección del programa de modelado estará en cierto modo ligado al tipo de motor de juego que se use y en la experiencia previa del desarrollador en el uso de software de modelado 3D.

En este apartado se describirán aspectos considerados muy importantes a tener en cuenta, además de tratar algunos de los problemas encontrados durante la realización del presente proyecto, para permitir a futuros desarrolladores tomar caminos diferentes.

Algo realmente importante a la hora de seleccionar el programa de modelado, e incluso el motor de juegos, es el qué tipo de formatos con los que son capaces de trabajar cada uno de ellos. Es más que probable que softwares como Solidworks o Inventor no permitan exportar el modelo 3D a algún tipo de formato compatible. Para ello se deberá instalar algún tipo de complemento (normalmente de pago) o usar otro software de modelado de paso.

En este caso se necesita saber cuál es el formato de importación de modelos 3D con los que trabaja el motor de juegos seleccionado, esta información ha sido consultada en la página oficial de los desarrolladores de Panda3D [3]. Una vez consultada se sabe que el formato de trabajo es específico para Panda3D, siendo este .egg. Se necesitará por tanto la conversión del modelo 3D a este formato. En la documentación dada por los desarrolladores de Panda3D se explica la forma de exportar el modelo creado a tal formato, para ello se necesitará usar Blender [4], Maya o 3dmax. Dado que el primero de ellos es el único gratuito será el que se utilice en este proyecto para tal menester. Blender por sí solo no tiene la capacidad de poder exportar el modelo al formato que interesa y será necesario el uso de un complemento adicional (YABEE R13) [5].

Se puede observar que la tarea de conseguir tener el modelo 3D en un formato legible por el motor de juegos no es nada fácil. La figura 3.2 muestra un esquema de los pasos que se han de realizar para conseguir el resultado deseado. De haber usado desde un principio Blender para el modelado del prototipo no se necesitarían los primeros pasos, a pesar de ello esto no es recomendable ya que realizar piezas tales como pueden ser el husillo, los engranajes etc. resulta infinitamente más fácil e intuitivo en Solidworks o inventor.

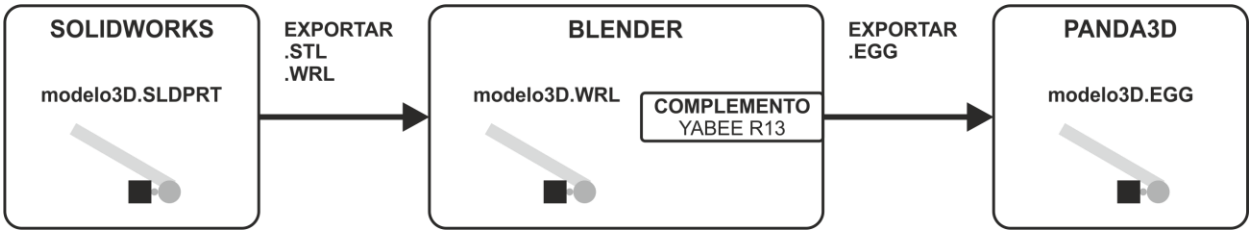

Figura 3.2. Pasos a seguir para conseguir formato .egg

Aunque se tenga el formato deseado, en la primera exportación es probable que se hayan perdido datos de texturas y por lo tanto el modelo aparezca sin color. Esto ocurrirá siempre que se exporte con el formato STL ya que no guarda las texturas. Si por el contrario se ha utilizado el formato WRL habrá piezas que conserven el color y otras que no (esto se debe a la forma en la que trabaja el formato WRL y sus limitaciones). En cualquiera de los dos casos es posible añadirle texturas y colores al modelo en Blender aunque es algo más complicado. Otros formatos que guardan las texturas son el .OBJ y .3DS, lamentablemente Solidworks no permite exportar en ese formato si no es mediante algún tipo de complemento de pago.

Aparentemente haber seleccionado Panda3D como motor de juegos parece a priori una mala decisión en cuanto a compatibilidad de formatos se refiere. Realmente de haber seleccionado otro motor de juegos diferente los problemas que aparecerían serian similares, ya que si se ha usado Solidworks para realizar el modelado, será necesario llevar a cavo un cambio de formato intermedio para conseguir el deseado.

Se puede concluir por tanto que para realizar el modelado 3D del sistema a simular será necesario disponer de diferentes softwares de modelado, siendo uno de ellos desde el que se realicen las piezas y otro el que permita exportar el archivo en el formato deseado. Además este último podrá ser utilizado para incluir a las piezas datos sobre los materiales (transparencia, color, reflexión…) y texturas dando al modelo un aspecto más realista. En algunos casos en los que se necesite algún tipo de animación programas como Blender, Maya o 3dmax son obligatorios ya que Solidworks no permite la implementación de lo que se conoce como animación esquelética.

3.2.3. *INTERFAZ GRÁFICA DE USUARIO*

En los requisitos se establecía que los parámetros de los diferentes componentes han de ser configurables, es por esto que se necesita algún tipo de nexo que permita al usuario interactuar con el simulador. Algo muy típico es el uso de la terminal o consola, que mediante la recepción comandos en forma de cadena de caracteres, permite al usuario usar el teclado para realizar las acciones que precise oportunas. Evidentemente esto supone que el usuario conozca previamente los comandos de control implementados por el programador, dado que esta forma de trabajar está algo desfasada y es engorrosa es mucho mejor la implementación de una interfaz gráfica. Además, el uso de una interfaz gráfica siempre será más intuitivo y los datos podrán ser representados en forma de grafica.

El principal uso de una interfaz grafica de usuario (a partir de ahora denominada GUI para acortar), consiste en proporcionar un entorno visual sencillo para permitir la interactuación del usuario con el simulador.

Todos los motores de juegos tienen funciones que permite implementar barras de deslizamiento, botones, entrada de textos y otros muchos complementos con los que realizar una GUI. En caso de no tenerlos existen librerías gratuitas como por ejemplo Crazy Eddie's GUI System o MyGUI que permiten realizar interfaces. El problema del uso de estas librerías o de las funciones proporcionadas por el motor de juegos es que es necesario implementar todo por código, tanto la función de cada botón, como su tamaño, posición, color etc. Más cómodo que todo eso es el uso de Qt designer [6]. Qt designer es una herramienta multiplataforma para el diseño construcción y personalización de interfaces gráficas, para ello usa las librerías de Qt. El uso de esta herramienta es muy sencillo puesto que, además de haber muchos tutoriales y manuales, permite crear la GUI arrastrando los botones, barras de deslizamiento… ya predefinidos a una ventana.

3.2.4. *ENTORNO DE DESARROLLO INTEGRADO*

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, que consiste en un editor de código, un compilador y un depurador.

Algunos de los entornos de programación más conocidos son Eclipse, Dev C++ y Microsoft Visual Studio. El desarrollador podrá utilizar cualquiera de los mencionados u otro al que esté más acostumbrado, en este caso se ha utilizado dos entornos diferentes, siendo uno de ellos la versión Microsoft Visual Studio 2008. La razón concreta para el uso de este entorno y no otro es que Panda3D tiene sus librerías precompiladas para él. Evidentemente podría utilizarse cualquier otro pero requiere volver a compilar las librerías.

El otro entorno de programación utilizado en el desarrollo del simulador ha sido Qt creator. La utilización de este otro entorno viene impuesta por la decisión anterior de utilizar Qt designer para la implementación de la interfaz gráfica de usuario. De todas las versiones ofrecidas por el fabricante se aconseja la instalación de Qt 5.3.0 para Windows 32-bit (MinGW 4.8.2, OpenGL) en su versión más actualizada.

3.2.5. *OTROS*

Además de las herramientas citadas en los puntos anteriores, es necesario el uso de librerías con funciones específicas dentro del simulador, más adelante se irán explicando cuales son estas librerías y para que se utilizan.

3.3. Resumen

- Se desarrollará un software de simulación del sistema real que estará dividido en tres partes, simulación electrónica, simulación mecánica y librería de interface.

- Se implementarán modelos matemáticos de los diferentes componentes del sistema que permitan representar su funcionamiento. Además, estos componentes ha de ser parametrizables para poder cambiar sus características.

- En el desarrollo del proyecto se usará software multiplataforma.

- El motor de juegos seleccionado para el renderizado será Panda3D. Para convertir el modelo 3D de Solidworks al formato compatible se usará Blender como software de paso.

- El entorno de programación para el desarrollo del código será por un lado Visual Studio 2008 y por otro Qt creator, el cual además permitirá realizar el interfaz grafica de usuario mediante Qt designer.

4. MODELOS MATEMÁTICOS

En el apartado de desarrollo de software se hablará de la simulación de diferentes componentes mecánicos y electrónicos, para realizar tal simulación se necesitaran los modelos matemáticos de cada uno de ellos. Por esta razón se le dedica un apartado individual a los diferentes modelos matemáticos, dividiendo éste en dos bloques (mecánica y electrónica) que contienen los modelos correspondientes. En cualquiera de los casos se ha intentado realizar modelos que se ajustasen a la realidad teniendo en cuenta la aplicación a la que están destinados.

Con la intención de conseguir un sistema lo más genérico posible se han utilizado modelos simplificados, por lo que en algunos casos no se corresponderán con el funcionamiento real del sistema pero si se aproximarán lo suficiente para esta aplicación. Antes de realizar modelos matemáticos más complejos es necesario analizar si con los propuestos el simulador se ajusta lo suficiente a la realidad o no. Para realizar esta tarea de ajuste es necesario tener construido el sistema físico antes de plantearse complicar los modelos.

No hay que olvidar que muchos de los parámetros de los diferentes modelos han de ser configurados por el usuario, es por esto que su simplificación hace más intuitivo y fácil la configuración del sistema sin necesidad de tener conocimientos avanzados sobre cada uno de sus componentes.

4.1. Mecánica

En este apartado se explica o se hace referencia a los anexos donde se detallan los modelos matemáticos utilizados para la simulación de los componentes mecánicos.

4.1.1. *MOTOR*

Para la simulación del motor de continua se ha utilizado un modelo matemático cuyo desarrollo completo puede verse en el anexo *01 - Motor de corriente continua* dentro de la carpeta *02 - Modelos matemáticos* [7].

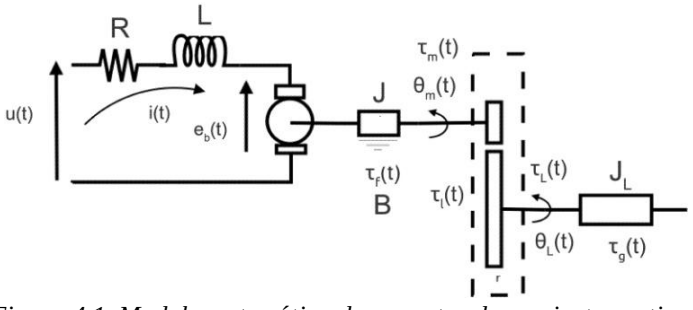

Figura 4.1. Modelo matemático de un motor de corriente continua

La figura 4.1 muestra el esquema eléctrico y mecánico de un motor CC cuyas ecuaciones son:

$$u(t) = L \cdot \frac{di(t)}{dt} + R \cdot i(t) + e_b(t) \tag{4.1}$$

$$\tau_m(t) = J \cdot \frac{d^2\theta_m(t)}{dt^2} + \tau_l(t) + \tau_f(t) \tag{4.2}$$

Donde u(t) es la tensión eléctrica aplicada al motor, i(t) la corriente eléctrica, $\theta_m(t)$ la posición angular del eje del motor, $e_b(t)$ la fuerza contraelectromotriz, $\tau_m(t)$ el par motor, $\tau_l(t)$ el par de la carga visto desde el eje del motor y $\tau_f(t)$ el par de fricción. Los parámetros L, R y J representan la inductancia, la resistencia eléctrica y el momento de inercia del rotor, respectivamente que consideraremos constantes. Esta clase de motor satisface además las siguientes ecuaciones de acoplo electromecánico:

$$\tau_m(t) = k_m \cdot i(t) \tag{4.3}$$

$$e_b(t) = k_b \cdot \dot{\theta}_m(t) \tag{4.4}$$

Siendo km la constante del par y k_b la constante de la fuerza contraelectromomotriz.

Se considera además el modelo clásico de fricción cuyas componentes son el par de fricción seca, el par de fricción viscosa y el par de fricción estática (τ_{fS}, τ_{fC} y τ_{fl} respectivamente) donde:

$$\tau_f(t) = \tau_{fC}(t) + \tau_{fV}(t) + \tau_{fS}(t) \tag{4.5}$$

$$\begin{aligned} &\tau_{fV}(t) = B \cdot \dot{\theta}_m(t) \\ &\tau_{fC}(t) = \tau_C \cdot \operatorname{sgn}\left(\dot{\theta}_m(t)\right) \\ &\tau_{fS}(t) = \begin{cases} \tau_e(t), & |\tau_e(t)| \le \tau_S, \quad \dot{\theta}_m(t) = 0, \quad \ddot{\theta}_m(t) = 0 \\ \tau_S \cdot \operatorname{sgn}(\tau_e(t)), & |\tau_e(t)| > \tau_S, \quad \dot{\theta}_m(t) = 0, \quad \ddot{\theta}_m(t) \neq 0 \end{cases} \end{aligned} \tag{4.6}$$

En este caso el par externo se representa con τ_e cuya ecuación es:

$$\tau_e(t) = \tau_m(t) + \tau_l(t) + J \cdot \ddot{\theta}_m(t) \tag{4.7}$$

Los parámetros B, τ_C y τ_S representan las constantes de fricción viscosa, de Coulomb y estática respectivamente.

Estas ecuaciones serán implementadas para permitir simular la dinámica del un motor de corriente continua de tal forma que se puedan observar los efectos de cambios de sentido en el eje del motor, inercias, retardos en el arranque del motor etc.

4.1.2. *HUSILLO*

En la simulación del husillo existen dos posibilidades diferentes ya que la tuerca puede ser trapezoidal o de bolas. La diferencia principal entre ambas es que en el caso del husillo con tuerca trapezoidal la fuerza axial producida por una carga no implicaría movimiento alguno en la tuerca. Esto no es así en la tuerca de bolas ya que no tiene autoenclavamiento (aunque se le podría aplicar una precarga para conseguir ese efecto). Desde un punto de vista del control esta circunstancia resulta interesante para la realización de prácticas ya que para mantener la tuerca en una posición fija se ha de regular continuamente el giro del husillo.

Para la tuerca trapezoidal se usarán las fórmulas de rozamiento de deslizamiento en tornillos (el desarrollo completo se puede consultar en el anexo *02 - Rozamiento en tornillos* [8] dentro de la carpeta *02 - Modelos matemáticos*) siendo el momento del par aplicado que hace inminente el deslizamiento del tornillo sobre la tuerca:

$$M = F \cdot r_m \cdot \tan(\varphi + \alpha) \tag{4.8}$$

$$\varphi = \tan^{-1}(\mu) \tag{4.9}$$

Donde F es la fuerza aplicada, r_m el radio medio del husillo, μ el rozamiento y α la inclinación de la rosca.

En el anexo *03 - Análisis del par de torsión de rotación* (correspondiente al fabricante THK [9]) se pueden consultar las ecuaciones del par de torsión requerido para convertir el movimiento de rotación del husillo de bolas en movimiento recto. De todas ellas nos interesará principalmente el par de torsión de fricción debido a una carga externa:

$$T_1 = \frac{Fa \cdot Ph}{2 \cdot \pi \cdot \eta} \cdot A \tag{4.10}$$

Siendo Ph el paso del husillo, η su eficiencia, Al factor de reducción y Fa la carga aplicada. T_1 junto con T_2 (par de torsión debido a la precarga) forman el par total de rotación requerido durante el movimiento uniforme.

Tanto en el caso del husillo con tuerca trapezoidal o de bolas su momento de inercia será el mismo, expresándose este como:

$$J = m \cdot \left(\frac{Ph}{2 \cdot \pi}\right)^2 \cdot A^2 + J_S \cdot A^2 + J_A \cdot A^2 + J_B \tag{4.11}$$

Donde J_S representa el momento de inercia del eje del husillo y J_A y J_B son los momentos de inercia de los engranajes, etc. conectados al lateral del eje de husillo y al lateral del motor respectivamente.

4.1.3. *GUÍA*

Para mantener el objetivo de tener un simulador genérico se ha implementado el modelo de la guía como una masa deslizándose por un plano inclinado con rozamiento. Esto implica que el modelo matemático no se corresponda exactamente con el de la guía utilizada en el sistema real.

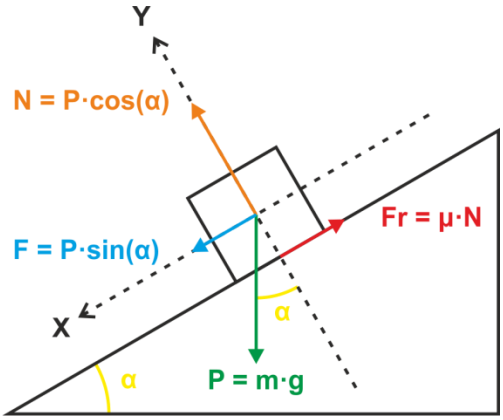

Figura 4.2. Rozamiento en plano inclinado

De la figura 4.2 se puede deducir que la fuerza total resultante es:

$$F_T = m \cdot g \cdot (\sin\alpha - \mu \cdot \cos\alpha) \tag{4.12}$$

Para la implementación en el código de la ecuación anterior se ha añadido la función de signo, de esta forma el rozamiento se opondrá siempre a la dirección del movimiento.

Se procurará que los títulos de apartados y subapartados no queden descolgados al final de una página, incluyendo un salto de página si es necesario para que esto no ocurra. Esto se hará así aun significando un aumento del espacio entre fin de párrafo y título.

4.1.4. *MASA*

El banco de ensayos que se pretende simular en este proyecto tiene 2 grados de libertad, una traslación y una rotación. Es necesario tener en cuenta que el propio sistema de traslación generará una inercia sobre el motor de rotación. Para simular esta situación se tomarán dos masas puntuales, una fija situada en el centro de gravedad del mecanismo de traslación y otra móvil que variará en función de la posición de la tuerca del husillo.

Dado que el mecanismo puede sufrir modificaciones, se le permitirá al usuario modificar datos como la posición X e Y del punto de gravedad respecto al eje de salida del reductor del motor de rotación, así como la masa en ese punto.

Con estas condiciones la inercia será:

$$J_T = J_F + J_V \tag{4.13}$$

Donde J_F corresponde a la inercia del mecanismo de traslación y J_V generada por la masa móvil de la tuerca más la carga aplicada.

Dado que la inercia de una masa puntual es:

$$J = m \cdot d^2 \tag{4.14}$$

Se puede deducir:

$$J_F = m_F \cdot \sqrt{g_X{}^2 + g_Y{}^2}^2 = m_F \cdot (g_X{}^2 + g_Y{}^2) \tag{4.15}$$

$$J_V = (m_T + m_C) \cdot \sqrt{p_X{}^2 + p_Y{}^2}^2 = (m_T + m_C) \cdot (p_X{}^2 + p_Y{}^2) \tag{4.16}$$

Donde m_f, m_t, y m_c, representan la masa del sistema de traslación, la masa de la tuerca y la masa de la carga respectivamente (todas ellas configurables por el usuario). Siendo g_x g_y y p_x p_y la posición de las masas con respecto al eje de salida del sistema de rotación.

De la misma forma que existen dos inercias (una variable y otra fija), también las masas crean un momento torsor en el eje de salida del reductor que pueden expresarse como:

$$M = m \cdot d \tag{4.17}$$

Siendo sus ecuaciones concretas:

$$M_F = m_F \cdot \sqrt{g_X{}^2 + g_Y{}^2} \tag{4.18}$$

$$M_V = (m_T + m_C) \cdot \sqrt{p_X{}^2 + p_Y{}^2} \tag{4.19}$$

Por lo que el par torsor total a la salida del eje del reductor es:

$$M_T = M_F + M_V \tag{4.20}$$

4.1.5. *REDUCTOR*

Cada uno de los motores está conectado a un bloque reductor por lo que es necesario tenerlo en cuenta a la hora implementar el modelo matemático en el código. Dado que el par y la inercia no serán los mismos en el eje de entrada que en el eje de salida del reductor, han de multiplicarse por el coeficiente de reducción. Como se puede observar en el punto 4.1.2 el modelo del husillo dado por el fabricante ya contempla esta circunstancia para el motor que lo hace girar. Para la simulación del reductor se ha seguido el punto 5. *Incorporación de la carga en el modelo del motor* del anexo *01 - Motor de corriente continua*.

Para este proyecto solo se incluye el efecto del coeficiente de reducción y de la eficiencia del reductor. Esto es debido a que la implementación de la simulación de la inercia de los engranajes, etc. presentaría más inconvenientes que ventajas. De todas formas, siempre será posible tal implementación en futuras versiones si el simulador funcionase de forma muy diferente a como se comporta el sistema real

4.2. Electrónica

El simulador electrónico está formado por varias secciones con funciones independientes. De todas ellas será especialmente interesante para su modelización matemática la sección referente a las señales de entrada y de salida del sistema. Dentro de esta sección se pueden diferenciar varios bloques que son:

Bloque sensor: es el bloque que se encarga de convertir una magnitud física en una variable eléctrica. La variable de entrada dependerá del tipo de sensor a simular y la variable de salida siempre será, en este caso, una tensión en cualquiera de las circunstancias.

Bloque de adaptación: este bloque se encarga de adaptar el rango de la señal de entrada (señal del bloque sensor) a otro rango deseado a la salida.

Bloque señal:
Señal de entrada: Si la señal de control entra al simulador, transformará su valor discreto de bits a una tensión equivalente.

Señal de salida: Si la señal de control sale del simulador, transformará la tensión en un valor de bits equivalente.

Se puede observar que todos los bloques tiene un funcionamiento idéntico ya que transforman un valor de entrada a otro de salida. La única diferencia entre ellos son las unidades con las que trabajan. Por esta razón se puede resumir el funcionamiento de todos ellos con el mismo modelo matemático.

Para su representación matemática se ha supuesto que cada uno de los bloques tiene una relación lineal entre el valor de entrada y el de salida. Esto es así ya que se ha considerado que los sensores son lineales o que se trabaja en su zona lineal.

Partiendo de estas premisas el modelo matemático de cada uno de los bloques mencionados se adaptará a la ecuación de una recta:

$$y = A \cdot x + B \tag{4.21}$$

Donde *A* es la pendiente, *B* la ordenada en el origen, *x* la señal de entrada e *y* la señal de salida.

El usuario tiene la capacidad de seleccionar el rango de entrada y de salida del bloque así como otros valores correspondientes a señales indeseadas. En la figura 4.3 el eje de abscisas representa el rango de valores de entrada para los diferentes bloques, de la misma forma el eje de ordenadas representa el rango de los valores de salida seleccionables por el usuario.

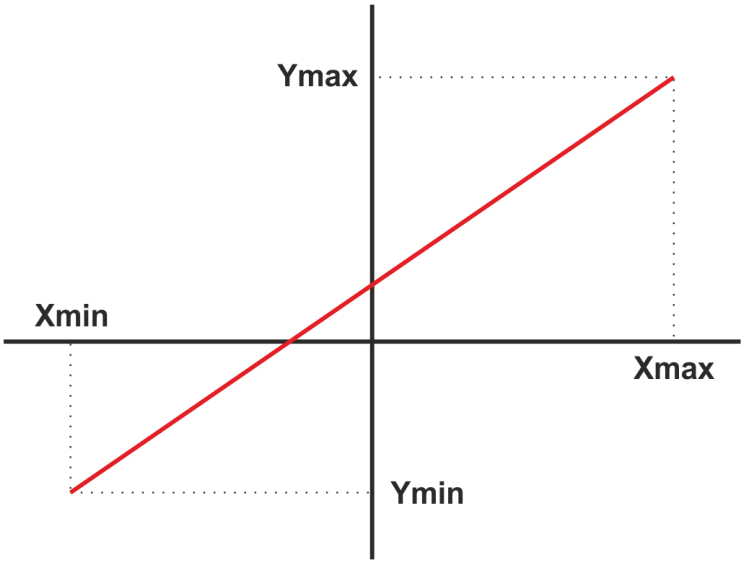

Figura 4.3. Recta

La pendiente se puede calcular fácilmente como:

$$A = \frac{Ymax - Ymin}{Xmax - Xmin} \tag{4.22}$$

Para calcular la ordenada en el origen se utiliza se sustituye *x* e *y* de la ecuación de la recta (4.21) por *Xmin* e *Ymin* respectivamente, al despejar la incógnita B el resultado será:

$$B = Ymin - A \cdot Xmin \tag{4.23}$$

En estos momentos el modelo simplificado de los bloques anteriormente mencionados quedaría de la siguiente forma:

$$y = \frac{Ymax - Ymin}{Xmax - Xmin} \cdot x + Ymin - A \cdot Xmin \tag{4.24}$$

Para aproximarlo algo mas a una situación real se le han incluido otros dos valores correspondientes a las señales indeseadas ya mencionadas, estas son el ruido y el offset. Estas señales pueden ser tanto positivas como negativas. El offset será un valor fijado por el usuario y se mantiene fijo en el tiempo, por el contrario, el ruido variará en un rango determinado (también fijado por el usuario) y será totalmente aleatorio.

Con todo esto, el modelo matemático de los diferentes bloques implementará de usando la siguiente fórmula:

$$y = \frac{Ymax - Ymin}{Xmax - Xmin} \cdot x + Ymin - A \cdot Xmin + offset + ruido \tag{4.25}$$

5. DESARROLLO DE SOFTWARE

El gran grueso del proyecto es meramente programación pura y dura, con este capítulo se pretende resumir el funcionamiento del código. Para ello se tratará brevemente los aspectos más importantes de cada una de las partes en las que se divide el software implementado para la realización del simulador.

El software consta de tres partes independientes pero complementarias entre sí. Antes de comentar cada una de ellas es interesante saber cuál ha sido el planteamiento de trabajo para la realización de estas. Se procede por tanto a detallar tal planteamiento.

Como se ha comentado en varias ocasiones, el prototipo real a simular ha sido diseñado de tal forma que pueda sufrir modificaciones, es decir, de forma que puedan cambiarse el tipo de motor, de husillo, la reducción etc. Gracias a esto se obtiene un dispositivo cuya configuración es variable, existiendo por lo tanto múltiples combinaciones entre los diferentes bloques que lo componen. Esto es muy importante de cara a la realización del software ya que ha de ser capaz de amoldarse a estos cambios en la configuración real del sistema. Además, dado que es un prototipo en desarrollo y no una versión final, el software ha de ser realizado pensando en facilitarle la vida a un futuro desarrollador, de tal forma que no necesite crear el código desde cero, sino que pueda aprovechar en la medida de lo posible la mayor cantidad del mismo.

Para conseguir esta versatilidad en el código es necesario olvidarse de la programación estructurada, a la que la gente suele estar más habituada, y conocer los entresijos de la programación orientada a objetos. La programación orientada a objetos (POO) es una forma especial de programar, más cercana a como se expresarían las cosas en la vida real que otros tipos de programación. Con la POO se tiene que aprender a pensar las cosas de una manera distinta, para escribir los programas en términos de objetos propiedades, métodos etc. Con este tipo de programación se obtienen los siguientes beneficios:

- Reusabilidad: Si se diseñan las clases adecuadamente, se pueden usar en diferentes partes del programa y en numerosos proyectos.

- Mantenibilidad: Debido a la sencillez para abstraer el problema, los programas orientados a objetos son más sencillos de leer y comprender, pues permiten ocultar detalles de implementación dejando visibles solo aquellos detalles más relevantes.

- Fiabilidad: Al dividir el problema en partes pequeñas se pueden probar de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir.

Se observa por lo tanto que la programación orientada a objetos es una buena opción para el desarrollo de este proyecto. La figura 5.1 muestra alguna de las características básicas de la POO con la que los desarrolladores han de estar familiarizados si se pretende, mejorar, cambiar o completar el código de este simulador.

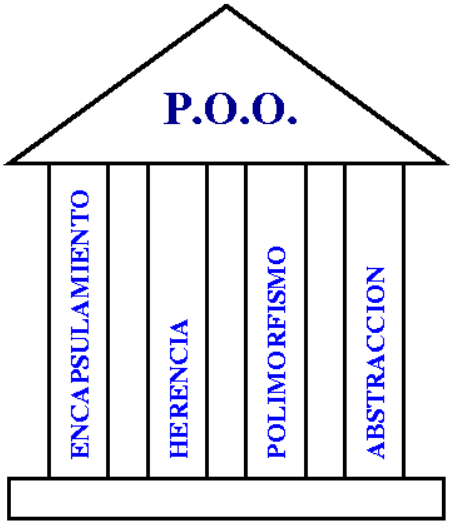

Figura 5.1. Pilares de la programación orientada a objetos

Aparece por tanto un nuevo requisito que nada tiene que ver con la forma de funcionar del simulador pero si con su desarrollo. Este nuevo requisito es la implementación del código orientándolo a la utilización de objetos, realmente se podría considerar una necesidad más que un requisito ya realizar el código de esta forma facilita el desarrollo y depuración del simulador.

Otra de las novedades que aparece respecto a la programación estructurada es la necesidad de eventos, como posteriormente se verá tanto en el simulador mecánico como en el simulador electrónico. La programación estructurada es aquella en la que se sabe cuál es el flujo del programa, es decir, se sabe de antemano en qué orden se ejecutaran las diferentes funciones del código. La figura 5.2 muestra un esquema de los diferentes tipos de programación estructurada, en cualquiera de los casos existe un flujo y las acciones se realizaran en el orden especificado.

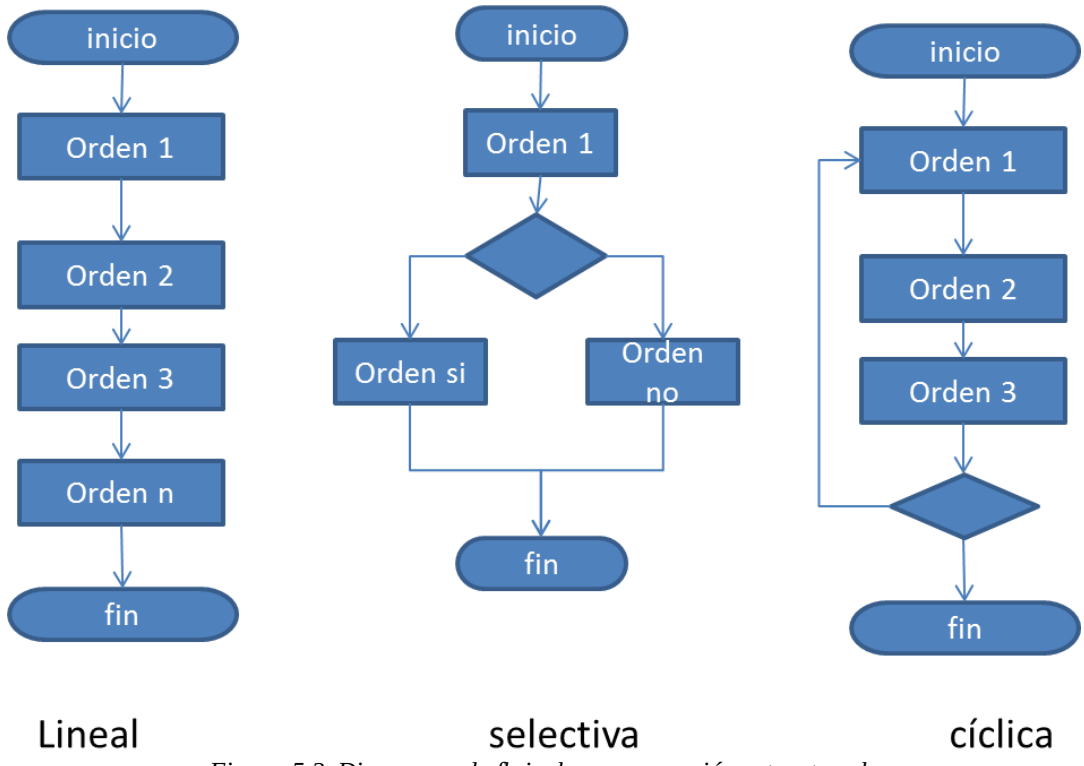

Figura 5.2. Diagramas de flujo de programación estructurada

Dado que se pretende realizar un simulador y la intención es permitir que el usuario interactué con él, resulta evidente que no se sabrá cual es el flujo del programa. Esto se debe a que no se puede saber cómo ni cuándo realizará una acción el usuario.

En la programación dirigida por eventos, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y demás código inicial y a continuación el programa quedará bloqueado hasta que se produzca algún evento. Cuando alguno de los eventos esperados por el programa tenga lugar, el programa pasará a ejecutar el código del correspondiente administrador de evento. Por ejemplo, si el evento consiste en que el usuario ha hecho click en el botón de play de un reproductor de películas, se ejecutará el código del administrador de evento, que será el que haga que la película se muestre por pantalla.

La programación dirigida por eventos es la base de lo que llamamos interfaz de usuario, aunque puede emplearse también para desarrollar interfaces entre componentes de Software o módulos del núcleo.

Con lo anteriormente descrito se deduce que, la programación orientada a objetos será lo que se utilice para hacer el código más comprensible y aprovechable dentro del presente proyecto y por futuros desarrolladores. Por otro lado, la programación orientada a eventos es la que permitirá crear el nexo entre el usuario y el código.

Hasta ahora se tiene una visión de cómo se estructurará el código y también se sabe que se dividirá en tres partes, pero no se han comentado cuales son ni como se relacionan entre ellas. La figura 5.3 muestra un lazo de control con las diferentes partes que compondrán la totalidad del proyecto. Se puede observar además la gran similitud con el esquema mostrado en la figura 3.1 en el apartado de planteamiento.

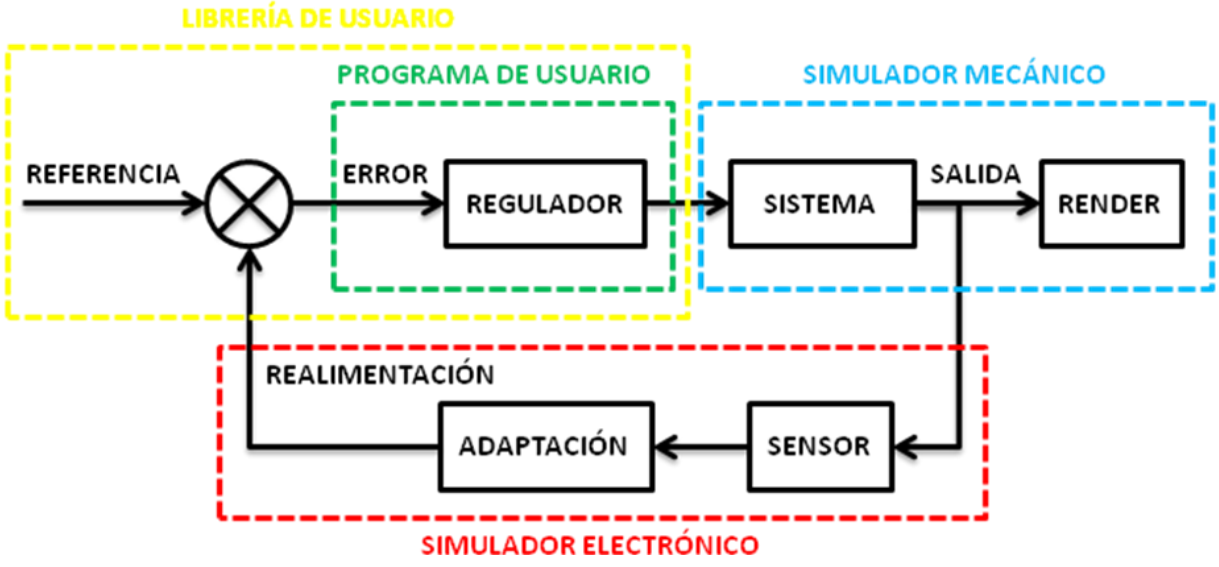

Figura 5.3. Lazo cerrado de control

Se pretende por tanto realizar una simulación mecánica correspondiente con el sistema real fabricado. Además se le dotara de una interfaz electrónica (también simulada) que permita comunicarse con la parte mecánica, gracias a esta comunicación se podrán obtener datos del estado del sistema. Para que el usuario pueda interactuar con el sistema se le facilitará una librería capaz de comunicase con ambos simuladores, siendo ésta por tanto el nexo entre ambos que permitirá al usuario cerrar el lazo de control.

La configuración de elementos intervinientes y sus parámetros se realiza mediante archivos con formato .ini. Se permitirá que algunos de los parámetros más relevantes sean modificables mediante el interfaz de usuario.

La comunicación entre los subsistemas se realiza mediante sockets de comunicación TCP/IP, lo que permite que cada subsistema pueda residir en el mismo o en diferente computador.

5.1. Simulador mecánico

Con intención de explicar los componentes que intervienen en esta parte del simulador se ha realizado el esquema de la figura 5.4.

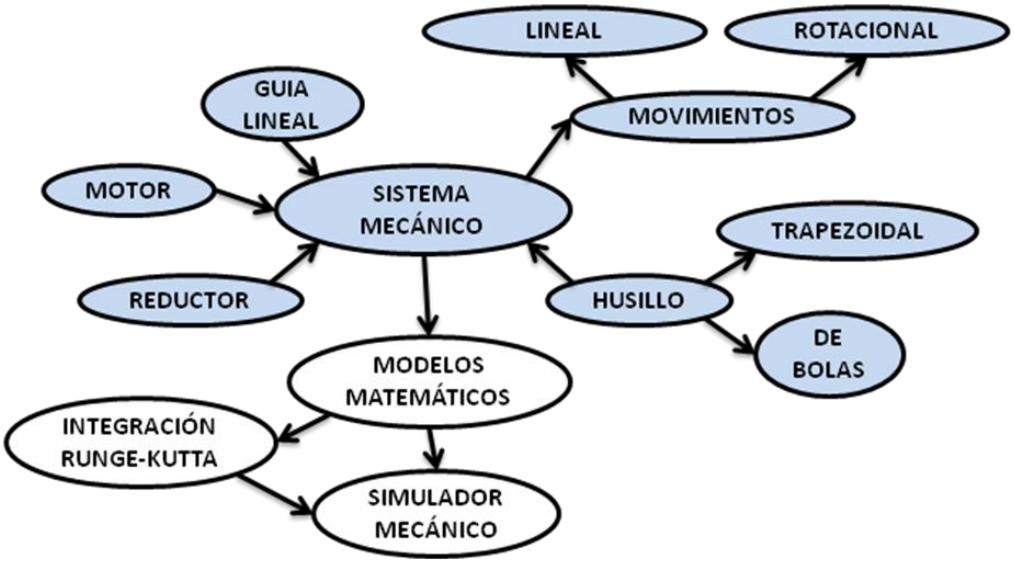

Figura 5.4. Componentes del sistema mecánico

El sistema mecánico está compuesto por diferentes elementos como los motores, la transmisión (con reducción o no), la guía prismática y el husillo, pudiendo ser este de bolas o trapezoidal. Con estos componentes se consiguen dos movimientos, uno rotacional y otro lineal. Todo esto ha de ser simulado, para ello se necesitará previamente los modelos matemáticos de los componentes mecánicos que forman el sistema. Para poder realizar la simulación mecánica de los movimientos del sistema se necesitará aplicar la integración Runge-Kutta, más adelante se comentará que es esto y por qué se necesita.

Otra de las cosas a mencionar, es como se usarán las herramientas que previamente se seleccionaron, para ello se sigue completando el esquema anterior como se muestra en la figura 5.5.

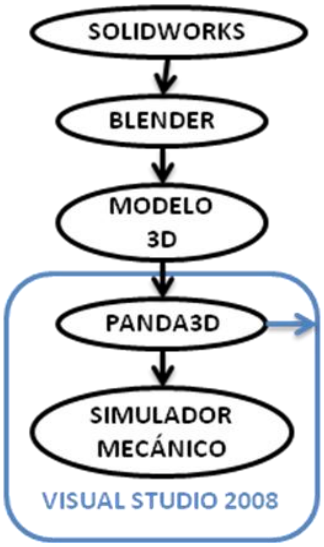

Figura 5.5. Herramientas usadas en el simulador mecánico

El esquema anterior no da información nueva que no se conociese anteriormente. Solidworks será utilizado para crear un modelo 3d, que gracias a Blender podrá ser exportado en el formato correcto del motor de juegos Panda3D. Las librerías de Panda3D vienen precompiladas para el uso de visual estudio 2008, razón por la cual se usará este entorno de programación para realizar el simulador mecánico.

De momento, nada se sabe sobre la estructura real del código a desarrollar, la figura 5.6 muestra vagamente tal estructura. En este esquema aparece el concepto de "hilo" por lo que antes de nada se explicará brevemente de qué se trata.

En sistemas operativos, un hilo de ejecución o subproceso es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. La creación de un nuevo hilo es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden a ese dato modificado inmediatamente.

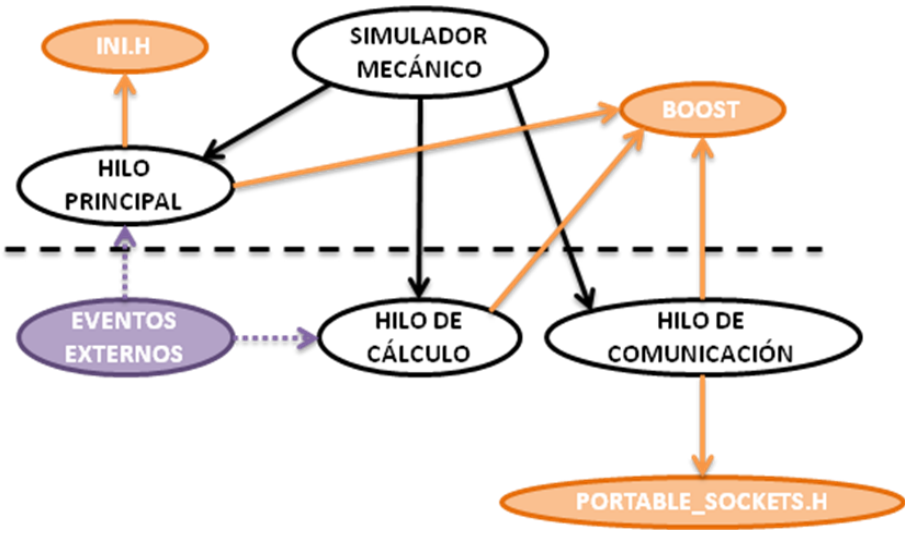

Figura 5.6. Estructura básica del código

De la definición anteriormente dada sobre que es un hilo podemos concluir que el simulador mecánico es un proceso del sistema que está compuesto por tres hilos, siendo estos:

- Hilo principal: Este hilo es el que ejecuta la mayor parte del código, es donde se ejecutan las funciones de lectura de archivos de configuración o todas aquellas funciones encargadas del renderizado del modelo 3D y de sus movimientos. El hilo principal será el creará a los otros dos, por lo que se convertirá en padre de estos

- Hilo de cálculo: Este hilo se encarga de realizar todos los cálculos matemáticos correspondientes con los modelos de los componentes mecánicos.

- Hilo de comunicaciones: El hilo de comunicaciones se mantiene a la espera de una conexión entrante. Su principal uso es el envío y recepción de tramas de datos con información del sistema al detectar algún dispositivo conectado.

El hilo principal y el hilo de cálculo manejan variables cuyo valor puede cambiar de forma fortuita durante la ejecución del programa debido a la interacción del usuario con el simulador, de ahí que en el esquema aparezca este suceso indicado como eventos externos.

En el *3.2. Selección de herramientas* dentro del apartado de *3. Planteamiento* se hizo referencia a la necesidad del uso de librerías adicionales, los bloques sombreados en naranja de la figura 5.6 muestran las librerías que se usarán en cada uno de los hilos.

Para crear los diferentes hilos podrían usarse las funciones específicas de Windows, dado que si hiciese de esta forma el simulador solo podría ejecutarse en este sistema operativo se ha decidido utilizar una librería multiplataforma. Un conjunto de librerías multiplataforma bastante conocido son las contenidas por la biblioteca C++ de Boost [10], gracias a su licencia, de tipo BSD, permite que sea utilizada en cualquier tipo de proyectos, ya sean comerciales o no.

A estas alturas ya se tiene un esquema general de los diferentes factores que entran en juego para el desarrollo e implementación del código de la parte del simulador mecánico, se explicará por tanto las funciones más importantes del código así como su funcionamiento general. Para ello se comentarán los hilos por separado ya que representan bloques independientes dentro del conjunto global del programa.

5.1.1. *HILO PRINCIPAL*

Se podría decir que el hilo principal de divide en dos bloques separados por el tiempo, estos dos bloques son el bloque de creación e inicialización de parámetros etc. y el bloque de renderizado (el que será afectado por eventos externos).

Antes de detallar las funciones que se llevan a cabo en este hilo, es conveniente hablar de las diferentes librerías y clases que componen la totalidad del código para tener una visión general y poder mencionarlas sabiendo cual es su utilidad. La tabla 5.1 resume brevemente estas librerías y clases utilizadas. Esta tabla no se incluye la librería portable_sockets.h ya que viene integrada en el motor de juegos panda3D.

LIBRERÍA	CLASE	DESCRIPCIÓN
device		Contiene las clases de cada uno de los elementos mecánicos independientes de los que se ha incluido algún tipo de simulación.
	Patin	Clase para la simulación de un patín genérico. Contiene datos de masa y funciones para leerla o modificarla.
	Guia	Clase para la simulación de una guía genérica. Contiene datos de masa y rozamiento estático y dinámico con el patín, también funciones para interactuar con esas variables
	Husillo	Clase para la simulación de un husillo. Contiene los parámetros que afectan al modelo matemático utilizado y las funciones para interactuar con ellas.
	Tuerca	Clase para la simulación de una tuerca (modeló relacionado con el husillo). Contiene los parámetros de masa y precarga, también las funciones para interactuar con ellas.
	Reductor	Clase para la simulación de un reductor. Contiene los parámetros que afectan al modelo matemático utilizado (reducción y rendimiento) y las funciones para interactuar con ellas.
	Motor	Clase para la simulación de un motor. Contiene los parámetros que afectan al modelo matemático utilizado y las funciones para interactuar con ellas.
stage		Esta librería contiene las clases de traslación y rotación, con ellas se pretende simular los dos grados de libertad del sistema real.
	Traslacion	Clase que hereda de cada uno de las clases implementadas en la librería device. Con ella se pretende tener todos los elementos mecánicos que forman el sistema de traslación.
	Rotacion	Esta clase hereda de Traslación y es la que se utiliza para implementar los modelos matemáticos ya que contiene todos los elementos del sistema.
Tcp_ip		Librería para la comunicación vía socket
	Tcpip	Esta clase contiene las funciones correspondientes para crear un hilo independiente para la comunicación con el simulador electrónico
Panda3d		Librería para la creación del programa.
	Panda	Esta clase crea el simulador completo, es decir, establece los eventos de teclado, los tiempos de simulación, inicializa los parámetros de los componentes contenidos en device …
ini ini_reader		Librerías multiplataforma para la lectura de archivos de configuración. Más información en https://code.google.com/p/inih/
parameters		Esta librería no contiene funciones, tan solo contiene variables globales y definiciones de macros útiles para una mejor comprensión y depuración de código

Tabla 5.1. Resumen de librerías y clases del simulador mecánico

A continuación se muestra un diagrama de flujo que explica el funcionamiento del hilo principal. La figura 5.7 muestra las dos partes, ya mencionadas, en las que se puede dividir este hilo.

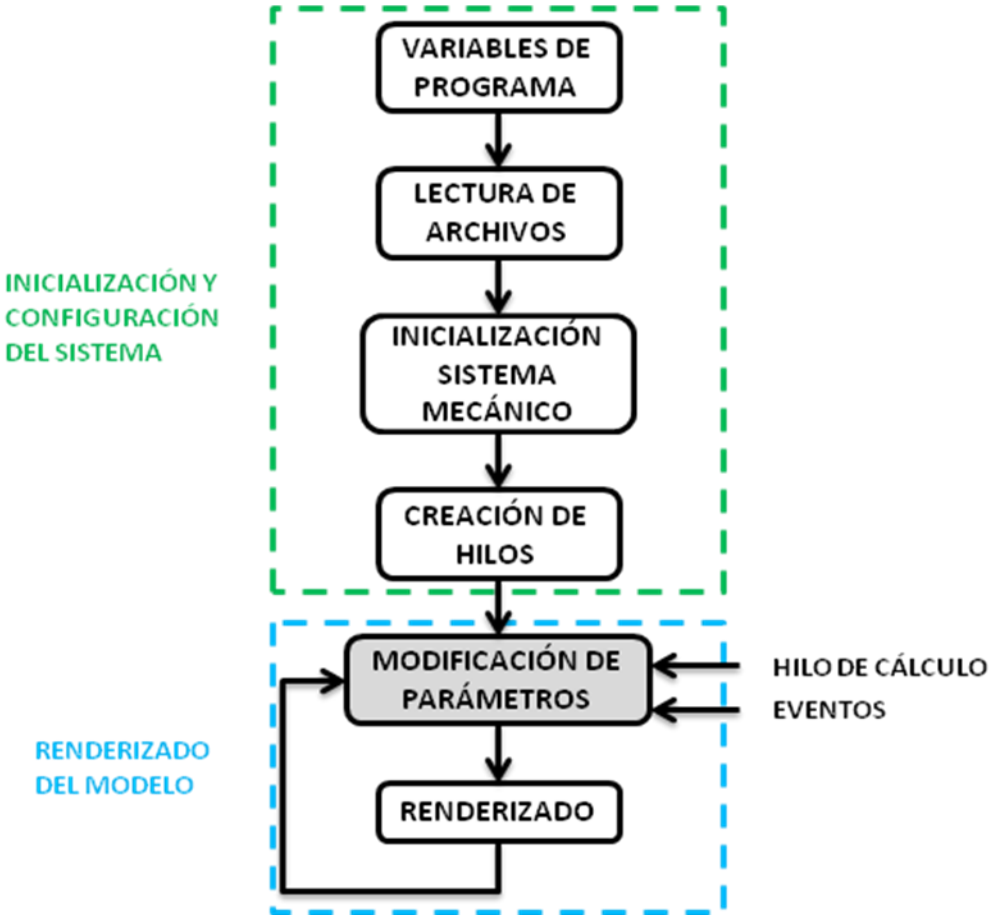

Figura 5.7. Diagrama de flujo de hilo principal.

El primer bloque se encarga de crear he inicializar las variables internas del programa, es decir, todas aquellas variables necesarias para el funcionamiento de la aplicación, contadores, objetos etc. En esta primera parte también se especifican los eventos de teclado (consultar tabla 5.2 dentro del punto *5.1.4 Eventos* y *manual de usuario* en el anexo *06 - Manual de usuario*).

Algunos de los objetos creados en el primer bloque, serán de alguna de las clases de la librería device (ver tabla 5.1) por lo que necesitaran ser inicializados. En este caso los valores de inicialización estarán guardados en un archivo modificable por el usuario, es aquí donde entra en juego las funciones de la librería de inicialización. Esta parte del código es la que permite que los valores de los componentes mecánicos sean parametrizables, ya que cada usuario podrá probar distintas configuraciones con el mismo simulador. Una vez leído los archivos de configuración e inicializadas todas las variables se crearan dos hilos fundamentales, hilo de comunicaciones e hilo de cálculo, estos se explicarán detalladamente más adelante. Ambos hilos se ejecutarán de forma paralela a la etapa de renderizado del modelo.

Después de la inicialización y configuración del sistema, el programa ya estará listo para renderizar el modelo 3D. En esta fase es donde el motor de juegos cobra mayor importancia y donde tiene sentido el uso de Panda3D que permite, gracias a sus funciones, visualizar el modelo y dotarle de diferentes movimientos. En este punto es necesario hacer mención al sistema de ejes usado y a la localización de centros de cada pieza individual.

El modelo 3D que se importa gracias al motor de juegos estará compuesto de varias piezas que hay que mover, estos movimientos en algunos casos serán traslaciones a lo largo de algunos de los ejes y en otros casos serán rotaciones tomando los como ejes de giro. Es de vital importancia saber cuál es el sistema de referencia de cada pieza así como la posición del origen de coordenadas. Para definir estos parámetros se ha utilizado Blender, con lo cual se observa que no solo se ha utilizado esta herramienta como elemento de paso de un tipo de archivo a otro legible por el motor de juegos, sino que también se puede utilizar para la definición del sistema de referencia. En el proceso de conversión del modelo 3D al formato .egg legible por Panda3D es posible que varíe el sistema de ejes referencia por lo que al mover una pieza en +X puede que con coincida con el movimiento esperado. Esto resulta incomodo a la hora de establecer los movimientos ya que si se tiene desconocimiento sobre esta circunstancia puede dar lugar a error pensando que el código está mal implementado cuando en realidad no es así. La imagen de la figura 5.8 muestra el problema mencionado. En este caso la exportación a formato .obj desde Solidworks para la posterior importación en Blender acarrea un cambio en el sistema de ejes, donde el eje Y pasa a ser el Z y viceversa.

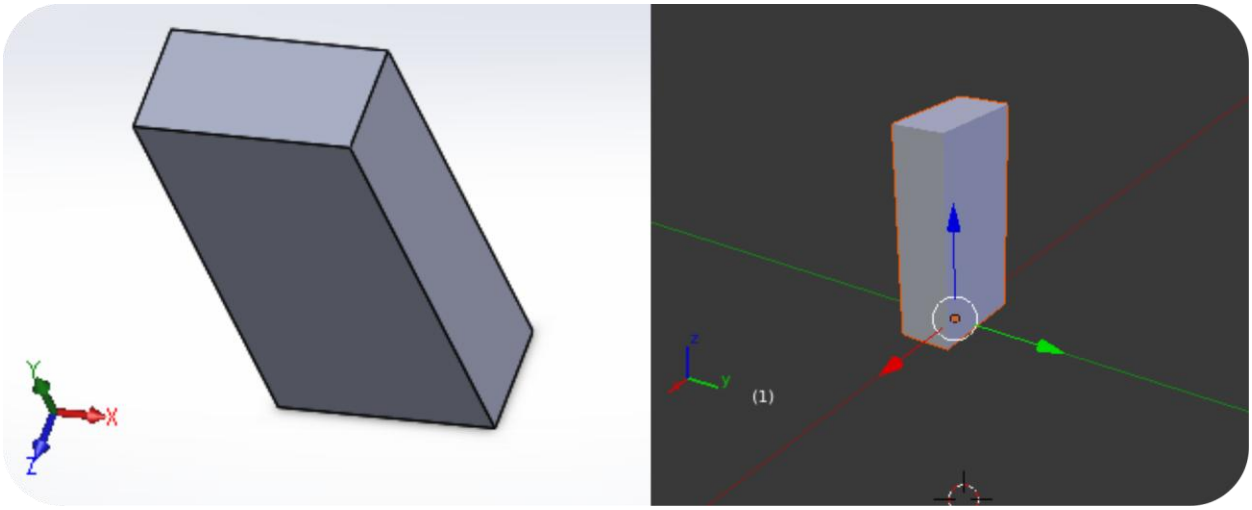

Figura 5.8. Cambio en el sistema de ejes de referencia.

Otro aspecto a tener en cuenta para la importación del modelo desde el motor de juegos es el número de vértices del mismo. Un gran número de vértices implica que se tendrán que procesar más datos y podría ralentizar el programa, por este motivo el modelo 3D del sistema real ha tenido que ser modificado eliminando tornillería, taladros y elementos superfluos.

Los movimientos de cada componente se realizarán con respecto a su nodo padre (componente de mayor prioridad, de tal forma que un cambio de posición sobre el implica un cambio de posición de todos sus hijos). Los parentescos se pueden establecer tanto por código como desde Blender, esta tarea es algo pesada por lo que otra ventaja añadida detener menos elementos es ahorrar tiempo en la designación de parentescos. En la figura 5.9 se muestra un árbol de parentescos entre las diferentes piezas que componen el sistema. Cada pieza tiene su nombre individual, de esta forma se podrá referir a ella desde el motor de juegos para moverla.

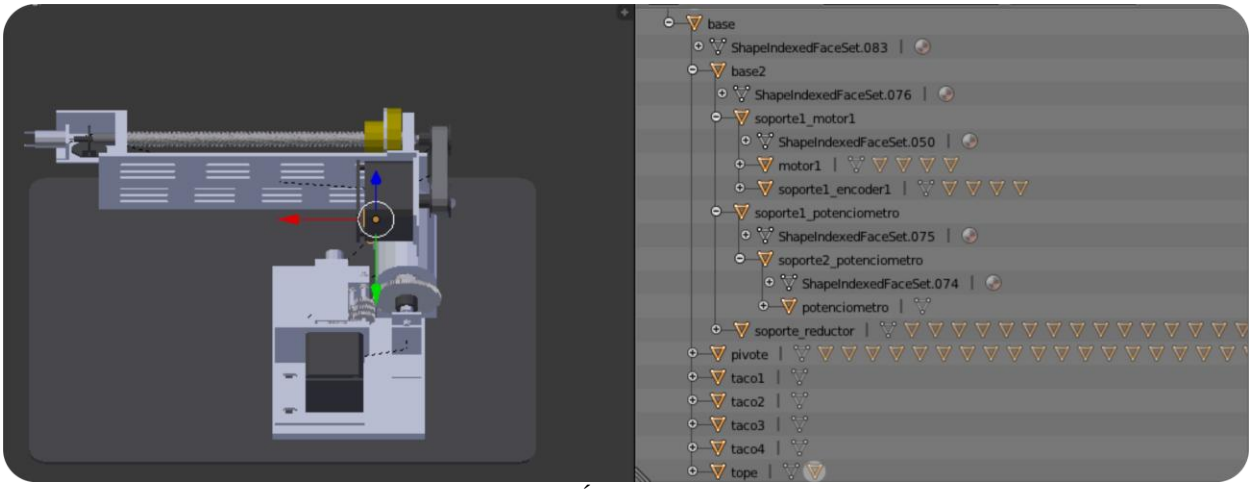

Figura 5.9. Árbol de parentescos.

Resulta evidente que para ver el sistema moviéndose hay variables que han de cambiar, este hecho se ha querido representar con el bloque sombreado en gris. La modificación en los parámetros de renderizado vendrá dada por diferentes circunstancias. Este bloque se comprenderá mejor si se explica una vez comentado el funcionamiento de los hilos de cálculo y de comunicaciones. De momento se puede deducir del esquema que los parámetros de renderizado podrán variar por eventos externos o nuevas soluciones en el hilo de cálculo.

El renderizado se realizará con una frecuencia concreta, en este caso se ha seleccionado 25 Hz ya que es la frecuencia con la que el ser humano interpreta que algo se está moviendo. Además esta frecuencia se ha considerado que es lo suficientemente baja como para no sobrecargar al sistema operativo con cálculos innecesarios.

5.1.2. *HILO DE CÁLCULO*

El hilo de cálculo es el encargado de realizar las operaciones necesarias para conseguir una simulación acorde al sistema real. Para ello necesita utilizar los modelos matemáticos descritos en el subapartado 4.1.1 Motor. Algunos de estos cálculos se harán de forma directa pero para otros es necesario aplicar algún tipo de método de integración. La integración se aplicará principalmente para resolver las ecuaciones del motor.

Antes de comentar cual ha sido el método de integración escogido y de explicar los entresijos del hilo de cálculo, se harán una serie de operaciones con el fin de relacionar todas las ecuaciones del motor. Dado que en el subapartado 4.1.1 ya se explico el significado de cada variable no se volverá a realizar aquí.

Primero se sustituirán los siguientes pares de ecuaciones:

$$\tau_m(t) = J \cdot \frac{d^2\theta_m(t)}{dt^2} + \tau_l(t) + \tau_f(t) \tag{5.1}$$

$$\tau_m(t) = k_m \cdot i(t) \tag{5.2}$$

$$u(t) = L \cdot \frac{di(t)}{dt} + R \cdot i(t) + e_b(t) \tag{5.3}$$

$$e_b(t) = k_b \cdot \dot{\theta}_m(t) \tag{5.4}$$

Con lo que se obtendrán estas nuevas ecuaciones:

$$k_m \cdot i(t) = J \cdot \frac{d^2\theta_m(t)}{dt^2} + \tau_l(t) + \tau_f(t) \tag{5.5}$$

$$u(t) = L \cdot \frac{di(t)}{dt} + R \cdot i(t) + k_b \cdot \frac{d\theta_m(t)}{dt} \tag{5.6}$$

De la ecuación 5.6 se desconocen la corriente y su derivada y también la velocidad angular, esta última se puede obtener de la ecuación 5.5 aunque para ello hay que seguir operando. Del modelo clásico de fricción se obtenía:

$$\tau_f(t) = \tau_{fC}(t) + \tau_{fV}(t) + \tau_{fS}(t) \tag{5.7}$$

$$\begin{aligned} &\tau_{fV}(t) = B \cdot \dot{\theta}_m(t) \\ &\tau_{fC}(t) = \tau_C \cdot \mathrm{sgn}\left(\dot{\theta}_m(t)\right) \\ &\tau_{fS}(t) = \begin{cases} \tau_e(t), & |\tau_e(t)| \le \tau_S, \quad \dot{\theta}_m(t) = 0, \quad \ddot{\theta}_m(t) = 0 \\ \tau_S \cdot \mathrm{sgn}(\tau_e(t)), & |\tau_e(t)| > \tau_S, \quad \dot{\theta}_m(t) = 0, \quad \ddot{\theta}_m(t) \neq 0 \end{cases} \end{aligned} \tag{5.8}$$

Si se sustituye 5.8 en 5.5 se deduce

$$k_m \cdot i(t) = J \cdot \frac{d^2\theta_m(t)}{dt^2} + B \cdot \frac{d\theta_m(t)}{dt} + \tau_c(t) \cdot sgn\left(\frac{d\theta_m(t)}{dt}\right) + \tau_{fS}(t) \qquad (5.9)$$

La ecuación anterior es de orden dos pero se puede pasar a orden uno fácilmente si se escribe de la siguiente forma:

$$k_m \cdot i(t) = J \cdot \frac{d\omega_m(t)}{dt} + B \cdot \omega_m(t) + \tau_c(t) \cdot sgn\big(\omega_m(t)\big) + \tau_{fS}(t) \qquad (5.10)$$

Si despejamos las derivadas en 5.6 y 5.10 obtendremos un sistema de ecuaciones que definen el funcionamiento del motor:

$$\frac{d\omega_m(t)}{dt} = \frac{k_m \cdot i(t) - B \cdot \omega_m(t) - \tau_c(t) \cdot sgn\big(\omega_m(t)\big) - \tau_{fS}(t)}{J} \qquad (5.10)$$

$$\frac{di(t)}{dt} = \frac{u(t) - R \cdot i(t) - k_b \cdot \omega_m(t)}{L} \qquad (5.11)$$

Las ecuaciones anteriores es necesario resolverlas mediante algún método de integración adecuado. En computación uno de los más habituales para resolver problemas de ecuaciones diferenciales ordinarias con condiciones iniciales es el método de Runge-Kutta. La razón por la que es ampliamente usado es porque es un método iterativo, lo hace que su implementación en el código sea sencilla.

Los métodos de Runge-Kutta se dividen en dos grupos que son de paso fijo y de paso variable. Este último ser acerca mejor a la solución aunque también es más difícil de implementar. Para poder explicar cuál es el inconveniente que hace que se descarte el método de paso variable se explicará primero el método de paso fijo seleccionado, siendo este Runge-Kutta de cuarto orden.

Problema general:

$$\frac{dy}{dx} = f(x,y), \quad y(x_0) = y_0 \qquad (5.12)$$

Método RK de cuarto orden:

$$y_{i+1} = y_i + \frac{1}{6} \cdot (k_1 + 2 \cdot k_2 + 2 \cdot k_3 + k_4) \qquad (5.13)$$

Donde:

$$k_1 = h \cdot f(x_i,\ y_i) \tag{5.14}$$

$$k_2 = h \cdot f\left(x_i + \frac{h}{2},\ y_i + \frac{k_1}{2}\right) \tag{5.15}$$

$$k_3 = h \cdot f\left(x_i + \frac{h}{2},\ y_i + \frac{k_2}{2}\right) \tag{5.16}$$

$$k_4 = h \cdot f(x_i + h,\ y_i + k_3) \tag{5.17}$$

En este método la h representa el paso de integración, que hablando en términos de software representaría en cierto modo cada cuanto se ejecuta el hilo de cálculo. Dado que el tiempo de ejecución del hilo es siempre el mismo, hace que la implementación del método RK de paso variable sea más complicado de implementar y por eso ha sido descartado. En cambio, para implementar el de paso fijo solo se necesita aplicar iterativamente sus ecuaciones.

Algo importante de cara a la simulación es seleccionar bien el paso de integración. Cuanto menor sea el paso mejor será la aproximación a la solución real, aunque también se necesitaran ejecutar un mayor número de instrucciones por unidad de tiempo. Windows no puede ejecutar hilos con un periodo menos a 10ms con lo que el periodo de muestreo seria de 100Hz. El teorema de Nyquist dice que para reconstruir una señal se necesita una frecuencia de muestreo el doble que la de la señal a reconstruir, esto implica que si la señal varía con una frecuencia mayor de 50Hz aparecerá aliasyng. Para solucionar este problema lo que se ha hecho es realizar cada vez que se ejecuta el hilo un total de 1000 iteraciones. La figura 5.10 muestra los diferentes tiempos de muestreo de los elementos comentados hasta ahora. En amarillo se muestran las iteraciones del método de integración, en rojo el periodo de ejecución del hilo de cálculo y en azul el del hilo de renderizado

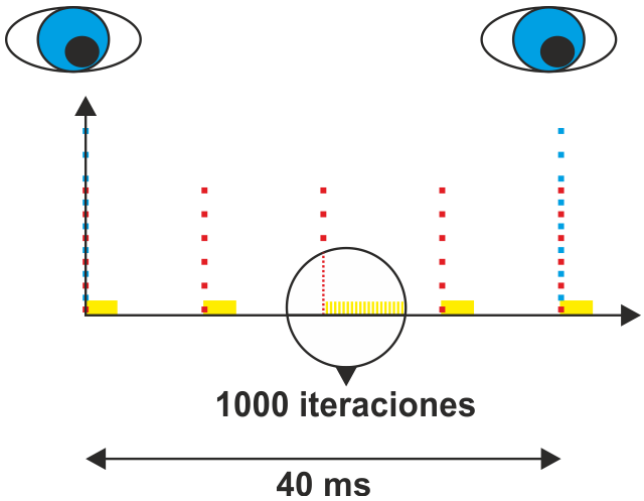

Figura 5.10. Tiempos de muestreo.

El usuario tiene la capacidad de modificar el tiempo de ejecución del hilo de cálculo pero no el número de iteraciones que se realizan. Si el tiempo es excesivamente alto se observara como el sistema empieza a funcionar de forma ilógica.

La figura 5.11 muestra el funcionamiento básico del hilo de cálculo. En él se observa de nuevo el bloque gris de modificación de parámetros. Con cada iteración se recalcularán los parámetros que se usarán para el renderizado.

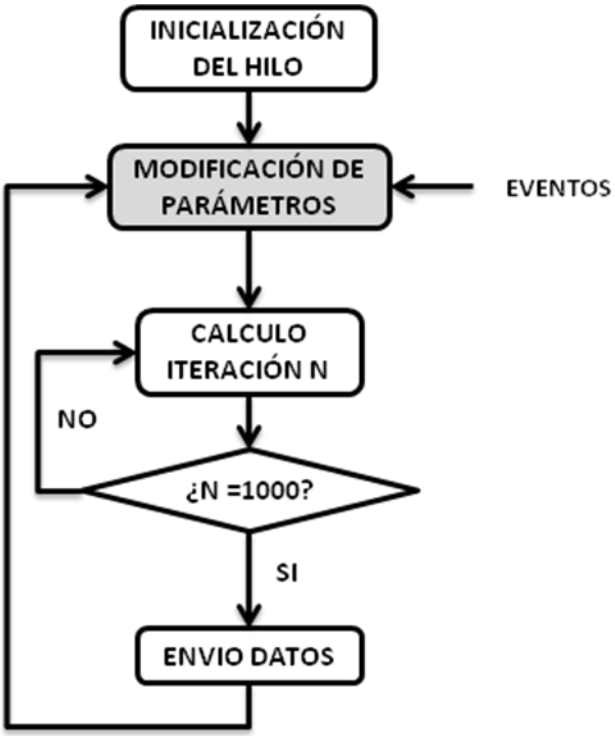

Figura 5.11. Diagrama de flujo de hilo de cálculo

En este esquema aparece algo nuevo que no se había comentado con anterioridad al hablar del hilo de cálculo y es la capacidad para enviar datos. Aunque hay un hilo especifico para las comunicaciones, ese solo tiene sentido para la información externa que entra al sistema. Los datos que se envíen fuera se harán desde el hilo de cálculo ya que es la manera más eficaz de asegurarnos de enviar datos totalmente actualizados y con una frecuencia constante. También desde este esquema se puede entender mejor la figura 5.7 donde se indicaba que el renderizado dependía de hilo de cálculo, ya que el bloque gris (modificación de parámetros) es el nexo entre ambos. Esta conexión puede darse gracias a que los parámetros que se modifican han sido declarados como variables globales y pueden ser accesibles desde cualquier parte del código.

5.1.3. *HILO DE COMUNICACIONES*

El hilo de comunicaciones será el encargado de recibir la información externa. Esta comunicación se realizará mediante comunicación mediante el protocolo tcp ip por su fácil implementación. Dada la aplicación de la que se trata no se realizara ningún tipo de cifrado por lo que cualquiera que esté conectado en la misma red puede ver los datos que se transmiten. Para explicar el funcionamiento global este hilo se hará una breve descripción siguiendo el diagrama de flujo de la figura 5.12.

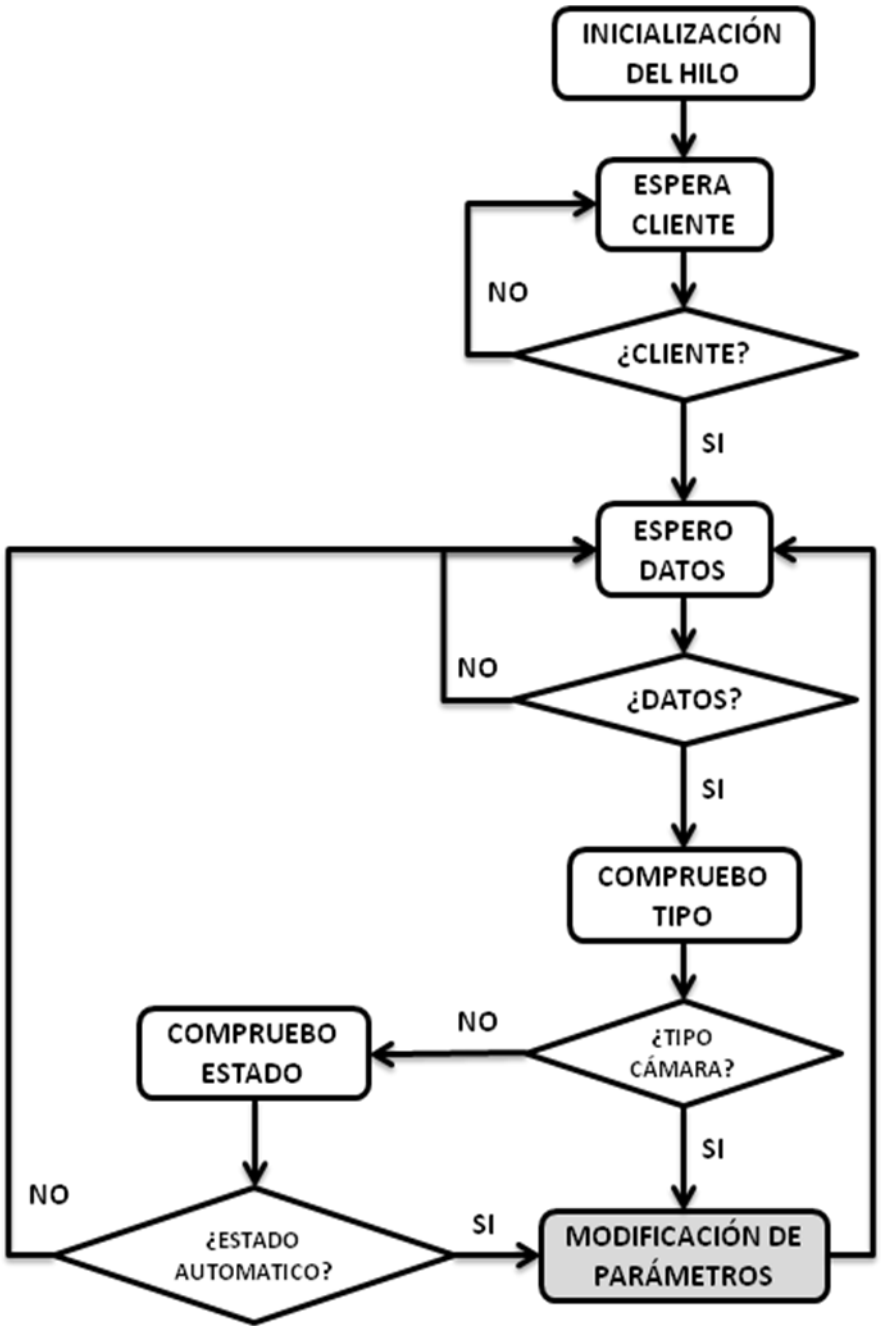

Figura 5.12. Diagrama de flujo de hilo de comunicaciones

Al igual que en los otros dos casos el primer paso es la inicialización y lanzamiento del hilo, en ese momento entrara en un bucle infinito esperando la conexión por parte de un cliente. Para evitar consumir recursos innecesarios esta comprobación se realizara cada 200ms.

Si al servidor del simulador mecánico le solicitan una conexión la aceptará y empezará a comprobar los datos que recibe. Estos datos solo podrán ser de dos tipos, movimiento de cámara o control de motor. En el caso de que sean de movimiento de cámara se modificarán los parámetros correspondientes para que en el próximo renderizado la cámara se sitúe donde le corresponda. Si el dato no es del tipo anterior quiere decir que se pretende realizar una acción de control en los motores, para ello se ha de comprobar si el estado del simulador esta en automático o manual. Si esta en automático implica que el lazo de control lo realiza un software externo y que se desean modificar los parámetros.

Cuando el simulador mecánico detecta la conexión por parte de un cliente conmuta a estado automático para permitirle al cliente el control del sistema, de todas formas se puede cambiar a estado manual aun con el cliente conectado para que el usuario recupere el control del sistema.

El protocolo de comunicaciones seguido por el simulador mecánico se comentará detalladamente más adelante en el simulador electrónico, ya que ahí se podrá entender mejor la lógica seguida para la realización de este protocolo. Cuando se habla de protocolo de comunicación, se refiere a como traducirán las cadenas de caracteres recibidas o como se implementarán las enviadas. Nada tiene que ver por lo tanto con la selección de un protocolo tcp ip para trasmitir tales cadenas.

5.1.4. *EVENTOS*

Los eventos son acciones que no se sabe cuándo tendrán lugar. En el simulador mecánico se han implementado varios eventos de teclado, la tabla 5.2 mostrada a continuación resume estos eventos y a que tecla están asociados.

TECLA	USO
^	Rota cámara hacia arriba (1)
v	Rota cámara hacia abajo (1)
<	Rota cámara hacia la izquierda (1)
>	Rota cámara hacia la derecha (1)
+	Acerca cámara hacia el modelo (1)
-	Aleja cámara del modelo (1)
1	Aumenta carga aplicada en 100 gramos (1)
2	Disminuye carga aplicada en 100 gramos (1)
4	Aumenta tensión aplicada al motor de rotación del husillo en 1 voltio (1)
5	Disminuye tensión aplicada al motor de rotación del husillo en 1 voltio (1)
7	Aumenta ángulo del sistema de traslación del husillo en 5 grados (1)
8	Disminuye ángulo del sistema de traslación del husillo en 5 grados (1)
A	Conmuta entre estado automático y manual

(1) se repite la ejecución si se mantiene la tecla pulsada

Tabla 5.2. Resumen de eventos

Todos los eventos descritos lo único que hacen es cambiar una variable que posteriormente será tratada. Da igual cuando ocurran los eventos ya que lo que interesa es saber el valor de los parámetros en un momento determinado y ese chequeo siempre se hace en el mismo lugar para establecer las condiciones de trabajo en ese instante. Por esta razón en los diagramas de flujo descritos con anterioridad se indicaba los eventos actuando sobre el bloque "modificación de parámetros".

Gracias a los eventos del teclado numérico y al de conmutación entre estado automático y manual se puede dotar al sistema mecánico de una independencia propia. El dibujo de la figura 5.13 pretende representar de forma visual como afectan los eventos al simulador y como se integran con el resto del programa.

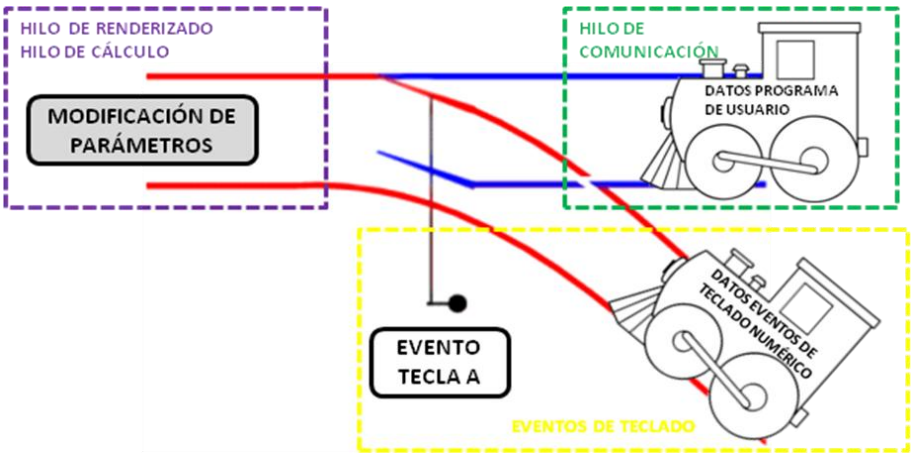

Figura 5.13. Integración de eventos con los hilos

5.2. Simulador electrónico

El simulador electrónico pretende ser una interfaz grafica que el usuario pueda utilizar para ver los datos de los sensores, además se realizará de tal forma que se puedan variar parámetros de forma fácil y ver cómo cambian los datos obtenidos respecto a la configuración anterior. Para realizar esta parte del proyecto se utilizará Qt desginer, esto implica que se cambie el entorno de trabajo y se pase a usar Qt creator para la implementación del código.

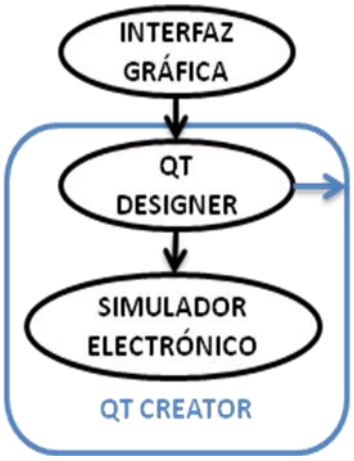

Figura 5.14. Herramientas usadas en el simulador electrónico

Qt creator es un entorno de desarrollo que además tiene sus propias librerías y una forma propia de trabajo. En la introducción de este apartado se hico referencia a la programación orientada a eventos y cómo funcionaba, para entender como gestiona Qt los eventos hay que darle una vuelta de tuerca más, el esquema 5.15 muestra su forma de trabajo.

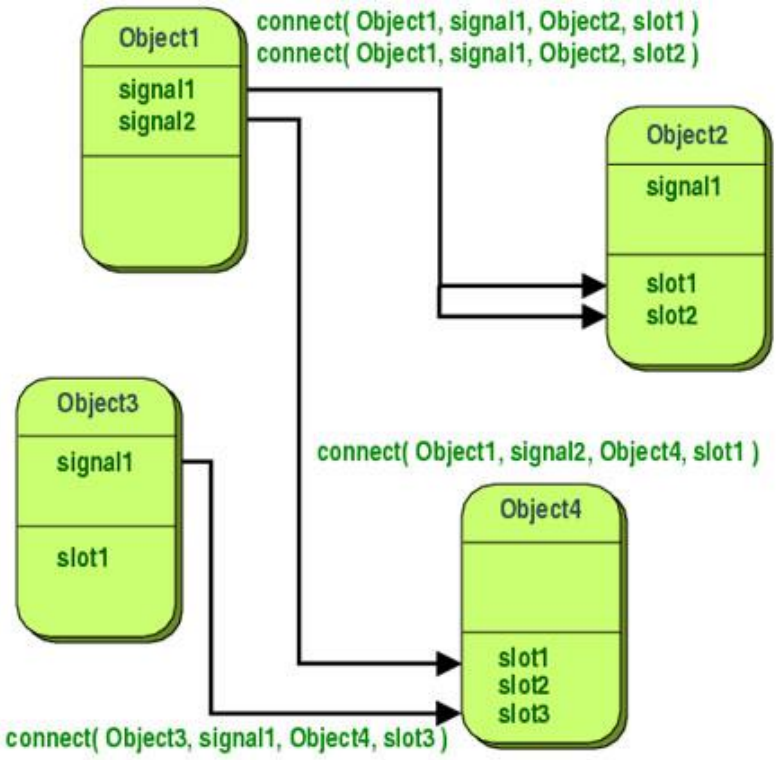

Figura 5.15. Funcionamiento de signals y slots en Qt

Para entender el esquema anterior tómese una señal (signal) como sinónimo de evento y un slot como sinónimo de función. Las señales y los slots se usan para las comunicaciones entre los objetos del interfaz. En general, la señal parte de un objeto "emisor" y llega a un objeto "receptor". El objeto receptor decide si ejecuta un slot y finaliza el proceso, o emite una nueva señal que propaga el evento hacia otro objeto receptor. El proceso se puede repetir. En la figura anterior se ve como un objeto puede emitir varias señales y como una misma señal puede ser recibida por diferentes slot. La relación entre señales y slot puede ser tan complicada como se quiera, es decir, una señal puede ser recibida por varios slot del mismo o distinto objeto, un slot puede recibir la misma señal de diferentes objetos, una señal puede emitir otra señal etc. Gracias a la forma en la que Qt gestiona los eventos, hace que la implementación de estos sea bastante sencilla en comparación a como se haría con la programación tradicional.

Si en el simulador mecánico se pretendían simular los sistemas mecánicos, resulta lógico que en simulador electrónico se simulen los componentes electrónicos. La figura 5.16 muestra una estructura básica de lo que se va a simular.

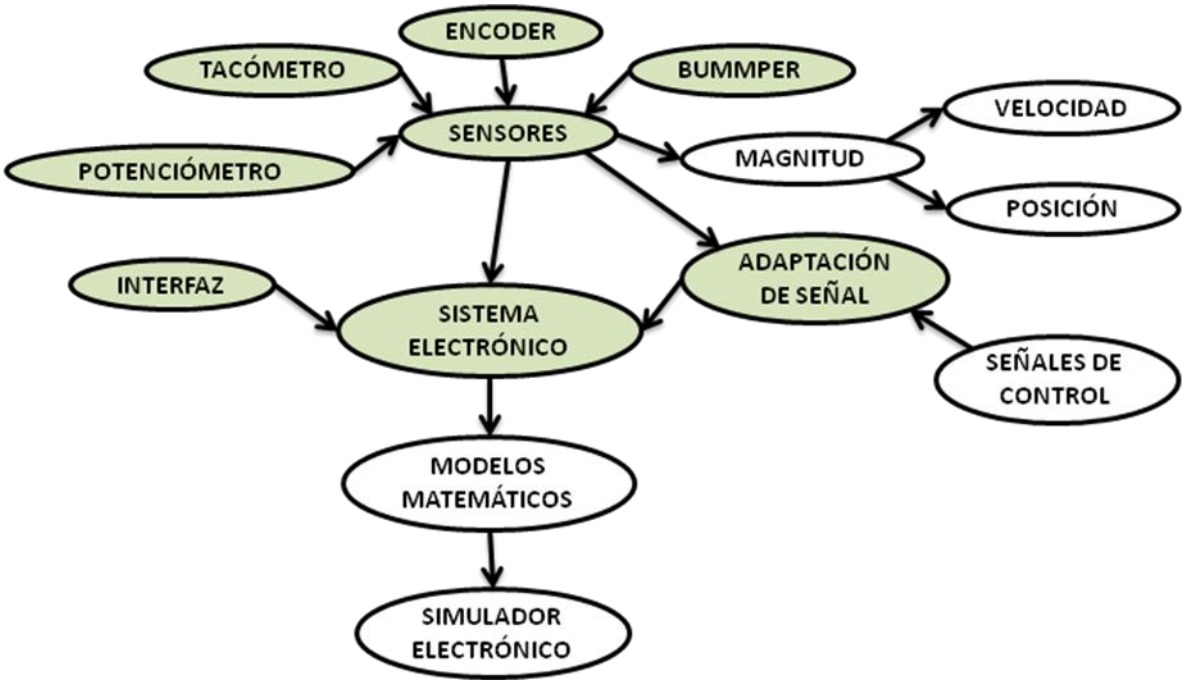

Figura 5.16. Componentes del sistema electrónico

Los elementos sombreados en verde son aquellos elementos físicos que se encuentran en la parte electrónica del sistema real.

El sistema electrónico está compuesto por diferentes elementos entre los que se encuentran la interfaz y los sensores (tacómetro, potenciómetro, encoder y bummper), estos serán utilizados para saber la posición y velocidad de algunos elementos mecánicos. Otro de los sistemas que componen la electrónica del prototipo real es el bloque de adaptación de señales. Este bloque es el encargado de adaptar las señales al rango de trabajo del dispositivo de lectura. En este caso también se simulará la adaptación necesaria en las señales de control de los motores. Todos estos sistemas tienen su modelo matemático que habrá que implementar para poder realizar correctamente el simulador electrónico.

Para entender cómo funciona el código y como se realizado la implementación de los elementos del esquema anterior, es necesario saber cómo es la estructura jerárquica de los diferentes conjuntos que componen el simulador.

De mayor a menor rango se tiene:

- Interfaz — Aplicación principal
  - Ventanas — Cada una de las ventanas que forman la aplicación
    - Widgets — Bloque con función propia e independiente dentro del simulador
      - Circuitos — Sensor mas adaptación que forman un circuito independiente
        - Bloques — Señal he indicador, ya sea un sensor a una adaptación
          - Configuración — Ventanas de configuración asociadas a un bloque en concreto

La tabla que se muestra a continuación se ciñe a como se ha organizado realmente el código, con lo que cada elemento corresponde a una librería independiente.

CONJUNTO	ELEMENTO	CONTIENE
Interfaz	main	Graficas y Simulador
Ventanas	Simulador	Camara, Lcd, Comunicaion, Etapa y Graficas
	Graficas	qcustomplot (1)
Widgets	Camara	
	Lcd	
	Comunicacion	
	Etapa	CircuitoInputPot, CircuitoInputTaco, CircuitoOutput y BloqueFdc
Circuitos	CircuitoInputPot	BloquePot, BloqueAo y Bloque Adc
	CircuitoInputTaco	BloqueTaco, BloqueAo y Bloque Adc
	CircuitoOutput	BloquePwm, BloqueDac, BloqueDig y BloqueAo
Bloque	BloquePot	ConfigPot
	BloqueTaco	ConfigTaco
	BloqueEnco	ConfigEnco
	BloqueFdc	
	BloquePwm	ConfigPwm
	BloqueDac	ConfigDac
	BloqueDig	ConfigDig
	BloaqueAdc	ConfigAdc
	BloqueAo	ConfigAo
Configuración	ConfigPot	
	ConfigTaco	
	ConfigEnco	
	ConfigPwm	
	ConfigDac	
	ConfigDig	
	ConfigAdc	
	ConfigAo	

(1) Librería no oficial de Qt necesaria para la implementación de gráficas

Tabla 5.3. Distribución de los elementos implementados

En la explicación de los modelos matemáticos de los componentes electrónicos se comentó que el funcionamiento de todo ellos es el mismo, es decir, todos convierten un tipo de variable en otra con de diferentes unidades. Esto hace pensar que la implementación de los elementos electrónicos podría hacerse creando varios objetos de la misma clase, en cierto modo este planteamiento es correcto. En este caso se ha ido más allá y se han tenido en cuenta los futuras modificaciones que se hagan del software, es decir, aunque el funcionamiento básico del potenciómetro y de un tacómetro sea el mismo puede que en el futuro se desee completar sus modelos haciéndolos más complejos. Gracias a tener varios bloques diferenciados (aunque en este simulador funcionen de la misma forma) se podrá sustituir fácilmente por otro más complejo sin afectar a los demás.

Las jerarquías y los elementos del simulador electrónico pueden verse en la figura 5.17 donde además se muestra el aspecto del interfaz con el que interactuara el usuario.

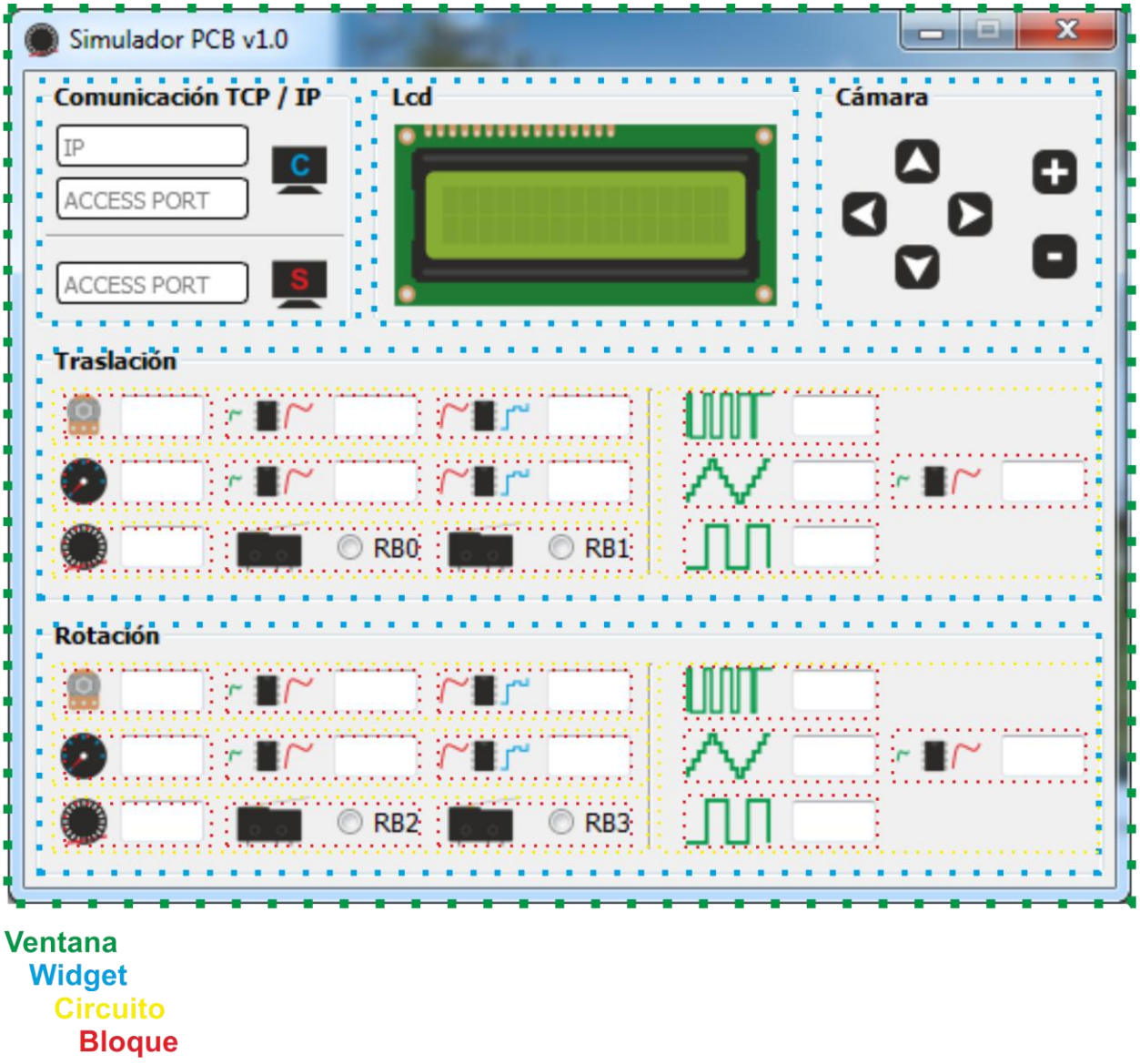

Figura 5.17. División jerárquica del interfaz de usuario

Para comunicar todos los bloques entre sí podrían utilizar variables globales, si se hiciera de esta forma, se tendría que implementar algún tipo de temporización para decir al software que chequee si se han modificado las variables. Resulta mucho más fácil y útil usar el sistema de señales y slots que ofrece Qt. El código se ha implementado de tal forma que los elementos de más rango (padres) estarán conectados con las señales de menos rango (hijos) y viceversa. Gracias a esto el flujo de información es bidireccional, es decir, puede ir aguas arriba o aguas abajo en la jerarquía de parentescos, de tal forma que si ocurre un evento en un bloque de menos jerarquía se desencadenara un torrente de señales que permita al elemento de más rango saber que algo ha sucedido. Parece ilógico que haya este torrente de señales pudiendo conectar directamente los elementos de menos peso jerárquico al de mayor peso, aunque tiene sus ventajas. La principal ventaja es poder implementar el código de una forma muy modular, si cada elemento contiene unas señales y slots básico, podrán interactuar entre ellos de forma fácil.

Para explicar cómo están interconectados los diferentes bloques resulta útil ver el ejemplo de la siguiente figura.

OBJETO	SEÑAL	SLOT	CONEXIONES	
			AGUAS ARRIBA	AGUAS ABAJO
OBJ_X	signal_X	slot_X	connect(OBJ_A(), signal_A(), OBJ_X(), slot_X()) connect(OBJ_B(), signal_B(), OBJ_X(), slot_X())	
OBJ_A	signal_A	slot_A	connect(OBJ_2(), signal_2(), OBJ_A(), signal_A()) connect(OBJ_1(), signal_1(), OBJ_A(), signal_A())	connect(parent(), signal_X(), OBJ_A(), signal_A())
OBJ_B	signal_B	slot_B	connect(OBJ_1(), signal_1(), OBJ_B(), signal_B()) connect(OBJ_2(), signal_2(), OBJ_B(), signal_B())	connect(parent(), signal_X(), OBJ_B(), signal_B())
OBJ_1	signal_1	slot_1		connect(parent(), signal_X(), OBJ_1(), slot_1())
OBJ_2	signal_2	slot_2		connect(parent(), signal_X(), OBJ_2(), slot_2())

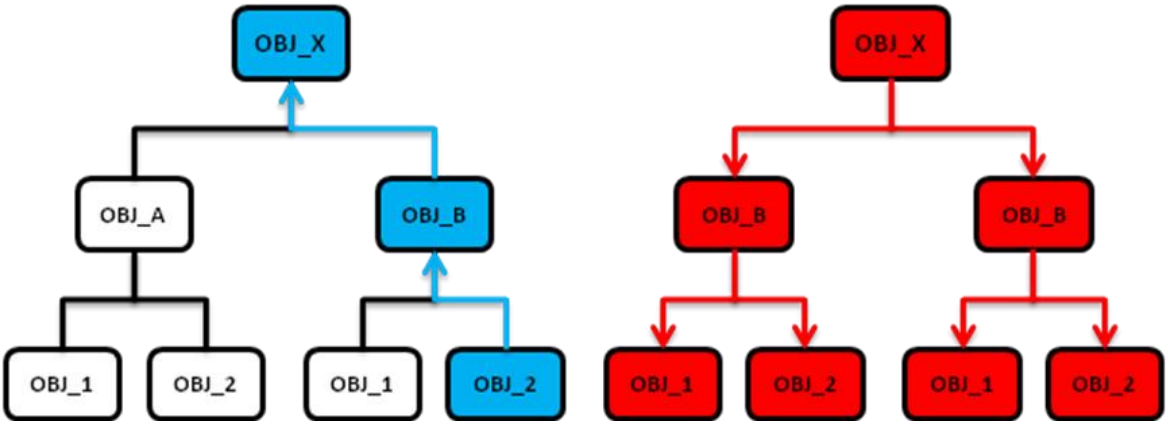

Figura 5.18. Ejemplo de funcionamiento signal y slots

En este ejemplo hay conexiones directas entre señales, con lo que si un evento ocurre en el bloque OBJ_2 desencadena una reacción en cadena de tal forma que el bloque de mayor rango, OBJ_X en este ejemplo, se enterará de tal hecho. El otro caso es que el evento ocurra en el bloque OBJ_X, al igual que en el caso anterior la relación entre señales es directa (una señal emite otra señal) por lo que todos los elementos de menor rango se enterarán de que algo ha ocurrido aguas arriba.

Algo que no se ha mencionado es que las señales emitidas pueden contener información que podrá ser utilizada por el slot del receptor. En este programa en concreto resulta de especial interés poder rastrear que elemento desencadenó el torrente de señales. Realmente, la lógica seguida para la implementación del código del simulador electrónico, es exactamente la misma que en el ejemplo descrito con una salvedad. La diferencia reside en que en este caso las señales llevan información y además una señal no emite otra sino que pasan por un slot intermedio donde se agrega nueva información. La información que se agrega es un sello identificativo del tipo de elemento por el que pasa. Gracias a esto se podrá tener un registro del camino seguido por el evento y saber quien ha sido el desencadénate y hacia donde hay que enviar la señal. La utilidad de todo esto se verá a continuación, pero antes se ha de explicar cómo se añaden los sellos identificativos y que protocolo siguen.

La figura 5.19 muestra esquemáticamente la diferencia entre el funcionamiento del ejemplo de la figura 5.18 y la implementación real del código.

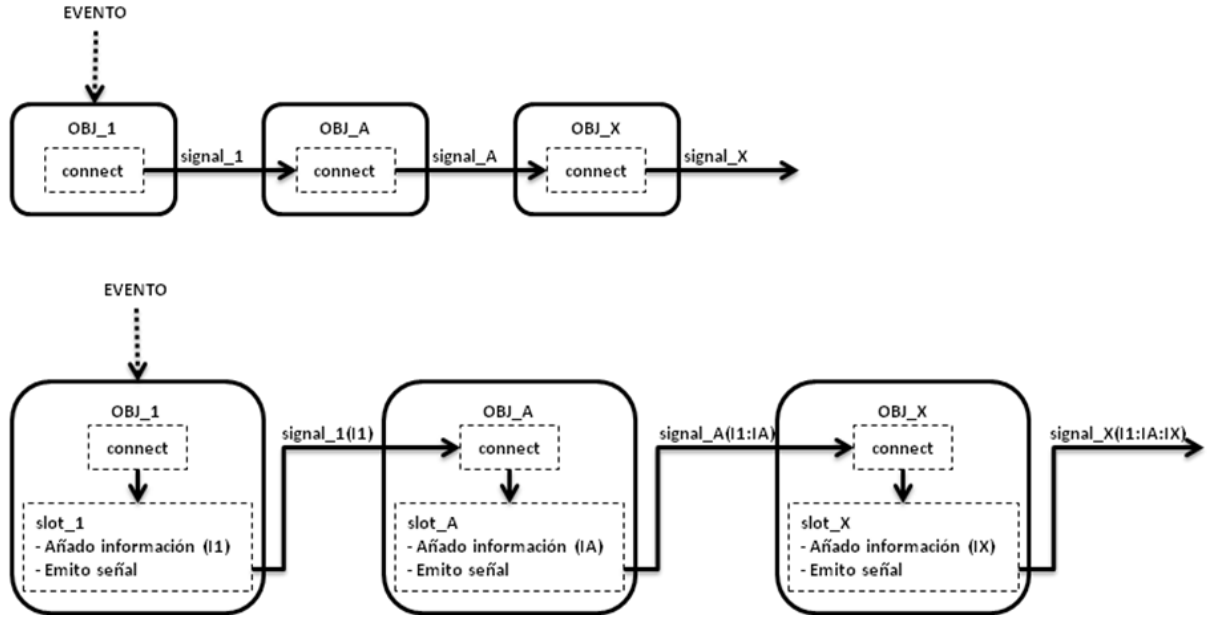

Figura 5.19. Diferencia entre conexión directa e indirecta de señales

La forma en la que se añadirán los sellos identificativos de cada bloque seguirá un protocolo concreto, además ese mismo protocolo se utilizará tanto para el rastreo de los eventos como para la comunicación entre el simulador mecánico y la librería de usuario. La tabla 5.4 resume el grupo de caracteres que se irán añadiendo mientras la señal avanza aguas arriba, en la tabla 5.5 se explica el significado de los caracteres. Nótese que algunas de las instrucciones en realidad no se añadirán sino que es el desarrollador quien las ha de incluir en su código para comunicarse con un bloque en concreto.

	WIDGET	CIRCUITO	BLOQUE	TIPO	VALOR
ETAPA	T-> R->	P: T: E: F: O:	--- PWM: DIG: DAC:	D: S:	*valores numéricos separados por comas*
LCD	LCD->	---	---	XY: TXT:	*valores numéricos separados por comas* *cualquier tipo de carácter*
CÁMARA	---	---	---	CAM:	arrow_up arrow_down arrow_left arrow_right

Tabla 5.4. Protocolo de comunicación

INSTRUCCIÓN	DESCRIPCIÓN
T->	Indicador añadido en la etapa de traslación
R->	Indicador añadido en la etapa de rotación
P:	Indicador añadido en el circuito donde se encuentra el potenciómetro
T:	Indicador añadido en el circuito donde se encuentra el tacómetro
E:	Indicador añadido por el encoder (no tiene circuito de adaptación)
F:	Indicador a añadir para establecer el estado de los finales de carrera
O:	Indicador añadido en el circuito de salida
PWM:	Indicador añadido por el bloque de señal PWM del circuito de salida
DIG:	Indicador añadido por el bloque de señal digital del circuito de salida
DAC:	Indicador añadido por el bloque de conversión digital analógico del circuito de salida
S:	Indicador de que el valor que le precede es para realizar una configuración
D:	Indicador de que el valor que le precede es para realizar una operación matemática
LCD->	Indicador a añadir para que la información se derive al widget de la pantalla LCD
XY:	Indicador a añadir para posicionar el texto que se desea a escribir
TXT:	Indicador a añadir para escribir en la pantalla LCD
CAM:	Indicador añadido por el widget cámara
arrow_up	Indicador añadido al pulsar la teda de posicionar mover cámara hacia arriba
arrow_down	Indicador añadido al pulsar la teda de posicionar mover cámara hacia abajo
arrow_left	Indicador añadido al pulsar la teda de posicionar mover cámara hacia la izquierda
arrow_right	Indicador añadido al pulsar la teda de posicionar mover cámara hacia la derecha

Tabla 5.5. Descripción de indicadores

Ahora que se conoce la lógica seguida para la implementación del código y el protocolo de comunicación (ya sea entre elementos del simulador o del simulador con programas externos) se puede ver como se ha utilizado. Uno de estos usos es la comunicación entre bloques de diferentes widgets para establecer una misma configuración, se entenderá mejor con el siguiente ejemplo. Supóngase un microcontrolador pic como dispositivo para la lectura de señales analógicas, al configurar el módulo AD se tiene que indicar con cuantos bits de resolución ha de trabajar y cuál será la tensión que tomará como referencia. Esta configuración será la misma para cada uno de los pines de entrada, por lo que si se establece que la lectura se hará con 10 bits, todas las señales entrantes se leerán con esa resolución. El simulador electrónico tendrá varias señales analógicas, correspondientes a los potenciómetros y los tacómetros, por lo que resultará de interés que si se configura un módulo AD, los demás tomen los mismos valores sin necesidad de que el usuario realice tal configuración en cada pin de entrada. Gracias a una adecuada conexión entre señales y slots se consigue enviar la información aguas arriba hasta el elemento padre que tengan en común. Todavía queda realizar el camino de vuelta y aquí es donde toma importancia haber implementado los indicadores para rastrear el elemento donde se dio el evento. En el camino de vuelta las señales estarán conectadas a un slot que se encarga de verificar que la información es correcta, es decir, si se está en un elemento de tipo Etapa pero la cadena que le llega es CAM:arrow_up la señal no será emitida aguas abajo ya que esta que esa información no es para él.

Como ya se ha dicho, el protocolo de comunicación entre el simulador electrónico y el simulador mecánico o entre el simulador electrónico y la librería de usuario seguirá esta lógica. El elemento encardado de realizar las comunicaciones es el widget Comunicacion. El simulador electrónico, al contrario que el mecánico, actuará como cliente (en la comunicación con el simulador mecánico) y como servidor (para establecer comunicación con la librería de usuario).

Volviendo a la figura 5.16 donde se indicaban los sistemas electrónicos que se iban a implementar en este simulador, se puede observar como cada uno de ellos corresponde con alguno de los elementos que aparecen en la tabla 5.3. En el manual de usuario adjunto en el anexo *06 - Manual de usuario* se detalla cuales son los parámetros que se pueden configurar de los sistemas electrónicos y entre que rangos. Con el manual de usuario y habiendo entendido las jerarquías y la conexión entre señales y slots, se puede disponer de una visión muy amplia de cómo funciona el simulador electrónico e incluso de entender el código adjunto en el anexo *04 - Código simulador electrónico*.

La frecuencia con la que se actualizarán los datos obtenidos por los sensores dependerá del simulador mecánico y no del simulador electrónico, ya que el primero es el que envía la nueva información cuando dispone de ella. Esta circunstancia afecta también al módulo encargado de realizar las graficas correspondientes a los parámetros de corriente velocidad y par de los motores, motivo por el cual en un gran número de ocasiones se dé el caso de no disponer de la suficiente información para realizar un graficado correcto. La imagen de la figura 5.20 muestra la problemática de tener pocos datos para la representación de los parámetros.

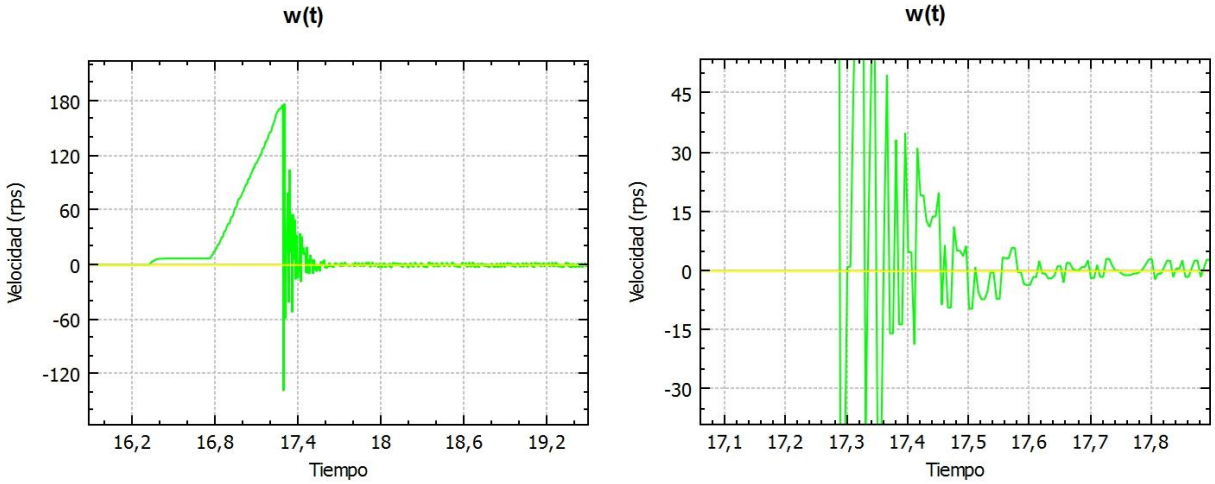

Figura 5.20. Gráficas de velocidad

En la imagen de la izquierda se ve una onda subamortiguada de la cual se pueden sacar conclusiones como que ha habido algún tipo de rebote de la tuerca ya que toma velocidades positivas y negativas. Si se quisiera sacar el modelo matemático o parámetros de configuración (algo realmente interesante en asignaturas de regulación y control) no se podría ya que al ampliar la imagen se observan incoherencias de las que se deducen que se está produciendo aliasing. En esta versión del simulador electrónico las graficas obtenidas no podrán utilizarse para sacar conclusiones sobre como es el sistema pero si de cómo funciona.

5.3. *Librería de usuario*

Se desea que el interfaz de los programas de control a realizar por los usuarios sea el mismo para el sistema simulado (objeto del presente proyecto) y el sistema real (objeto del proyecto complementario). Esto permitirá que múltiples usuarios trabajen sobre el sistema simulado para realizar sus diseños y programas, y que puedan después validarlo en el sistema real sin cambios en su desarrollo.

Para ello, tanto el sistema simulado como el sistema real deben ofrecer las mismas funcionalidades para interacción con ellos, donde la interacción únicamente podrá consistir en:

- Leer datos de sensores.
- Establecer valores en accionadores.
- Escribir en pantalla LCD auxiliar.

La siguiente figura muestra como interactuará la librería de usuario con el presente proyecto y el complementario.

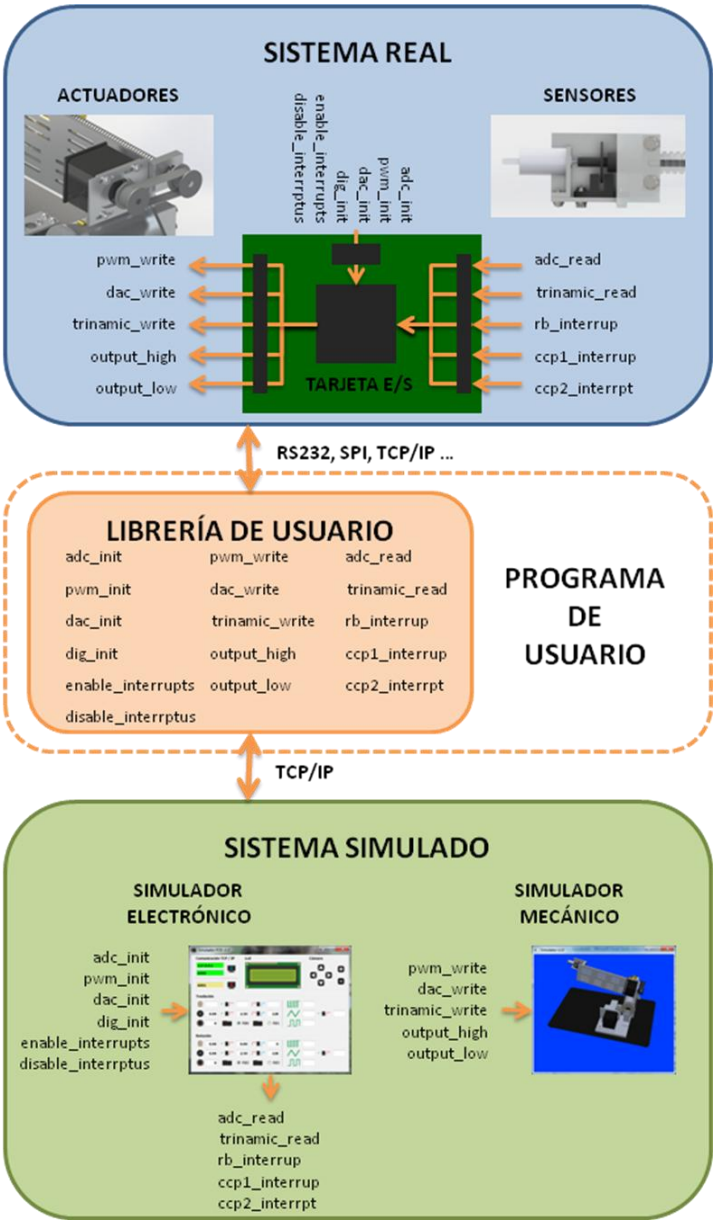

Figura 5.21. Parte común entre el sistema real y el simulado.

La figura anterior muestra con que bloque (software o hardware) actúa cada una de las diferentes funciones implementadas en la librería de usuario. Esta librería ha de ser llamada por el propio usuario cuando se desee realizar un código con el que controlar el sistema mecánico, además se comunicará con el simulador o el sistema real para realizar la transmisión de datos leídos o escritos por el usuario. La configuración interna de sistema real y el simulado puede diferir en algunos aspectos, por lo que habrá funciones que sean diferentes, de todas formas la librería de usuario ha de tener en ambos casos las siguientes funcionalidades:

- Funciones de configuración:

```
void enable_interrupts(int flags);
void disable_interrupts(int flags);
void adc_init(int bits, float ref);
void pwm_init(int f, int bits, float vout);
void dac_init(int bits, float vout);
void dig_init(float vout);
```

- Funciones de lectura de datos

```
double adc_read(int canal);
char *trinamic_read(int motor);
```

- Funciones de escritura de datos

```
void pwm_write(float bits,int canal);
void dac_write(int bits,int canal);
void trinamic_write(char *cadena);
void output_high(int pin);
void output_low(int pin);
```

- Interrupciones

```
void ccp1_interrupt();
void ccp2_interrupt();
void rb_interrupt();
```

Algunos de los parámetros que se pasan a la función solo serán necesarios en él la librería del simulador ya que se realizó de tal forma que pueda utilizarse en dispositivos que trabajen con tensiones de salida de 5 voltios y de 3,3 voltios, razón por la cual se ha añadido en algunas funciones el parámetro `float vout`.

5.3.1. *IMPLEMENTACIÓN EN EL SIMULADOR*

Para la implementación de la librería usuario con el simulador, se ha tenido en cuenta el funcionamiento de un microcontrolador pic y las librerías de CCS utilizadas para su programación. Se ha intentado, en la medida de lo posible, realizar esta librería de tal forma que el funcionamiento de las funciones implementadas no fuese muy diferente a como trabajan las funciones de CCS. Para conseguir mayor versatilidad de la librería y poder hacer conectarlo con el sistema real se han añadido varias funciones auxiliares, como por ejemplo las referentes al motor Trinamic (En esta versión no se ha implementado el código correspondiente, pero queda reflejada su existencia en las cabeceras para futuros desarrollos)

La librería de usuario tiene que poder comunicarse tanto con el simulador electrónico para pedir los datos de los sensores, como con el simulador mecánico para enviar datos a los motores. Para realizar esta comunicación se ha necesitado usar de nuevo la librería Portable_socket. Las funciones que se comunican con los simuladores siguen el protocolo de indicadores explicado en el punto anterior, de esta forma se puede direccionar que sensor en concreto se quiere leer o a que motor se le quiere enviar una señal.

Además de las funciones para el envío y recepción de datos también se han implementado varias interrupciones. Una de ellas es una interrupción externa que se dará cuando algún final de carrera se active o se desactive. Para simular esta circunstancia se lanza un hilo (en caso de que sea habilitada la interrupción) que pedirá información cada cierto tiempo al simulador electrónico. Otra de las interrupciones añadidas son las de modulo de comparación y captura. Su funcionamiento se basa en la cuenta de pulsos hasta llegar a un numero especificado, momento en el cual se lanzara la interrupción. Este modulo ha sido simulado para contar los pulsos de los encoder. Al igual que en el caso anterior es necesario lanzar un hilo que pregunte al simulador el numero de pulsos.

El numero de hilos que tendrá el programa que realice el usuario dependerá de cuentas interrupciones habilite. Para realizar estos hilos ha sido necesario utilizar una nueva librería llamada TinyThread [11]. Puede resultar curioso que no se utilice la biblioteca boost ya usada con anterioridad en los hilos del simulador mecánico, esto se debe a que no se quiere que el usuario necesite tener instalado elementos externos para dotar al sistema de una mayor independencia

5.4. *Software adicional*

Durante el desarrollo del proyecto surgieron complicaciones que hubo que subsanar. Para ayudar a encontrar y resolver los problemas se desarrollaron aplicaciones de depuración de código de los que se pueden destacar dos.

5.4.1. *SOFTWARE DE DEPURACIÓN GLOBAL*

En la figura 5.22 se muestra el software de depuración utilizado para el desarrollo de los simuladores (incluido *Comprobación de comandos* dentro del anexo *07 - Software de depuración*).

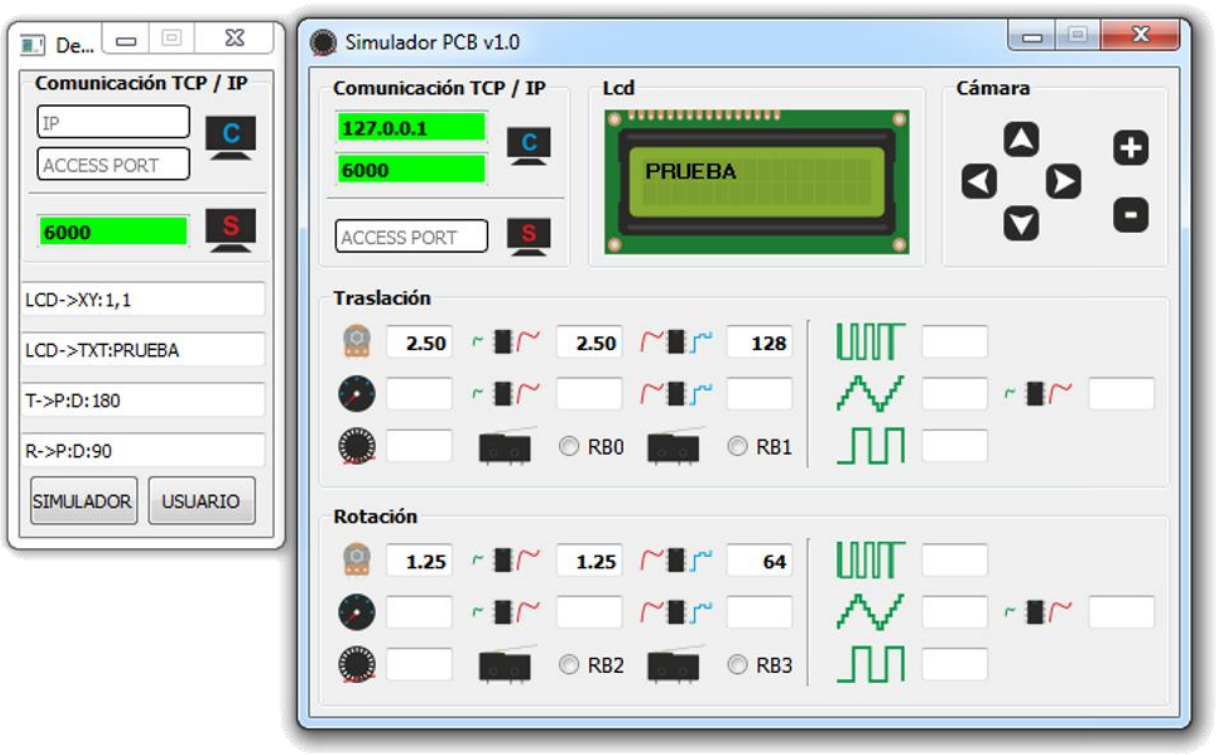

Figura 5.22. Software de depuración global

Se puede observar que el programa de depuración, mostrado a la izquierda de la imagen, contiene el mismo bloque de comunicación. Con esto, queda patente que haber realizado la programación orientada a objetos facilitó el desarrollo del simulador. Su funcionamiento se basa en la introducción de un grupo de indicadores en las líneas de edición de texto y enviarlos al cliente o al servidor, gracias a este software y esta manera de trabajo se ha podido comprobar las siguientes partes:

- Implementación correcta del protocolo de comunicaciones.
- Comprobación del funcionamiento del cliente y el servidor, tanto el simulador mecánico como en el electrónico y en la librería de usuario.
- Comprobación de conexiones entre señales y slots.
- Comprobación de que los modelos matemáticos de los componentes electrónicos funcionases correctamente.

5.4.2. *SOFTWARE DEPURACIÓN DE SEÑALES*

Dado que la aplicación anterior no permite escribir múltiples puntos a la vez se necesito un software adicional para comprobar la parte grafica del simulador electrónico. Su funcionamiento se basa en el mismo bloque de comunicación, aunque en este caso los indicadores para la trasmisión de datos es añadirán automáticamente según se mueva la barra. El código de esta aplicación se encuentra en el anexo *Comprobación de gráficas* dentro del anexo *07 - Software de depuración*. La figura 5.23 muestra su funcionamiento.

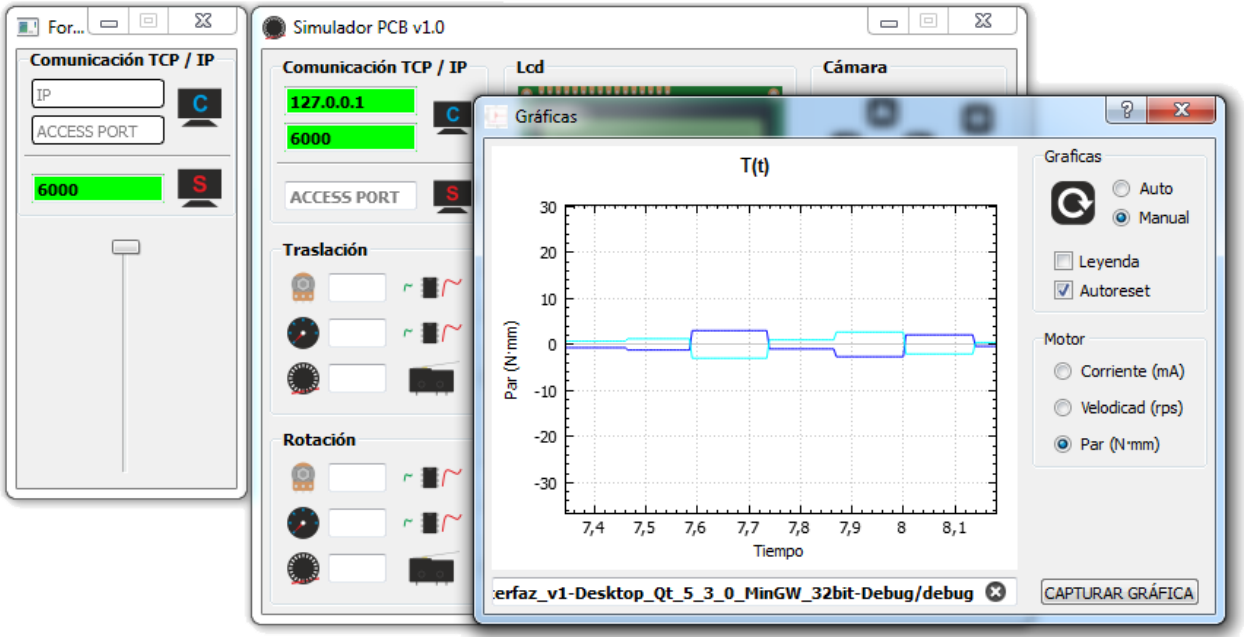

Figura 5.23. Software de depuración de señales

6. CONCLUSIONES Y MEJORAS

6.1. Competencias necesarias

Para la realización de este proyecto se requieren conocimientos sobre algunas materias, que de no tenerlos imposibilitarían la realización del mismo, al menos en un plazo razonable de tiempo.

Dado que el presente proyecto trata de simular un sistema mecatrónico, resulta evidente tener conocimientos avanzados sobre programación. Además es necesario comprender como funcionan los diferentes componentes a simular. Esto permite poder realizar una implementación de sus modelos matemáticos, de tal forma que se puedan detectar incoherencias en el funcionamiento en caso de haberlas.

Para poder realizar un código ordenado y entendible resulta imprescindible saber cómo funciona la programación orientada a objetos, además de ser capaz de desarrollar el código según este planteamiento de trabajo. La programación orientada a objetos es realmente importante en este proyecto, puesto que las librerías de Qt creator utilizan este tipo de programación para realizar las aplicaciones gráficas. Además Qt creator tiene una forma particular para la gestión de eventos que también es necesario comprender.

E simulador está dividido en diferentes partes y estas se tienen que poder comunicar entre sí, por esta razón resulta interesante conocer o aprender como comunicar aplicaciones diferentes entre sí.

Para poder realizar una visualización de un modelo 3D al que dotarle de movimientos hay que conocer los entresijos del funcionamiento de un motor de juegos. No solo valdrá con saber que funciones tienen sus librerías y para qué sirven, sino que además será necesario tener conocimientos sobre que es una luz ambiental, características de los materiales (color, reflexión, difusión…).

Con todo esto se estará en disposición de poder realizar un simulador de mayor o menor calidad pero totalmente funcional.

6.2. Competencias adquiridas

En este caso se ha necesitado adquirir conocimientos avanzados sobre la programación orientada a objetos, además, también ha sido necesario comprender el funcionamiento de los eventos en Qt para poder implementarlos en el simulador electrónico.

Para la visualización del modelo 3D del sistema mecatrónico a simular, no solo se ha necesitado adquirir conocimientos sobre el funcionamiento de un motor de juegos, sino que también se ha tenido que aprender las características de los formatos en los que se guardan los modelos.

Por otra parte, el proyecto no solo es didáctico para el que lo use sino que lo ha sido para el propio proyectante, ya que ha podido observar cómo funcionan los diferentes sistemas (mecánicos y electrónicos) con la implementación de los mismos. Además, ha ayudado al proyectante a conocer algunos tipos de integración numérica.

El sistema a simular consta de diferentes componentes mecánicos y electrónicos. Para la realización de modelo del husillo de bolas se han tenido que consultar las hojas de datos de diferentes fabricantes, lo que ha implicado un aprendizaje del funcionamiento de estos, además de las diferentes características de las tuercas.

Debido a que una gran mayoría de la documentación consultada (manuales, foros, tutoriales…) para realizar el proyecto está escrita en inglés, se ha adquirido vocabulario técnico en este idioma.

6.3. Opciones de mejora

A todo proyecto tiene aspectos a mejorar, ya sea para incluir más funcionalidades o para subsanar deficiencias que no se hayan tenido en cuenta en el diseño etc. En este punto se comentarán algunas de las posibles funciones adicionales que se podrían implementar además de comentar algunos aspectos que no se han conseguido solucionar.

En este proyecto el motor que se ha simulado para el control de los mecanismos móviles ha sido un motor de corriente continua. Resulta interesante la implementación de otro tipo de motores como un motor paso a paso o uno del fabricante trinamic cuya driver de control viene incluida en el propio dispositivo.

Dado que no se disponía del prototipo real construido mientras se realizaba el simulador, no se han podido ajustar los parámetros de los modelos correspondientes a los mecanismos utilizados. Para conseguir una simulación más acorde la realidad han de realizarse estos ajustes, una vez hechos se podría plantear completar los modelos matemáticos de tal forma que el sistema sea aun más abierto. Esta complejidad en los modelos también puede aplicarse en los componentes electrónicos simulando también la respuesta en frecuencia según sea el lugar de las raíces del sistema etc. Además el bloque de adaptación podría sustituirse por varias etapas de amplificación y filtrado donde incluso es pudiera seleccionar la topología de estas.

Este tipo de mejoras podrían considerarse como un proyecto final de grado o master por si solas dependiendo la complejidad a la que se quiera llegar. Aunque se complicaría el sistema también sería mucho más universal y didáctico.

En este proyecto los parámetros han de ser establecidos en un archivo de configuración, por lo que para ver como actúa el sistema con otro tipo de características es necesario reiniciarlo. Como mejora se propone realizar un menú de configuración dentro del propio simulador, lo que permitiría ver en tiempo real como afectan las distintas configuraciones.

Haciendo referencia a aspectos que no se han podido solucionar durante el transcurso del proyecto cabe destacar las visualización del modelo 3D. El modelo visualizado tiene una baja calidad de gráficos y de sombras. La falta de información al respecto ha impedido realizar una visualización del modelo más realista. Por otra parte, no se ha conseguido implementar una línea de texto sobre la pantalla del simulador mecánico, por lo que es necesario la terminal para mostrar los datos de carga, tensión aplicad a los motores etc. cuando se está en estado manual. Resultaría interesante evitar el uso de la terminal para la visualización de estos datos.

Finalmente algo que no se ha podido solucionar dado que no se ha encontrado la fuente de error, es el movimiento residual del husillo cuando no se le aplica tensión alguna al motor. Esta circunstancia apenas es perceptible y solo se da en casos puntuales.

Evidentemente las mejoras a aplicar no quedan solo aquí sino que podrán realizarse tantas como se le ocurran a los futuros desarrolladores.

6.4. Conclusiones

Partiendo de las condiciones iniciales, se puede concluir que se ha conseguido realizar un simulador totalmente funcional cumpliendo los requisitos impuestos. Este simulador puede ser usado, por lo tanto, para la realización de prácticas o como elemento de apoyo en clases teóricas, independientemente de las posibles mejoras que se le puedan implementar para hacer al sistema aun más didáctico.

BIBLIOGRAFÍA

[1] http://www.sbgames.org/papers/sbgames10/computing/short/Computing_short29.pdf

[2] http://ima.udg.es/iiia/GGG/TIC2001-2416-C03-01/docs/EnginesUJI.pdf

[3] https://www.panda3d.org/manual/index.php/Converting_from_Blender

[4] http://www.blender.org/download/

[5] https://code.google.com/p/yabee/downloads/detail?name=YABEE_r13_1_b266.zip&can=2&q=

[6] http://qt-project.org/downloads

[7] http://www.robolabo.etsit.upm.es/asignaturas/seco/apuntes/motor_dc.pdf

[8] http://mahara.uji.es/artefact/file/download.php?file=25300&view=74

[9] https://tech.thk.com/es/products/pdf/es_a15_053.pdf

[10] http://www.boost.org/users/history/version_1_55_0.html

[11] http://tinythreadpp.bitsnbites.eu/