Facultat d'Informàtica de Barcelona (FIB) Universitat: Universitat Politècnica de Catalunya (UPC) BarcelonaTech

Integración de un sensor laser en un robot SCARA y desarrollo de un aplicación de escaneado de superficies con el sistema integrado robot-laser

Autor: Guillermo Sotomayor Millán

Directora : Alicia Casals Gelpi , ESAII

22 de julio del 2011

Título: *Integración de un sensor laser en un robot SCARA y desarrollo de una aplicación de escaneado de superficies con el sistema integrado robot-laser.*

Autor: *Guillermo Sotomayor Millán*

Data: *22 de Julio del 2011*

Directora: *Alicia Casals Gelpi*
Departamento de la directora: *ESAII*

Titulación: *Ingeniería en Informática*

Centro: Facultat de Informática de Barcelona (FIB),*Universidad:* Universitat Politècnica de Catalunya (UPC) BarcelonaTech

Índice

1 Introducción

El objetivo de este proyecto es recuperar un manipulador robótico SCARA, que ha estado en desuso durante bastante tiempo. Además de darle una utilidad a este manipulador y facilitar el trabajo de este manipulador para futuros usuarios dándoles una herramienta para trabajar con el SCARA y documentación sobre su funcionamiento y utilización. Con la realización de este proyecto el SCARA quedará listo para su utilización.

Los manipuladores SCARA normalmente están en cadenas de montaje, en la ubicación actual del SCARA no puede hacer ningún trabajo parecido por lo que hemos buscado una aplicación que se pueda llevar a cabo con su ubicación actual y de también la posibilidad de explorar otros campos de la informática como puede ser la generación de modelos de superficies a partir de una nube de puntos.

Para cumplir este objetivo se pondrá en marcha el manipulador y se creara una interfaz que permita su configuración y su utilización de una forma fácil e intuitiva tanto para un usuario sin conocimientos previos sobre manipuladores de este tipo como para un usuario experimentado, dando a futuros usuarios una herramienta para utilizar el SCARA.

También se creara una aplicación de escaneado de superficies con el SCARA y l ayuda de un sensor de distancia, por lo que será necesario integrar el sensor con el SCARA. Esta aplicación nos permitirá aplicar los conocimientos de modelado y visualización de gráficos obtenidos durante la carrera.

En definitiva separando los objetivos del proyecto por puntos más definidos nos quedaría la siguiente lista de objetivos:

1. Puesta en marcha SCARA YK-640.
2. Diseño y desarrollo de la aplicación para la manipulación del SCARA.
3. Integración del sensor de fuerza y ampliación de la aplicación anterior.
4. Diseño y desarrollo de la aplicación para el escaneado de superficies con el sistema creado.

2 Descripción del proyecto

2.1 Puesta en marcha SCARA YK-640

Para la realización de esta parte del proyecto disponemos del manipulador SCARA mencionado en la introducción. Es un robot de los años ochenta, por lo que para la industria actual ya no es competitivo pero aun se le pueda dar uso en investigación y educación, y un PC. Para empezar el proyecto lo primero que se debe hacer es poner en funcionamiento el SCARA que lleva mucho tiempo sin utilizarse, para ello disponemos inicialmente del propio SCARA con su unidad de control y un PC, conectaremos ambos con un cable serie (RS-232). En cuanto a documentación para llevar a cabo este paso se dispone del manual de usuario del SCARA[1], que al ser de los años ochenta está un poco desfasado respecto a los PC actuales.

Antes de explicar cómo ha sido el proceso de puesta en marcha especificaremos cómo es y cómo funciona el SCARA.

2.1.1 El SCARA YK-640

Los manipuladores SCARA son robots normalmente utilizados en cadenas de montaje realizando tareas repetitivas y que precisen de precisión, como puede ser paletización de productos o soldado de los pines de un chip en una placa base, en definitiva son robots diseñados para la automatización de trabajos de ensamblado.

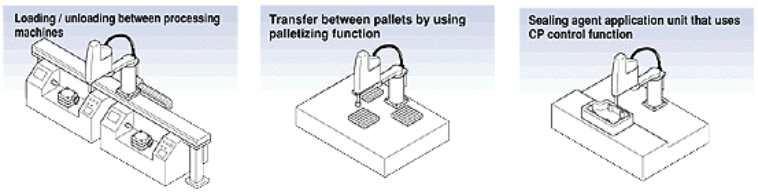


Figura 1- Ejemplos de uso de un SCARA [0]

Las siglas SCARA provienen de *Selective Compliance Assembly Robot Arm*, lo que quiere decir que son manipuladores con una gran movilidad en un plano del espacio pero una gran rigidez en lo que respecta al resto, en este caso el manipulador tiene altas prestaciones en cuanto al movimiento por los ejes X e Y pero no es así sobre el eje Z.

Los parámetros más importantes para especificar un robot son sus grados de libertad y su espacio de trabajo y dimensiones así como la capacidad de sus actuadores. Se puede ver una especificación completa en el manual [1].

Empezaremos por sus grados de libertad y su espacio de trabajo. En cuanto a los grados de libertad del SCARA son 4 que se los dan sus cuatro actuadores, como se pueden ver en los siguientes diagramas. Los cuatro actuadores permiten que el SCARA se desplace por las tres dimensiones del espacio y darle una orientación al terminal. La posición de los actuadores nos da las coordenadas articulares del SCARA, en

este caso como se puede observar en el diagrama la nomenclatura para cada componente de las coordenadas es la letra D seguida de un número (1 - 4) que indica el actuador.

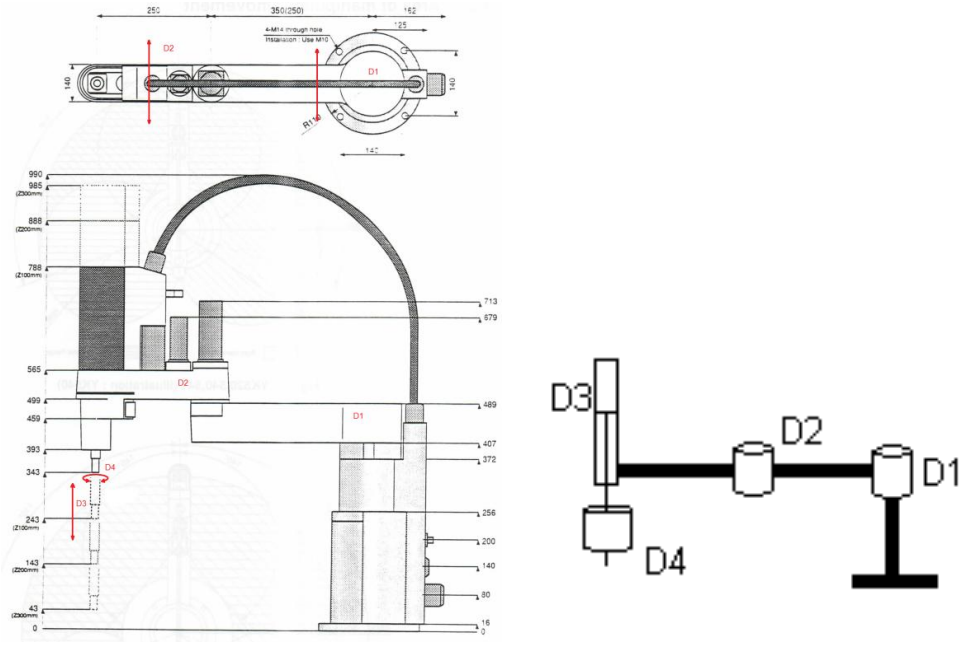


Figura 2 - Esquema actuadores SCARA y diagrama grados de libertad (Manual usuario SCARA YK-640)

Como se puede observar en la figura los actuadores D1, D2 y D4 son actuadores de movimiento angular mientras que el actuador restante D3 es un actuador de movimiento lateral.

Como se puede ver en la figura 2 los motores D1 y D2 se encargan del movimiento del brazo robótico, estos desplazan el terminal sobre el plano XY, el motor D3 se encarga de posicionar el terminal a la altura deseada y el motor D4 se encarga de la orientación del terminal.

En cuanto a la localización del terminal o la posición actual del manipulador, el manipulador utiliza dos opciones. La primera las coordenadas cartesianas del terminal del manipulador y la segunda las coordenadas articulares que se han mencionado más arriba.

En cuanto a las coordenadas cartesianas su origen está centrado en la base del manipulador a la altura del terminal en su posición inicial, como se ve en la figura 3. El desplazamiento será positivo en el sentido de las flechas de la figura. Coordenada X desplazamiento positivo alejándose de la base del robot, coordenada Y desplazamiento positivo hacia la izquierda, y la coordenada Z cuando más cerca este el terminal del suelo más alto será su valor.

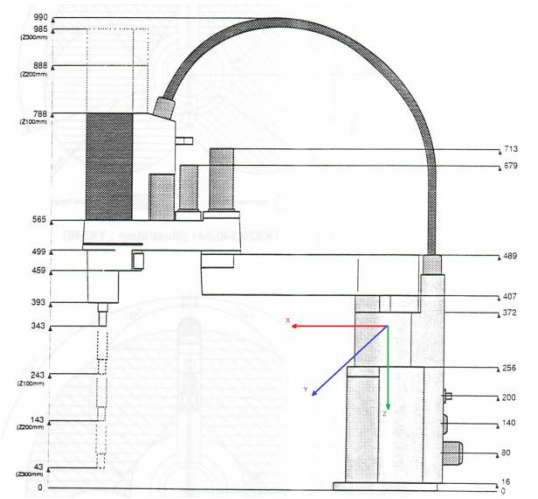


Figura 3 - Origen de los ejes de las coordenadas cartesianas. (Manual usuario SCARA YK-640)

En lo que respecta al espacio de trabajo viene definido por el tamaño del brazo mecánico (figura 2 y 3) y por su rango de movimiento (figura 4).

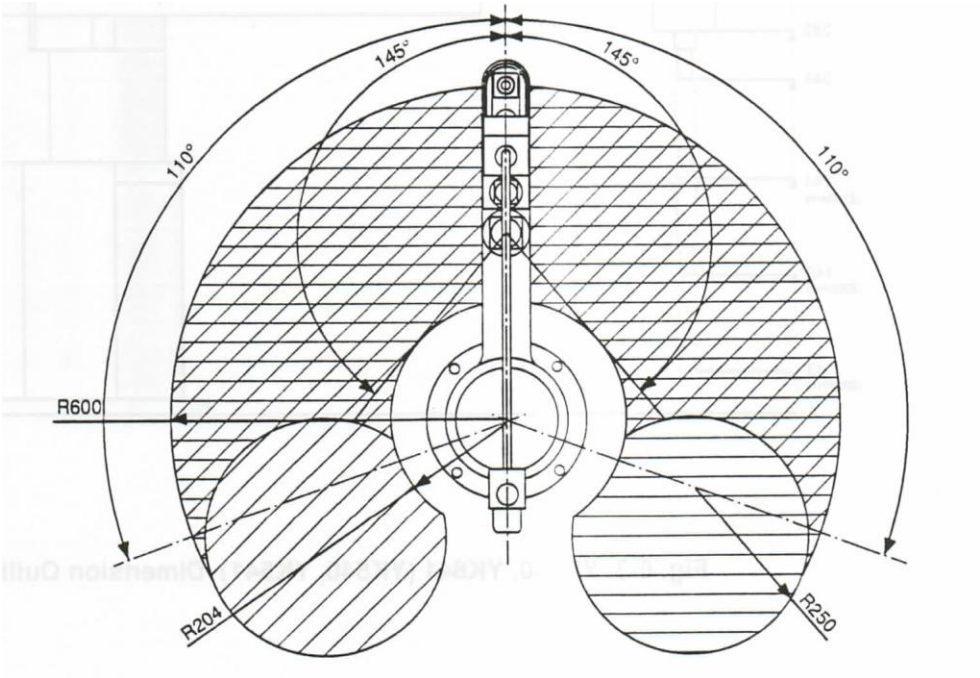


Figura 4 - Espacio de trabajo del SCARA (Manual usuario SCARA YK-640)

Como podemos ver en la figura 4 el actuador D1 tiene una libertad de movimiento de ± 110° desde la posición inicial. Y el actuador D2 de ± 145° desde la posición inicial. Siempre que se trabaje con el manipulador hay que tener en cuenta su espacio de trabajo.

2.1.2 La puesta en marcha

Para poner en funcionamiento el SCARA el primer paso fue conectar el manipulador con el PC. Para lo que hace falta un cable serie (null-modem), para un portátil actual será necesario un adaptador USB a RS-232.

Una vez se ha conectado físicamente el manipulador y el PC nos encontramos con el primer problema. Con tal de que la comunicación entre el manipulador y el PC sea satisfactoria la configuración del puerto serie (Baud rate, paridad, bits de stop,…) debe ser la misma en ambos componentes. El problema radica en que solo con el PC no puedes saber la configuración del manipulador.

Para solucionar este problema se puede recurrir al método de la prueba y error, no muy recomendado ya que el número de posibilidades es bastante elevado.

Otro sistema es conectar al controlador del sistema un terminal MPB (figura 6). Este terminal se conecta directamente al controlador y ya tiene un protocolo de comunicación propio con el controlador. El terminal permite configurar y manipular el SCARA. Con este terminal se puede consultar y modificar la configuración del puerto serie, por lo que se consulto la configuración del puerto serie y se estableció la comunicación.

Una vez se ha establecido la comunicación con el manipulador y se ha comprobado que funciona correctamente solo falta empezar a probar las distintas instrucciones y familiarizarse con el SCARA, lo que no será útil para el resto del proyecto.

También nos ha servido para comprobar que algunas de las instrucciones del manual daban errores no esperados por culpa de algunos parámetros de configuración. Por lo que es necesario reconfigurar algunos parámetros de la configuración del SCARA.

Durante este apartado del proyecto se ha creado un manual de usuario más simplificado y una lista con los errores que más nos hemos encontrado durante la manipulación del manipulador y la solución que se ha aplicado (Apéndices A y B).

2.2 Aplicación consola SCARA YK-640

Antes de empezar con la primera aplicación debemos definir la metodología que utilizaremos para el desarrollo del software de este proyecto. Durante el desarrollo del proyecto seguiremos la metodología UP, me he decidido por esta metodología por su versatilidad a la hora de adaptarse a cualquier tipo de proyecto, y por su sistema de funcionamiento en iteraciones lo que permite ir desarrollando la aplicación por pasos.

2.2.1 Metodología de desarrollo

El diseño se hará siguiendo un proceso unificado de desarrollo que consta de las siguientes etapas planificación, requisitos, especificación y diseño, implementación, pruebas y por último evaluación (figura 5). Este método es iterativo e incremental, por lo se puede hacer tantas iteraciones como se vea necesario hasta conseguir una aplicación que satisfaga todos los requisitos de los usuarios.

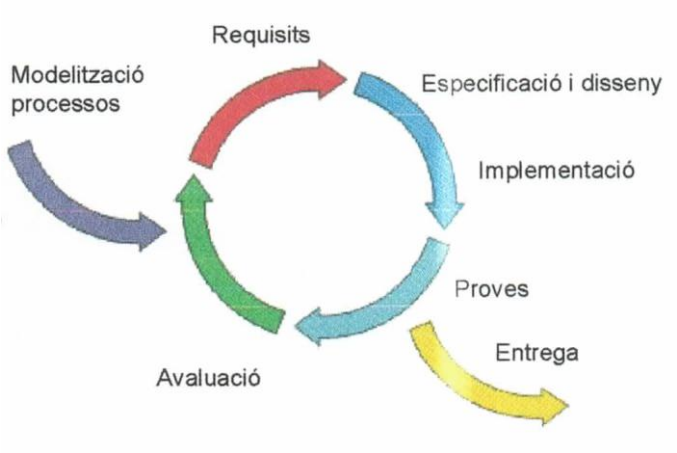


Figura 5 - Etapas UP (Transparencias de clase Ingeniería del Software)

Una vez visto la metodología que utilizaremos para el desarrollo de las aplicaciones del proyecto pasamos a ver el desarrollo de esta primera aplicación.

2.3 La aplicación

La finalidad básica de esta aplicación es crear una interfaz que permita el uso del SCARA desde un PC conectado a este. Puesto que no se tiene ninguna otra forma de utilizarlo hay que crear una aplicación que permita aprovechar todas las funcionalidades que tiene el SCARA. Para ver las necesidades que debe cubrir la aplicación nos hemos basado en las aplicaciones que cubre la consola externa del propio manipulador (figura 6), la cual no se dispone en la universidad y para esta parte del proyecto se ha tenido que conseguir uno externo. En definitiva esta primera aplicación tiene el objetivo de sustituir el terminal externo.

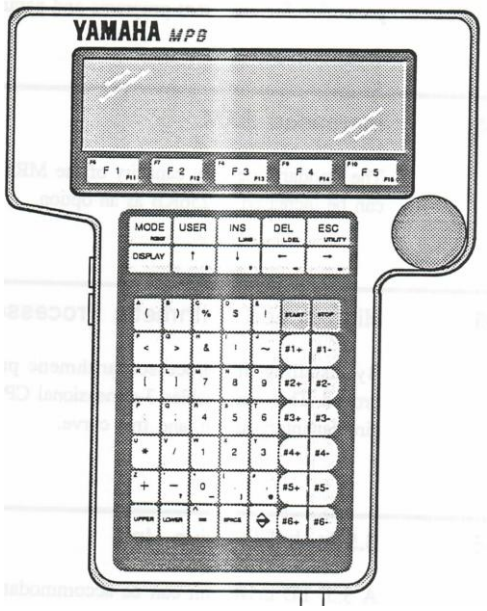


Figura 6 - Terminal programación MPB (Manual de usuario SCARA YK-640)

Como se puede ver en la consola externa la aplicación nos debe permitir mover el manipulador. Ya sea haciendo un movimiento sobre las coordenadas cartesianas, en que el propio controlador calculara la posición de cada actuador. O definiendo la posición de cada actuador por separado.

Este terminal también nos permite la gestión de archivos del controlador, estos pueden ser archivos de configuración, de posición y programas principalmente. Se pueden generar nuevos programas, ejecutar programas existentes y eliminar programas. Así como configurar todos los parámetros del robot y del controlador, como velocidad, modo de funcionamiento.

El terminal muestra constantemente la posición del terminal y los mensajes que producen los errores que se produzcan en el SCARA, esta utilidad también la debería incluir la aplicación.

En el diseño de la aplicación hay que tener en cuenta que se pretende hacer una aplicación adaptable a la posible adición de nuevos sensores al manipulador que también tendrán que ser gestionados desde el PC y funcionar como un único sistema, por lo que tiene que ser una aplicación modular y robusta.

Para la implementación y diseño de esta aplicación y del resto de aplicaciones del proyecto hay que tener en cuenta que se trata de aplicaciones con componentes que reciben datos e interactúan con el entorno en tiempo real, son lo que se llaman sistemas en tiempo real, este tipo de sistemas no solo tienen que ser funcionalmente correctos sino que además deben hacerlo dentro de intervalo de tiempo adecuado. Este tipo de sistemas nos llevan a seguir la siguiente arquitectura a la hora de diseñarlos e implementarlos.

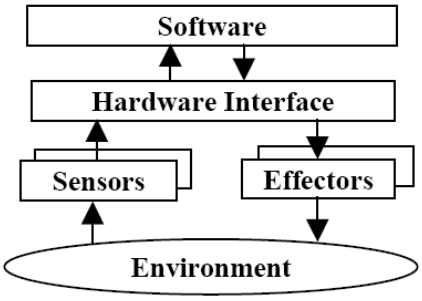


Figura 7 – Esquema arquitectura sistemas en tiempo real [2].

A continuación explicaremos los diferentes conceptos de esta arquitectura y sus funcionalidades:

Entorno: Por definición los sistemas en tiempo real interactúan con el entorno, en nuestro caso la interacción con el entorno está limitada al espacio de trabajo del SCARA.

Actuadores y sensores: los sistemas en tiempo real pueden adquirir información del entorno a través de los sensores y producir cambios en el entorno mediante los actuadores, en nuestro caso los actuadores serán los servos del SCARA que a su vez nos harán de sensores indicándonos la posición del brazo mecánico del manipulador, y más adelante añadiremos el sensor de distancias que nos proporcionara más información del entorno.

Interfaz hardware: Es la capa de abstracción entre el hardware y el sistema software, hace posible una comunicación satisfactoria entre el software y el hardware.

Para la obtención de los datos de los sensores se pueden seguir dos métodos, “time triggered” y “event triggered”.

En el primer método, se coordinan las acciones mediante un reloj global, el sistema software recibe los datos de los sensores por encuesta, esta encuesta se hace cada cierto intervalo de tiempo del reloj, el intervalo de reloj se debe ajustar para que como máximo sea igual al menor tiempo de servicio, tiempo que tarda el sistema desde que se produce un cambio hasta que es percibido.

En el método basado en eventos es el propio sistema hardware el que produce un evento que nos indica qué se ha producido algún cambio en el sistema, por lo que el software no está constantemente haciendo encuestas y solo consulta los sensores cuando es necesario o es el propio hardware el que le manda la información necesaria junto a la notificación del evento [2].

En un primer momento nos hemos decidido en utilizar el sistema basado en las encuestas cada cierto intervalo de tiempo (Time triggered).

Requisitos

Como se ha visto después de un primer análisis funcional la aplicación debe de ser capaz de llevar a cabo las siguientes tareas:

- Configuración puertos serie.

- Inicialización manipulador.
- Gestión de los archivos de la memoria interna del manipulador.
- Movimiento del manipulador, tanto en coordenadas cartesianas como por pulsos de los actuadores.
- Retorno a la posición origen.
- Configuración del manipulador.
- Ejecución de cualquier instrucción del manipulador.

2.3.1.1 Configuración puertos serie

Para el correcto funcionamiento de toda la aplicación es necesaria la comunicación entre el PC y el SCARA. Por lo que la configuración del puerto serie al cual está conectado el SCARA es un requisito básico. La aplicación permitirá que el usuario indique a que puerto debe conectarse la aplicación para comunicarse con el sistema hardware.

2.3.1.2 Inicialización manipulador

La aplicación debe inicializar el robot. Puesto que sin una inicialización inicial el robot no funcionara correctamente. Aparte se aprovecha para inicializar parámetros del robot y de los sensores que se añadan en adelante, como el sensor laser. Entre los parámetros del robot inicializará todos los que no se pueden consultar y pueden variar entre los distintos usos de la aplicación. Así conseguiremos que cada vez que se utiliza la aplicación el sistema sea consciente de la configuración de sus componentes hardware.

2.3.1.3 Gestión de los archivos de la memoria interna del manipulador

La aplicación debe dar apoyo al usuario para que gestione la memoria interna del SCARA, permitiendo al usuario:

- Consultar el directorio de programas almacenado en la memoria.
- Descargar/Subir el archivo de configuración del SCARA.
- Descargar/Subir el archivo de puntos del controlador.
- Descargar programas contenidos en la memoria del SCARA.
- Almacenar programas nuevos en la memoria del SCARA.

Una funcionalidad que se podría añadir a la gestión de archivos, es el ejecutar los programas. Pero esto se puede hacer mediante el envío de instrucciones al manipulador.

2.3.1.4 Movimiento del manipulador

La aplicación debe permitir al usuario mover el brazo mecánico del SCARA de forma fácil e intuitiva. Permitiendo al usuario definir los movimientos que realizara el SCARA, tanto en coordenadas cartesianas como en coordenadas articulares.

2.3.1.5 Retorno a posición origen

La aplicación debe permitir al usuario dar la instrucción de retorno a la posición de origen del brazo mecánico del SCARA. Este requisito es necesario puesto que después de algunas acciones, como la

ejecución de un programa, es necesario el retorno a la posición de origen para continuar trabajando con el SCARA.

2.3.1.6 Configuración manipulador

Los usuarios podrán configurar el manipulador a través del archivo de configuración. Esta funcionalidad solo es útil para los usuarios que ya tienen costumbre a tratar con este tipo de manipuladores y conocen profundamente los parámetros de configuración del SCARA. En el apéndice A se pueden ver los distintos parámetros de configuración en caso de que fuera necesario modificar alguno.

También hay que tener en cuenta parámetros configurables que afectan al funcionamiento del manipulador como es el modo de funcionamiento (manual o automático) y la velocidad. Estos parámetros se pueden gestionar mediante instrucciones, por lo que con el siguiente requisito quedan cubiertos.

2.3.1.7 Ejecución de instrucciones del manipulador

El usuario tiene que poder enviar al SCARA cualquier instrucción de las que es capaz de interpretar, para ello el usuario debe conocer las instrucciones y su correcta sintaxis, en caso de ser necesario hay una relación de instrucciones que se pueden utilizar en el manual del SCARA.

2.3.2 Especificación y diseño

Empezaremos definiendo los distintos componentes físicos que integrarán el sistema y como se organizarán. Los distintos elementos físicos son el robot el PC y el sensor. Los distintos elementos ya están explicados en profundidad en otros puntos de la memoria por lo que nos centraremos en su estructura.

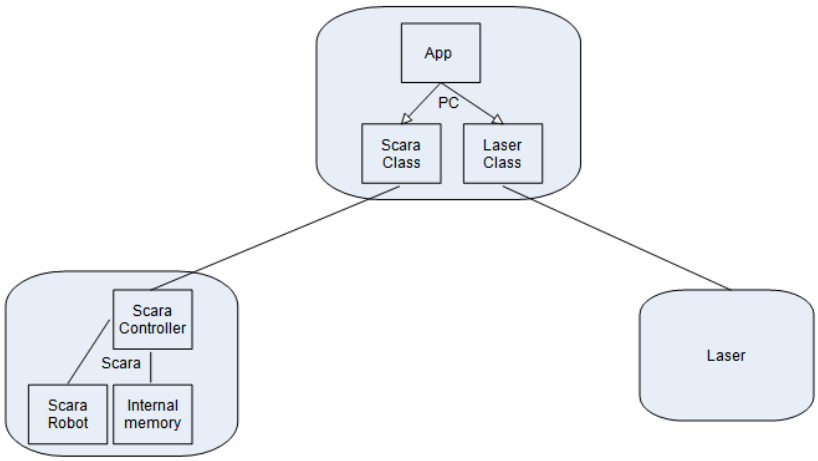


Figura 8 – Arquitectura hardware

Como podemos ver siguiendo la arquitectura de los sistemas en tiempo real. El PC tendrá una interfaz para cada componente hardware añadido y un sistema software que gestionara los componentes.

Una vez vistos los componentes de hardware seguiremos definiendo los actores del sistema. En nuestra aplicación intervienen dos actores los usuarios de la aplicación y el hardware externo al PC, es decir, el SCARA, que interactuará con el sistema en respuesta a las distintas acciones del usuario.

El usuario del sistema es alguien acostumbrado a tratar con este tipo de manipuladores y que ya conoce al menos parte de las instrucciones que se le pueden dar al manipulador desde un terminal.

Una vez definidos los actores y los componentes del sistema veremos los distintos casos de uso del sistema necesarios para cubrir los requisitos funcionales surgidos de un primer análisis funcional del sistema.

2.3.2.1 Diseño de las interfaces y mapa navegacional

Después de ver los casos de uso que tenemos para nuestra aplicación hemos diseñado la aplicación con las siguientes interfaces (figuras 9 y 10).

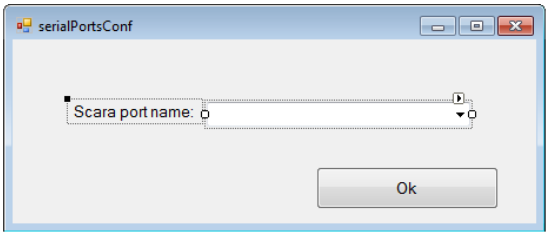


Figura 9 - Interfaz inicial Terminal SCARA

Dado que sin saber el puerto serie al que está conectado el SCARA el resto de la aplicación no tiene ninguna funcionalidad la aplicación empiece mostrando la interfaz (figura 9) en la que se especifica que puerto es el que utilizamos para el SCARA y una vez le damos al botón de OK se cerrara y pasaremos a la siguiente interfaz (figura 10).

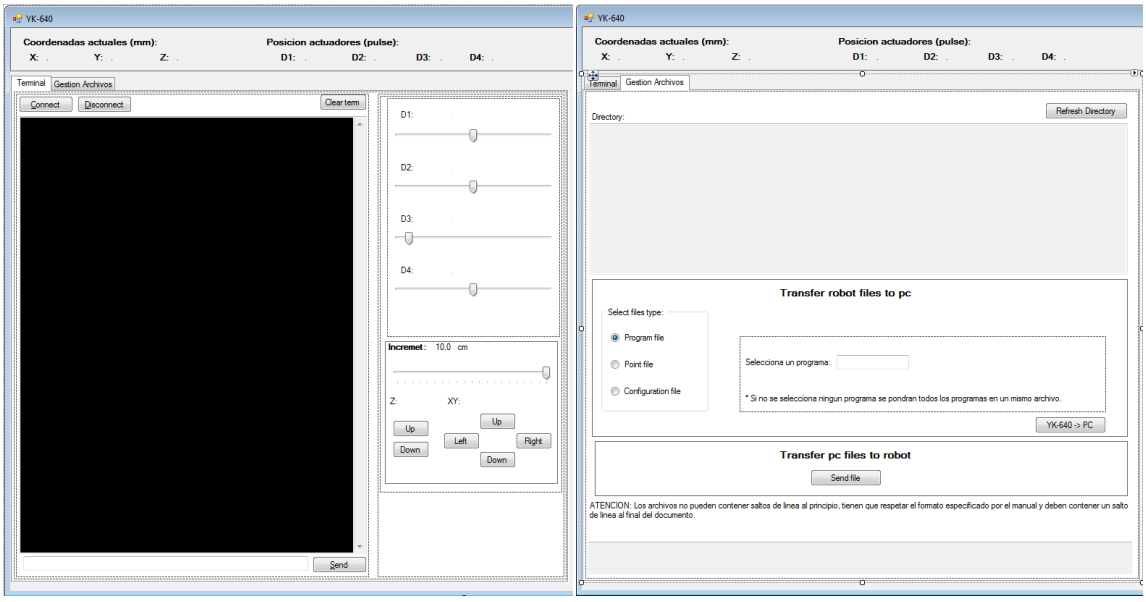


Figura 10 – Interfaz consola YK-640

Esta interfaz es la que da al usuario la posibilidad de interacción con el manipulador SCARA. La interfaz está dividida en dos secciones, una en la parte superior en la que se muestra permanentemente al usuario la posición del terminal del SCARA tanto en coordenadas cartesianas como en coordenadas articulares. A continuación la segunda sección con la que se puede interactuar con el SCARA cubriendo todos los requisitos surgidos del análisis funcional.

La segunda sección de la interfaz se divide en dos secciones más, separadas por pestañas. En la primera pestaña, Terminal, hay la opción de conectar y desconectar el puerto serie, un terminal para comunicarse con el SCARA mediante las instrucciones interpretables por el controlador y al lado del terminal hay cuatro sliders para indicar el número de pulsos de los actuadores del SCARA, un slider por actuador (ver figura 10), y seis botones para desplazar el terminal incrementalmente sobre sus coordenadas cartesianas, el incremento se puede definir con el slider que hay justo encima de estos botones.

La segunda pestaña se encarga de la gestión de los archivos de la memoria interna del SCARA. La cual tiene un botón para cargar el directorio de programas, que se visualizara como texto plano en el cuadro de texto situado justo debajo de este botón (refresh directory). A continuación esta la sección para descargar archivos de la memoria del SCARA. Para descargar el archivo de configuración o de puntos será suficiente con seleccionar el tipo de archivo, con los botones del lado izquierdo de la sección (Select file) y pulsar el botón “YK-640->PC”. En caso de querer descargar un programa hay que seleccionar la opción de programa en el group box y escribir el nombre del programa en el campo de texto creado para ello y por ultimo darle al botón “YK-640->PC”. Después de apretar dicho botón se abrirá un “Save file dialog” para elegir en que fichero quieres salvar el programa descargado de la memoria del SCARA. Por último hay un botón para subir archivos a la memoria interna del SCARA, al pulsar este botón se abrirá un navegador de ficheros para que se seleccione el archivo que se quiere guardar en la memoria del SCARA.

A partir del diseño de las interfaces nos queda el mapa navegacional de la figura 11.

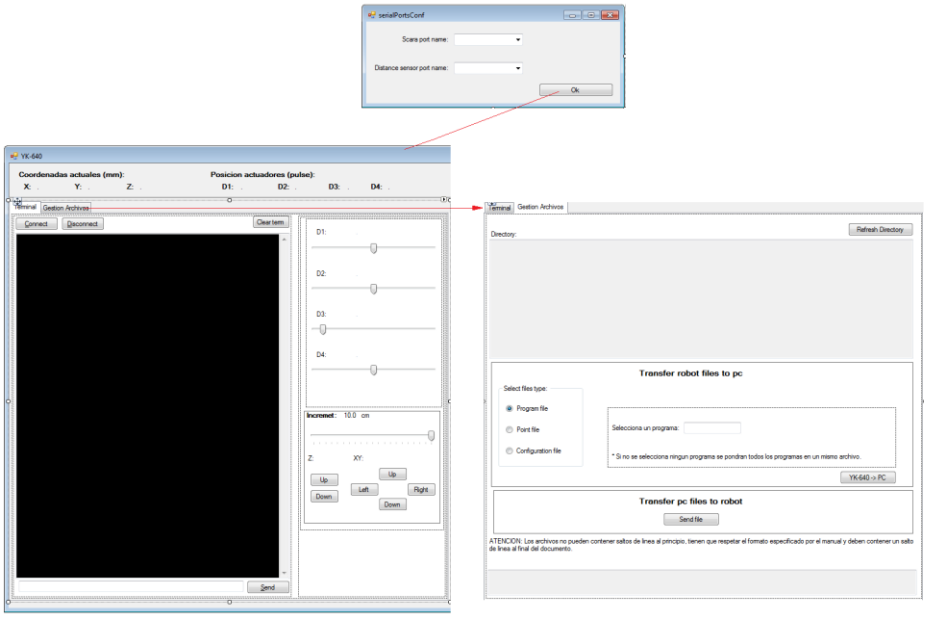


Figura 11 - Mapa navegacional 1 aplicación consola YK-640

2.3.2.2 Especificación, casos de uso y diseño de la aplicación

A continuación podemos ver un diagrama en el que se reflejan todos los casos de uso de la aplicación y la especificación de cada uno de ellos.

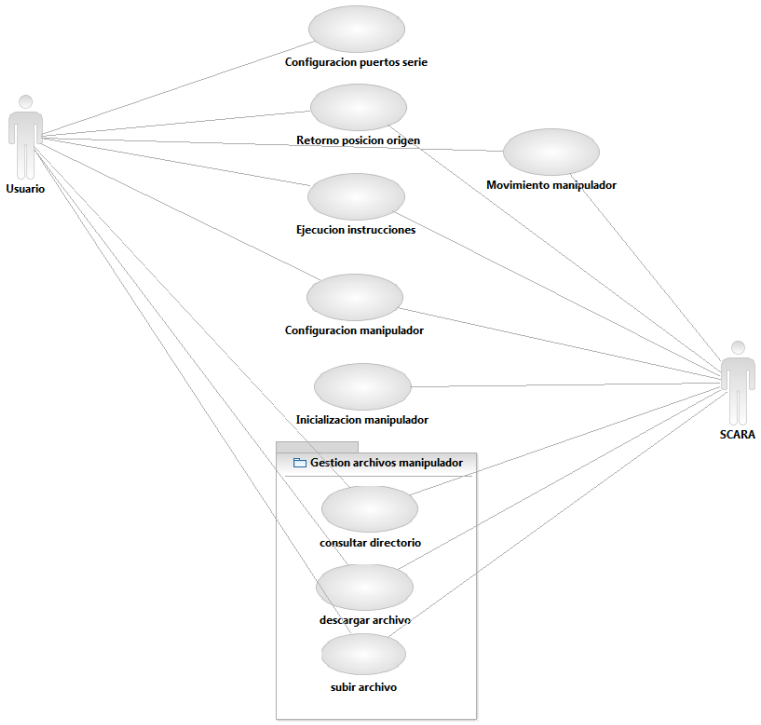
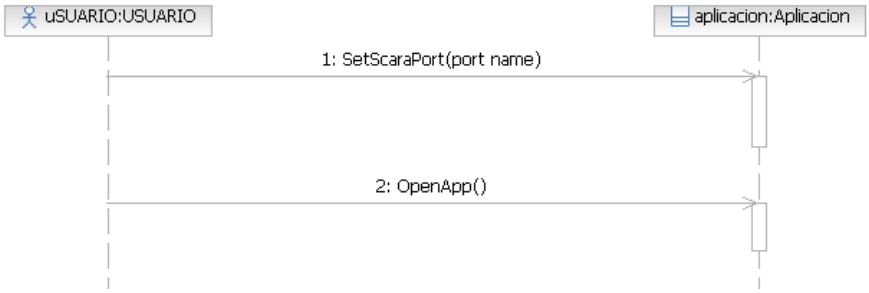


Figura 12 – Diagrama de casos de uso 1 aplicación consola YK-640

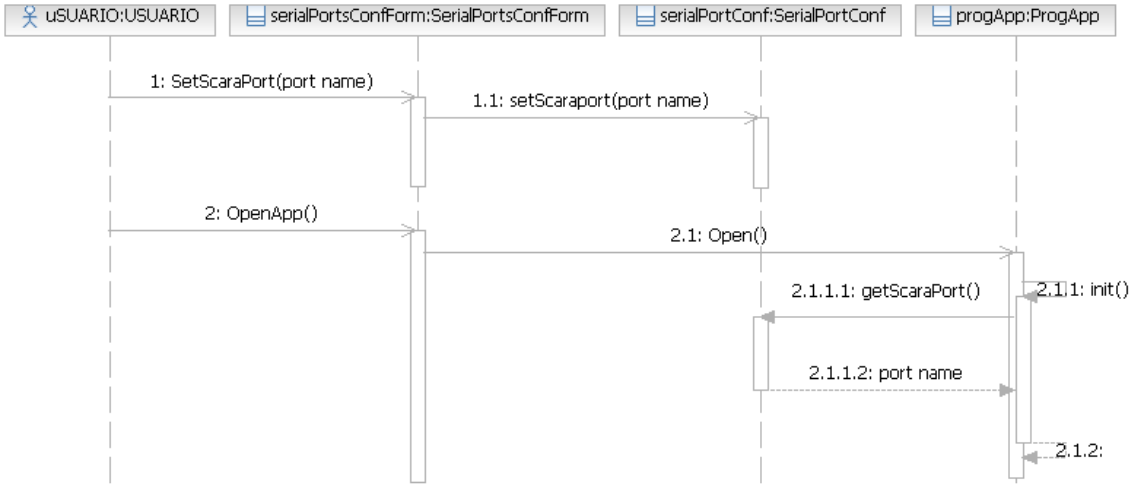
2.3.2.2.1 Configuración puertos serie

El usuario introduce en el sistema el nombre del puerto serie al que está conectado el SCARA, una vez introducido el siguiente paso es pasar a la interfaz para interactuar con el SCARA.

2.3.2.2.1.1 *Diagrama actor-sistema*



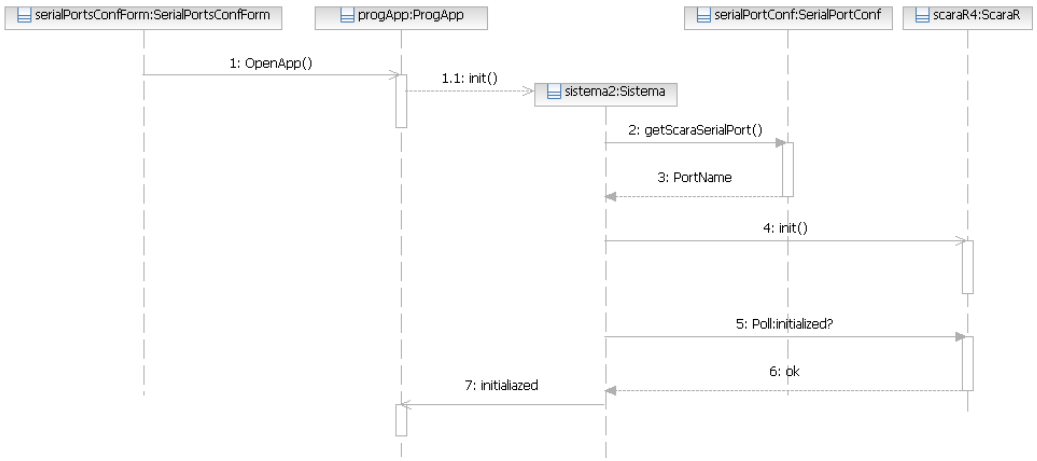
2.3.2.2.1.2 *Diagrama de secuencia*



2.3.2.2.2 Inicialización manipulador

Después de configurar el puerto serie el usuario abre la interfaz principal de la aplicación para empezar a trabajar con el SCARA, para ello necesita el SCARA inicializado. Por lo que el sistema inicializa el sistema con el puerto serie que ha sido configurado anteriormente.

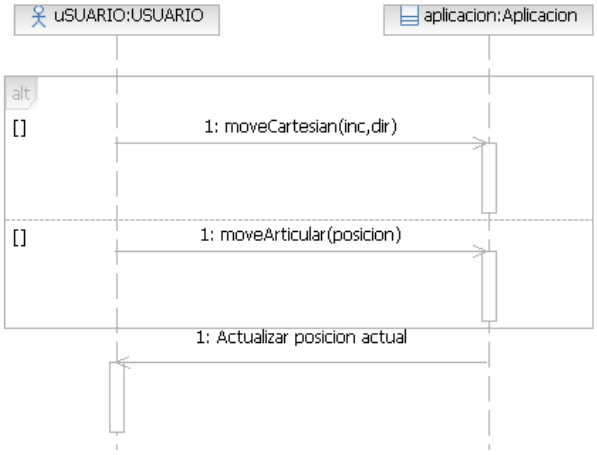
2.3.2.2.2.1 *Diagrama de secuencia*



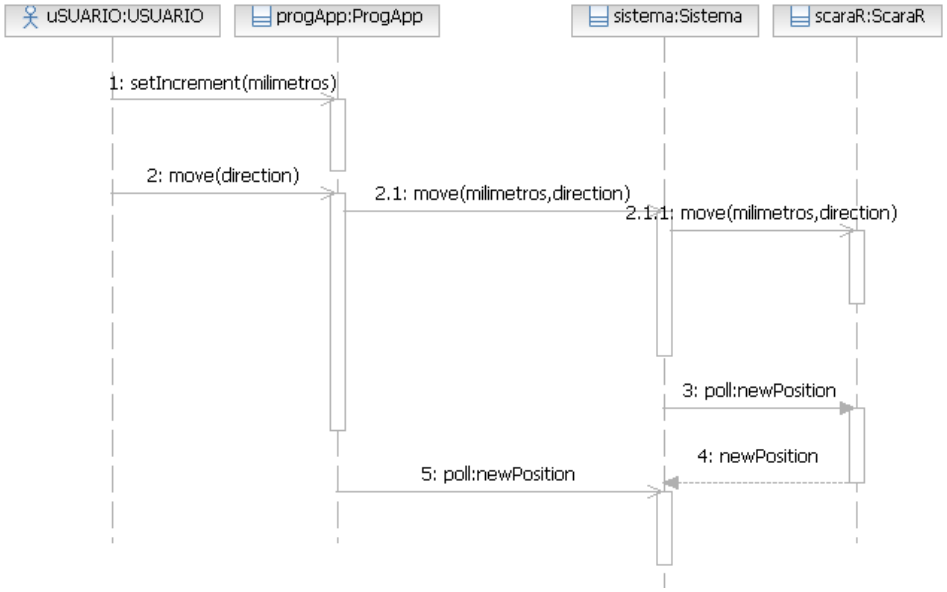
2.3.2.2.3 Movimiento manipulador

El usuario indica un desplazamiento en coordenadas cartesianas o en coordenadas articulares del SCARA y este se mueve a la posición indicada siempre que esté dentro de su rango de acción, si no puede efectuar el movimiento muestra un mensaje de error.

2.3.2.2.3.1 *Diagrama actor-sistema*



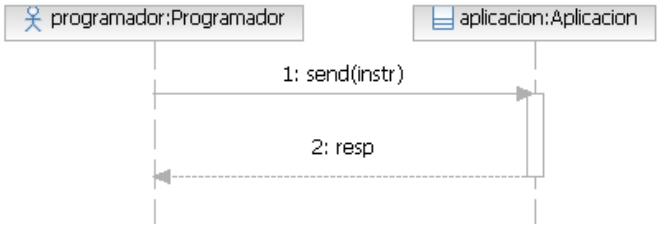
2.3.2.2.3.2 *Diagrama de secuencia*



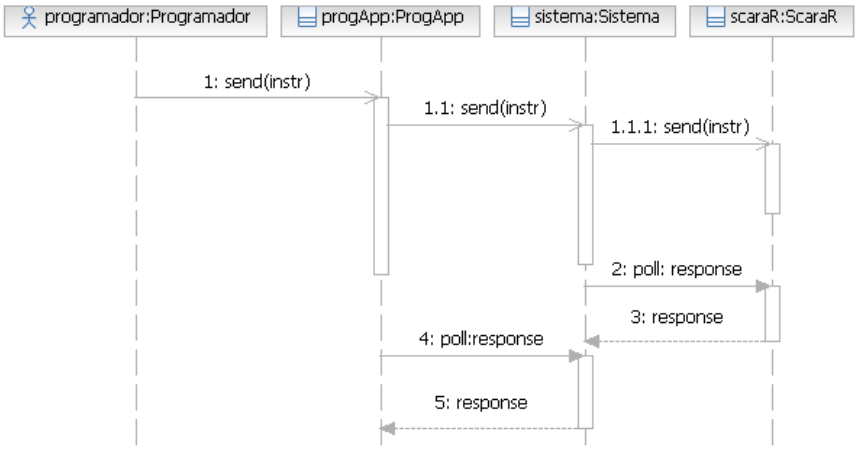
2.3.2.2.4 Ejecución instrucciones

El usuario le manda una instrucción al SCARA, si es correcta y posible de ejecutar se ejecuta inmediatamente sino muestra el mensaje de error, que nos devuelve el controlador del SCARA, por el terminal de la aplicación.

2.3.2.2.4.1 *Diagrama actor-sistema*



2.3.2.2.4.2 *Diagrama de secuencia*



2.3.2.2.5 Consultar directorio

El usuario pide actualizar el directorio del programa o verlo por primera vez y el sistema consulta el directorio del SCARA y lo muestra por pantalla, muestra directamente el mensaje que recibe del controlador del SCARA.

Como se puede ver todos los diagramas de secuencia y de actor-sistema son muy parecidos por lo que de ahora en adelante los omitiré, a cambio se explicara por escrito con más detalle cómo funcionarán cada uno de los casos de uso.

2.3.2.2.6 Descargar archivo

El usuario selecciona un archivo de los que están almacenados en la memoria interna del SCARA, y el sistema descarga el archivo del SCARA y lo almacena en un archivo del sistema de ficheros del PC. La selección del archivo se llevare a cabo mediante un campo de texto en el que el usuario introducirá el nombre del programa que quiere descargar.

2.3.2.2.7 Subir archivo

El usuario escoge un archivo del sistema de ficheros del PC mediante un navegador de archivos del sistema y la aplicación se encarga de almacenarlo en la memoria interna del SCARA. El navegador no nos dejara coger archivos de tipo no válidos para el controlador del SCARA. En caso de que haya algún tipo de error durante la transmisión la aplicación lo notificara al usuario.

Antes de continuar con la especificación de la aplicación hay que tener en cuenta que las operaciones en las que haya comunicación entre el PC y el SCARA no siguen un comportamiento típico, una operación que nos devuelve la respuesta, en este caso se envía una instrucción por la línea serie y cuando la respuesta este lista nos llegara por la línea serie. Por lo que hay que tener en cuenta siempre en qué estado está el sistema, que nos puede llegar como respuesta a la acción realizada y a que estado nos llevara cada respuesta. Para solucionar esto la aplicación tendrá diversos estados en función de lo que

se espera recibir. En la siguiente figura (fig. 13) se pueden ver los diferentes estados y las transiciones entre ellos según los datos que nos lleguen por el puerto serie.

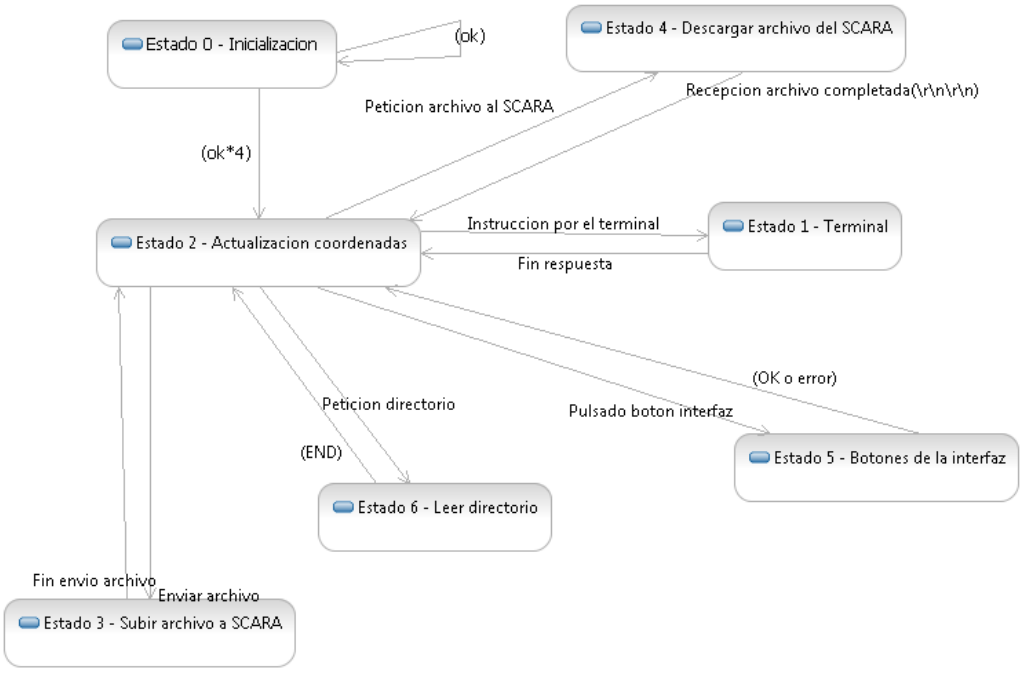


Figura 13 - Diagrama de estados SCARA

Como se puede ver en la figura hay 7 estados a continuación describiré cada estado:

- **Estado 0:** este estado esta para la inicialización del SCARA, se abre el puerto de comunicación y se envían instrucciones para inicializar el SCARA con los parámetros deseados, en este caso son modo de trabajo manual, velocidad al 50%, activar los servos y desplazar el terminal a la posición de origen. En este caso se espera recibir una confirmación por cada instrucción que se envía al SCARA, una vez se recibe la última confirmación el sistema pasa al estado 2.

- **Estado 1:** cuando se manda al SCARA una instrucción por el terminal se entra en este estado, esperara respuesta y la mostrara por el terminal, una vez haya recibido toda la respuesta pasara al estado 2. Se habrá llegado al final de la respuesta cuando se cumpla esta condición: `(received == "OK\r\n" || received == "\r\n" || received.Contains("END\r\n")`.

- **Estado 2:** es el estado de espera, en este estado va actualizando las coordenadas del SCARA constantemente, cada un cierto de intervalo de tiempo (por el momento lo hemos dejado fijado en 7 ms.), para que el usuario pueda saber las coordenadas actuales del manipulador. Cuando se lleva a cabo cualquier otra operación sale de este estado y pasa al estado pertinente según la operación que se quiera llevar a cabo.

- **Estado 3:** entra en este estado cuando se le da la instrucción al SCARA de que se prepare para recibir un archivo, en este estado hay tres posibles cadenas de datos que pueden llegar al sistema. La primera que debe llegar es `"*** Please Enter !"` a partir de este momento el controlador está listo para recibir el archivo. También puede recibir `"OK\r\n"`, cuando lo reciba es que la transmisión del archivo se ha completado correctamente, en este caso se pasara al estado 2. Por último puede recibir cualquier otra cadena de error, si recibe un error se transmitirá el error al usuario y se pasara al estado de espera (estado 2).

- **Estado 4:** en este estado entra cuando se envía la instrucción para descargar un archivo de la memoria interna del SCARA, la transmisión del archivo empezara de forma inmediata a no ser que él archivo solicitado no se encuentre en la memoria del controlador, en se caso llegara un mensaje de error. Mientras se esté recibiendo el archivo se permanecerá en este estado una vez se acabe la descarga del archivo se pasara al estado 2, la transmisión del archivo habrá finalizado cuando se reciba `"\r\n\r\n"`**.**

- **Estado 5:** cuando se pulsa un botón de la interfaz la aplicación manda una instrucción y entra en este estado, se espera la respuesta si es diferente a `"OK\r\n"` será un mensaje de error que se transmitirá al usuario. Una vez recibida la respuesta sea un error o la orden se haya realizado correctamente el sistema pasara al estado 2.

- **Estado 6:** se entrara en este estado cuando se envíe al SCARA la instrucción de consulta del directorio de programas que hay en la memoria del controlador, lo que se recibe por la línea serie se almacena para mostrarlo al usuario, una vez ha concluido el envío del directorio por parte del SCARA (`"END\r\n"`) se pasa al estado 2. En este estado no esperamos ningún mensaje de error puesto que aun que el directorio este vacío el controlador nos enviara el directorio.

Viendo los diferentes estados en los que puede estar el sistema vemos que es muy importante un correcto diseño de la función que se encargara de recibir los datos recibidos por el puerto serie y que nos define en cada momento en el estado en el que estamos, a continuación podemos ver su diseño (figura 14).

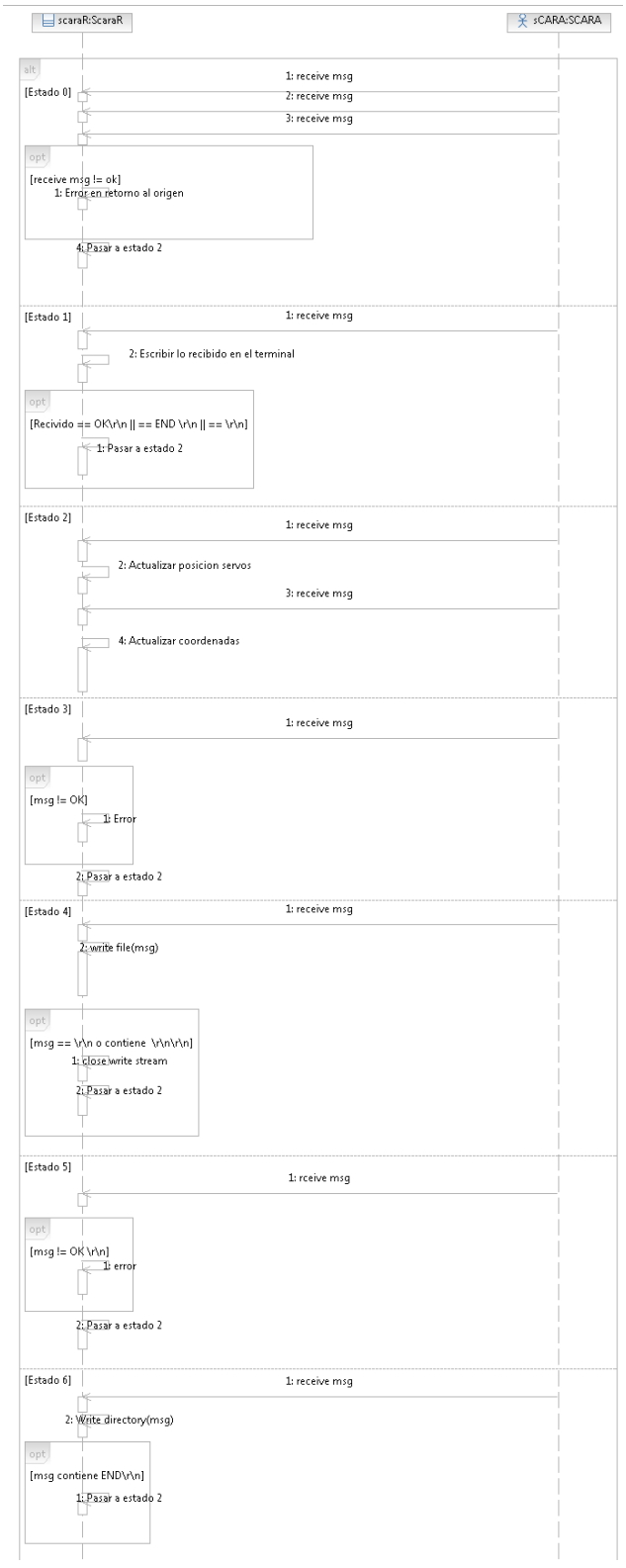


Figura 14 – Diagrama de secuencia recepción de datos por el puerto serie

En este diagrama de secuencia no se ve reflejado la parte relativa al sistema y el resto de clases de la aplicación, pero el funcionamiento del resto del sistema es por encuesta por lo que cada cierto intervalo de tiempo el sistema encuestaría a la instancia de la clase ScaraR para obtener el estado actual de todas las variables de interés para él y las transmitiría por el mismo método a la interfaz para que fuera visible por los usuarios.

2.3.2.3 Diagrama de clases

Otro aspecto a tener en cuenta en el diseño de esta aplicación es que el sistema que se crea tiene que ser expandible con nuevos sensores y reutilizable en otros programas, en definitiva es un controlador del sistema hardware que se quiera montar.

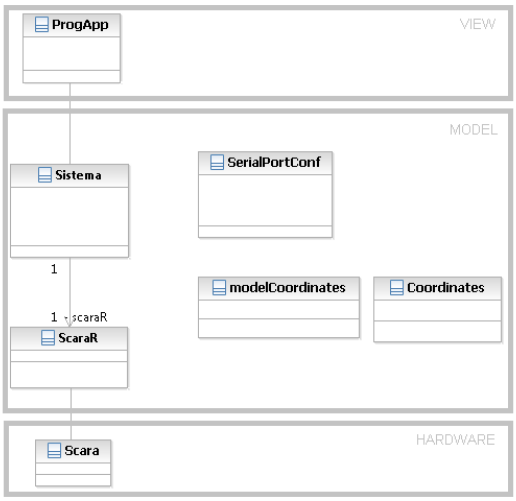


Figura 15 - Diagrama de clases aplicación consola YK-640

La clase ScaraR es la clase que abstrae el hardware del manipulador al resto del sistema esta clase se encargara de gestionar la comunicación hardware-software y gestionara los estados del sistema en lo que refiere al SCARA.

La clase sistema esta creada para cuando se añada más hardware al sistema será la clase que unificará y sincronizara todos los elementos hardware con los que en un futuro se amplíe el sistema. Nos dará una visión del sistema como un único elemento y gestionara las acciones que requieran la sincronización de varios elementos hardware.

Las clases Coordinates y modelCoordinates representan las coordenadas del SCARA, cartesianas y articulares.

La clase serialPortConf representa la configuración de los puertos serie de todos los elementos hardware del sistema, será una clase “singleton” lo que nos permitirá mantener la configuración del sistema en futuras aplicaciones.

2.3.3 Evaluación

En esta evaluación voy a incluir todos los cambios que se han visto necesarios durante la realización del proyecto, no es una evaluación de una primera iteración únicamente así que los cambios realizados en el siguiente apartado llegarán hasta el estado actual de la aplicación.

Después de la implementación y prueba del primer diseño realizado se ha visto necesario la mejora de la usabilidad del apartado de gestión de archivos del SCARA.

En cuanto al movimiento del manipulador se ha comprobado que es una buena herramienta dar la opción de desactivar los servos del SCARA para que el usuario pueda desplazar el terminal de forma manual y situar este en una posición determinada.

Otro aspecto importante que ha surgido después de las pruebas realizadas sobre la primera iteración es la creación de otra interfaz para usuarios que no tengan conocimientos sobre este SCARA o ningún otro, es decir, una aplicación usable por cualquier persona, aunque es bueno mantener la otra para dar a usuarios experimentados la posibilidad de explotar al máximo las funcionalidades del SCARA.

Mucho más adelante en el proyecto se ha visto la necesidad de cambiar la forma en la que se gestiona la obtención de datos de los sensores por lo que hemos tenido que rediseñar y re-implementar la aplicación para que funcione por eventos en vez de por encuesta cada cierto periodo de tiempo. Esto se debe a que el tiempo de servicio del SCARA es muy variable según la acción que se lleve a cabo lo que dificulta mucho ajustar el tiempo de encuesta de forma que el funcionamiento de la aplicación sea aceptable para situaciones en las que se necesita una sincronización exacta de distintos actuadores y sensores del sistema.

El último cambio que se ha visto necesario es el de diferenciar un estado para controlar la finalización de los movimientos del SCARA con tal de poder sincronizarlos con el funcionamiento del sensor de distancias que se integrará más adelante.

Ahora pasaremos a aplicar los cambios que se han visto necesarios en las distintas evaluaciones que se han hecho durante el desarrollo del proyecto.

2.3.4 Segundas iteraciones en el diseño de la aplicación

Como hemos visto en la evaluación de la iteración anterior hay que añadir y cambiar algunos aspectos de la aplicación, a continuación se irán añadiendo los cambios en la aplicación, intentaremos solo mostrar los aspectos que cambian para evitar hacer repetitiva la memoria.

Como se ha mencionado en la evaluación anterior ha surgido la necesidad de tener en cuenta un nuevo tipo de usuarios para nuestra aplicación, para la mayoría de los casos de uso de la aplicación siguen funcionando exactamente igual por lo que no variara su diseño.

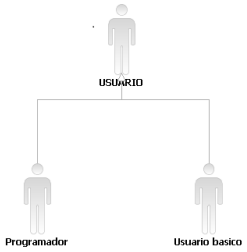


Figura 16 - Esquema usuarios de la aplicación

A continuación vemos los nuevos requisitos que nos han surgido en la segunda iteración.

2.3.4.1 Requisitos nuevos de la aplicación

En cuanto a los requisitos se añade tres nuevos requisitos:

- Conexión y desconexión de los actuadores del manipulador.
- Retorno a la posición inicial.
- Gestión de los archivos de la memoria interna del manipulador.
    - Ejecutar programa.

2.3.4.1.1 Retorno a la posición inicial

El SCARA vuelve a su posición inicial, en algunos casos es necesario que el SCARA vuelva a su posición inicial antes de realizar otra nueva tarea. Antes el programador lo podía hacer ejecutando la instrucción "@ORIGIN" en el terminal pero para el usuario básico es necesario dar la funcionalidad a la aplicación.

2.3.4.1.2 Conexión y desconexión de los actuadores del manipulador

Este nuevo requisito permite que los usuarios de la aplicación muevan el terminal del manipulador a mano lo permite a los usuarios ubicar el terminal en una posición determinada de forma rápida y cómoda.

2.3.4.1.3 Gestión de los archivos de la memoria interna del manipulador

Dentro de este grupo nos surge una nueva funcionalidad fruto del nuevo usuario y de la falta de usabilidad de esta sección en la anterior iteración, ejecutar los programas pulsando un botón de forma que sea más fácil y evitar que el usuario pueda cometer errores.

2.3.4.2 Especificación y diseño

Como se ha dicho anteriormente nos ha aparecido un nuevo usuario para la aplicación, la aparición de este nuevo usuario amplía la cantidad de gente que podrá utilizar la aplicación y el manipulador. Este nuevo usuario es el usuario que no conoce la semántica propia de las instrucciones del SCARA y por lo cual necesita una aplicación más usable e intuitiva. Por lo cual ahora la aplicación tendrá dos usuarios, programador, el usuario más experimentado, y usuario básico, un usuario que no tiene por que tener conocimientos previos sobre manipuladores SCARA.

La aplicación tendrá una nueva interfaz para el usuario básico y la interfaz para el programador será mejorada en cuanto a gestión de archivos se refiere.

2.3.4.2.1 Diseño de las interfaces y mapa navegacional

En cuanto a la interfaz inicial será igual que la resultante del primer diseño solo que añadiéndole un botón para poder acceder a la interfaz diseñada para el usuario que hemos llamado usuario básico.

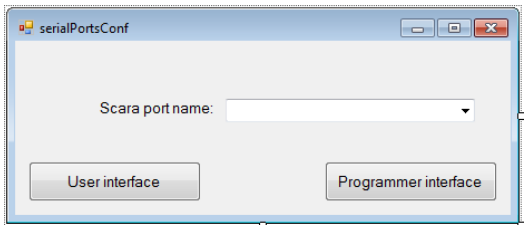


Figura 17 – Interfaz inicial aplicación consola YK-640

En cuanto a la interfaz para el usuario programador es la interfaz que habíamos diseñado anteriormente con la mejora de usabilidad que explicaremos a continuación y la adición de un check box para activar y desactivar los servos del SCARA.

Las mejoras en la usabilidad de la gestión de archivos de la memoria del controlador se han hecho haciendo lo siguiente: se ha cambiado la forma en que se visualiza el directorio de programas en vez de texto plano ha pasado a ser una tabla en la que además se permite mediante botones en la misma tabla eliminar o descargar programas de la memoria del SCARA. En cuanto al archivo de configuración y el archivo de puntos del controlador se ha añadido un botón su descarga y otro para actualizarlos con archivos del PC. Como se puede ver en la siguiente figura (figura 18).

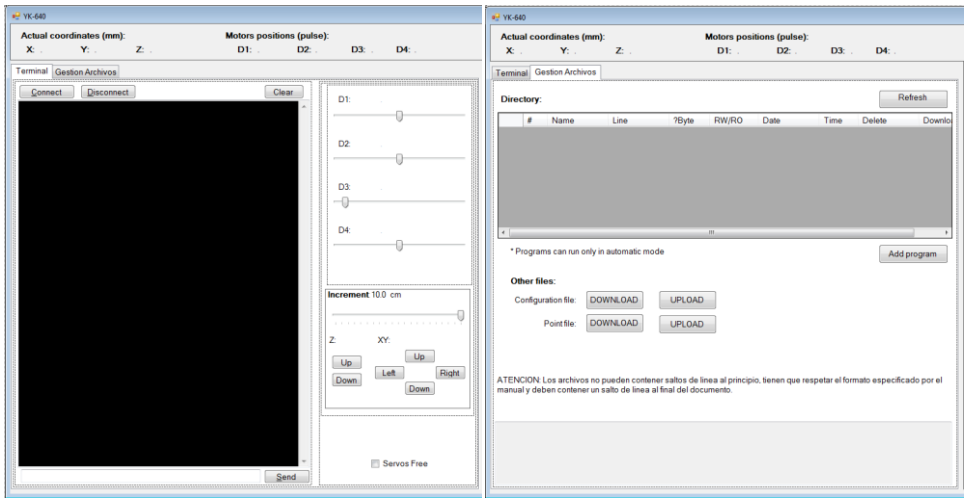


Figura 18 – Interfaz para programador de la aplicación consola YK-640

En cuanto a la interfaz para el usuario básico (figura 19) es parecida a la interfaz para el programador pero se ha suprimido el terminal para mandar instrucciones al SCARA, se ha añadido algunas opciones de configuración del SCARA a la interfaz, como la velocidad y el modo de trabajo, y se ha simplificado toda la interfaz en una sola pestaña.

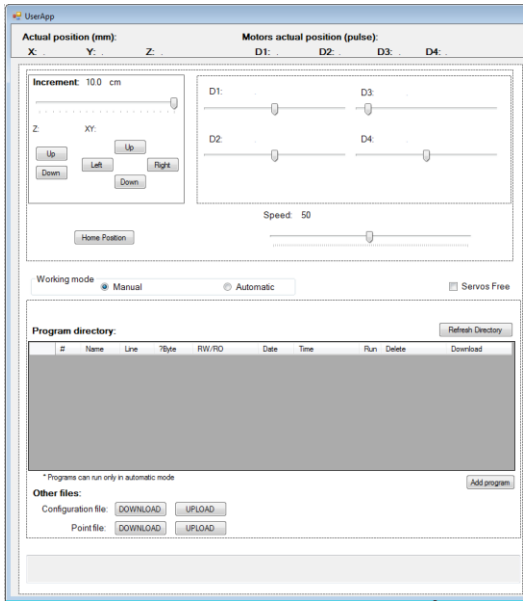


Figura 19 – Interfaz para usuario básico aplicación consola YK-640

El mapa navegacional de la aplicación (figura 20) es el mismo que en la iteración anterior pero añadiendo la interfaz nueva a la que se accede desde el botón que se ha añadido a la interfaz para configurar el puerto serie.

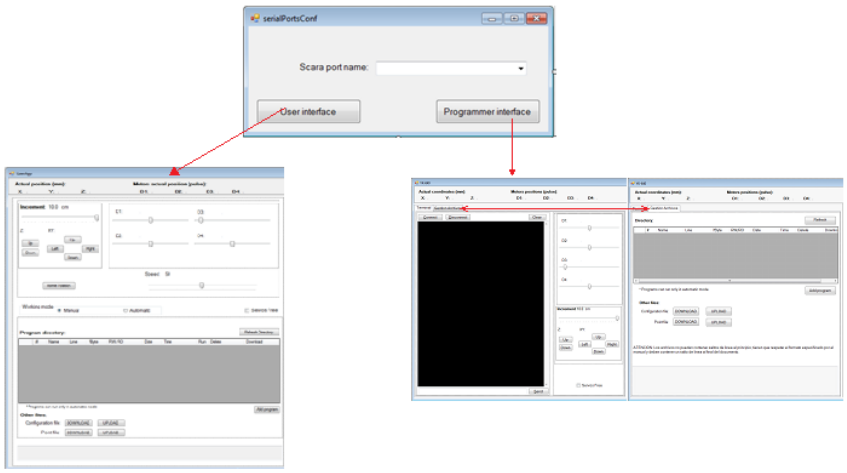


Figura 20 – Mapa navegacional 2 aplicación consola YK-640

2.3.4.2.2 Especificación casos de uso y diseño de la aplicación

En el diagrama de casos de uso general (figura 21) aparecen los tres nuevos casos de uso el nuevo usuario.

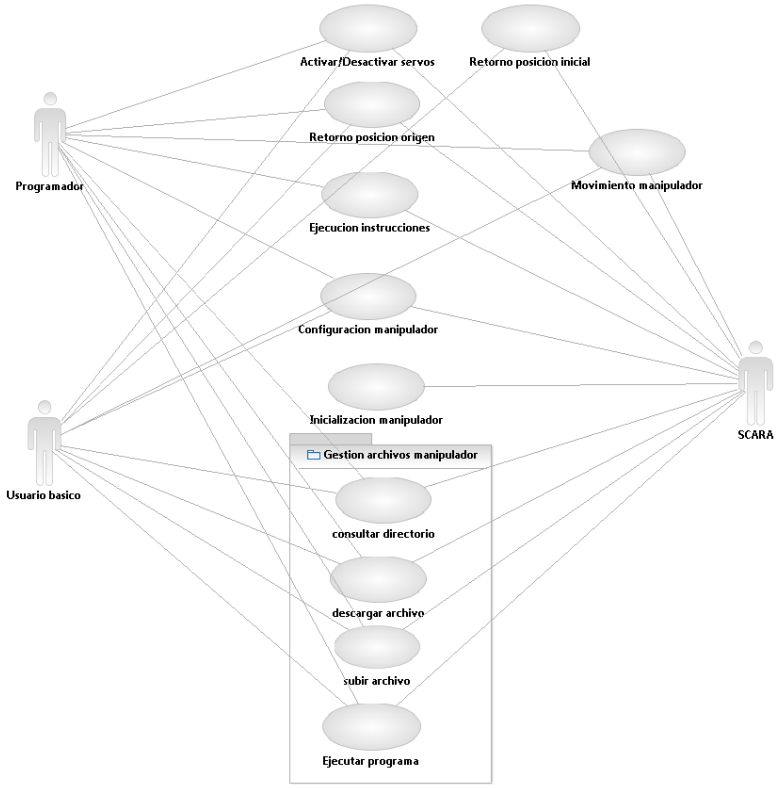


Figura 21 – Diagrama casos de uso 2 aplicación consola YK-640

2.3.4.2.2.1 *Inicialización manipulador*

Después de configurar el puerto serie el usuario abre la interfaz principal de la aplicación para empezar a trabajar con el SCARA, para ello necesita el SCARA inicializado. Este de caso de uso no varía respecto al diseño anterior, lo único que cambia es como notificará el SCARA al resto del sistema que la inicialización ha sido completada que en esta ocasión será mediante un evento.

2.3.4.2.2.2 *Movimiento manipulador*

El usuario indica un desplazamiento en coordenadas cartesianas o indica las coordenadas articulares a las que quiere que se desplace el SCARA, éste se mueve a la posición indicada siempre que esté dentro de su rango de trabajo, si no puede efectuar el movimiento muestra un mensaje de error. La variación será la de la creación del nuevo estado, estado 8, que generara este caso de uso y la creación del evento para notificar al sistema que ya se ha llegado al punto de destino.

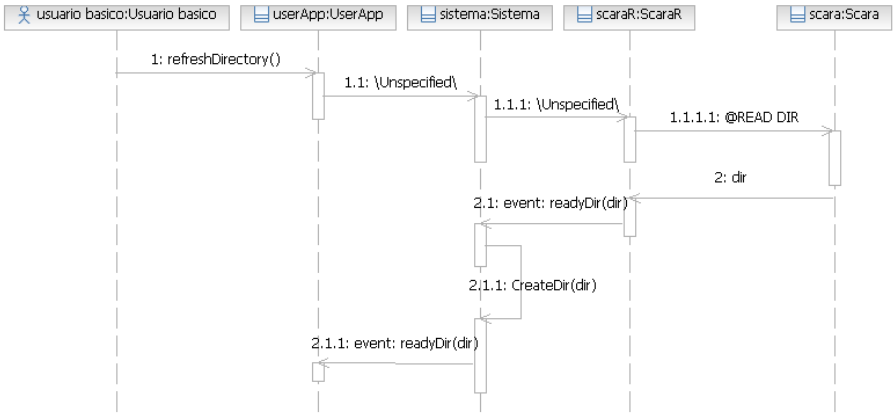
2.3.4.2.2.3 *Consultar directorio*

El usuario pide actualizar el directorio del programa o verlo por primera vez y el sistema consulta el directorio del SCARA y lo muestra por pantalla. La variación está en que ahora la clase sistema generará

la tabla para que se pueda mostrar en las interfaces y el evento que notificara que el directorio está listo.

2.3.4.2.2.3.1 Diagrama de secuencia

A continuación podemos ver un ejemplo de cómo funcionara ahora el sistema utilizando eventos.



2.3.4.2.2.4 *Descargar archivo*

El usuario selecciona un archivo de los que están almacenados en la memoria interna del SCARA, y el sistema descarga el archivo del SCARA y lo almacena en un archivo del sistema de ficheros del PC. Lo que más va a cambiar de este caso de uso es la forma de llevar a cabo la acción, el usuario no tendrá que introducir a mano el nombre del programa o el tipo de archivo que quiere descargar, simplemente apretando un botón se descargara el archivo seleccionado.

Como se puede ver en la figura 19 el directorio contiene el número de programa, el nombre, el tamaño en bytes y la fecha y hora de creación.

2.3.4.2.2.5 *Ejecutar programa*

El usuario básico escoge un programa del directorio y este se ejecuta. Nos crea un estado nuevo, el estado 7, que se encargara de que se ejecute el programa notificarnos su finalización y informarnos en caso de error.

2.3.4.2.2.6 *Activar/Desactivar servos*

Los actuadores del SCARA se activarán o desactivarán en función del estado de un check box en la interfaz. Dando a los usuarios la posibilidad de mover el terminal de forma manual, para esta acción no hace falta un evento que notifique que se ha realizado ni un estado nuevo ya que la incluiremos en el estado 5.

2.3.4.2.2.7 *Retorno posición inicial*

El terminal vuelve a su posición inicial. Es una funcionalidad que se había tenido en cuenta en el diseño inicial pero no se había implementado porque se podía hacer mediante el terminal, al añadir el nuevo

usuario nos obliga a implementarla de forma que el usuario básico pueda ejecutar esta operación que es imprescindible para el funcionamiento de la aplicación.

Hay que tener en cuenta que al hacer estos cambios han aparecido estados nuevos y se ha modificado la forma de gestionar los datos que se reciben de los sensores por lo que el diseño de la aplicación ha cambiado considerablemente en este aspecto. A continuación mostrare los cambios en el diagrama de estados y explicaremos a fondo los dos estados nuevos y las variaciones en el comportamiento del resto de estados.

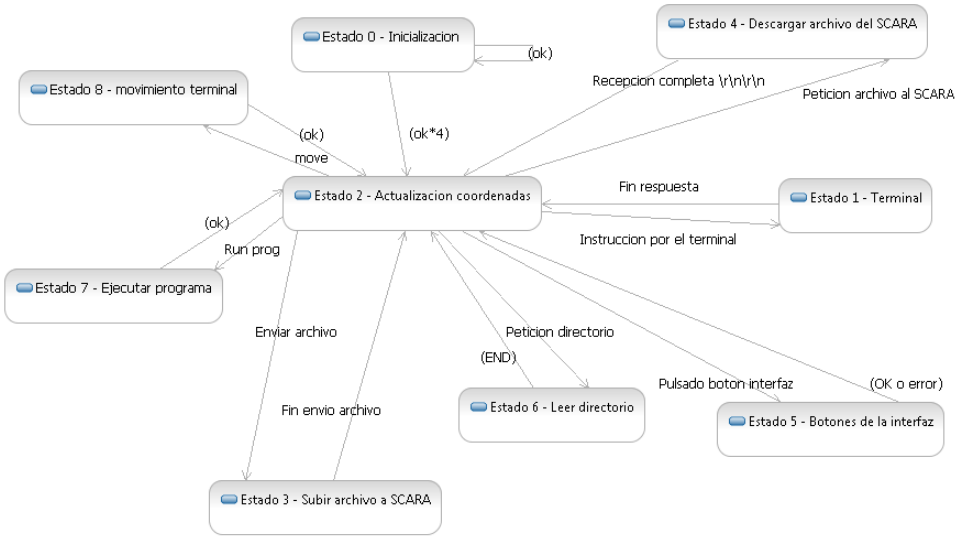


Figura 22 – Diagrama de estados SCARA

Por lo que los estados y sus transiciones quedan de la siguiente forma:

- **Estado 0:** tendrá el mismo comportamiento que en el diseño anterior pero antes del cambio de estado se disparara un evento indicando que el SCARA está inicializado, o un evento indicando el error producido.

- **Estado 1:** como en el caso anterior la variación será que se crea un evento para notificar que hay nuevos datos que escribir en el terminal.

- **Estado 2:** sigue siendo el estado de espera pero ahora el sistema ya no estará constantemente encuestando al SCARA, esperara a la recepción de eventos para comprobar la nueva posición del terminal.

- **Estado 3:** mismo comportamiento que en el diseño anterior pero en caso de error se notificara con un evento.

- **Estado 4:** mismo comportamiento que en el diseño anterior.

- **Estado 5:** mismo comportamiento que en el diseño anterior con la excepción de que notifica los errores mediante un evento.

- **Estado 6:** mismo comportamiento que en el caso anterior pero se notificara el nuevo directorio mediante un evento.

- **Estado 7:** este estado es para cuando se está ejecutando un programa a petición del usuario, durante este estado se muestra todo lo que llegue al puerto de comunicación por pantalla una vez se sale finaliza la ejecución del programa se pasa al estado 2.

- **Estado 8:** se pasara a este estado cuando se ejecute una instrucción de movimiento, cuando este finalice y se reciba el `"OK\r\n"` se lanzara un evento que indique que el movimiento ha finalizado y se pasara al estado 2.

Después de todos estos cambios el diagrama de secuencia de la función que se encarga de la recepción de datos por el puerto serie nos quedará de la siguiente forma:

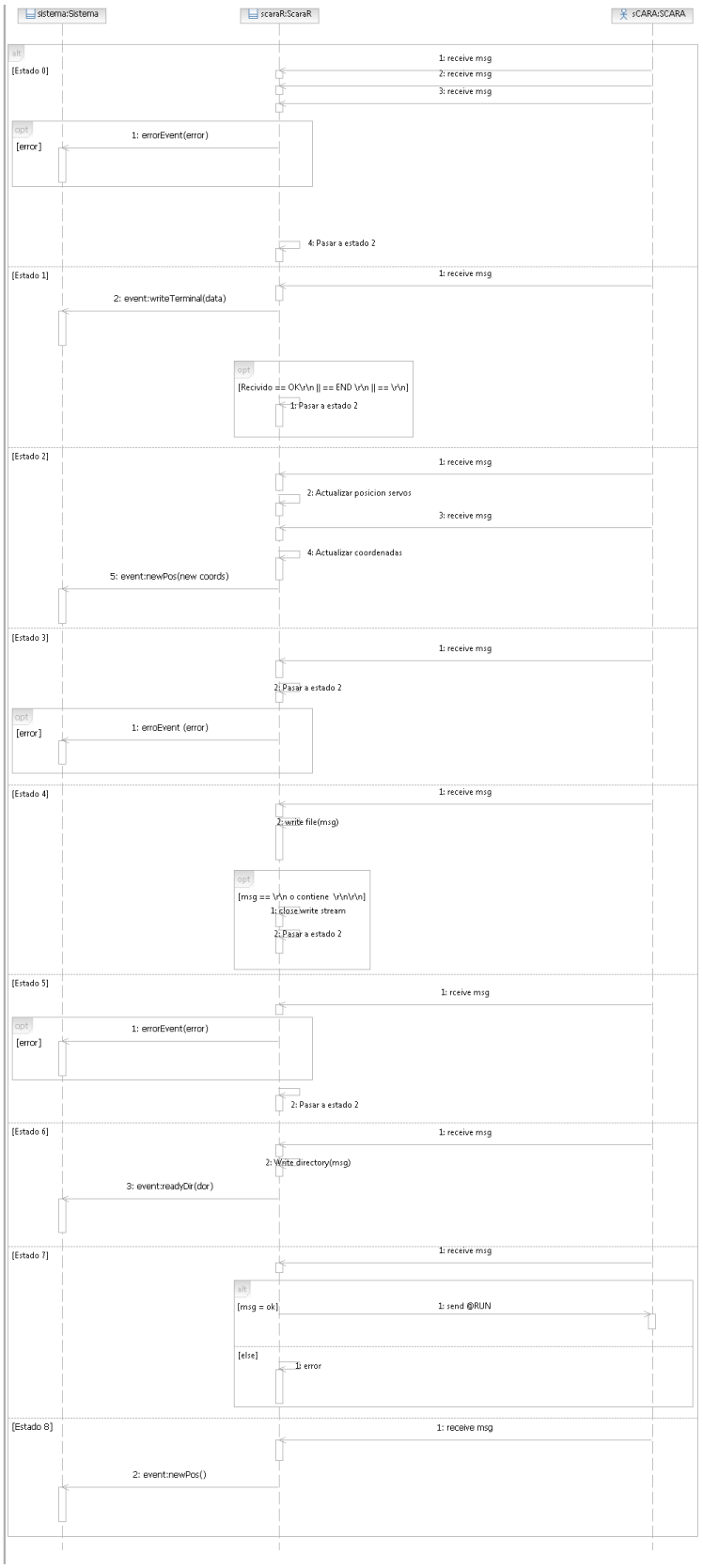


Figura 23 – Diagrama secuencia comunicación SCARA-SISTEMA por eventos

Como se puede observar ahora la información pasa al sistema mediante eventos, por lo que nos ahorramos que el sistema este continuamente encuestando a sus componentes y solo este ocupado en los momentos que son estrictamente necesarios.

2.3.4.2.3 Diagrama de clases

En cuanto al diagrama de clases la única variación que hay después de los cambios que se han tenido que producir para cubrir todos los requisitos y funcionalidades nuevas después de las distintas iteraciones es la creación de una nueva clase para controlar la nueva interfaz para el usuario básico.

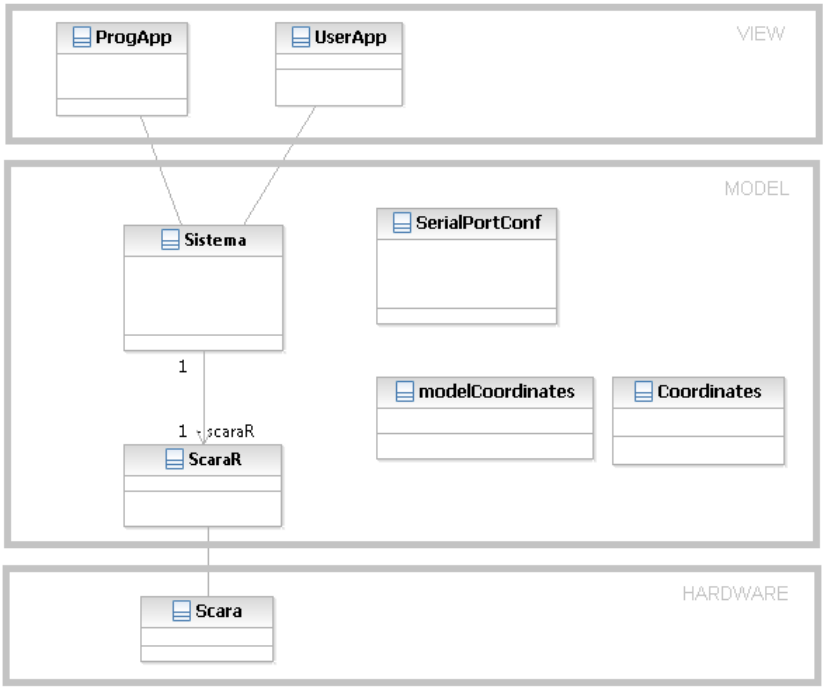


Figura 24 – Diagrama de clases aplicación consola YK-640

2.3.4.3 Pruebas y Evaluación

Después de todos los cambios que hemos hecho después de las múltiples iteraciones y evaluaciones hechas durante la realización del proyecto hemos llegado a un punto en el que no hemos visto la necesidad de seguir trabajando en esta aplicación por lo que la damos por terminada.

Antes de pasar al siguiente punto del proyecto haremos una comparación entre las dos aplicaciones resultantes que nos han quedado, la primera que obtiene los datos mediante polling, y la segunda que está basada en eventos. Para realizar la comparación hemos hecho una serie de operaciones controlando el tiempo de respuesta de las dos aplicaciones en terminar el proceso y comunicarlo al usuario.

Las pruebas se han realizado con la integración del laser finalizada, por lo que también hay operaciones relativas al laser. En las siguientes tablas podemos observar las operaciones y el tiempo utilizado por estas.

Tabla 1 – Relación tiempos de servicio operaciones, Time Triggered

Operación	1	2	3	4	5	6	7	8	9	10	media
Inicializacion	10864	10710	10712	10718	10766						10753.6813
Manual Sample	1435	1372	1431	1169	1394	1123	1218	1079	1493	1155	1270.45455
Actualizar posición	362	286	295	477	364	122	180	455	191	316	257.448686
Actualizar directorio	754	439	615	603	512	599	686	605	912	427	586.16099

Tabla 2 - Relación tiempos de servicio operaciones, Event Triggered

Operación	1	2	3	4	5	6	7	8	9	10	media
Inicialización	10607	10586	10591	10606	10680						10613.8915
Manual Sample	1023	1010	1012	1012	1012	1019	1011	1020	1011	1034	1016.34829
Actualizar posición	53	92	565	31	89	480	44	137	20	49	56.3318522
Actualizar directorio	292	219	494	325	200	472	333	207	404	204	282.311581

Tabla 3 – Comparación tiempos de servicio

Operación	Time Triggered	Event triggered
Inicialización	10753.68	10613.89
Manual Sample	1270.45	1016.34
Actualizar posición	257.45	56.33
Actualizar directorio	586.16	282.311

Como se puede ver por los tiempos de servicio el método que menos tiempo consume es el método basado en eventos. Por lo cual es mejor para este sistema basarse en eventos.

Que el tiempo de servicio sea más pequeño para las operaciones de ambos componentes hardware nos ayudará a mejorar la sincronización entre componentes del sistema, por lo que es el método ideal para desarrollar aplicaciones que requieran de sincronización entre componentes.

2.4 Integración sensor laser y ampliación consola para el sistema

Antes de empezar con la integración del sensor al sistema anterior vamos a ver como es el sensor que utilizaremos y su especificación.

2.4.1 El sensor laser

EL sensor laser que disponemos para este proyecto es un sensor laser de la marca Acuity modelo AR200-50, este sensor se basa en la triangulación, que más adelante explicamos como funciona, para obtener las muestras. Es un sensor muy preciso pero con un espacio de trabajo muy limitado por sus características de funcionamiento.

2.4.1.1 Sistema de triangulación para el muestreo

Para la toma de muestras el sensor proyecta el laser sobre una superficie el reflejo es captado por un conjunto de lentes y focalizado sobre una cámara lineal, se procesa la imagen de la cámara y es lo que nos determina la distancia de la superficie (figura 25 a).

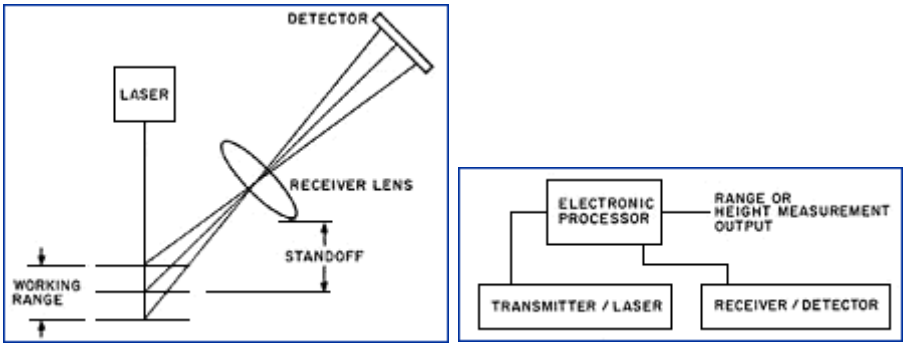


Figura 25 –a) Esquema funcionamiento sensor láser. B) Módulos del sensor laser [3]

- **Span o rango de trabajo:** Distancia de trabajo a la que el sensor medirá de forma fiable desplazamientos.
- **Standoff:** distancia entre el sensor y el centro del span.

Como se puede observar (figura 25 b) el sensor consta de tres módulos el emisor de luz, el receptor (formado por las lentes y el sensor CMOS) y el módulo que procesa los datos adquiridos para obtener el valor de la muestra final.

Para determinar la distancia se usa trigonometría, se conoce el ángulo (A) con el que se proyecta el laser y la distancia entre el laser y las lentes (d), el sensor CMOS nos da el ángulo con el que llega el laser a las lentes (B), con estos datos podemos ya saber a la distancia que se encuentra la superficie que refleja el laser. En el caso del sensor que utilizamos el ángulo A es de 90°, por lo que los calculas serán bastante simples (figura 26).

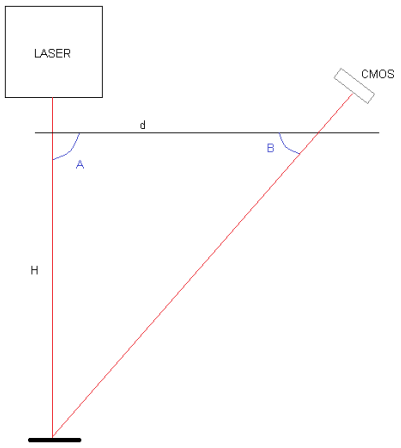


Figura 26 – Cálculo de la distancia

En cuanto a la resolución del sensor (vertical) nos viene dada por el tamaño del sensor CMOS el span, y las lentes que concentran el haz de luz reflejado sobre el CMOS.

Un factor importante que nos determina la resolución lateral del sensor es el diámetro del haz de luz emitido, ya que si el haz de luz es mayor que la superficie u objeto sobre el que vamos a utilizar el sensor las medidas que nos devuelva el sensor no serán fiables. En nuestro caso el haz laser tiene un diámetro de 50µm en el standoff y de 220µm al final del span (figura 27).

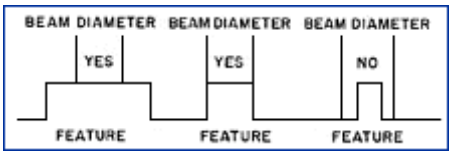


Figura 27 - Resolución lateral [3]

2.4.1.2 *Especificación y configuración del laser AR200-50*

En cuanto la especificación del sensor se puede consultar en manual de usuario [4] y en la siguiente tabla los elementos más básicos.

Tabla 4 – Especificación AR200-50 [5]

AR200 model	AR200-50
Span	2.00 [50.80]
Standoff to middle of span	1.67 [42 mm]
Linearity +/- 0.2% of span	0.004 [101.6 µm]
Resolution 0.03% of span	0.0006 [15.2 µm]
Laser spot size at Standoff, at ends of span	50 µm, 220 µm

Hay tres parámetros importantes de configuración que afectan significativamente al funcionamiento del sensor:

1. **Eliminación de la luz de fondo ON/OFF:** Cuando esta activada la eliminación de luz de fondo se toman dos imágenes una con luz y otra sin y se restan las dos imágenes para reducir los efectos de la luz ambiente, en esta situación el intervalo de muestreo máximo es de 600 muestras por segundo, en el caso de que este desconectada la eliminación de luz ambiente el intervalo será de 1250 muestras por segundo.
2. **Intervalo de muestreo:** el intervalo de muestreo se define en décimas de milisegundo. Utilizando 8 como parámetro el intervalo de muestro de definirá en 0.8 milisegundos o lo que es lo mismo 1250 muestras por segundo, el intervalo más amplio que se puede definir es de 5 segundos entre muestras (50000).
3. **Prioridad de muestreo:** este parámetro define la prioridad que tendrá el sensor para dar una medida puedes priorizar cumplir el intervalo de muestreo o la calidad de las muestras. Si la cantidad de luz reflejada por el objeto que se mide es baja el tiempo necesitado por el sensor para tomar la medida puede ser mayor que el intervalo especificado si se prioriza el intervalo de tiempo se devolverá 0 si no ha dado tiempo a tomar la medida por falta de luz.

Hay que tener en cuenta que el sensor nos devolverá medida 0 siempre que no pueda tomar la muestra ya sea por falta de tiempo, de calidad del haz de luz reflejado por culpa de las propiedades del material a escanear o porque el objeto este fuera del rango de acción del sensor.

También son configurables los parámetros relativos a la comunicación por el puerto serie.

2.4.1.2.1 Comunicación por el puerto serie

La comunicación entre este sensor y el PC es bastante sencilla, cuando el sensor ha tomado una muestra inmediatamente la manda por la línea serie al PC, si desde el PC se quiere mandar una instrucción al sensor se manda por la línea serie si es correcta el sensor la ejecutará de inmediato, en caso contrario el sensor no dará ningún tipo de información indicando que es una instrucción incorrecta. En cuanto a funcionamiento el laser tiene dos modos de funcionamiento automática o manual en manual el sensor espera la instrucción de tomar una medida, en modo automático el sensor va tomando medidas cada cierto intervalo de tiempo ininterrumpidamente.

2.4.2 Integración del sensor laser en la aplicación

Una vez vistas las características del sensor y sus parámetros configurables nos disponemos a integrarlo con el SCARA en la aplicación anterior. Para la integración tenemos que tener en cuenta los requisitos funcionales propios del sensor y los requisitos del sistema en conjunto.

El objetivo de integrar este sensor es dar al SCARA más percepción del entorno. Permitiendo que pueda desempeñar más funciones y ser más competitivo. Como ya habíamos dicho el objetivo del proyecto es poner en marcha un sistema para uso educativo y de investigación. Pero no hay que olvidar que este manipulador es preciso y rápido, con la integración de este sensor podría utilizarse el sistema para el control de calidad de piezas en una cadena de montaje. En cuanto a aplicaciones de uso educativo y de investigación, se podría aplicar para instruir en la integración de sensores o en la creación de sistemas hardware distribuidos por ejemplo.

2.4.2.1 Requisitos

Como requisitos funcionales relativos al sensor surgen los siguientes requisitos:

- Configuración del sensor.
  - Configurar el intervalo de muestreo.
  - Seleccionar la prioridad de muestreo.
  - Activar/desactivar la eliminación de luz ambiente.
- Modo de funcionamiento automático o manual.
- On/off muestreo automático.
- Tomar muestra, en modo manual.

En cuanto al sistema completo (SCARA, sensor) añadiremos el siguiente requisito:

- Almacenar la posición y la lectura del sensor en un archivo.
- Escanear una posición del terminal.

2.4.2.1.1 Configurar intervalo de muestreo

Cuando se está trabajando en modo automático el escáner toma muestra cada cierto intervalo de tiempo, este intervalo de tiempo es configurable y tiene que estar comprendido entre 8 y 50000.

2.4.2.1.2 Seleccionar prioridad de muestreo

Como se ha explicado en el punto anterior se puede priorizar entre tomar las muestras de forma más fiable, que puede tardar un poco más de lo esperado y modificar el tiempo entre muestras, o cumplir el intervalo de tiempo entre muestras. El usuario podrá configurar lo que más le interese en cada momento.

2.4.2.1.3 Activar/desactivar la eliminación de luz ambiente

El usuario tendrá la posibilidad de configurar si quiere que se elimine la luz ambiente en la toma de muestras o no. Por defecto estará desactivada ya que así se permite un menor intervalo de muestreo. En caso de que las muestras sean deficientes por el exceso de luz en el entorno el usuario podrá activar la eliminación de luz ambiente.

2.4.2.1.4 Modo de funcionamiento

Según en qué modo de funcionamiento este el sensor tomara muestras de forma ininterrumpida o esperara la orden del usuario para tomar una muestra, en modo automático se generará una grafica que nos permite ver la evolución de las muestras tomadas desde un intervalo de tiempo hasta ahora. El usuario tiene que ser capaz de cambiar el modo de funcionamiento del sensor.

2.4.2.1.5 On/Off muestreo automático

Una vez esta configurado el sensor para trabajar en modo automático el usuario puede activar o desactivar el muestreo.

2.4.2.1.6 Tomar muestra modo manual

Cuando se está funcionando en modo manual el usuario tiene que ser capaz de tomar muestras en cualquier momento, se tomara una muestra y se mostrara al usuario por pantalla la muestra obtenida.

2.4.2.1.7 Almacenar posición y lectura en un archivo

Un requisito indispensable para este sistema es asociar a cierta coordenada una lectura del sensor, para ello se almacenará las coordenadas del momento del muestreo y el valor del muestreo en un archivo cuando se requiera por el usuario. Tanto en modo manual como en modo automático.

2.4.2.1.8 Escanear una posición del terminal

Este requisito no es necesario para la primera aplicación puesto que ya se puede hacer con las herramientas que había antes de implementar el caso de uso del que es responsable este requisito, pero para futuras aplicaciones como la del escáner es una funcionalidad necesaria para el escáner.

2.4.2.2 Especificación y diseño

La especificación ya la hacemos conscientes de la estructura de la aplicación anterior y de los usuarios y agentes externos que tenemos, ahora se añade un actor al sistema, el sensor laser, y los dos usuarios que tenemos. Para esta parte de la aplicación los trataremos como un único usuario, puesto que ambos van a tener exactamente la misma interacción con el escáner y todos los requisitos son necesarios para ambos, por lo que en nuestro diagrama de casos de uso solo pondremos un usuario. Esta parte de la aplicación también se hizo primero basada en el método de encuesta y luego la pasamos al método basado en eventos, la especificación que hay a continuación solo corresponde a la última versión que se ha realizado.

2.4.2.2.1 Diseño de la interfaz

La interfaz que se propone a continuación servirá para los dos usuarios por lo que ira unida a las dos interfaces anteriores. La apariencia final será la de la siguiente figura.

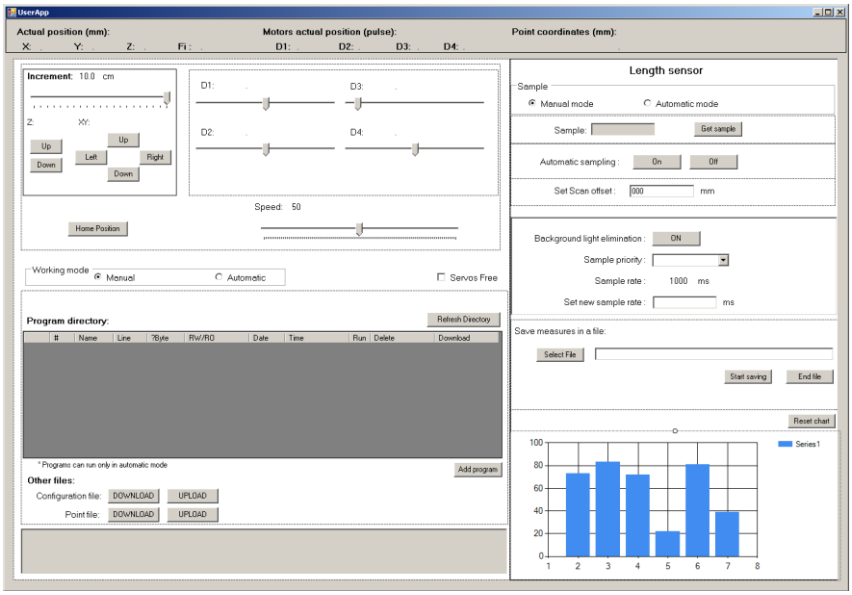


Figura 28 – Interfaz consola YK-640 y sensor de distancia

Como se puede ver la interfaz para la interacción con el sensor laser se organiza en siete secciones ordenadas verticalmente. En la primera sección se determina en qué modo está trabajando el sensor, según el modo se nos activara la segunda sección (modo manual) o la tercera (modo automático). La

sección del modo manual nos permite visualizar la muestra cogida bajo petición del usuario, en cuanto la sección del modo automático nos permite activar o desactivar el muestreo.

Las dos siguientes secciones nos permiten configurar el sistema y el sensor, el primer campo que se puede configurar es la separación entre el final del terminal del SCARA y el sensor laser para poder determinar la altura exacta de las muestras obtenidas por el sistema. En el siguiente apartado se pueden configurar los tres parámetros configurables del sensor que ya se han explicado anteriormente.

Las dos últimas secciones nos permiten almacenar las coordenadas del terminal y las muestras del sensor en un archivo y visualizar las últimas muestras tomadas por el sistema.

2.4.2.2.2 Especificación casos de uso y diseño de la aplicación

Para llevar a cabo los requisitos funcionales extraídos del análisis funcional vamos a desarrollar los siguientes casos de uso.

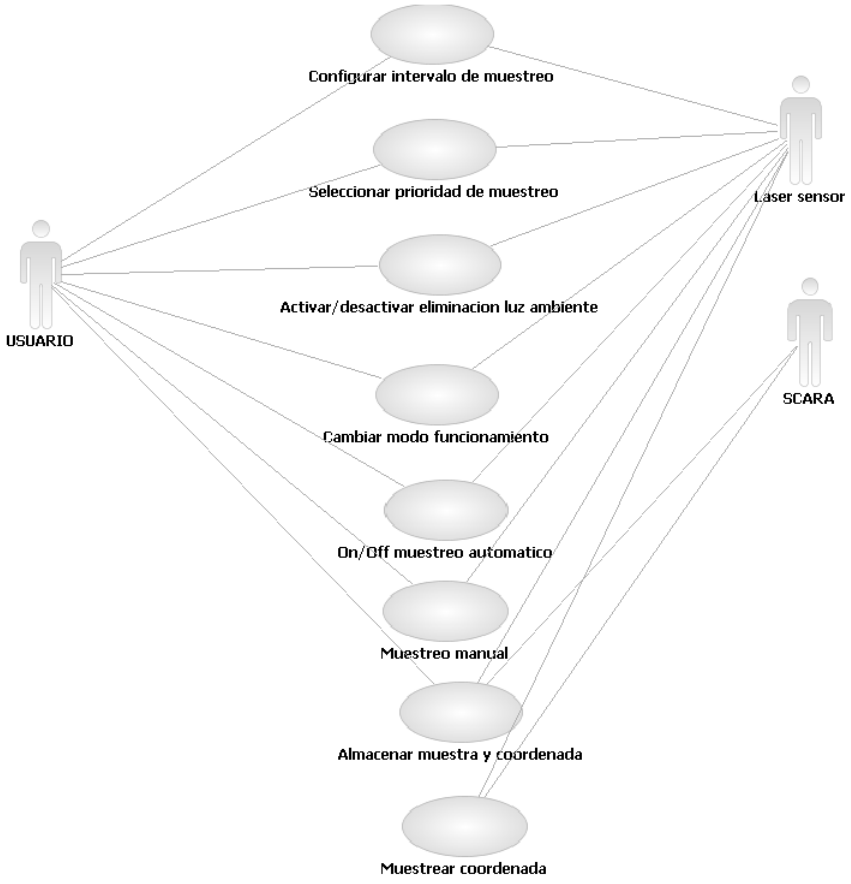


Figura 29 – Casos de uso sensor de distancia integrado en la consola YK-640

2.4.2.2.2.1 *Configurar intervalo de muestro*

En este caso se configura el intervalo de muestreo del sensor laser. El intervalo de muestreo puede estar entre 5000ms y 0.8ms, cuando el intervalo es muy bajo la eliminación de luz ambiente dejara de funcionar.

El usuario introducirá el nuevo intervalo de tiempo, si esta dentro del rango de tiempo aceptado se configurara el sensor laser con este intervalo.

2.4.2.2.2.2 *Seleccionar prioridad de muestreo*

Como se ha explicado anteriormente se puede configurar el muestreo automático para que priorice la calidad del muestro o el cumplimiento del intervalo de tiempo, en este caso no hay posibilidad a errores ya que la interfaz no da opción a cometerlos por lo que solo hay que pasar la orden a través del sistema hasta que le llegue al hardware.

2.4.2.2.2.3 *Activar/desactivar luz ambiente*

En la aplicación hay un botón que indica al usuario en qué estado se encuentra la eliminación de luz ambiente (activada/desactivada) y que permite modificarlo, al igual que en el caso anterior no hay que tener en cuenta la posibilidad de errores en esta interacción.

2.4.2.2.2.4 *Alternar modo de funcionamiento*

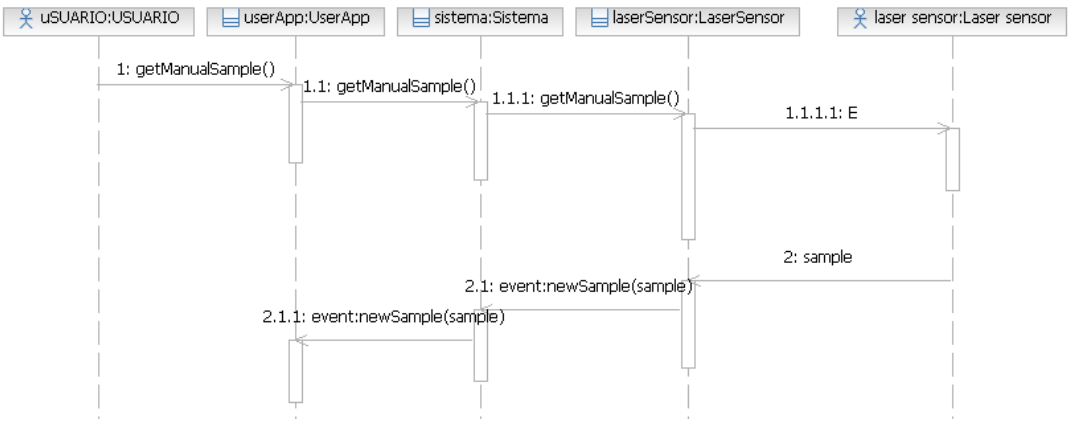
En este caso se cambia el modo de funcionamiento del sensor, de manual a automático o al revés. Como en los casos anteriores no hay errores que contemplar ni información que pasar al usuario, solo hay que hacer llegar la orden necesaria al hardware.

2.4.2.2.2.5 *On/off muestreo automático*

El usuario tiene la posibilidad de activar y desactivar el muestreo automático a su antojo, cuando el sistema está haciendo tomando muestras de modo automático las ira almacenando en la interfaz del sensor hasta que sean requeridas por otra parte del sistema para evitar el trafico constante de datos por todo el sistema.

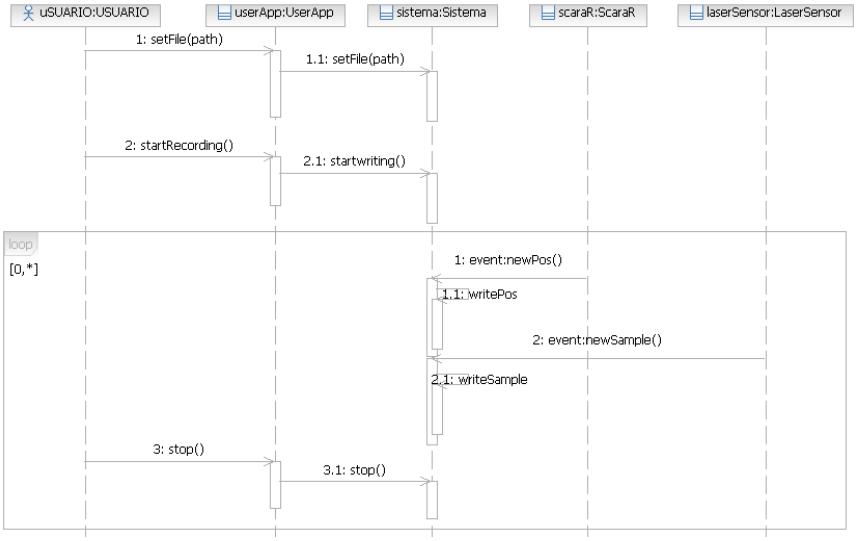
2.4.2.2.2.6 *Tomar una muestra de modo manual*

Con el sensor funcionando en modo manual el usuario podrá dar la orden que se tome una muestra, en este caso cuando la muestra esté lista se le transmitirá al usuario mediante un evento.



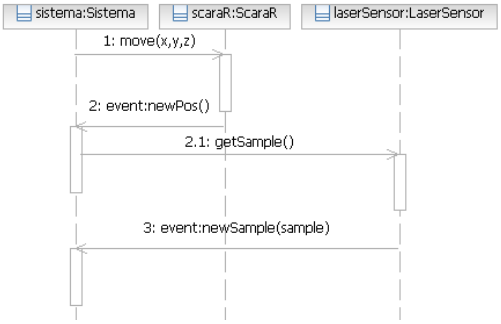
2.4.2.2.2.7 *Almacenar muestras y coordenadas en un archivo*

Mientras se quieran escribir los datos en un archivo se irán escribiendo las coordenadas del manipulador y las muestras del sensor en un archivo de forma que entre dos coordenadas se escriban todas las muestras que se han tomado durante el movimiento del terminal de una coordenada a la otra.



2.4.2.2.2.8 *Muestreo de una coordenada*

Como se ha explicado este caso de uso es necesario para futuras aplicaciones del sistema, pero no para la aplicación actual ya que la aplicación actual era capaz de hacer esta acción con las herramientas que ya tenía antes de añadir este caso de uso.



2.4.2.2.3 Diagrama de clases

En cuanto al diagrama de clases el único cambio que hay es la aparición de la clase que se encarga de gestionar el sensor laser.

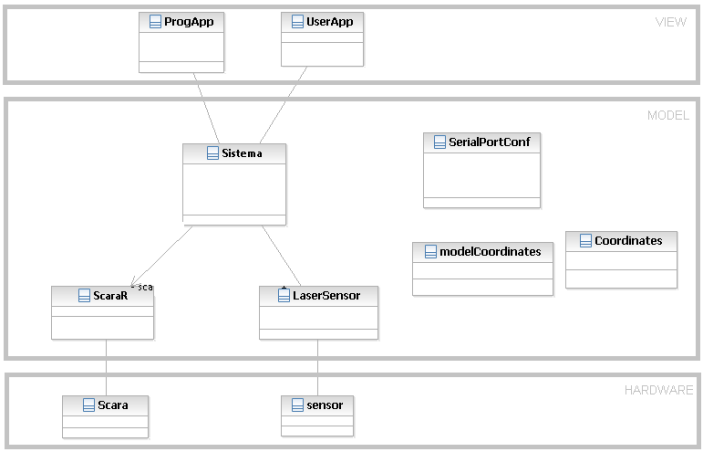


Figura 30 – Diagrama de clases aplicación consola YK-640 con el sensor de distancias ya integrado

2.4.2.3 Evaluación

Para probar esta parte de la aplicación se ha ido probando la funcionalidad y la usabilidad de cada uno de sus casos de uso, el resultado ha sido satisfactorio en todos los puntos menos en el de la escritura de datos en un archivo una mejora que queda pendiente para esta aplicación es la mejora del almacenado de los datos en el archivo sobretodo en cuanto a su formato. Otro aspecto que en un futuro se podría modificar el diseño visual de la aplicación con la intención que sea más atractiva y más intuitiva.

2.5 Integración de un sensor de fuerza

La integración de este sensor se escapa del desarrollo del proyecto, pero es importante tener una visión de la dirección que puede tomar el proyecto.

La integración de un sensor de fuerza basado en galgas extensiométricas puede tener diversas aplicaciones. Como podrían ser el escaneado de modelos por contacto, el guiado manual del SCARA o la inserción de ejes y pines controlando la presión ejercida para no dañar las piezas en caso de errores entre muchas otras.

Otra ventaja que nos daría el sensor de fuerza es el de hacer un guiado inteligente evitando zonas peligrosas en caso de estar en un entorno con más obstáculos en el espacio de trabajo del manipulador.

La integración de un sensor de fuerza abriría muchos campos de investigación y posibilidades de nuevos estudios así como creación de nuevas aplicaciones para futuros usuarios del sistema montado en este proyecto.

La aplicación más inmediata y que mejor se acopla con la configuración actual del sistema sería el guiado manual del SCARA. Con el guiado manual podríamos conseguir un movimiento del SCARA preciso y variable en velocidad según la fuerza aplicada. Además de cómodo ya que el movimiento no sería costoso al no tener que aplicar fuerza, como es el caso actual cuando se desconectan los servos.

El diseño seguiría la misma línea, una interfaz que se comunicase con el hardware y con el controlador del sistema completo que gestionaría y sincronizaría el sensor de fuerza con el resto de componentes.

2.6 Integración de un sistema de visión por computador

Otro campo interesante a aplicar, que también está fuera del desarrollo de este proyecto sería el de la introducción de un sistema de visión por computador, se puede ver una posible aplicación en el artículo *Robotic manipulation of ophtalmic lenses assited by a dedicated visión system,* [6].

Una posible aplicación sería la localización automática de múltiples objetos a escanear. Por lo que no sería necesario, por parte del usuario, dar las coordenadas del objeto a escanear.

La integración del sistema de visión puede tener muchas aplicaciones y está fuera del proyecto, pero creo que es un campo que sería interesante para futuros proyectos, y con un gran número de aplicaciones.

2.7 Aplicación escáner superficies

La finalidad de esta aplicación es proporcionar una aplicación que escanee superficies y cree un modelo, dando una utilidad al sistema creado anteriormente. La creación de un escáner nos da principalmente dos problemas la adquisición de los datos y el modelado de la superficie escaneada.

Para la adquisición de los datos hoy en día hay diversos métodos para hacerlo bastantes más avanzados pero con el hardware que tenemos se basara en hacer un barrido por encima de la superficie a escanear tomando medidas con el sensor laser de forma que para cierta coordenada tendremos una medida dada por el sensor con lo que podremos crear las coordenadas del modelo y a partir de estas sacar una malla de triángulos que represente la superficie escaneada. Para dar mayor funcionalidad al escáner la aplicación debe permitir hacer un primer escaneado de poca densidad y luego permitir al usuario seleccionar las zonas de interés en las que quiera realizar un escaneado más preciso, es decir de mayor densidad, para así acelerar el proceso de escaneo de forma que solo se escaneen de forma precisa las zonas de interés.

En cuanto al modelado de los datos obtenidos para permitir al usuario visualizar lo escaneado y trabajar con el modelo se hará utilizando OpenGL y creando modelos exportables a otro programas para trabajar con modelos como pueden ser Blender, Rhino o muchos otros.

Para crear los modelos se usaran dos algoritmos distintos el primer algoritmo solo será útil para los primeros barridos que se hagan con el escáner, ya que para que funcione es necesario que los puntos que se van a modelar sean representables por una matriz, por lo que debe ser una distribución de puntos uniforme por lo que si hay zonas escaneadas con mayor densidad no será posible aplicar este algoritmo. La idea principal del algoritmo es ir recorriendo la matriz de puntos creando los triángulos de la malla con los puntos adyacentes.

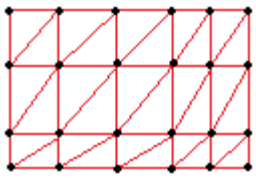


Figura 31 – Triangulación de una matriz

Este algoritmo solo puede hacerse es un único recorrido de la matriz por lo que tendrá un coste muy bajo O(n) y será bastante rápido comparado con otros algoritmos de triangulación.

La reconstrucción de superficies a partir de puntos no distribuidos uniformemente es un proceso bastante complicado. Hay que tener en cuenta a parte de la distribución no uniforme, normalmente hay ruido debido a la imprecisión del muestreo y los errores del escáner.

Para solventar el problema de modelado de estas nubes de puntos no uniformes obtenidas por el escáner hemos estudiado diferentes algoritmos de reconstrucción de superficies. Hemos llegado a la

conclusión que utilizaremos un algoritmo basado en la triangulación de Delaunay con coste O(n log n) en las implementaciones más rápidas. Ya que este algoritmo nos permite hacer la triangulación sin tener en cuenta las normales de los puntos y nos aporta una triangulación más apropiada para la representación de superficies.

2.7.1.1 Algoritmo de reconstrucción de superficies

Como se ha comentado anteriormente utilizaremos para la creación de la malla de puntos utilizaremos un algoritmo basado en la triangulación de Delunay.

La idea de la triangulación de Delunay es dada una nube de puntos en el plano, hallar una triangulación en la que los puntos próximos estén conectados entre sí por una arista. Lo que producirá una triangulación lo más regular posible.

Para la triangulación de Delunay cumple las siguientes dos propiedades. Siendo P el conjunto de puntos del plano que vamos a triangular.

Propiedad 1: Tres puntos pi,pj,pk pertenecientes a P son vértices de la misma cara de la triangulación resultante, si y solo si, el circulo que pasa por los puntos pi,pj y pk no contiene otros puntos de P en su interior (figura 32).

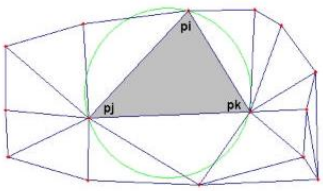


Figura 32 – Propiedad 1 Triangulación Delunay [7]

Propiedad 2: Dos puntos pi y pj pertenecientes a P forman un lado de la triangulación, si y solo si, existe un círculo que contiene a pi y pj en su circunferencia y no contiene en su interior ningún punto de P.

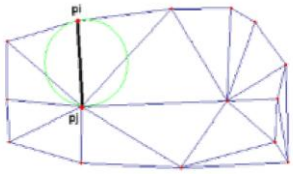


Figura 33 – Propiedad 2 Triangulación Delunay [7]

En una triangulación de Delunay diremos que una arista es ilegal si dados los dos triángulos adyacentes a esta arista, al cambiar los vértices que forman la arista aumenta el ángulo mínimo de los triángulos adyacentes (figura 34).

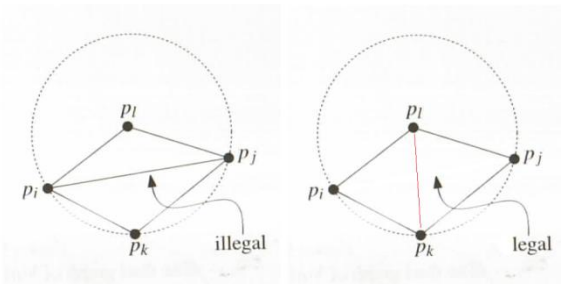


Figura 34 – Arista ilegal y arista legal [8]

Con una arista ilegal se puede hacer un flip, es el cambio de arista ilegal a legal que se ve en la figura 34.

Teniendo en cuenta y aplicando los conceptos explicados hasta ahora la triangulación de Delunay se podría obtener fácilmente de una triangulación cualquiera.

El algoritmo a seguir para obtener la triangulación de Delunay sería el siguiente:

***Entrada**: Un conjunto finito de puntos en el plano.*
***Paso 1**: Sean p1, p2 y p3 tres puntos tales que P está contenido en el triángulo que forman.*
***Paso 2**: Inicializar T como una triangulación de un único triángulo (p1-p2-p3).*
***Paso 3**: Realizar una permutación cualquiera p1, p2,..., pn de P.*
***Paso 4**: for r=1 to n*
***Paso 5**: hacer (* Insertar pr en T *)*
***Paso 6**: Encontrar un triángulo pi-pj-pk de T que contenga a pr*

***Paso 7**: si pr cae en el interior del triángulo pi-pj-pk*
***Paso 8**: entonces añadir aristas desde pr a los tres vértices de pi-pj-pk, dividiendo este triángulo en tres*
***Paso 9**: legaliza _ lado (pr, pi-pj, T)*
***Paso 10**: legaliza_lado (pr, pj-pk ,T)*
***Paso 11**: legaliza _ lado (pr, pk-pi,T)*
***Paso 12**: en caso contrario (* pr cae encima de uno de los lados del triángulo pi-pj pk, por ejemplo el lado pi-pj *)*
***Paso 13**: Añadir aristas desde pr a pk y al tercer vértice pl del otro triángulo que comparte la arista pi-pj, de esta forma dividimos los dos triángulos que comparten la arista pi-pj en cuatro triángulos.*
***Paso 14**: legaliza _ lado (pr, pi-pl,T)*
***Paso 15**: legaliza _ lado (pr, pl-pj,T)*
***Paso 16**: legaliza _ lado (pr, pj-pk,T)*
***Paso 17**: legaliza _ lado (pr, pk-pi,T)*
***Paso 18:** descartar p1, p2 y p3 y todas las aristas que parten de ellos de T*
***Paso 19**: devuelve T*
***Salida**: La triangulación de Delaunay del conjunto de puntos*

El algoritmo legaliza_ lado, comprueba si la legalidad de la arista, en caso de que sea necesario hade un flip.

***Paso 1**: El punto que se está insertando es pr, y pi-pj es la arista de T a la que puede ser necesario hacer un intercambio de aristas*
***Paso 2**: si pi-pj es no válida*
***Paso 3**: entonces, sea pi-pj-pk el triángulo adyacente a pr-pi-pj compartiendo la arista pi-pj*
***Paso 4**: (* Flip pi-pj *) Reemplazar pi-pj por pr-pk*
***Paso 5**: legaliza _ lado (pr, pi-pk, T)*
***Paso 6**: legaliza _ lado (pr, pk-pj, T)*

2.7.2 Diseño y especificación de la aplicación

De lo que se ha podido observar anteriormente se desprenden los siguientes requisitos que a continuación describiremos.

2.7.2.1 Requisitos

- Escanear una nueva superficie.
- Crear un modelo de los datos escaneados.
- Visualizar el modelo de la superficie.
- Visualizar los puntos obtenidos por el escáner.
- Volver a escanear el modelo o parte de él.
- Rehacer un modelo a partir de los datos guardados.
- Configurar los puertos serie del sistema.
- Exportar el modelo creado a archivos útiles para otros programas de modelado.

2.7.2.2 Escanear una nueva superficie

Este requisito es la principal funcionalidad de esta aplicación y la que da uso al sistema hardware creado anteriormente, el usuario podrá escanear una superficie u objeto y visualizar el modelo creado a partir de los datos obtenidos por el escáner.

Para escanear la superficie el usuario tiene que ser capaz de definir el área de escaneado y la precisión con la que quiere que se escanee la superficie.

2.7.2.2.1 Definición del área de escaneado

La aplicación escaneara un área definida por el usuario, para definir esta área se hará mediante la selección de un conjunto de coordenadas, como mínimo dos, que delimitarán el área a escanear por el escáner.

El conjunto de puntos seleccionado por el usuario definirá el área a escanear. Esta área debe poder ser de cualquier forma para acotar al máximo la superficie a escanear, para evitar escanear partes de la superficie que no interesan o están vacías. Un requisito obvio para esta funcionalidad es que el área seleccionada por el usuario no se salga del radio de acción del SCARA.

Para definir el área de escaneado el usuario tiene que poder mover el SCARA con facilidad para poder seleccionar los puntos que marcan el perímetro, por lo que la aplicación necesita de las funcionalidades básicas de movimiento del SCARA.

2.7.2.2.2 Tipos de escaneado

La aplicación debe dar al usuario distintas posibilidades de escanear una superficie, puesto que el sensor laser tiene un campo de trabajo muy pequeño, para dar la máxima funcionalidad a la aplicación.

En una primera versión se darán tres posibilidades para realizar el escaneado:

- Escaneado a una altura fija por puntos: el escáner ira punto por punto de la superficie escaneando la superficie a una altura fija.

- Escaneado por puntos con altura variable: el espacio de trabajo del sensor de distancia es limitado, para permitir el escaneo de superficies con una altura mayor que lo que nos permite el sensor, el sistema ira ajustando la altura del terminal de forma automática para ajustarse a la altura del modelo.
- Escaneado por filas: será un escaneo a una altura fija pero no será por puntos, se escaneara por filas escaneando de forma automática y luego se calculara a qué posición pertenece cada muestra del sensor según la velocidad de muestreo y del SCARA y la longitud de la fila. Es el método que da menos precisión en el escaneado pero mejora mucho la velocidad del escaneo.

2.7.2.2.3 Resolución

El usuario tiene que poder escoger la resolución del escaneado, la resolución máxima que nos puede dar el escaneado es la resolución que nos proporciona el SCARA que es el que situara el sensor laser sobre la superficie a escanear. La resolución la daremos en número de muestras por centímetro cuadrado y será como máximo de 900 muestras por centímetro cuadrado, aunque no en todos los tipos de escaneado se podrá llegar a esta resolución.

2.7.2.3 Modelado a partir de los datos obtenidos

La aplicación a partir de los datos obtenidos por el escáner creara un modelo, de la nube de puntos que se obtiene se creara una malla de triángulos, mediante los algoritmos explicados anteriormente.

2.7.2.4 Visualización del modelo

El usuario podrá visualizar la malla de triángulos creada por la aplicación. El modelo se creara utilizando OpenGL. El usuario tendrá la posibilidad de interactuar con el modelo visualizado para inspeccionarlo a fondo, podrá rotarlo, desplazarse por él y hacer zoom para verlo con más detalle.

2.7.2.5 Volver a escanear el modelo

Una vez escaneado el modelo el usuario puede definir una zona del modelo que quiere que se vuelva a escanear. Este segundo escaneo de una zona en concreto del modelo se tiene que poder hacer con más resolución que el escaneo inicial, permitiendo así un primer escaneo rápido de toda la superficie y un posterior escaneo más preciso de las zonas de interés.

Esto solo se podrá hacer mientras el modelo no se haya movido de su ubicación inicial puesto que sino el sistema no será capaz de ubicar correctamente la zona a escanear por segunda vez.

2.7.2.6 Configuración de los puertos serie

El nombre de los puertos variara según el PC que se conecte al sistema hardware por lo que la aplicación tiene que ser capaz de soportar estos cambios para ello será necesario que los usuarios puedan reconfigurar los puertos serie a los que se conectará el PC.

2.7.2.7 Exportar un modelo creado por la aplicación para su utilización en programas

Para que la aplicación sea totalmente funcional se debe dar la posibilidad que los modelos obtenidos por esta sean exportables a otras plataformas de modelado y otros programas de modelado como puede ser 3dStudio, Blender. Para ello los modelos creados daremos la posibilidad de exportarlos a archivos .OBJ que es un tipo de fichero ampliamente soportado por los programas de modelado.

2.7.3 Especificación y diseño

2.7.3.1 Usuarios y actores

Para esta aplicación solo hay un tipo de usuario, el usuario que quiere escanear una superficie con el escáner. Para los usuarios debe ser una aplicación fácil de utilizar y funcional que cubra todos los posibles requisitos que requiere un escáner de este tipo.

Como actor del sistema a parte del usuario visto anteriormente tendremos el sistema hardware que se encargara del escaneado de la superficie que deseemos escanear que estará gestionado por el sistema creado en la primera parte del proyecto.

2.7.3.2 Prototipo aplicación y mapa navegacional

Para cumplir las funcionalidades explicadas en el punto anterior hemos diseñado la aplicación de la siguiente manera. A continuación se muestran las distintas interfaces de las que constara la aplicación y una explicación de cómo utilizarlas, además de una explicación de cómo sería la navegación entre las distintas interfaces.

2.7.3.3 Interfaz inicial

La interfaz inicial será el portal de entrada a la aplicación, nos permitirá configurar los puertos serie del sistema para que la comunicación con el hardware sea satisfactoria y nos llevara a las demás interfaces para poder realizar el escaneado y la visualización de superficies ya escaneadas.

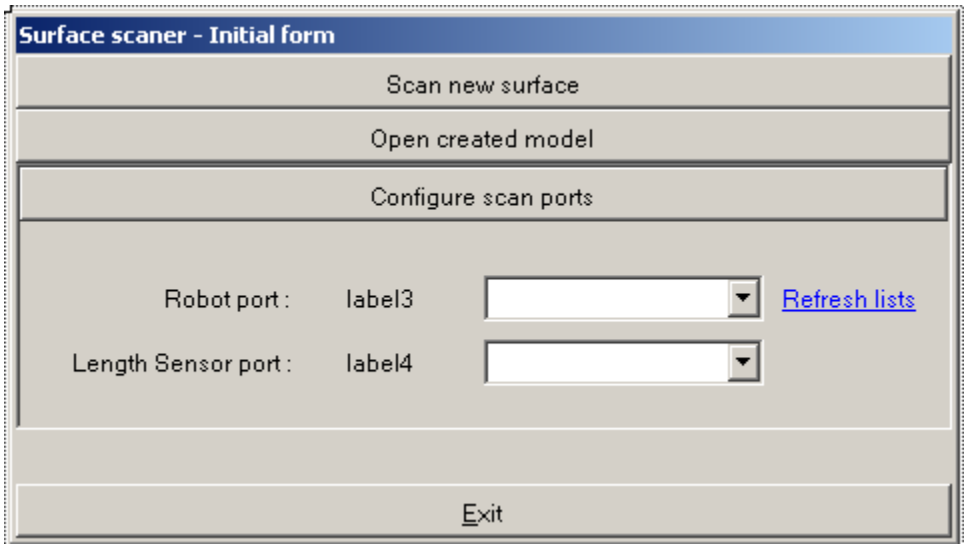


Figura 35 – Interfaz inicial aplicación escaner

Como se puede observar es una interfaz bastante sencilla, en la que se nos permite configurar los puertos serie a los que se tiene que conectar el sistema mediante dos listas donde se mostraran los puertos disponibles en el PC, en caso de que se haya conectado algún componente después de haber inicializado la aplicación se podrá actualizar las listas de puertos disponibles pulsando en el campo de texto “Refresh lists”, para que la configuración se finalice hay que pulsar el botón “Configure scan ports”.

2.7.3.4 *Interfaz escaneado*

Esta es la interfaz que nos permitirá delimitar el área a escanear, darle un nombre a la superficie para poder identificarla y seleccionar el tipo de escaneado entre los posibles.

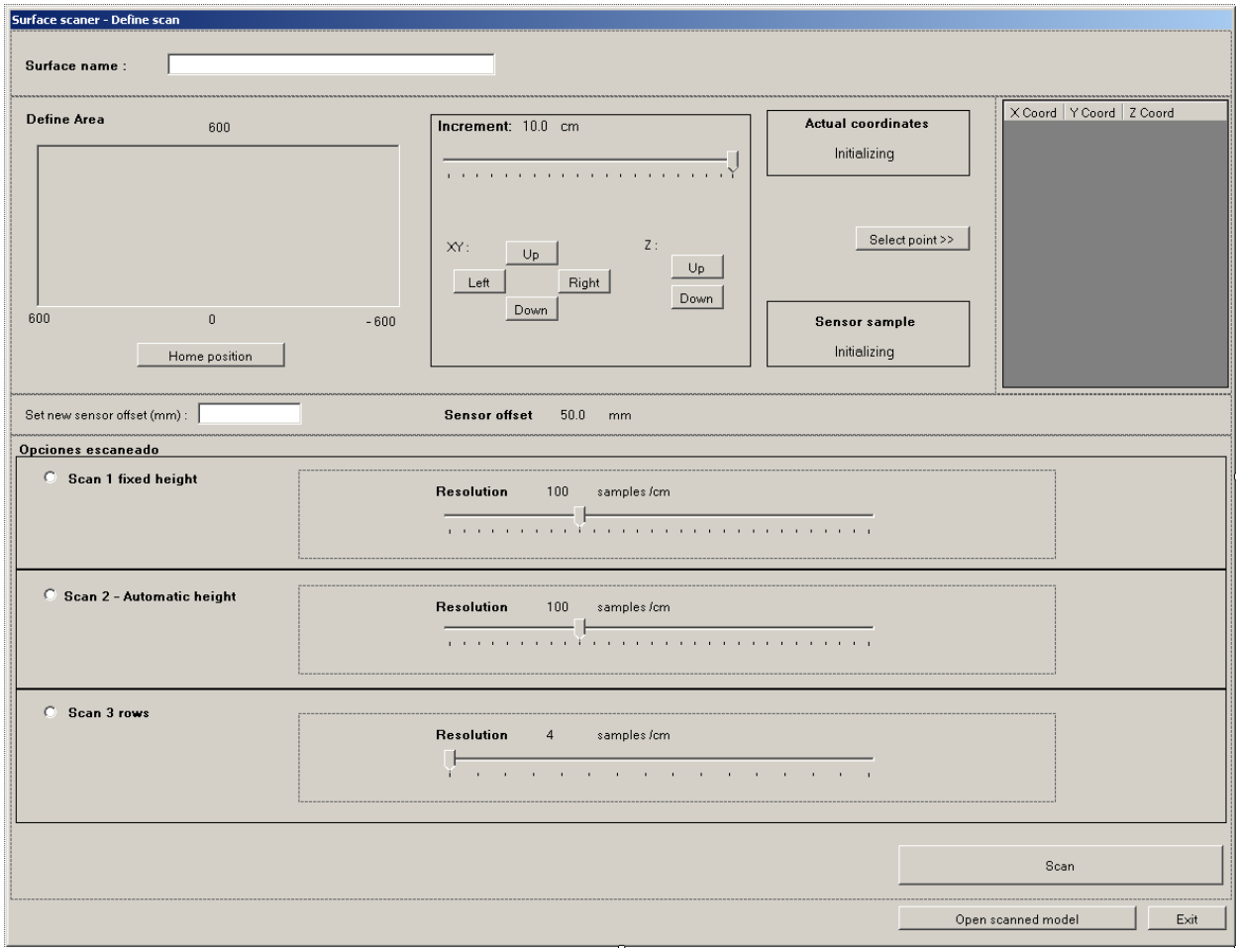


Figura 36 - Interfaz escaneado aplicación escáner

Lo primero que nos encontramos en esta interfaz es un campo de texto para darle nombre a la superficie, a continuación hay una sección para definir las coordenadas que delimitaran el area a escanear. A continuación hay un campo de texto que nos permite definir la distancia entre el final del terminal y el sensor laser, es necesario por que es una distancia que se puede modificar a antojo del usuario. Por último antes de escanear hay que decidir que tipo de escaneado se va ha hacer y la resolución deseada,en caso de escanear por filas la resolución esta limitada por la velocidad del hardware y del sensor de distancias. En caso de que el usuario no quiera escanear una nueva superficie puede abrir el modelo de una superficie ya escaneada desde esta interfaz o cerrar la interfaz y volver al menu principal.

2.7.3.5 *Interfaz visulizador de superficies escaneadas*

Con esta interfaz se nos permitirá la visualización de la superficie escaneada y la interacción con el modelo que se visualiza, además si acabamos de escanear la superficie se nos permitirá seleccionar zonas en las que se tiene que volver a escanear porque ha habido errores o para obtener una mayor resolución en esas zonas.

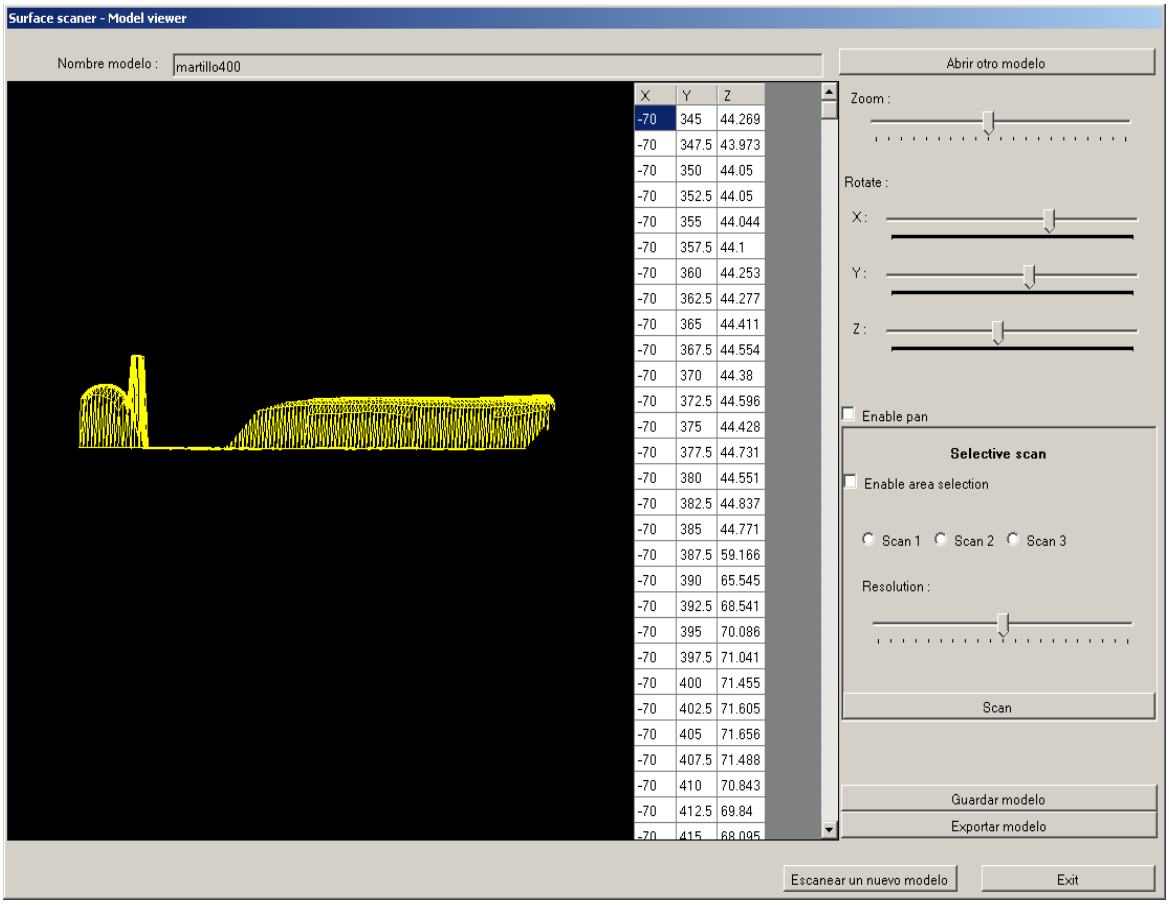


Figura 37 – Interfaz visualización de superficies

Esta interfaz está dividida en tres secciones ordenadas verticalmente, en la primera sección la de la parte superior de la interfaz te muestra el nombre de la superficie que se está visualizando y te da la opción de abrir un modelo ya escaneado. En la sección inferior te da la opción de escanear un nuevo modelo o de cerrar la aplicación y volver al menú principal.

En la sección intermedia se visualiza el modelo 3d de la superficie y una tabla con todas las coordenadas de los vértices del modelo. Se permite al usuario interactuar con el modelo, rotarlo sobre cualquier eje de coordenadas mediante sliders al igual que hacer zoom sobre él. Para desplazarse sobre la imagen bastara con habilitar el pan, con el check box que pone “Enable pan”, y arrastrar el cursor apretando el botón izquierdo del mouse sobre el modelo.

También se nos permite exportar el modelo aun archivo de tipo de archivo que permita abrir el modelo en otros programas de modelado y guardar el modelo.

Así como definir una zona sobre el propio modelo para volver a escanearla, definiendo otra vez el tipo de escaneado y la resolución con la que se quiere escanear la zona. Para usar esta funcionalidad el modelo tiene que ser un modelo recién escaneado para asegurar que el escáner es capaz de encontrar el área seleccionada sobre el modelo de la superficie en la superficie real. El área seleccionada se quedara marcada con otro color para que el usuario vea que área de la superficie se va a volver a escanear.

2.7.3.6 Casos de uso y diagramas actor-sistema

Con el diseño de las interfaces y con los requisitos funcionales que tenemos que cubrir nos quedan definidos los casos de uso que cubriremos con nuestra aplicación que podemos ver en el siguiente diagrama.

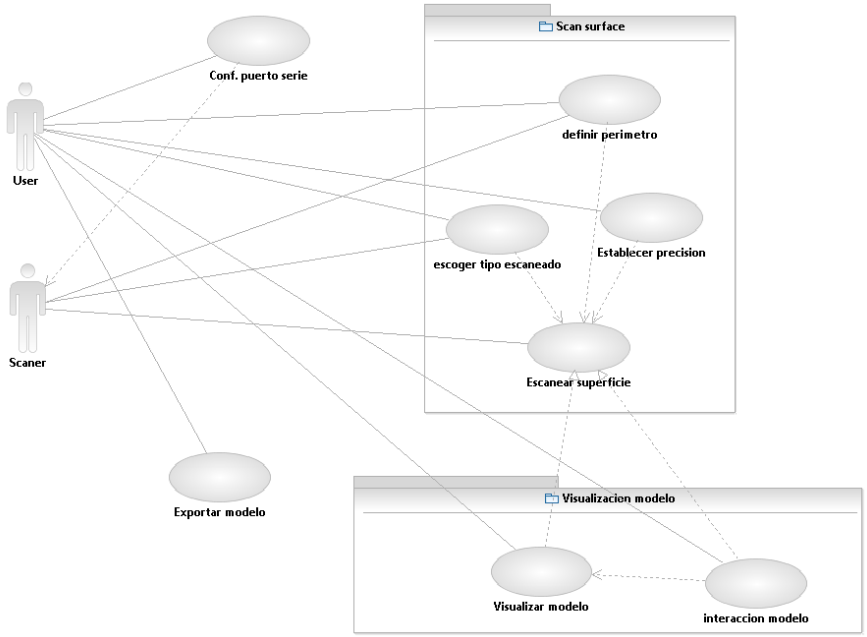


Figura 38 – Casos de uso de la aplicación escaner

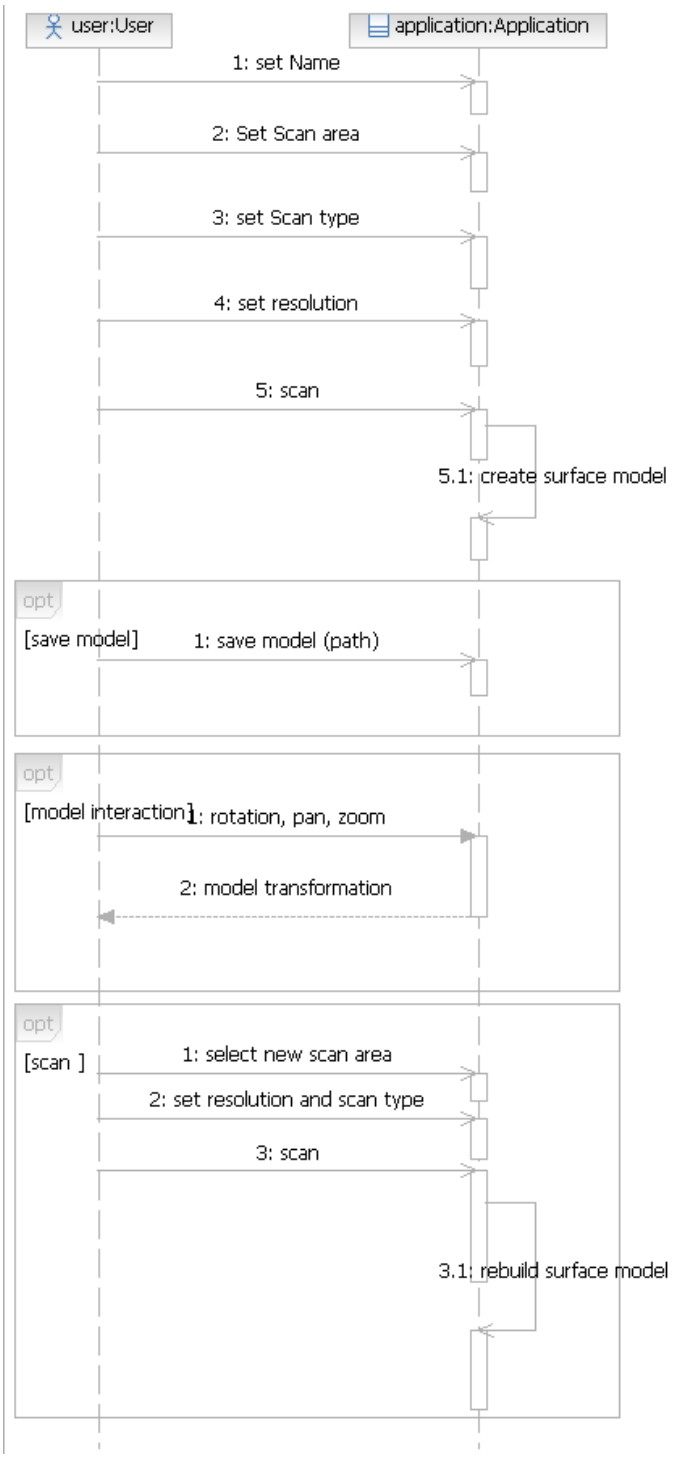
2.7.3.6.1 Configuración de los puertos serie

El usuario introduce los nombres de los puertos serie que utiliza el sistema hardware y pulsa el botón de configurar puertos. Para seleccionar los nombres el usuario los tiene que seleccionar de dos listas desplegables que hay en la interfaz inicial de la aplicación.

Esta operación se realizará en un único paso configurando los dos puertos serie de manera simultánea.

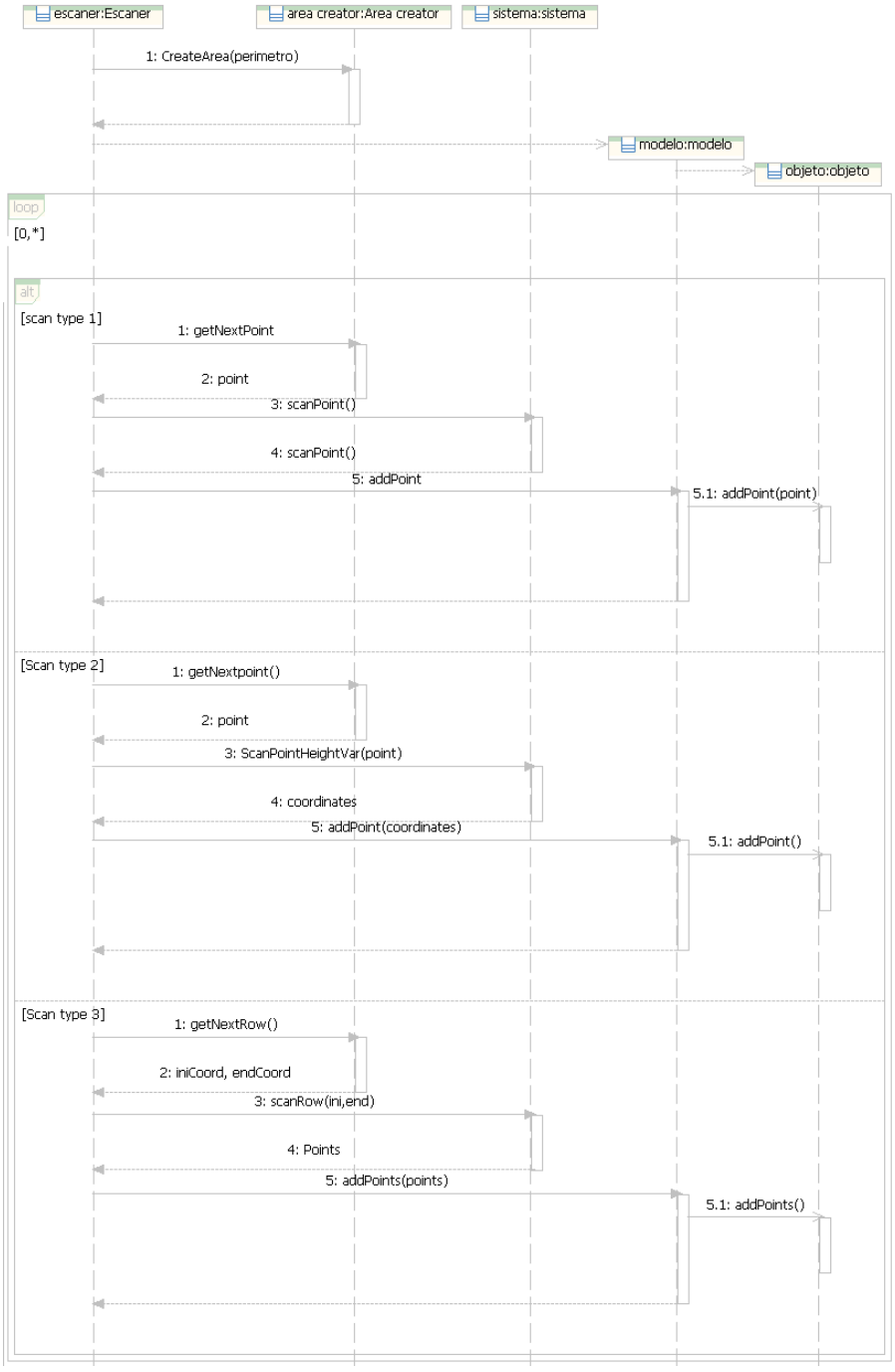
2.7.3.6.2 Escanear una nueva superficie

Para escanear una nueva superficie primero hay que definir el nombre que se le asignara, luego definir el área de escaneado junto al tipo de escaneado y la resolución deseada. Una vez se haya escaneado la superficie se visualizará en la siguiente interfaz. Entonces se podrá volver a escanear alguna zona con mayor una mayor resolución si se desea.

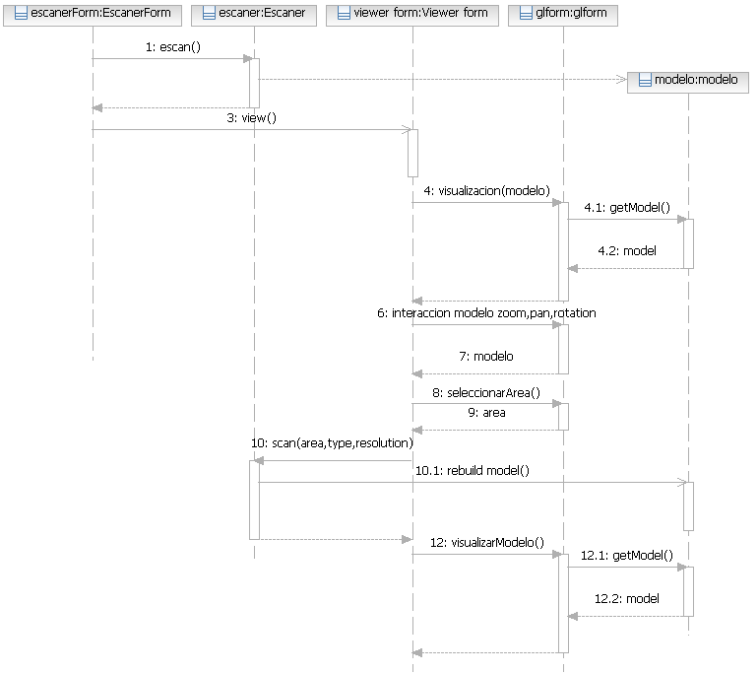


Las funciones más importantes que se extraen de este caso de uso son las de escanear la superficie, y las dos de construir los modelos gráficos. Como hemos dicho anteriormente el primer modelado no se hará con el mismo algoritmo que el segundo. El primero será un barrido por la matriz de puntos generando las caras del modelo, y el segundo será el algoritmo de reconstrucción de superficies explicado anteriormente.

A continuación vemos el diagrama de secuencia de la operación de escaneo. Durante la ejecución de esta operación se crea el modelo de la superficie, entonces se pasaría a la siguiente interfaz donde se visualizaría el modelo y en este caso nos dejaría volver a escanear una zona determinada de la superficie.

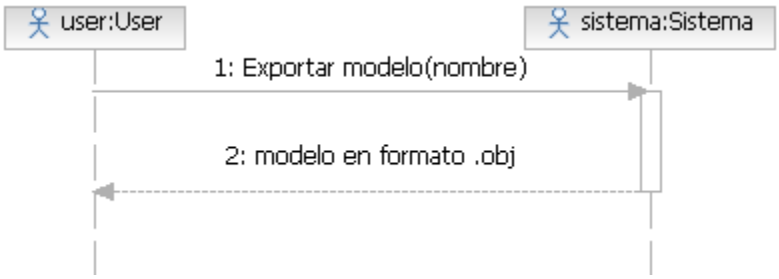


El siguiente diagrama de secuencia pertenece a la visualización de un modelo con y la selección de una zona para el nuevo escaneado, para ello el usuario seleccionará la nueva zona a escanear con el ratón sobre el modelo de la superficie.



2.7.3.6.3 Exportar el modelo

El usuario podrá exportar los modelos creados por la aplicación para poder utilizarlos en otros programas, como por ejemplo programas de modelado, para ello se exportarán como ficheros .OBJ.



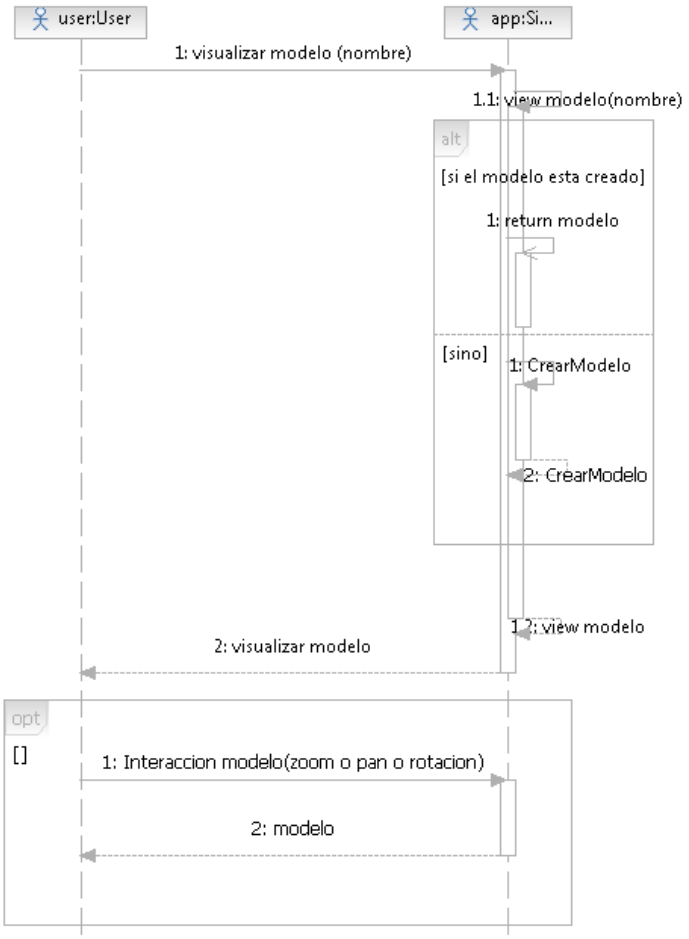
Esta operación se hará desde la interfaz de visualización del modelo. Se puede hacer en cualquier momento para permitir tener la evolución del modelo durante todo el proceso de escaneado (los distintos escaneados).

Dada la estructura de datos que se ha utilizado para almacenar el modelo en la aplicación la exportación a archivos del tipo OBJ será trivial.

2.7.3.6.4 Visualización del modelo

En este caso de uso se incluye la visualización del modelo y la interacción con este por parte del usuario para poder inspeccionar el modelo desde el mismo programa dándole la posibilidad de rotar el modelo y moverse por él así como hacer zoom.

Durante la visualización del modelo el usuario podrá interactuar con él, con el fin de poder inspeccionarlo a fondo. Pudiendo así definir las nuevas zonas de escaneado o inspeccionar la superficie para la búsqueda de imperfecciones o errores.



Será el modulo de OpenGL, glForm, el que se encargue de hacer la visualización del modelo.

2.7.3.7 Diagrama de clases

Del diseño que se ha hecho de la aplicación sale el siguiente diagrama de clases.

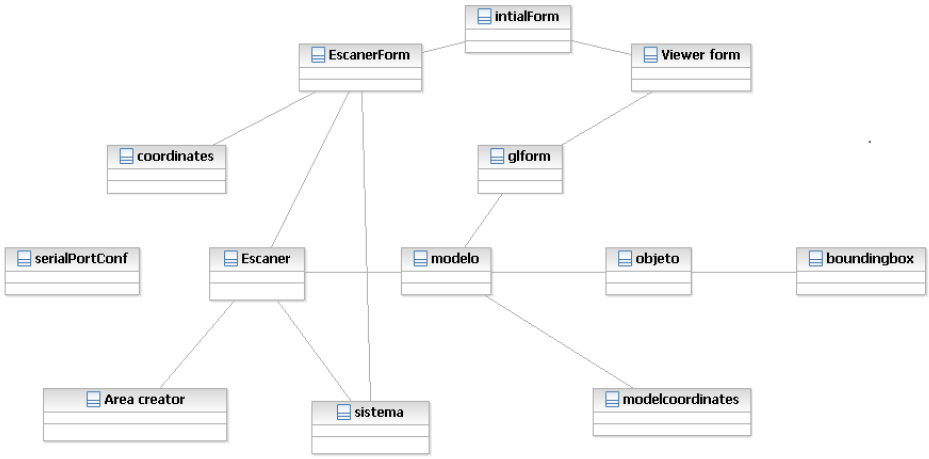


Figura 39 – Diagrama de clases aplicación escaner

Las clases EscanerForm, initialForm, y viewerForm corresponden a los controladores de las vistas son clases que únicamente se centran en el funcionamiento de las interfaces y en las llamadas a las funciones de la aplicación.

La clase escáner es la clase que se encargara del control del escaneado. Utilizando el sistema creado anteriormente (en este diagrama no hemos incluido las dos interfaces del hardware) y la clase Area creator. Esta última clase se encarga de calcular el punto o fila siguiente a escanear, a partir del perímetro y la resolución determinados por el usuario.

El escaneado creará un modelo, el modelo contendrá un objeto que es la estructura de datos necesaria para poder representar el modelo de la superficie en 3D. Para la visualización de la superficie utilizaremos openGl, la clase glform es la clase que se encarga de gestionar y crear el modelo en openGl y de todos los tratamientos de esta para poder interactuar con el usuario.

2.7.4 Evaluación

Como evaluación de esta aplicación no la podemos hacer puesto que el desarrollo de la aplicación no se ha finalizado. Pero evaluaremos la parte que se ha completado.

En cuanto a la configuración de los puertos serie y la interfaz inicial son completamente funcionales y usables, no permiten al usuario que cometa errores de forma fácil, por lo que podemos dar esta parte de la aplicación por finalizada.

En cuanto a la selección del área de escaneado y configuración de los parámetros de escaneado funciona correctamente, aunque sería conveniente en posteriores fases de desarrollo mejor un poco su usabilidad y apariencia, permitiendo una selección del área de escaneado más rápida. En un futuro se podría mejorar con alguno de los dos sensores introducidos anteriormente, permitiendo guiar al robot manualmente, o dándole autonomía al sistema para encontrar el mismo los objetos con un sistema de visión.

En cuanto a los diferentes tipos de escaneados solo está totalmente terminado el escaneado por puntos a una altura fija, falta ajustar el escaneado por filas y el otro tipo de escaneado falta la implementación es su totalidad aunque esta es muy similar a la implementación del escaneado por puntos a una altura fija.

Con lo que se tiene implementado hasta el momento se ha podido ver la necesidad de un segundo escaneo, ya que el sensor puede dar lecturas erróneas. Como se puede ver a continuación.

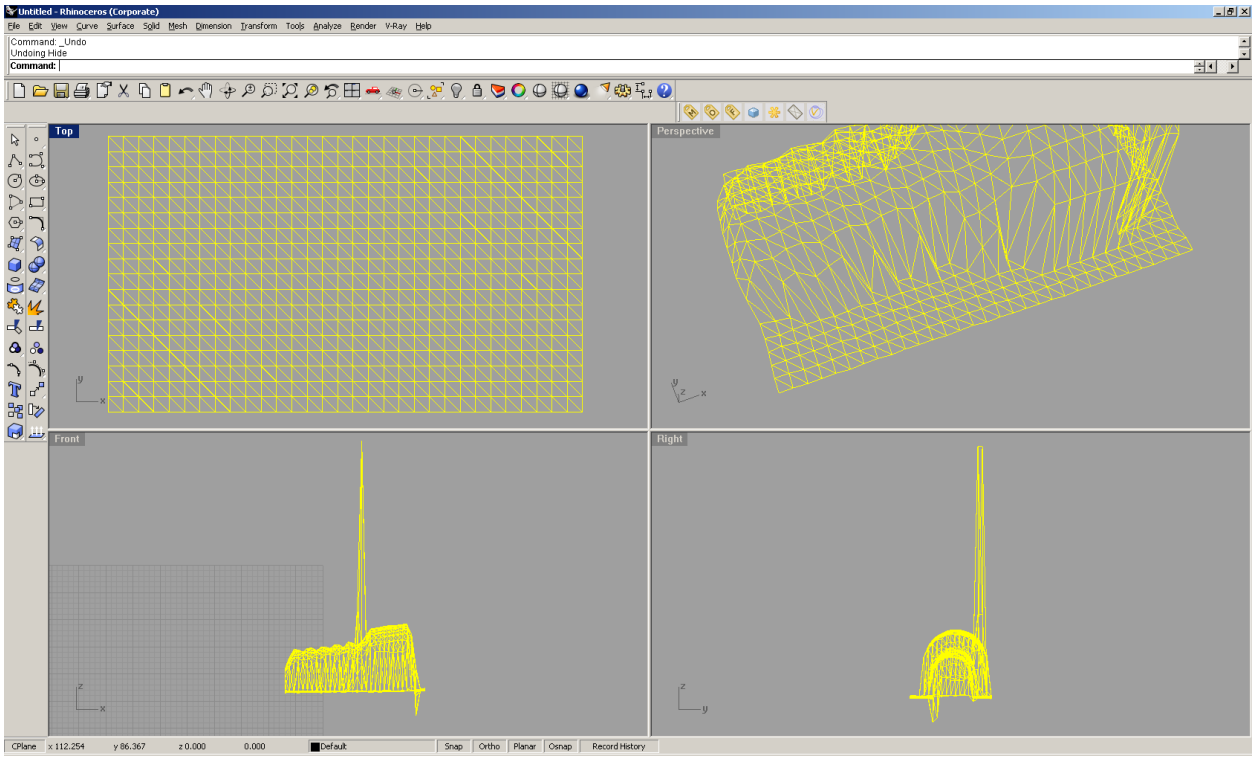


Figura 40 – Modelo creado por la aplicación, en Rhino. Errores lectura escáner.

También hay que tener en cuenta que según la especularidad del material que se escanee, y también del ángulo con el que se escanee puede haber zonas en las que el sensor sea incapaz de tomar las muestras correctamente (Figura 41).

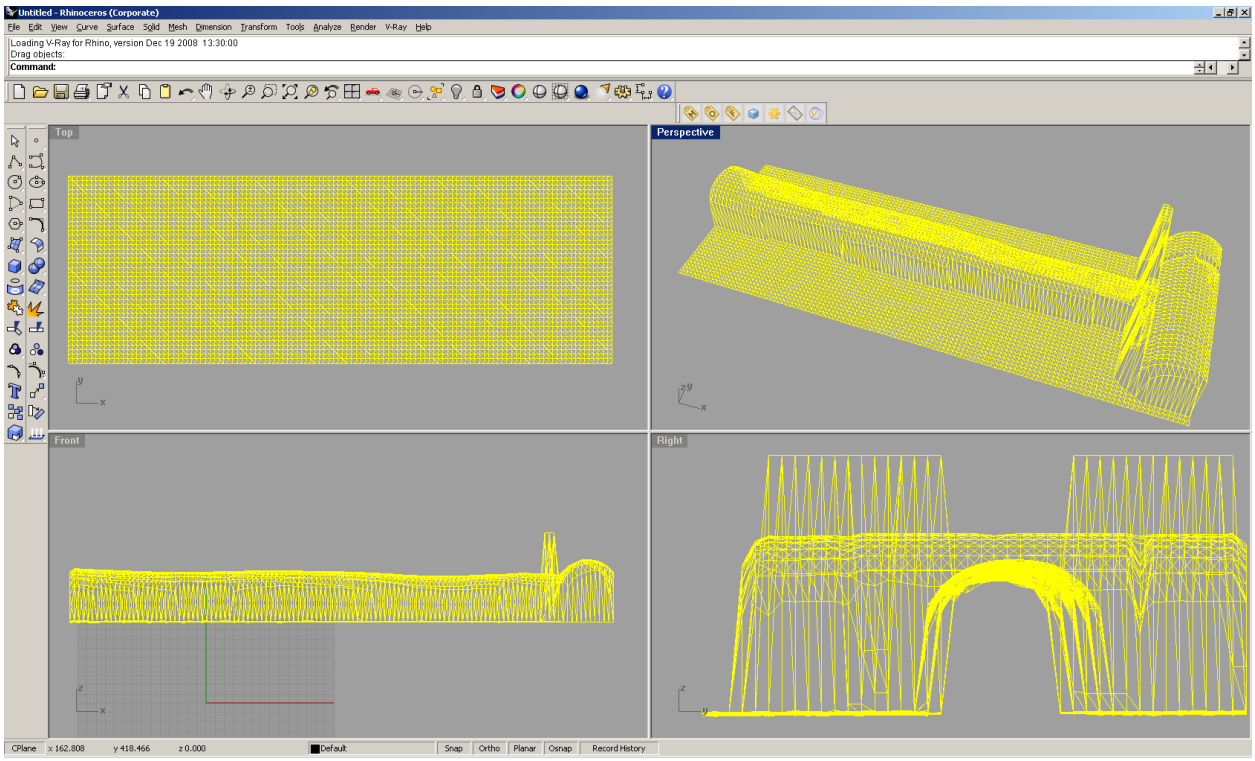


Figura 41 – Errores por limitación del hardware

En cuanto a la interacción del usuario con la visualización del modelo en la aplicación se tendría que dar más usabilidad. Permitiendo que el usuario interactuase con el modelo directamente con el ratón, sin necesidad de utilizar los sliders.

3 Planificación y presupuesto

En este apartado del proyecto veremos la planificación seguida hasta el momento. La compararemos brevemente con la planificación inicial, e introduciremos la planificación del tiempo necesario para finalizar la parte del proyecto que no está finalizada.

En cuanto al presupuesto, debemos tener en cuenta que el hardware utilizado ha sido aprovechado de hardware que no se utilizaba en el departamento. Aun así contabilizaremos el hardware en el presupuesto, también se tendrán en cuenta las horas de implantación del sistema en su entorno de trabajo.

3.1 Planificación

En cuanto a la planificación del proyecto hay que decir que no se ha cumplido la planificación inicial. El primer objetivo del proyecto ha sido mucho más costoso de lo esperado, debido a la falta de experiencia en este campo y la dificultad de la tarea en sí. En esta tarea se ha doblado el número de horas planificadas en un inicio.

En cuanto al diseño y desarrollo de la primera aplicación en una primera iteración se cumplió la planificación, pero la segunda iteración del desarrollo de la aplicación se preveía más corta. Además hay que añadir que durante el desarrollo del escáner, se vio la necesidad de hacer una tercera iteración sobre el diseño del sistema a utilizar.

Para la segunda aplicación la etapa de diseño se hizo en las horas planificadas, pero se empezó más tarde de lo previsto por los retrasos de las tareas anteriores, y la etapa de desarrollo aun no se ha finalizado.

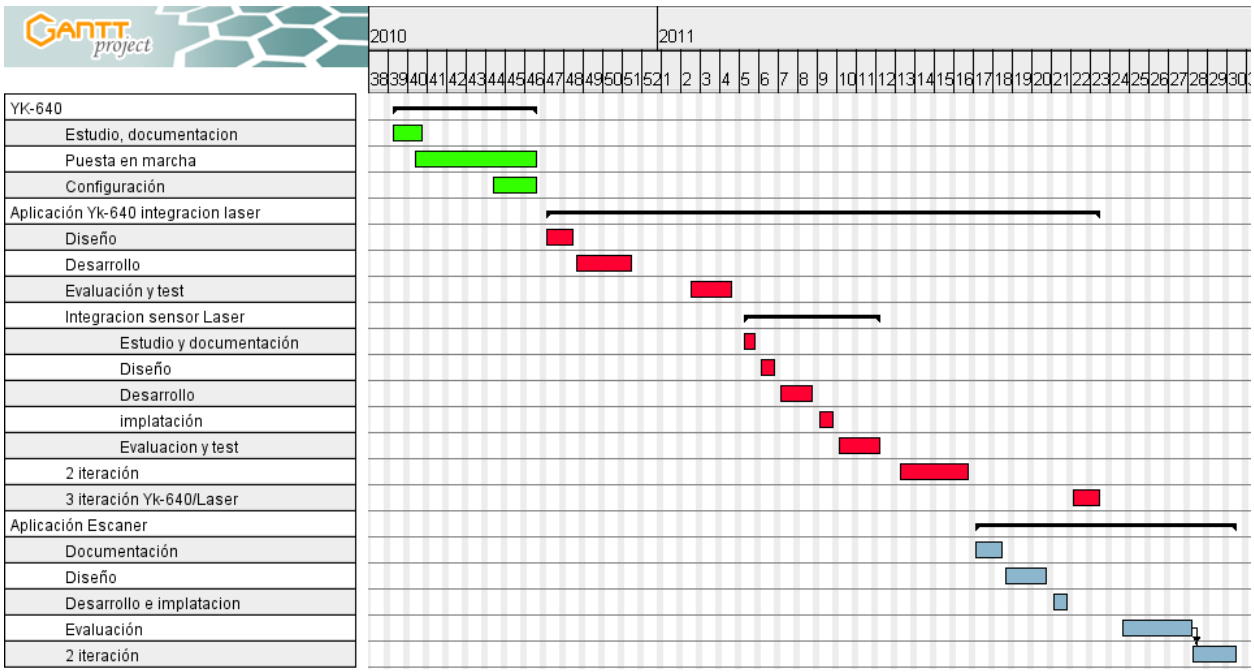


Figura 42 – Diagrama Gantt desarrollo proyecto

El número de horas trabajado por semana no está distribuido de forma uniforme durante la primera mitad del proyecto solo se ha trabajado medias jornadas y en el resto jornadas completas. Este diagrama de Gantt representa la distribución del trabajo durante la realización del proyecto.

En cuanto al número de horas empleado en cada apartado del proyecto lo definiremos con más detalle en el presupuesto.

3.2 Presupuesto

Para hacer el presupuesto del proyecto nos centraremos en dos campos, coste del hardware utilizado y horas empleadas para la consecución del proyecto. Las horas de trabajo las separaremos según la función realizada, separaremos por analista y programador para dar un presupuesto más cercano al presupuesto que daría una empresa especializada en este ámbito.

Table 5 – Horas programador y analista

	Puesta en marcha YK-640	Consola YK-640	Integración sensor	Escaner	Precio Hora	Total
Analista	100	100	50	100	40	14000
Programador	0	160	120	200	35	16800

En cuanto a gastos de diseño y programación el coste llega a 30800€.

En cuanto a la parte de hardware, hay que tener en cuenta que el material utilizado ya esta amortizado. Se está recuperando material que ya estaba fuera de servicio. Por lo que el coste de material es nulo.

En el caso de que se realizase este proyecto con hardware nuevo el presupuesto aumentaría considerablemente. Por la necesidad de adquirir el SCARA y el sensor. Una aproximación del precio sería el de la siguiente tabla.

Tabla 6 – Precio hardware

	Precio
Scara	20000-30000
Sensor distancia	1000

4 Conclusiones

El objetivo principal del proyecto era reactivar el SCARA YK-640. Este manipulador estaba parado sin ninguna utilidad, por lo cual había que encontrarle una utilidad y ponerlo en funcionamiento. Al ser un manipulador que ya está descatalogado no se le podía dar ninguna funcionalidad comercial. Pero aun puede ser útil para fines educativos y de investigación. El objetivo de este proyecto era ponerlo en funcionamiento e integrarle algunos sensores para ampliar sus funcionalidades.

Después de la realización del proyecto podemos decir que el sistema creado es totalmente útil y funcional todavía, por lo que podría ser utilizado perfectamente para fines educativos y en proyectos de investigación.

Después de finalizar la puesta en marcha y puesta a punto del manipulador, hemos visto la dificultad de tratar con un hardware totalmente desconocido es una tarea que puede llevar mucho tiempo. Por lo que hay que tener en cuenta a la hora de planificar los proyectos un tiempo de familiarización con el hardware. Además para que este proyecto sea realmente útil hay que dejar mucha documentación para futuros usuarios, para acortar al máximo el tiempo de familiarización con el manipulador.

Una vez en funcionamiento el manipulador se ha creado una aplicación que emula el terminal externo del propio manipulador. Con ello se pretende que los usuarios puedan manipular el SCARA de una forma sencilla e intuitiva, y no por ello perdiendo funcionalidades del SCARA. Además el sistema montado pretende ser una plataforma que facilite la integración de diversos sensores en el SCARA.

Durante la realización del proyecto se ha construido la misma aplicación utilizando dos métodos distintos para obtener los datos de los sensores. (Time triggered y event triggered) Lo cual nos ha servido para comprobar que el método basado en eventos para este caso es el más apropiado a implementar, ya que reduce el tráfico de datos por la aplicación software y tiene un tiempo de respuesta mejor. También hay que reconocer que el método Time triggered es mucho más sencillo de implementar y muy válido para algunas aplicaciones, que no requieren mucha sincronización.

En cuanto a la integración del sensor laser al manipulador ha ayudado a hacer un sistema más modular. El ser más modular facilitará la integración de nuevos sensores en un futuro.

Como fines educativos y continuación del proyecto se podría extender el número de sensores integrados en el sistema. Como podría ser el sensor de fuerza mencionado en la memoria. Y otro campo en el que podría ser útil el sistema creado es en el campo de la visión por computador.

Para demostrar que se ha creado un sistema útil con la integración del sensor laser, se ha creado un escáner de superficies, capaz de crear modelos 3D de las superficies escaneadas. Con esta aplicación he podido ver la necesidad de hacer un código basado en eventos para una buena sincronización entre componentes.

La creación del escáner me ha permitido profundizar en el campo de la reconstrucción (modelado) de superficies a partir de una nube de puntos, y ver aplicaciones para el sistema creado. Otro campo en el

que se pueden realizar más proyectos en un futuro dándole más posibilidades al sistema en el campo de la investigación y de la educación.

En cuanto a la realización del proyecto ha podido ver la necesidad de una buena planificación inicial y la importancia de tener en cuenta los riesgos que tienen todas las tareas a llevar a cabo. Al no haber tenido la suficiente consideración con los riesgos me ha llevado a no cumplir la planificación del proyecto. Por lo que parte de la aplicación del escáner no ha podido ser implementada en su totalidad.

Finalmente podemos decir que aparte de futuras aplicaciones en el campo de la investigación y la educación. La aplicación de escaneo diseñada es funcional y podrían dársele utilidades como modelado de maquetas, control de calidad de objetos de una cadena de montaje. El siguiente paso que yo daría para continuar este proyecto es la incorporación de un sistema de visión que permitiese escanear todos los objetos en el espacio de trabajo del robot y la localización automática de los objetos.

5 Bibliografía

[0] Página oficial Yamaha Robotics

http://www.yamaharobotics.com/business/robot/ykx/what/index.html

[1] Manual del Programador YK-X Series.

[2] Real time software patterns

Ross Albert Mckegney, *Application of patterns to real-time object-oriented software design*, Queen's University,Kingston Ontario Canada July 2000.

[3] Documentación sensores de triangulación

http://archives.sensorsmag.com/articles/0598/tri0598/

[4] Manual usuario Acuity AR200

[5] Acuity AR200 data sheet

[6] J.Amat y X.Fernandez. *Robotic manipulation of ophtalmic lenses assited by a dedicated visión system, Proceedings of the 1998 IEEE International Conference on Robotics & Automation. Leuven, Belgium Mayo 1998.*

[7]J.Priego de los Santos y M.Porres de la Haza. *La triangulación de Delunay aplicada a los modelos digitales del terreno.* Departamento de Ingeniería Cartográfica, Geodesia y Fotogrametría de la Universidad Politécnica de Valencia

[8] Mark de Berg, Otfried Cheong, Marc van Kreveld y Mark Overmars. *Computational Geometry.* SRINGER 2008.

J.-R. Sack y J.Urrutia. Handbook of Computational Geometry. Elsevier 2000.

Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Patrones de diseño. Addison Wesley

William F. Riley, Leroy D. Sturges. Ingeniería mecánica, Reverté, 1995-1996.

J.M. Selig. Introductory robotics, Prentice Hall, 1992.

Manual programación C# - http://msdn.microsoft.com/en-us/library

OpengGL – The OpenGL Red Book

Michael Kazhdan, Matthew Bolitho y Hugues Hoppe. *Poisson surface Reconstruction*. Eurographics Symposium on Geometry Processing (2006).

Hugues Hoppe, Tony deRose, Tom Duchamp, John McDonald, Werner Stuetzle. *Surface Reconstruction from unorganized points.* University of Washintong Seattle.

Geoff Leach. *Improving Worst-Case Optimal Delunay Triangulation Algorithms.* Department of computer Science, Royal Melbourne IT. Melbourne, Australia 1992.

Markus Gross y Hanspeter Pfister. *Point-based graphics.* Morgan Kauffman, Elsevier (2007)

6 Apéndices

6.1 Apéndice A. Manual usuario SCARA YK-640

6.1.1 Introducción

Se estructurara de la siguiente forma: primero hablaremos sobre la comunicación entre el PC y el manipulador, luego sobre la configuración del manipulador básica para poder utilizar este a pleno rendimiento, posteriormente trataremos sobre la programación del robot. Hay que tener en cuenta que este manual no está completo es un manual centrado sobre las necesidades que han surgido durante el desarrollo de este proyecto, en caso de no encontrar algún punto en este manual consultar el manual del propietario.

6.1.2 Comunicación

La comunicación con el manipulador se hará mediante el puerto serie del PC, para que esta comunicación sea posible se tiene que tener las mismas configuraciones en el controlador del manipulador y el puerto serie del ordenador, si la configuración de los dos extremos de la línea serie no es la misma la comunicación no será satisfactoria y el controlador nos devolverá un código de error con el problema correspondiente de configuración del puerto. La configuración tiene que estar dentro de los parámetros de la siguiente tabla.

Table 1 – Configuración Puerto serie

Método transmisión	Full Duplex
Sincronización	Start stop
Baud rates (bps)	600, 1200, 2400, 4800, 9600, 19200 *
Tamaño de los caracteres	7 o 8 bits *
Tamaño carácter Stop	1 o 2 bits *
Control ocupación de los buses	DTR durante la recepción, CTS/DSR para la transmisión
Código terminación	CR o CRLF *
Control XON/XOFF	Si o no *(XON = 11H, XOFF = 13H)
Buffer recepción	255 bytes
Buffer transmisión	127 bytes

*Parámetros configurables según las opciones dadas.

Hay que tener en cuenta que según el tipo de unidad externa al que conectemos el robot la especificación de los pines de la conexión rs-232 pueden variar, para nuestro caso que lo conectamos a un ordenador y controlamos la disponibilidad del bus mediante RTS nos tendremos que atener a la siguiente especificación (figura 1).

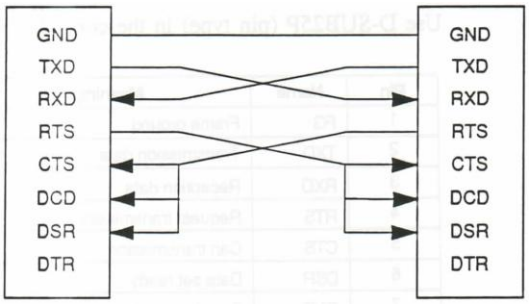


Figura 1 – Diagrama conexión de los pines del conector RS-232

Por lo que necesitaremos conectar el robot y el ordenador con un cable rs-232 null-modem sino la comunicación no será satisfactoria.

6.1.2.1.1 Tipos de comunicación

Una vez establecida la comunicación de forma satisfactoria con el ordenador ya se pueden mandar instrucciones al manipulador, se puede hacer de dos formas usando la comunicación offline o la comunicación online. Básicamente hemos utilizado la comunicación online, por lo que para comunicación offline mirar el manual.

6.1.2.1.1.1 *Método Offline*

Para la comunicación offline las instrucciones deben seguir el siguiente formato:

SEND <source file> TO CMU (robot -> unidad externa)

SEND CMU TO <source file> (unidad -> externa robot)

6.1.2.1.1.2 *Método Online*

Para la comunicación online el formato debe ser el siguiente:

@[_] <online command> [<_command option c/r>]

@: Codigo de inicialización (40H).

[_]: espacio en blanco, no es obligatorio.

<online command> : comando de la tabla de comandos del manual. [Scara owner´s manual (6-16)].

[<_command option c/r>]: parámetros del comando en el caso de que sean necesarios.

Hay cuatro tipos de comandos online:

1. *Keyboard operation command*
2. *Utility command*
3. *Data handling command*
4. *Ejecución de una instrucción en lenguaje del manipulador.*

6.1.2.1.2 Configuración actual del puerto serie del manipulador

Método comunicación	Online
Longitud datos	8 bits
Baud rate	9600
Tamaño Stop bits	1
Paridad	Odd
Código terminación	CRLF
Xon/Xoff	Si

6.1.3 Archivos del controlador

6.1.3.1.1 Archivos programa (.PGM)

Estos archivos contienen programas ejecutables por el SCARA, en el programa debe constar su nombre y debe seguir la siguiente plantilla. En el programa se pueden definir puntos que solo servirán para ese programa y de pueden modificar parámetros de configuración. Los archivos deben seguir los siguientes formatos para que sean interpretados correctamente por el controlador, figura 2 para un solo programa y figura 3 para un conjunto de programas.

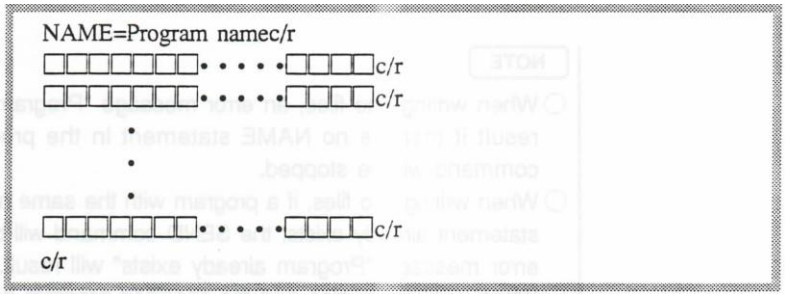


Figura 2 – Formato programa

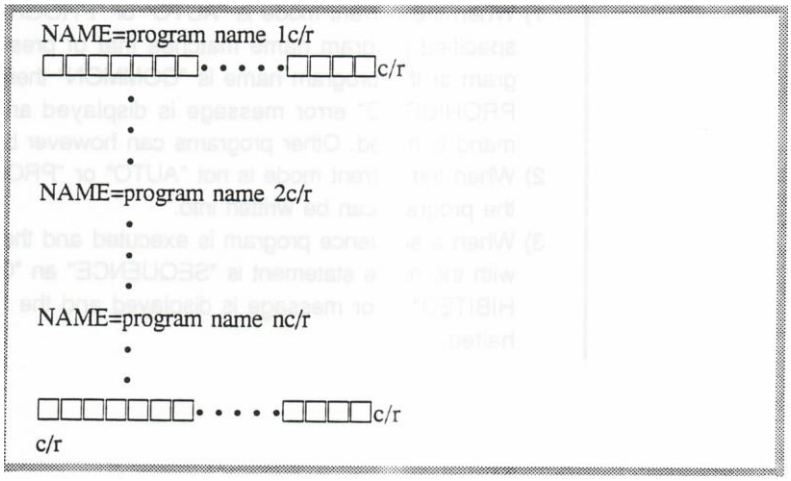


Figura 3 – Formato conjunto de programas

6.1.3.1.2 Archivo de posiciones (.PNT)

Este archivo contiene un conjunto de coordenadas a las que el SCARA accederá sin necesidad de escribir toda la coordenada con solo el nombre de esta será suficiente. Los nombres serán una p mayúscula seguida de un número. Las coordenadas pueden ser tanto cartesianas como pulsos de los actuadores. El formato del archivo se puede ver en la siguiente figura.

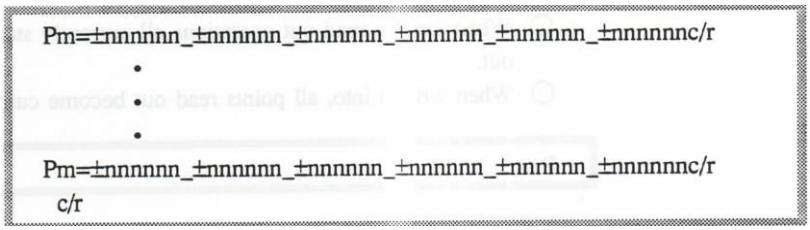


Figura 43 – Formato archivo coordenadas

6.1.3.1.3 Archivo configuración (.PRM)

Aquí solo mostrare el formato del archivo, en el apartado de Configuración del yk-640 de este apéndice se explica en detalle cada parámetro de configuración.

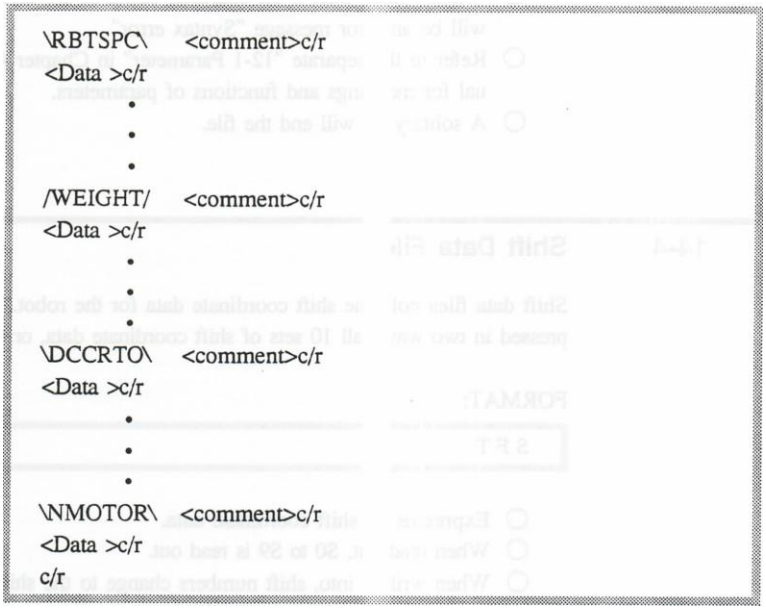


Figura 5 – Formato archivo configuración

6.1.4 Configuración del yk-640

Hay que tener en cuenta que si hay parámetros de configuración que no están definidos correctamente o no están definidos el manipulador puede perder algunas funcionalidades, es importante configurar correctamente el manipulador, a continuación detallare cada parámetro de configuración.

Hay dos tipos de parámetros de configuración, Axis y Robot. Aunque en el archivo de configuración no se hace ningún tipo de distinción entre ellos.

Los parámetros de configuración tienen una nomenclatura especial, que es la siguiente:

<Nombre>……<Clasificación>/<Etiqueta comunicación>

Nombre: nombre del parámetro de configuración.

Clasificación: Puede contener tres iniciales (I, P, C) que indican lo siguiente

- **I:** parámetro para la inicialización.
- **P:** parámetro que puede ser modificado por un programa.
- **C:** parámetro que puede ser modificado por el puerto de comunicación.

Etiqueta comunicación: Nombre por el que se accede al parámetro desde el puerto de comunicación.

6.1.4.1 Parámetros ROBOT

A continuación detallare los parámetros de configuración de este tipo, omitiré los parámetros que no son útiles a nuestros propósitos como por ejemplo el cambio de idioma del display.

6.1.4.1.1 Tip Weight [kg] ……I,PC/WEIGHT

Este parámetro define el peso que debe desplazar el robot incluyendo la pieza que tenga que desplazar y la herramienta que se este utilizando. El peso se expresa en kilos y el máximo viene determinado automáticamente por la capacidad del modelo.

6.1.4.1.2 Origin sequence ……I,C/ORIGIN

Este parámetro define el orden en el que los distintos actuadores del robot volverán a su posición de origen, se hace mediante los numerales [1-6] que corresponden a los ejes X, Y, Z, R, A, B. Para ello se escribe una secuencia con los seis números en el orden que se debe realizar el retorno a la posición de origen.

Ejemplo: 123456

6.1.4.1.3 R axis orientation ……I,C/RORIEN

Este parámetro indica al controlador si durante los movimientos sobre los ejes XY debe mantener la orientación del eje R. Puede tomar dos valores KEEP, FREE.

6.1.4.2 Parámetros AXIS

6.1.4.2.1 Accel coefficient [%] ……I,P,C/ACCEL

Aceleración del manipulador en modo automático. Se define para cada actuador del robot.

6.1.4.2.2 +Soft limit [pulse] ……I,C/PLMT+ -Soft limit [pulse] ……I,C/PLMT-

Define el rango de movimiento del robot por software, se define un límite para actuador del robot.

6.1.4.2.3 Tolerance [pulse] ……I,P,C/TOLE

Define la tolerancia al posicionamiento para los posicionamientos intermedios de una trayectoria PTP, con un valor alto reduce el tiempo de posicionamiento del robot. Se utiliza solo para trayectorias definidas por un programa.

6.1.4.2.4 Out position [pulse] ……I,P,C/OUTPOS

Con este parámetro se define cuando se puede ejecutara el siguiente comando o no según el número de pulsos antes de que llegue a la posición destino, aunque si la siguiente instrucción es de movimiento también esperara a ejecutar la siguiente instrucción esperara a que se llegue al margen impuesto por la tolerancia.

6.1.4.2.5 Arch position [pulse] ……I,P,C/ARCH

Para los movimientos arch este parámetro define el movimiento por el número de pulsos previos a la posición objetivo. Para ejecutar movimientos ARCH sobre el plano XY, los puntos deben estar expresados en pulsos.

6.1.4.2.6 Origin speed [%] ……I,C/ORGSPD

Este parámetro define la velocidad durante la ejecución del movimiento a la posición de origen.

6.1.4.2.7 Manual accel [%] ……I,C/MANACC

Especifica la aceleración en modo manual (1-100%).

6.1.4.2.8 Arm length [mm] ……I,C/ARMLEN

Define la longitud de cada brazo del SCARA, cuando las coordenadas estándar están definidas la longitud de los brazos X, Y se define de forma automática. La definición de este parámetro y de las coordenadas estándar es importante cuando se trabaja en coordenadas cartesianas para garantizar la precisión.

6.1.4.2.9 Offset pulse [pulse] ……I,C/OFFSET

Cuando el robot esta en el origen de cada eje este parámetro muestra los offsets de todos los ejes en pulsos.

6.1.4.2.10 Axis tip weight [kg] ……I,C/AXSTIP

Define el peso de la carga con el que trabajara el robot.

6.1.5 Programación del manipulador

Se utiliza un lenguaje propio desarrollado por Yamaha Motor Company con una semántica muy similar a la de BASIC. Para la programación del SCARA consultar el manual de programación del manual del propietario.

6.1.5.1 Ejecución de programas

Para ejecutar los programas en el SCARA hay que estar en modo automático y utilizando la comunicación Online hay que ejecutar las siguientes instrucciones:

@SWI <Program name> para decir al SCARA que es el programa que se quiere ejecutar.

@RUN para que comience la ejecución del programa.

El programa debe estar en la memoria interna del SCARA y debe ser correcto, no tener errores de compilación. En caso de que tenga errores de compilación el SCARA nos dará un error de comando ilegal, en caso de que no esté en la memoria nos dirá que el programa no existe.

6.2 Apéndice B. Listado errores y soluciones del manipulador.

Este no es un listado completo, es solo una memoria de los errores que más nos hemos encontrado durante la realización del proyecto y las soluciones aplicadas para solventar estos errores.

Hay un conjunto de errores relacionados con el puerto de comunicación, pero al ser un problema de comunicación solo se pude ver desde la unidad externa MPB y no desde el PC, por lo que con el hardware del que se dispone no se pueden ver. En el manual del proyecto hay una relación completa de todos los posibles errores del manipulador.

Error 0.0 Origin incomplete

Es un error muy común que puede pasar en los siguientes casos:

a. Retorno al origen incompleto.
b. Retorno al origen interrumpido.
c. Falta algún parámetro en la configuración del retorno.

La solución es dejar que acabe el retorno al origen si está en proceso o sino hacer el retorno al origen. En caso de que el terminal del manipulador este en coordenadas con la y menor que 0 antes de hacer el retorno al origen hay que asegurarse de mover previamente el terminal ya que el movimiento de vuelta al origen va de +y hacia –y.

Error 2.1 Oversoft limit

Este error indica que se ha intentado hacer un movimiento fuera de los límites marcados por software, como solución se pueden modificar los límites de software si es necesario y hay posibilidad de hacerlo por hardware.

Error 2.2 Std. Coordinate doesn't exist

Este error nos indica que las coordenadas estándar, en ese caso no se permiten los movimientos a puntos por coordenadas cartesianas. Para solucionar este error hay que reconfigurar los parámetros que definen el sistema de coordenadas estándar arm length y offset pulse.

Error 3.2 Program already exists

Ya existe un programa con este nombre, no se pueden reescribir los programas, para guardar un programa con el mismo nombre primero hay que eliminar el programa que hay ya almacenado en la memoria del SCARA.

Error 3.3 Program doesn´t exist

Cuando nos da este error puede ser por dos razones o el programa no está en la memoria del SCARA o el programa que hay en la memoria con ese nombre tiene algún error de compilación. La solución es bastante obvia.

Error 5.6 Digit number error

Este error se produce cuando al controlador le llega un número con un formato incorrecto, refiriéndose con formato al número de dígitos que se le envían. El número de dígitos de cada tipo es:

a. Binario: 8 dígitos.
b. Octal: 6 dígitos.
c. Decimal: 7 dígitos.
d. Hexadecimal: 4 dígitos.

6.3 Apéndice C. Código

6.3.1 Código aplicación consola, SCARA-sensor event triggered.

En éste apéndice no se encuentra el código completo, solo hay las clases más representativas. Para ver el código completo se puede consultar el archivo adjunto entregado junto a esta memoria.

6.3.1.1 Sistema

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    public partial class sistema : Component
    {

        //delegados
        public delegate void errorEventHandler(object sender, errorEventArg a);
        public delegate void newPosEventHandler(object sender, newPosEventArg a);
        public delegate void directoryEventHandler(object sender, directoryEventsArg a);
        public delegate void newSamplesEventHandler(object sender, sistemSamplesEventArgs
a);
        public delegate void scaraTerminalHandler(object sender, scaraTerminalEventArgs
a);

        //eventos
        public event errorEventHandler errorEvent;
        public event newPosEventHandler newPosEvent;
        public event directoryEventHandler directoryEvent;
        public event newSamplesEventHandler newSamplesEvent;
        public event EventHandler scaraIniciatialized;
        public event scaraTerminalHandler writeTerminal;

        //elementos necesarios del scara
        private coordinates crd = new coordinates();
        private articularCoord acrd = new articularCoord();
        private int speed = 50;

        public System.Data.DataTable directory;

        //elementos del sensor
        private double sensOffset = 70.0;
        private const double height = 343.0;
        private double distance = 0.00;
        private double distanceFn = 0.00;
        private int interval = 10000;
        private List<double> lastsMeasures = new List<double>();
        private List<double> exList = new List<double>();

        //variables de control del sistema
        private bool write = false;

        public string err;
```

```
        public int error = 0;
        public bool initialized = false;
        public bool readyDir = false;
        public bool sampleReady = false;

        //Parametros propios del sistema
        private System.IO.StreamWriter strWr; //para guardar las coordenadas de los
puntos muestreados por el sistema.
        private coordinates lastCoord;

    #region "Init"
        public sistema(string scaraPort, string laserPort, bool scOn)
        {
            try
            {
                InitializeComponent();
                laserSensor1 = new LaserSensor(laserPort);
                scaraR1 = new ScaraR(scaraPort);
                scaraR1.init(scOn);
                directory = new System.Data.DataTable();
                for (int i = 0; i < 7; i++) directory.Columns.Add();

                //Subscripcion a eventos del SCARA
                scaraR1.errorEvent +=new ScaraR.errorEventHandler(scaraR1_errorEvent);
                scaraR1.newPositionEvent += new EventHandler(scaraR1_newPositionEvent);
                scaraR1.redyDirEvent += new
ScaraR.scaraReadyDirHandler(scaraR1_readyDirEvent);
                scaraR1.writeTerminal += new
ScaraR.scaraTerminalHandler(scaraR1_writeTerminalEvent);
                scaraR1.scaraInitialized += new EventHandler(scaraR1_scaraInitialized);

                //Subscripcion a eventos del sensor Laser
                laserSensor1.newSampleEvent += new
LaserSensor.newSampleHandler(laserSensor1_newSampleEvent);
            }
            catch (Exception ex)
            {
                errorEvent(this,new errorEventArg(string.Format("Error en la
inicializacion del sistema:{1}",ex.Message)));
            }
        }

        /// <summary>
        /// Se recalculan los datos necesarios para el sistema y se reenvia al form los
datos que ellos necesitan
        /// distancia y medida del laser.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="a"></param>
        private void laserSensor1_newSampleEvent(object sender, newSampleEventArgs a)
        {
            distanceFn = height - sensOffset - a.Sample;
            distance = a.Sample;
            newSamplesEvent(this,new sistemSamplesEventArgs(distanceFn,distance));
        }

        private void scaraR1_scaraInitialized(object sender, EventArgs e)
        {
```

```
            initialized = true;
            scaraIniciatialized(this, new EventArgs());
        }

        private void scaraR1_writeTerminalEvent(object sender, scaraTerminalEventArgs a)
        {
            writeTerminal(this, a);
        }

        private void scaraR1_readyDirEvent(object sender, scaraReadyDirEventArg a)
        {
            string[] dir = a.Directory.Split(new Char[] { '\n' },
StringSplitOptions.RemoveEmptyEntries);
            createDir(dir);
        }

        private void scaraR1_newPositionEvent(object sender, EventArgs e)
        {
            crd = new coordinates(scaraR1.getCoord());
            acrd = new articularCoord(scaraR1.getCoordAc());
            newPosEventArg nP = new newPosEventArg();
            nP.Articular = acrd;
            nP.Cartesian = crd;
            newPosEvent(this, nP);
        }

        private void scaraR1_errorEvent(object sender, errorEventArg a)
        {
            errorEvent(this, a);
        }

        public sistema(IContainer container)
        {
            container.Add(this);

            InitializeComponent();
        }
        #endregion

    #region "Polling"

        /// <summary>
        /// Encuesta al modulo laser para coger las medidas tomadas por el sensor en caso
de que el sensor este funcionando de en modo
        /// automatico, si hay que escribir las medidas en un archivo tambien se hara
aqui.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void samplingTime_Tick(object sender, EventArgs e)
        {

            List<double> dist = laserSensor1.getMeasures();
            if (dist.Count > 0)
            {
                lastsMeasures.AddRange(dist);
                exList.AddRange(dist);
```

```
            distance = dist.Last();
            distanceFn = height - sensOffset - distance;

            if (write)
            {
                if (crd != lastCoord)
                {
                    strWr.WriteLine();
                    strWr.WriteLine(crd);
                    lastCoord = crd;
                }
                //Escribir medidas
                writeLengths();

            }
        }
    }

    #endregion

#region "Sistema"

    public void writeLengths()
    {
        List<double> aux = getLastsMeasures();
        for (int i = 0; i < aux.Count; i++)
        {
            double val = height - crd.getZ() - sensOffset - aux[i];
            strWr.Write(val.ToString() + ", ");
        }
    }

    public void setsensOffset(double offset)
    {
        sensOffset = offset;
    }

    public coordinates getCoord()
    {
        return  crd;
    }

    public articularCoord getArCoord()
    {
        return acrd;
    }

    //Se cogen las ultimas muestras del sistema
    public List<double> getLastsMeasures(bool clear = true)
    {
        List<double> aux = new List<double>(lastsMeasures);
        if(clear) lastsMeasures.Clear();
        return aux;
    }

    public List<double> getExList()
    {
        List<double> aux = new List<double>(exList);
```

```
            exList.Clear();
            return aux;
        }

        public void setWritingFile(string name)
        {
            strWr = new System.IO.StreamWriter(name);
        }

        public void startWriting()
        {
            write = true;
        }

        public void stopWriting()
        {
            write = false;
            strWr.Close();
        }

        private void createDir(string[] lines)
        {
            directory.Rows.Clear();
            int j = 0;
            if (lines[0].Contains("DIR")) j = 2;
            else
                for(int i=1;i<lines.Length - 1; i++)  if (lines[i].Contains("DIR")) { j =
i + 2; break; }
            int rws = 0;
            for (int i = j; i < lines.Length - 1; i++)
            {
                if (lines[i].Contains("END")) break;
                directory.Rows.Add();
                directory.Rows[rws].ItemArray = lines[i].Split(new Char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
                rws++;
            }
            directoryEvent(this, new directoryEventsArg(directory));
        }
#endregion

    #region "Scara"

        public void scMove(directions dir, double distance)
        {
            scaraR1.move(dir, distance);
        }

        public void scMoveI(double x, double y, double z = 0, double fi = 0)
        {
            coordinates aux = new coordinates(new double[] { x, y, z, fi });
            scaraR1.moveCrd(aux.scaraCoords());
        }

        public void scDrive(int actuador, int pulse)
        {
            scaraR1.drive(actuador, pulse);
        }
```

```
public string scConectDisconect(bool on)
{
    if (!on) return scaraR1.disconnect();
    else return scaraR1.connect();
}

public void scClearTerm()
{
    scaraR1.clearTerm();
}

public void scSetSpeed(int speedin)
{
    scaraR1.setSpeed(speedin.ToString());
    speed = speedin;
}

public void scHome()
{
    if(scaraR1.on)scaraR1.goOrigin();
}

public void scManualMode()
{
    scaraR1.manual();
}

public void scAutoMode()
{
    scaraR1.automatic();
}

public void scServosOnOff(bool free)
{
    if (free) scaraR1.servosFree();
    else scaraR1.servosOn();
}

public void scSendIns(string order)
{
    scaraR1.instr(order, 1);
}

public void scRefreshDir()
{
    scaraR1.getDirectory();
}

public void scFileUpload(string name,string ext)
{
    scaraR1.pc_ykFile(name, ext);
    readyDir = false;
    if(ext == "PGM")scRefreshDir();
}

public void scFileDownload(string name, string ext)
{
```

```
            scaraR1.yk_pcFile(name, ext);
        }

        public void scRun(string name)
        {
            scaraR1.runProg(name);
        }
#endregion

    #region "LaserSensor"

        public void scDelete(string name)
        {
            scaraR1.deleteProg(name);
            readyDir = false;
            scRefreshDir();
        }

        public void lsManualSample()
        {
            laserSensor1.sampleManual();
            samplingTime.Enabled = false;
        }

        public void lsAutSample()
        {
            samplingTime.Interval = interval*2;
            samplingTime.Enabled = true;
        }

        public void lsOnSen()
        {
            laserSensor1.on();
            samplingTime.Start();
        }

        public void lsOffSen()
        {
            laserSensor1.off();
            samplingTime.Stop();
        }

        public void lsGetSample()
        {
            sampleReady = false;
            laserSensor1.getManualSample();
        }

        public void lsBgLight(bool enable)
        {
            laserSensor1.switchBGlight(enable);
        }

        public void lsChangePref(string pref)
        {
            laserSensor1.changePref(pref);
        }
```

```
        public void lsSetInterval(int intervalIn)
        {
            //comprobar que el intervalo sea valido entre 16 y 10000 el intervalo del
sensor es 8 50000
            if (intervalIn < 8 || intervalIn > 50000) throw new
IndexOutOfRangeException();
            else
            {
                interval = intervalIn;
                samplingTime.Interval = intervalIn*2;
                laserSensor1.setsampleRate((interval * 10));
            }
        }

        public int getInterval()
        {
            return interval;
        }

        public double[] getLastDistance()
        {
            if (lastsMeasures.Count > 0)
            {
                return new[] { lastsMeasures.Last(), height - crd.getZ() - sensOffset -
lastsMeasures.Last() };
            }
            else
                return new[] { distance, distanceFn };
        }

        public List<double> getDistances()
        {
            List<double> res = new List<double>(lastsMeasures);
            lastsMeasures.Clear();
            return res;
        }

 #endregion
    }
}
```

6.3.1.2 Scara

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;


namespace WindowsFormsApplication1
{

    public partial class ScaraR : Component
    {
```

```
        //EVENTOS
        //delegados
        public delegate void errorEventHandler(object sender, errorEventArg a);
        public delegate void scaraReadyDirHandler(object sender, scaraReadyDirEventArg
a);
        public delegate void scaraTerminalHandler(object sender, scaraTerminalEventArgs
a);

        //eventos
        public event errorEventHandler errorEvent;
        public event EventHandler newPositionEvent;
        public event EventHandler scaraInitialized;
        public event scaraReadyDirHandler redyDirEvent;
        public event scaraTerminalHandler writeTerminal;

        const char LF = '\x0A';
        const char CR = '\x0D';
        const int TIME = 400;

        private int[] coordAc = new int[4];
        private double[] coord = new double[4];
        private double[] coordAux = new double[4];
        public bool newPos = false;

        public string terminal;
        public string directory;

        public bool readyDir = false;
        public bool initialized = false;
        public bool servOn = true;
        public bool on = false;

        private int mode;
        private string received;
        private System.IO.StreamWriter sw;
        private bool auto;
        private int initcount = 0;
        private bool idle = false;

        public ScaraR(string portName)
        {
            InitializeComponent();
            yk640.PortName = portName;
        }

        public ScaraR(IContainer container)
        {
            container.Add(this);
            InitializeComponent();
        }

        ~ScaraR()
        {
            if (yk640.IsOpen) yk640.Close();
        }

        public void init(bool open = false)
        {
```

```
            try
            {
                yk640.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(yk640_DataReceived);
                terminal = "";
                mode = -1;
                internPoll.Interval = TIME;
                yk640.Open();
                yk640.ReceivedBytesThreshold = 1;
                yk640.DiscardOutBuffer();
                yk640.DiscardInBuffer();
                instr("@MANUAL", 0);
                instr("@SPEED 50", 0);
                instr("@SERVO ON", 0);
                instr("@ORIGIN", 0);
                if (open)
                {
                    idle = false;
                    on = true;
                    internPoll.Enabled = true;
                }
                else on = false;
                yk640.DiscardInBuffer();
                coord[0] = 600.0;
                coord[1] = coord[2] = coord[3] = 0.0;
                coordAc[0] = coordAc[1] = coordAc[2] = coordAc[3] = 0;
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public string connect()
        {
            try
            {
                yk640.Open();
                if (initialized)
                {
                    mode = 2;
                    idle = true;
                    internPoll.Enabled = true;
                }
                return "OK";
            }
            catch (Exception ex)
            {
                throw ex;
            }
        }

        public string disconnect()
        {
            try
            {
                internPoll.Enabled = false;
                mode = -1;
```

```
            yk640.DiscardOutBuffer();
            yk640.DiscardInBuffer();
            yk640.Close();
        }
        catch (Exception ex)
        {
            terminal += ex.Message + LF;
            return ex.Message;
        }
        return "OK";
    }

    public void clearTerm()
    {
        terminal = "";
    }

    public void instr(string text, int mod = -1)
    {
        yk640.DiscardOutBuffer();
        mode = mod;
        idle = false;
        if (mode == 1)
        {
            yk640.DiscardOutBuffer();
            yk640.DiscardInBuffer();
            if (text == "@AUTO" || text == "@SYSTEM") auto = true;
            else if (text == "@MANUAL") auto = false;
        }
        if (mode == 5)
            yk640.DiscardInBuffer();
        yk640.Write(text + CR + LF);
    }

    public void drive(int actuador, int value)
    {
        instr("@DRIVE(" + actuador + ", " + value + ")", 5);
    }

    public void servosOn()
    {
        instr("@SERVO ON", 5);
        servOn = true;
        idle = false;
    }

    public void servosFree()
    {
        instr("@SERVO FREE", 5);
        servOn = false;
        idle = false;
    }

    public void online()
    {
        instr("@ONLINE",5);
    }
```

```
public void move(directions dir, double incr)
{
    string coord = "";
    switch (dir)
    {
        case directions.up:
            coord = incr.ToString() + " 0.0 0.0 0.0 0.0 0.0";
            break;
        case directions.down:
            coord = (-incr).ToString() + " 0.0 0.0 0.0 0.0 0.0";
            break;
        case directions.left:
            coord = "0.0 " + incr.ToString() + " 0.0 0.0 0.0 0.0";
            break;
        case directions.right:
            coord = "0.0 " + (-incr).ToString() + " 0.0 0.0 0.0 0.0";
            break;
        case directions.zup:
            coord = "0.0 0.0 " + (-incr).ToString() + " 0.0 0.0 0.0";
            break;
        case directions.zdown:
            coord = "0.0 0.0 " + incr.ToString() + " 0.0 0.0 0.0";
            break;
    }
    instr("@MOVEI PTP," + coord, 8);
}

public void moveCrd(string coord, int state = -1)
{
    instr("@MOVE PTP, " + coord, 8);
}

public void setSpeed(string speed)
{
    idle = false;
    instr("@SPEED " + speed, 5);
}

public void goOrigin()
{
    idle = false;
    instr("@ORIGIN ", 5);
}

public void getDirectory()
{
    if (!readyDir)
    {
        yk640.DiscardOutBuffer();
        mode = 6;
        idle = false;
        yk640.DiscardInBuffer();
        directory = "";
        instr("@READ DIR", 6);
    }
}

public void runProg(string name)
```

```
{
    if (auto)
    {
        idle = false;
        instr("@SWI <" + name + ">", 7);
    }
}

public void deleteProg(string name)
{
    instr("@ERA <" + name + ">", 5);
    readyDir = false;
}

public void automatic()
{
    auto = true;
    instr("@AUTO", 5);
}

public void manual()
{
    auto = false;
    instr("@MANUAL", 5);
}

public int[] getCoordAc()
{
    return coordAc;
}

public double[] getCoord()
{
    return coord;
}

public void pc_ykFile(string filePath, string name)
{
    idle = false;
    mode = 3;
    System.IO.StreamReader sr = new System.IO.StreamReader(filePath);
    yk640.Write("@WRITE " + name + CR + LF);
    while (!sr.EndOfStream)
    {
        string data = sr.ReadLine();
        yk640.Write(data+CR+LF);
    }
    sr.Close();
    readyDir = false;
}

public void yk_pcFile(string filePath, string name)
{
    idle = false;
    mode = 4;
    sw = new System.IO.StreamWriter(filePath);
    yk640.Write("@READ " + name + CR + LF);
}
```

```
        //Tratamiento de los datos que se reciben por el puerto serie.
        private void yk640_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            //hay que variar el tratamiento de los datos segun el tipo de comunicacion
que se este haciendo
            //hay que diferenciar entre la recepcion/envio de archivos al controlador, el
uso de botones del formulario
            // y la actualiacion de las variables de posicion del robot.
            // Otro factor a tener en cuenta es que no se tratara una linea de codigo
recibida del controlador hasta que no se llegue el salto de linea para evitar la perdida
de informacion.
            //una vez se ha tratado los datos de entrada se vuelve a poner en marcha el
timmer para que en los ratos que el controlador este ocioso actualice los parametros de
posicion.
            /**
             * Mode 0 -> init
             * Mode 1 -> terminal, mostraremos los datos que nos devuelvan por el
terminal.
             * Mode 2 -> actualizacion parametros en momentos ociosos. d1-d4
             * Mode 3 -> pc->yk640
             * Mode 4 -> yk640->pc
             * Mode 5 -> botones form solo nos interesa mostrar si hay algun tipo de
error
             * Mode 6 -> get directory
             * Mode 7 -> run program in user interface
             * Mode 8 -> Desplazamientos, coordenadas cartesianas, queremos saber cuando
terminan.
             * */
            if (yk640.IsOpen)
            {
                if (yk640.BytesToRead >= 1)
                {
                    try
                    {

                        received = yk640.ReadTo("\r\n");
                        if (mode == 1)
                        {
                            received += "\r\n";
                        }
                        if (mode == 2 && received.Contains("[POS]"))
                        {
                            string[] aux2;
                            aux2 = received.Split(new Char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
                            if (aux2.Length > 5)
                            {
                                if (aux2[1].Contains('.'))
                                {
                                    coordAux[0] = double.Parse(aux2[1]);
                                    coordAux[1] = double.Parse(aux2[2]);
                                    coordAux[2] = double.Parse(aux2[3]);
                                    coordAux[3] = double.Parse(aux2[4]);
                                    if (coordAux != coord)
                                    {
```

```
                                    coord = new double[] { coordAux[0], coordAux[1],
coordAux[2], coordAux[3] };
                                    newPositionEvent(this, new EventArgs());
                                }
                                if (mode == 2) idle = true;
                            }
                            else
                            {
                                coordAc[0] = int.Parse(aux2[1]);
                                coordAc[1] = int.Parse(aux2[2]);
                                coordAc[2] = int.Parse(aux2[3]);
                                coordAc[3] = int.Parse(aux2[4]);
                            }
                        }
                    }
                    switch (mode)
                    {
                        case 0:
                            initcount++;
                            if (received != "OK")
                            {
                                errorEvent(this, new errorEventArg(received));
                            }
                            if (initcount == 4)
                            {
                                initialized = true;
                                scaraInitialized(this, new EventArgs());
                                if (!on) yk640.Close();
                                else
                                {
                                    mode = 2;
                                    idle = true;
                                }
                            }
                            break;
                        case 1:
                            if ((received == "OK\r\n" || received == "\r\n" ||
received.Contains("END\r\n")) && !auto)
                            {
                                idle = true;
                                mode = 2;
                            }
                            writeTerminal(this, new
scaraTerminalEventArgs(received));
                            break;
                        case 2:
                            mode = 2;
                            break;
                        case 3:
                            if (received != "OK" && received != "*** Please enter !")
                            {
                                errorEvent(this, new errorEventArg(received));
                            }
                            if (received == "OK")
                            {
                                idle = true;
                                mode = 2;
                            }
```

```
                                break;
                            case 4:
                                sw.Write(received);
                                if (received == "")
                                {
                                    sw.Close();
                                    mode = 2;
                                    idle = true;
                                }
                                break;
                            case 5:
                                if (received != "OK" && !received.Contains("[POS]") )
                                {
                                    errorEvent(this, new errorEventArg(received));
                                }
                                idle = true;
                                mode = 2;
                                break;
                            case 6:
                                directory += received+'\n';
                                if (received.Contains("END"))
                                {
                                    idle = true;
                                    readyDir = true;
                                    redyDirEvent(this, new
scaraReadyDirEventArg(directory));
                                    directory = "";
                                }
                                break;
                            case 7:
                                if (received == "OK") instr("@RUN", 5);
                                else
                                {
                                    errorEvent(this, new errorEventArg(received));
                                }
                                break;
                            case 8:
                                if (received != "OK" && !received.Contains("[POS]"))
                                {
                                    errorEvent(this, new errorEventArg(received));
                                }
                                idle = true;
                                mode = 2;
                                newPositionEvent(this, new EventArgs());
                                break;
                            default:
                                mode = 2;
                                break;
                        }
                    }
                    catch (Exception ex) { }
                }
            }
        }

        private void internPoll_Tick(object sender, EventArgs e)
        {
            if (idle && mode == 2)
```

```
            {
                instr("@?WHERE", 2);
                instr("@?WHRXY", 2);
            }
        }


    }
}
```

6.3.1.3 Laser

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    public partial class LaserSensor : Component
    {
        //delegado evento
        public delegate void newSampleHandler(object sender, newSampleEventArgs e);

        //evento
        public event newSampleHandler newSampleEvent;

        private char LF = '\x0A';
        private char CR = '\x0D';
        //light true  cuando BGL Elimination On
        //      flase cuando BGL Elimination Off
        private string received = "";


        private int interval = 10000;
        public bool sampleReady = false;
        private bool manualMode = false;
        private List<double> measureList = new List<double>();

        public LaserSensor(string portName)
        {
            try
            {
                InitializeComponent();
                sensor.PortName = portName;
                sensor.ReceivedBytesThreshold = 7;
                sensor.ReadTimeout = 10;
                sensor.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(sensor_DataReceived);
                sensor.Open();
                sensor.DiscardInBuffer();
                dataformat("ASCII");
                //en la configuracion inicial del Sensor la eliminacion de luz ambiente
esta activada, la desactivamos.
```

```
                switchBGlight(false);
                sampleManual();
                //el intervalo inicial del sensor sin cambiar su configuracioon es de
200ms, lo inicializamos a 1s
                //importante acordarse de que el interval del sensor se configura en
decimas de milisegundo (1s -> 10000)
                setsampleRate(interval);
                sensor.DiscardInBuffer();
            }
            catch (Exception ex) { throw new Exception("SENSOR: INIT -> " + ex.Message);
}
        }

        public LaserSensor(IContainer container)
        {
            container.Add(this);
            InitializeComponent();
        }

        public List<double> getMeasures()
        {
            try
            {
                List<double> res = new List<double>(measureList);
                sampleReady = false;
                measureList.Clear();
                return res;
            }
            catch (Exception ex)
            {
                measureList.Clear();
            }
            return measureList;
        }

        //Funciones para configurar el sensor
        public void on()
        {
            string ins = "H1";
            manualMode = false;
            sendIns(ins);
        }

        public void off()
        {
            string ins = "H2";
            sendIns(ins);
        }

        public void setsampleRate(int interv)
        {
            if (interv >= 8 && interv <= 50000)
            {
                string ins = "S" + interv;
                interval = interv;
                sendIns(ins);
            }
            else throw new Exception("SENSOR:intervalo fuera de rango");
```

```
}

public void sampleManual()
{
    string ins = "H2";
    sendIns(ins);
    manualMode = true;
    sensor.DiscardInBuffer();
}

public void getManualSample()
{
    sensor.DiscardInBuffer();
    string ins = "E";
    sampleReady = false;
    sendIns(ins);
}

public void switchBGlight(bool enable)
{
    //Comando L 1=on, 2=off
    string ins;
    if (enable) ins = "L1";
    else ins = "L2";
    sendIns(ins);
}

public void changePref(string pref) {
    string ins = "";
    switch (pref)
    {
        case "Rate":
            ins = "P2";
            break;
        case "Quality":
            ins = "P1";
            break;
        default:
            throw new Exception("SENSOR: Wrong parameter in preference choice");
    }
    sendIns(ins);
}

private void sincroSport()
{
    sensor.DiscardInBuffer();
    sensor.Close();
    sensor.Open();
}

public void dataformat(string format) {
    string ins = "";
    sensor.DiscardInBuffer();
    switch (format)
    {
        case "binary":
            ins = "N";
            break;
```

```
                case "ASCII":
                    ins = "D";
                    break;
                default:
                    throw new Exception("SENSOR: Wrong parameter in data format");
            }
            sendIns(ins);
            sincroSport();
        }

        //envio de una instruccion al sensor
        public void sendIns(string ins)
        {
            sensor.Write(ins+CR+LF);
        }

        //Comunicacion sensor con form y tratamiento de los datos recividos
        private void sensor_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
                    //en principio tienes que recibbir un string de estas caracteristicas
                    //DD.DDD<CR><LF>   '\r\n'
                    //Hay que tratar las medidas de forma que no se pierdan datos o se
pierdan los menos posibles.
                    // una opcion es que si se cogen los datos de forma automatica se
guarden en un archivo y los cogidos manualmente se muestren en el display.
            if (sensor.BytesToRead >= 7)
            {
                try
                {
                    received = sensor.ReadTo("\r\n");
                    double rec = double.Parse(received);
                    if (manualMode)
                    {
                        newSampleEvent(this, new newSampleEventArgs(rec));
                    }
                    else
                    {
                        measureList.Add(rec);
                        sampleReady = true;
                    }
                }
                catch (Exception es) { Console.WriteLine("LASER: "+ es.Message); }
            }
        }
    }
}
```

6.3.1.4 Controladores Interfaz-modelo

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using System.Threading;

namespace WindowsFormsApplication1
{
    public partial class UserApp : Form
    {
        private bool lsauto = false;
        private bool BGE = false;


        public UserApp()
        {
            InitializeComponent();
            try
            {
                serialPortsConf sport = serialPortsConf.Instance;
                sys = new sistema(sport.scara, sport.laser, true);
                this.panel6.Enabled = false;
                this.pollTimer.Enabled = true;
                this.comboBox1.SelectedValue = 1;
                createChart();

                //Subscripcion a eventos del sistema
                sys.errorEvent += new sistema.errorEventHandler(sys_errorEvent);
                sys.directoryEvent += new
sistema.directoryEventHandler(sys_directoryEvent);
                sys.newPosEvent += new sistema.newPosEventHandler(sys_newPosEvent);
                sys.newSamplesEvent += new
sistema.newSamplesEventHandler(sys_newSamplesEvent);
                sys.scaraIniciatialized += new EventHandler(sys_scaraIniciatialized);
            }
            catch (Exception ex) { throw ex; }
        }

        private delegate void newDirectoryDL(object sender,directoryEventsArg a);
        private void sys_directoryEvent(object sender, directoryEventsArg a)
        {
            if (directory.InvokeRequired)
            {
                this.Invoke(new newDirectoryDL(sys_directoryEvent),this,a);
            }
            else
            {
                writeDir(a.Directory);
            }
        }

        private delegate void newPosEventHandler(object sender, newPosEventArg a);
        private void sys_newPosEvent(object sender, newPosEventArg a)
        {
            if (this.xCoord.InvokeRequired)
            {
                this.Invoke(new newPosEventHandler(sys_newPosEvent),this,a);
            }
            else
```

```
    {
        string[] coord = a.Cartesian.coordT();
        this.xCoord.Text = coord[0];
        this.yCoord.Text = coord[1];
        this.zCoord.Text = coord[2];
        this.FiCoord.Text = coord[3];

        coord = a.Articular.coordT();
        this.d1Value2.Text = d1Value.Text = coord[0];
        this.d2Value2.Text = d2Value.Text = coord[1];
        this.d3Value2.Text = d3Value.Text = coord[2];
        this.d4Value2.Text = d4Value.Text = coord[3];
    }
}


private delegate void inicialized(object sender, EventArgs e);
private void sys_scaraIniciatialized(object sender, EventArgs e)
{
    if (this.panel6.InvokeRequired)
    {
        this.Invoke(new inicialized(sys_scaraIniciatialized),this,e);
    }
    else
    {
        this.panel6.Enabled = true;
    }
}

private delegate void newSampleDL(object sender,sistemSamplesEventArgs e);
private void sys_newSamplesEvent(object sender, sistemSamplesEventArgs a)
{
    if (sensorDisp.InvokeRequired)
    {
        this.Invoke(new newSampleDL(sys_newSamplesEvent), this, a);
    }
    else
    {
        this.sensorDisp.Text = a.Sample.ToString();
        this.modelCoord.Text = a.Distance.ToString();
    }
}

private void sys_errorEvent(object sender, errorEventArg e)
{
    MessageBox.Show(e.Message);
}

#region LaserSensor

private void radioButton4_CheckedChanged(object sender, EventArgs e)
{
    if (radioButton4.Checked)
    {
        getSample.Enabled = true;
        panel9.Enabled = true;
        panel10.Enabled = false;
        autoSampPannel.Enabled = false;
```

```
            sys.lsManualSample();
        }
    }

    private void radioButton5_CheckedChanged(object sender, EventArgs e)
    {
        if (radioButton5.Checked)
        {
            getSample.Enabled = false;
            panel9.Enabled = false;
            panel10.Enabled = true;
            autoSampPannel.Enabled = true;
            sys.lsAutSample();
            offSensor.Enabled = false;
            onsSensor.Enabled = true;
        }
    }

    private void onsSensor_Click(object sender, EventArgs e)
    {
        lsauto = true;
        pollTimer.Enabled = true;
        pollTimer.Start();
        sys.lsOnSen();
        offSensor.Enabled = true;
        onsSensor.Enabled = false;
    }

    private void offSensor_Click(object sender, EventArgs e)
    {
        pollTimer.Stop();
        lsauto = false;
        sys.lsOffSen();
        offSensor.Enabled = false;
        onsSensor.Enabled = true;
    }

    private void getSample_Click(object sender, EventArgs e)
    {
        sys.lsGetSample();
    }

    private void bgLight_Click(object sender, EventArgs e)
    {
        if (BGE) bgLight.Text = "OFF";
        else bgLight.Text = "ON";
        sys.lsBgLight(!BGE);
        BGE = !BGE;
    }

    private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string select = comboBox1.SelectedItem.ToString();
        sys.lsChangePref(select);
    }

    private void textBox4_KeyPress(object sender, KeyPressEventArgs e)
    {
```

```
    if (e.KeyChar == (char)Keys.Return)
    {
        try
        {
            int intervalo = int.Parse(textBox4.Text);
            if (intervalo <= 5000 && intervalo >= 1)
            {
                sys.lsSetInterval(intervalo);
                rate.Text = textBox4.Text;
            }
            else throw new Exception("5000 <= value >= 1");
            textBox4.Text = "";
        }
        catch (Exception ex)
        {
            MessageBox.Show("Wrong value!!\n"+ex.Message);
        }
    }
}

private void createChart()
{
    measureChart.Series["Series1"].ChartType = SeriesChartType.FastLine;
    measureChart.Series["Series1"].Points.Clear();
}

private void updateChart()
{
    List<Double> aux = new List<double>(sys.getExList());
    try
    {
        foreach (double item in aux)
        {
            try
            {
                measureChart.Series["Series1"].Points.Add(item);
            }
            catch (Exception ex)
            {
                measureChart.Series["Series1"].Points.Clear();
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private void button1_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        sys.setWritingFile(saveFileDialog1.FileName);
        textBox5.Text = saveFileDialog1.FileName;
        button2.Enabled = true;
        button1.Enabled = false;
    }
```

```
}

private void button2_Click(object sender, EventArgs e)
{
    sys.startWriting();
    button3.Enabled = true;
}

private void button3_Click(object sender, EventArgs e)
{
    button2.Enabled = false;
    button3.Enabled = false;
    button1.Enabled = true;
    sys.stopWriting();
}

#endregion

#region Scara

private void upButton_Click(object sender, EventArgs e)
{
    sys.scMove(directions.up, incrementalTrack.Value);
}

private void downButton_Click(object sender, EventArgs e)
{
    sys.scMove(directions.down, incrementalTrack.Value);
}

private void leftButton_Click(object sender, EventArgs e)
{
    sys.scMove(directions.left, incrementalTrack.Value);
}

private void rightButton_Click(object sender, EventArgs e)
{
    sys.scMove(directions.right, incrementalTrack.Value);
}

private void upZ_Click(object sender, EventArgs e)
{
    sys.scMove(directions.zup, incrementalTrack.Value);
}

private void downZ_Click(object sender, EventArgs e)
{
    sys.scMove(directions.zdown, incrementalTrack.Value);
}

private void refreshDir_Click(object sender, EventArgs e)
{
    sys.scRefreshDir();
}

private void incrementalTrack_ValueChanged(object sender, EventArgs e)
{
    label20.Text = ((double)(incrementalTrack.Value) / 10.0).ToString();
```

```
}

private void servoControl_CheckedChanged(object sender, EventArgs e)
{
    sys.scServosOnOff(servoControl.Checked);
    if (servoControl.Checked) panel6.Enabled = false;
    else panel6.Enabled = true;
}

private void d4Value_TextChanged(object sender, EventArgs e)
{
    d4TB.Value = int.Parse(d4Value.Text);
}

private void d3Value_TextChanged(object sender, EventArgs e)
{
    d3TB.Value = int.Parse(d3Value.Text);
}

private void d2Value_TextChanged(object sender, EventArgs e)
{
    d2TB.Value = int.Parse(d2Value.Text);
}

private void d1Value_TextChanged(object sender, EventArgs e)
{
    d1TB.Value = int.Parse(d1Value.Text);
}

private void d1TB_MouseCaptureChanged(object sender, EventArgs e)
{
    sys.scDrive(1, d1TB.Value);
    d1Value.Text = d1TB.Value.ToString();
}

private void d2TB_ValueChanged(object sender, EventArgs e)
{
    sys.scDrive(2, d2TB.Value);
    d2Value.Text = d2TB.Value.ToString();
}

private void d3TB_ValueChanged(object sender, EventArgs e)
{
    sys.scDrive(3, d3TB.Value);
    d3Value.Text = d3TB.Value.ToString();
}

private void d4TB_ValueChanged(object sender, EventArgs e)
{
    sys.scDrive(4, d4TB.Value);
    d4Value.Text = d4TB.Value.ToString();
}

private void speedBar_MouseCaptureChanged(object sender, EventArgs e)
{
    speedLab.Text = speedBar.Value.ToString();
    sys.scSetSpeed(speedBar.Value);
}
```

```
        private void butOrigin_Click(object sender, EventArgs e)
        {
            sys.scHome();
        }

        private void writeDir(DataTable dt)
        {
            try
            {
                this.directory.Rows.Clear();
                for (int i = 0; i < dt.Rows.Count; i++)
                {
                    this.directory.Rows.Add();
                    for (int j = 0; j < 7; j++) this.directory.Rows[i].Cells[j].Value =
dt.Rows[i].ItemArray[j];
                }
            }
            catch (Exception ex)
            { }
        }

        private void prmDwn_Click(object sender, EventArgs e)
        {
            saveFileDialog1.Filter = "Configuration files (*.PRM)|*.PRM";
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                sys.scFileDownload(saveFileDialog1.FileName, "PRM");
            }
        }

        private void pntDwn_Click(object sender, EventArgs e)
        {
            saveFileDialog1.Filter = "Point files (*.PNT)|*.PNT";
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                sys.scFileDownload(saveFileDialog1.FileName, "PNT");
            }
        }

        private void prmUp_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "Configuration file(*.PRM)|*.PRM";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                sys.scFileUpload(openFileDialog1.FileName, "PRM");
            }
        }

        private void pntUp_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "Point file(*.PNT)|*.PNT";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                sys.scFileUpload(openFileDialog1.FileName, "PNT");
            }
        }
```

```
        private void upLoad_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "Program files(*.PGM)|*.PGM";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                sys.scFileUpload(openFileDialog1.FileName, "PGM");
            }
        }

        private void directory_CellContentClick(object sender, DataGridViewCellEventArgs
e)
        {
            // Controlamos que tecla se ha apretado si es uno de los botones haremos una
de las distintas operaciones posibles (run, delete, download)
            // solo nos interesan las columnas 7->RUN,8->DELETE,9->DOWNLOAD
            switch (e.ColumnIndex)
            {
                case 7:
                    sys.scRun(directory.Rows[e.RowIndex].Cells[1].Value.ToString());
                    break;
                case 8:
                    sys.scDelete(directory.Rows[e.RowIndex].Cells[1].Value.ToString());
                    break;
                case 9:
                    saveFileDialog1.Filter = "Program files (*.PGM)|*.PGM";
                    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                    {
                        sys.scFileDownload(saveFileDialog1.FileName, "PGM");
                    }
                    break;
                default:
                    break;
            }
        }

        private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            if (radioButton1.Checked) sys.scManualMode();
            if (!servoControl.Checked) panel6.Enabled = true;
        }

        private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
            if (radioButton2.Checked) sys.scAutoMode();
            panel6.Enabled = false;
        }

        #endregion

        private void pollTimer_Tick(object sender, EventArgs e)
        {
            if (lsauto)
            {
                double[] aux = sys.getLastDistance();
                sensorDisp.Text = aux[0].ToString();
                modelCoord.Text = aux[1].ToString();
                updateChart();
            }
```

```
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Return)
            {
                try
                {
                    double off = double.Parse(textBox1.Text);
                    sys.setsensOffset(off);
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Wrong value!!");
                }
            }
        }

        private void chartRefresh_Click(object sender, EventArgs e)
        {
            createChart();
        }
    }
}
```

6.3.1.5 Otros

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WindowsFormsApplication1
{
    //Esta clase contiene la configuracion de los puertos serie del sistema
    class serialPortsConf
    {
        private static serialPortsConf instance;
        private string scaraPort, laserPort;

        private serialPortsConf()
        {
            scaraPort = "COM11";
            laserPort = "COM9";
        }

        public static serialPortsConf Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new serialPortsConf();
                }
                return instance;
            }
```

```
        }

        public string laser
        {
            get { return laserPort; }
            set { laserPort = value; }
        }

        public string scara
        {
            get { return scaraPort; }
            set { scaraPort = value; }
        }
    }
}
```

6.3.2 Código aplicación escáner de superficies.

6.3.2.1 Escaner

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace surfaceScan
{
    //Clase principal de la aplicacion.
    //Esta clase recibe la informacion de los forms, se comunica con el escaner y crea el
modelo
    class escaner
    {
        const double HEIGHT = 343;
        modelo model;
        coordinates ini, end, actual,old;
        double incx,incy;
        double offset;
        double xway;
        double num_samples;
        double num_rows;
        int x = 0;
        int y = 0;

        sistema sys;

        public delegate void newPointEventHandler(object sender, newModelPointEventArgs
e);

        public event EventHandler escanEndEvent;
        public event newPointEventHandler newPoint;

        public escaner(sistema sistem)
        {
            sys = sistem;
            sys.confEscaner();
        }

        public void escan(List<coordinates> area, double precision, int tipo,double
offset)
        {
            if(tipo == 1 || tipo == 2) sys.newSamplesEvent +=new
sistema.newSamplesEventHandler(sys_newSamplesEvent);
            if(tipo == 3) sys.scanedRow +=new sistema.rowScanEventhandler(sys_scanedRow);
            model = modelo.Instance;
            model = model.newInstance;
            createPerimetro(new coordinates(area[0]), new coordinates(area[1]));
            this.offset = offset;
            double dist = end.Y - ini.Y;
            num_samples = (double)precision * Math.Abs(dist) / 10;
            double xini = ini.X;
            double xend = end.X;
            xway = xend - xini;
            num_rows = Math.Abs(xway) * ((double)precision / 10);
            incx = Math.Abs(xway / num_rows);
```

```
            incy = Math.Abs(dist / num_samples);

            num_rows = Math.Round(num_rows, 0);
            num_samples = Math.Round(num_samples, 0);

            model.setColumnsRows((int)num_samples, (int)num_rows);
            Console.WriteLine("-----ESCANEAMOS----\nfilas:{0}\ncolumnas:{1}\ntotal:{2}\n-
--------------", (int)num_rows, (int)num_samples, (int)num_rows * (int)num_samples);
            if (tipo == 1)//Por puntos altura fija
            {
                //Tenemos que conseguir los dos puntos entre los que se va a escanear
                //con la distancia y el intervalo de muestreo tenems que definir la
velocidad
                // a la que queremos que vaya el robot
                //Probamos con un intervalo de 0.02 segundos 100ms
                //ESCANEAMOS PUNTO A PUNTO
                actual = new coordinates(ini);
                scanPoint(actual);
            }
            else if (tipo == 2) { }
            else if (tipo == 3) //por filas altura fija
            {
                actual = new coordinates();
                actual.X = ini.X;
                actual.Y = end.Y;
                actual.Z = ini.Z;
                old = new coordinates(ini);
                sys.scanRow(ini, actual);
            }
        }

        private void sys_newSamplesEvent(object sender, sistemSamplesEventArgs a)
        {
            //creamos la coordenada actual y la guardamos en el modelo
            modelo mdl = modelo.Instance;
            double altura = HEIGHT - actual.Z - offset - a.Sample;
            modelCoord mcrd = new modelCoord(actual.X, actual.Y, altura);
            Console.WriteLine("Escaneada posicion --> {0},{1}  de
{2},{3}",this.x,this.y,num_rows,num_samples);
            newPoint(this, new newModelPointEventArgs(mcrd));
            mdl.setPoint(mcrd);
            coordinates pnt = new coordinates();
            y++;
            if (y < num_samples)
            {
                pnt.X = actual.X;
                pnt.Y = actual.Y + incy;
                pnt.Z = actual.Z;
                Console.WriteLine("Escaneando posicion --> {0},{1} de {2},{3}", this.x,
this.y,num_rows,num_samples);
                scanPoint(pnt);
            }
            else
            {
                x++; y = 0;
                if (x < this.num_rows)
                {
                    pnt.X = actual.X + incx;
```

```
                    pnt.Y = ini.Y;
                    pnt.Z = actual.Z;
                    Console.WriteLine("Escaneando posicion -->{0},{1}", this.x, this.y);
                    scanPoint(pnt);
                }
                else
                {
                    escanEndEvent(this, new EventArgs());
                }
            }
        }

        private void scanPoint(coordinates pnt)
        {
            actual = new coordinates(pnt);
            sys.scanPoint(pnt);
        }

        private void sys_scanedRow(object sender, rowScanEventArgs a)
        {
            Console.WriteLine("Escaneada fila {0} de {1}",x.ToString(),num_rows);
            x++;
            if (x < num_rows)
            {
                old.X = actual.X =  old.X + incx;
                sys.scanRow(old, actual);
            }
            else
            {
                //escanEndEvent(this, new EventArgs());
                Console.WriteLine("Mejorar el muestreo de filas");
            }
        }

        public void createPerimetro(coordinates fr, coordinates scn)
        {
            double xmin = Math.Min(fr.X, scn.X);
            double ymin = Math.Min(fr.Y, scn.Y);
            double xmax = Math.Max(fr.X, scn.X);
            double ymax = Math.Max(fr.Y, scn.Y);

            ini = new coordinates(new double[] {xmin,ymin,fr.Z,0 });
            end = new coordinates(new double[] { xmax, ymax, fr.Z, 0 });
        }
    }
}
```

6.3.2.2 Sistema

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Linq;
using System.Text;

namespace surfaceScan
```

```
{
    public partial class sistema : Component
    {

        //delegados
        public delegate void errorEventHandler(object sender, errorEventArg a);
        public delegate void newPosEventHandler(object sender, newPosEventArg a);
        public delegate void directoryEventHandler(object sender, directoryEventsArg a);
        public delegate void newSamplesEventHandler(object sender, sistemSamplesEventArgs
a);
        public delegate void scaraTerminalHandler(object sender, scaraTerminalEventArgs
a);
        public delegate void rowScanEventhandler(object sender, rowScanEventArgs a);

        //eventos
        public event errorEventHandler errorEvent;
        public event newPosEventHandler newPosEvent;
        public event directoryEventHandler directoryEvent;
        public event newSamplesEventHandler newSamplesEvent;
        public event EventHandler scaraIniciatialized;
        public event scaraTerminalHandler writeTerminal;
        public event rowScanEventhandler scanedRow;

        //elementos necesarios del scara
        private coordinates crd = new coordinates();
        private articularCoord acrd = new articularCoord();
        private int speed = 50;

        public System.Data.DataTable directory;
        //public string terminal;

        //elementos del sensor
        private double sensOffset = 70.0;
        private const double height = 343.0;
        private double distance = 0.00;
        private double distanceFn = 0.00;
        private double interval = 10000;
        private List<double> lastsMeasures = new List<double>();
        private List<double> exList = new List<double>();

        //variables de control del sistema
        bool scanpoint = false;
        int scanrow = 0;
        coordinates next;

        public string err;
        public int error = 0;
        public bool initialized = false;
        public bool readyDir = false;
        public bool sampleReady = false;

        //Parametros propios del sistema
        //private System.IO.StreamWriter strWr; //para guardar las coordenadas de los
puntos muestreados por el sistema.
        //private coordinates lastCoord;

    #region "Init"
        public sistema(string scaraPort, string laserPort, bool scOn)
```

```
        {
            try
            {
                InitializeComponent();
                laserSensor1 = new LaserSensor(laserPort);
                scaraR1 = new ScaraR(scaraPort);
                scaraR1.init(scOn);
                directory = new System.Data.DataTable();
                for (int i = 0; i < 7; i++) directory.Columns.Add();

                //Susscripcion a eventos del SCARA
                scaraR1.errorEvent +=new ScaraR.errorEventHandler(scaraR1_errorEvent);
                scaraR1.newPositionEvent += new EventHandler(scaraR1_newPositionEvent);
                //scaraR1.redyDirEvent += new
ScaraR.scaraReadyDirHandler(scaraR1_readyDirEvent);
                scaraR1.writeTerminal += new
ScaraR.scaraTerminalHandler(scaraR1_writeTerminalEvent);
                scaraR1.scaraInitialized += new EventHandler(scaraR1_scaraInitialized);

                //Suscripcion a eventos del sensor Laser
                laserSensor1.newSampleEvent += new
LaserSensor.newSampleHandler(laserSensor1_newSampleEvent);
            }
            catch (Exception ex)
            {
                errorEvent(this,new errorEventArg(string.Format("Error en la
inicializacion del sistema:{0}",ex.Message)));
            }
        }

        public void close()
        {
            scaraR1.disconnect();
            scaraR1.Dispose();
            laserSensor1.close();
        }

        public void confEscaner()
        {
            lsManualSample();
            samplingTime.Enabled = false;
            scaraR1.disableInternPoll();
            scaraR1.writeTerminal -= scaraR1_writeTerminalEvent;
        }

        /// <summary>
        /// Se recalculan los datos necesarios para el sistema y se reenvia al form los
datos que ellos necesitan
        /// distancia y medida del laser.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="a"></param>
        private void laserSensor1_newSampleEvent(object sender, newSampleEventArgs a)
        {
            if (scanpoint)
            {
                scanpoint = false;
            }
```

```
            distanceFn = height - sensOffset - a.Sample;
            distance = a.Sample;
            newSamplesEvent(this, new sistemSamplesEventArgs(distanceFn, distance));
        }

        private void scaraR1_scaraInitialized(object sender, EventArgs e)
        {
            initialized = true;
            scaraIniciatialized(this, new EventArgs());
        }

        private void scaraR1_writeTerminalEvent(object sender, scaraTerminalEventArgs a)
        {
            writeTerminal(this, a);
        }

        //private void scaraR1_readyDirEvent(object sender, scaraReadyDirEventArg a)
        //{
        //    string[] dir = a.Directory.Split(new Char[] { '\n' },
StringSplitOptions.RemoveEmptyEntries);
        //    createDir(dir);
        //}

        private void scaraR1_newPositionEvent(object sender, EventArgs e)
        {
         if (scanrow == 2)
            {
                lsOffSen();
                scanrow = 0;
                lastsMeasures.Clear();
                lastsMeasures = laserSensor1.getMeasures();
                scanedRow(this, new rowScanEventArgs(new List<double>(lastsMeasures)));
            }
            else if (scanrow == 1)
            {
                scSetSpeed(20);
                lsSetInterval(1);
                scanrow = 2;
                lastsMeasures.Clear();
                scaraR1.moveCrd(next.scaraCoords());
                lsOnSen();
            }
            if (scanpoint)
            {
                lsGetSample();
            }
            else
            {
                crd = new coordinates(scaraR1.getCoord());
                acrd = new articularCoord(scaraR1.getCoordAc());
                newPosEventArg nP = new newPosEventArg();
                nP.Articular = acrd;
                nP.Cartesian = crd;
                newPosEvent(this, nP);
            }
        }

        private void scaraR1_errorEvent(object sender, errorEventArg a)
```

```
        {
            errorEvent(this, a);
        }

        public sistema(IContainer container)
        {
            container.Add(this);

            InitializeComponent();
        }
        #endregion


    #region "Polling"

        /// <summary>
        /// Encuesta al modulo laser para coger las medidas tomadas por el sensor en caso
de que el sensor este funcionando de en modo
        /// automatico, si hay que escribir las medidas en un archivo tambien se hara
aqui.
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void samplingTime_Tick(object sender, EventArgs e)
        {

            List<double> dist = laserSensor1.getMeasures();
            if (dist.Count > 0)
            {
                lastsMeasures.AddRange(dist);
                exList.AddRange(dist);

                distance = dist.Last();
                distanceFn = height - sensOffset - distance;
                newSamplesEvent(this, new sistemSamplesEventArgs(distanceFn,distance));
            }
        }

        #endregion

    #region "Sistema"

        public void setsensOffset(double offset)
        {
            sensOffset = offset;
        }

        public coordinates getCoord()
        {
            return  crd;
        }

        public articularCoord getArCoord()
        {
            return acrd;
        }

        //Se cogen las ultimas muestras del sistema
```

```
        public List<double> getLastsMeasures(bool clear = true)
        {
            List<double> aux = new List<double>(lastsMeasures);
            if(clear) lastsMeasures.Clear();
            return aux;
        }

        public List<double> getExList()
        {
            List<double> aux = new List<double>(exList);
            exList.Clear();
            return aux;
        }

        public void scanPoint(coordinates point) {
            scanpoint = true;
            scaraR1.moveCrd(point.scaraCoords());
        }

        public void scanRow(coordinates ini, coordinates end)
        {
            scanrow = 1;
            scSetSpeed(100);
            next = end;
            scaraR1.moveCrd(ini.scaraCoords());
        }

#endregion

    #region "Scara"

        public void scMove(directions dir, double distance)
        {
            scaraR1.move(dir, distance);
        }

        public void scMoveI(double x, double y, double z = 0, double fi = 0)
        {
            coordinates aux = new coordinates(new double[] { x, y, z, fi });
            scaraR1.moveCrd(aux.scaraCoords());
        }

        public void scDrive(int actuador, int pulse)
        {
            scaraR1.drive(actuador, pulse);
        }

        public string scConectDisconect(bool on)
        {
            if (!on) return scaraR1.disconnect();
            else return scaraR1.connect();
        }

        public void scClearTerm()
        {
            scaraR1.clearTerm();
        }
```

```
        public void scSetSpeed(int speedin)
        {
            scaraR1.setSpeed(speedin.ToString());
            speed = speedin;
        }

        public void scHome()
        {
            if(scaraR1.on)scaraR1.goOrigin();
        }

        public void scManualMode()
        {
            scaraR1.manual();
        }

        public void scAutoMode()
        {
            scaraR1.automatic();
        }

        public void scServosOnOff(bool free)
        {
            if (free) scaraR1.servosFree();
            else scaraR1.servosOn();
        }

        public void scSendIns(string order)
        {
            scaraR1.instr(order, 1);
        }

        public void scRefreshDir()
        {
            scaraR1.getDirectory();
        }

        public void scFileUpload(string name,string ext)
        {
            scaraR1.pc_ykFile(name, ext);
            readyDir = false;
            if(ext == "PGM")scRefreshDir();
        }

        public void scFileDownload(string name, string ext)
        {
            scaraR1.yk_pcFile(name, ext);
        }

        public void scRun(string name)
        {
            scaraR1.runProg(name);
        }
#endregion

    #region "LaserSensor"

        public void scDelete(string name)
```

```
        {
            scaraR1.deleteProg(name);
            readyDir = false;
            scRefreshDir();
        }

        public void lsManualSample()
        {
            lastsMeasures.Clear();
            laserSensor1.sampleManual();
            samplingTime.Enabled = false;
        }

        public void lsAutSample()
        {
            samplingTime.Interval = (int)interval*2;
            samplingTime.Enabled = true;
        }

        public void lsOnSen()
        {
            laserSensor1.on();
            samplingTime.Start();
        }

        public void lsOffSen()
        {
            laserSensor1.off();
            samplingTime.Stop();
        }

        public void lsGetSample()
        {
            sampleReady = false;
            laserSensor1.getManualSample();
        }

        public void lsBgLight(bool enable)
        {
            laserSensor1.switchBGlight(enable);
        }

        public void lsChangePref(string pref)
        {
            laserSensor1.changePref(pref);
        }


        /// <summary>
        /// Definimos el intervalo de tiempo de muestreo.
        /// </summary>
        /// <param name="intervalIn">intervalo de muestreo en ms.</param>
        public void lsSetInterval(double intervalIn)
        {
            //Los valores aceptados por el sensor son entre 8 y 50000 decimas de
milisegundo.
            if (intervalIn < 0.8 || intervalIn > 5000) throw new
IndexOutOfRangeException();
```

```
            else
            {
                interval = intervalIn;
                samplingTime.Interval = (int)intervalIn*2;
                laserSensor1.setsampleRate((int)(interval * 10));
            }
        }

        public double getInterval()
        {
            return interval;
        }

        public double[] getLastDistance()
        {
            if (lastsMeasures.Count > 0)
            {
                return new[] { lastsMeasures.Last(), height - crd.Z - sensOffset -
lastsMeasures.Last() };
            }
            else
                return new[] { distance, distanceFn };
        }

        public List<double> getDistances()
        {
            List<double> res = new List<double>(lastsMeasures);
            lastsMeasures.Clear();
            return res;
        }

 #endregion
    }
}
```

6.3.2.3 Objeto

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using OpenTK;
using OpenTK.Graphics.OpenGL;

namespace surfaceScan
{

    /// <summary>
    /// Representa la caja englobante de un objeto.
    /// </summary>
    public class boundingBox
    {
        modelCoord max;
        modelCoord min;

        public boundingBox(modelCoord first)
```

```
        {
            max = new modelCoord(first.X, first.Y, first.Z);
            min = new modelCoord(first.X, first.Y, first.Z);
        }

        /// <summary>
        /// Este parametro es el centro de la caja englobante, por lo que es el centro
del modelo.
        /// </summary>
        public modelCoord Center
        {
            get { return new modelCoord((max.X + min.X) / 2, (max.Y + min.Y) / 2, (max.Z
+ min.Z) / 2); }
        }

        public double getRadio()
        {
            double x = Math.Pow(max.X - Center.X, 2);
            double y = Math.Pow(max.Y - Center.Y, 2);
            double z = Math.Pow(max.Z - Center.Z, 2);
            return Math.Sqrt(x + y + z);
        }


        public void update_boundingBox(modelCoord point)
        {
            if (max.X < point.X) max.X = point.X;
            if (max.Y < point.Y) max.Y = point.Y;
            if (max.Z < point.Z) max.Z = point.Z;

            if (min.X > point.X) min.X = point.X;
            if (min.Y > point.Y) min.Y = point.Y;
            if (min.Z > point.Z) min.Z = point.Z;
        }
    }

    public class face
    {
        public List<int> verts;
        public face()
        {
            verts = new List<int>();
        }
    }

    public class objeto
    {
        public List<modelCoord> vertices;
        public List<face> faces;
        public string name;
        public boundingBox bounding;

        bool ready = false;

        public bool Ready
        {
            get { return ready; }
        }
```

```
        public objeto()
        {
            vertices = new List<modelCoord>();
            faces = new List<face>();
            name = "";
        }

        /// <summary>
        /// Renderizamos el objeto
        /// </summary>
        public void render()
        {
            //pintado sin materiales.
            modelCoord point;
            Matrix4d mat = Matrix4d.CreateTranslation(-bounding.Center.X, -
bounding.Center.Y, -bounding.Center.Z);
            GL.MultMatrix(ref mat);
            foreach (face f in faces)
            {
                GL.Begin(BeginMode.Polygon);
                GL.Material(MaterialFace.Front, MaterialParameter.AmbientAndDiffuse, 1);
                foreach (int mc in f.verts)
                {
                    //indexado de 1 a ... en un .obj
                    point = vertices[mc-1];
                    GL.Vertex3(point.X, point.Y, point.Z);
                }
                GL.End();
            }
        }

        /// <summary>
        /// Leemos un objeto (.OBJ), y creamos el modelo para el sistema.
        /// </summary>
        /// <param name="path">path del objeto</param>
        public string readSimpleObj(string path)
        {
            //Solo suponemos que estan los vertices y las caras, separados por lines.
            try
            {
                System.IO.StreamReader strw = new System.IO.StreamReader(path);
                while (!strw.EndOfStream)
                {
                    string[] temp = strw.ReadLine().Split(new char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
                    if (temp.Length != 0)
                    {
                        if (temp[0] != "#")
                        {
                            if (temp[0] == "g") name = temp[1];
                            else if (temp[0] == "v")
                            {
                                modelCoord a = new modelCoord(double.Parse(temp[1]),
double.Parse(temp[2]), double.Parse(temp[3]));
                                vertices.Add(a);
                                if (bounding == null)
                                {
```

```
                                bounding = new boundingBox(a);
                            }
                            else
                            {
                                bounding.update_boundingBox(a);
                            }
                        }
                        else if (temp[0] == "f")
                        {
                            face f = new face();
                            for (int i = 1; i < temp.Length; i++)
                            {
                                f.verts.Add(int.Parse(temp[i]));
                            }
                            faces.Add(f);
                        }
                    }
                }
                strw.Close();
                Console.WriteLine("Centro en {0},{1},{2}", bounding.Center.X,
bounding.Center.Y, bounding.Center.Z);
                Console.WriteLine("Radio = {0}", bounding.getRadio());
                ready = true;
                return name;
            }
            catch (Exception ex) { return "ERROR"; }
        }

        /// <summary>
        /// A partir de los vertices obtenidos del escaneado crea las caras del modelo.
        /// Solo sirve para mallas triangulares en la que los puntos no esten dispersos
(una matriz)
        /// y todas las zonas tengan la misma densidad de puntos.
        /// Primer algoritmo del diseño
        ///
        /// PRE: Tienen que estar todas las caras del modelo creadas o como minimo dos
filas completas.
        /// </summary>
        /// <param name="y">numero de columnas</param>
        public void createFaces(int nx,int ny)
        {
            if (vertices.Count <= ny) return;
            int x = 1;
            int y = 0;
            if (vertices.Count < 4) return;
            for (int i = GetIndex(1, 0, ny); i < vertices.Count-1; i++)
            {

                    ///caso normal: posicion (x,y) -> cara (x,y)(x-1,y+1)(x-1,y),
cara(x,y)(x,y+1)(x-1,y+1).
                    if (y > ny - 2 && x < nx)
                    {
                        x++; y = 0;
                    }
                    else
                    {
                        face aux = new face();
```

```
                        aux.verts.Add(i + 1);
                        aux.verts.Add(GetIndex(x - 1, y + 1, ny) + 1);
                        aux.verts.Add(GetIndex(x - 1, y, ny) + 1);
                        faces.Add(aux);

                        aux = new face();
                        aux.verts.Add(i + 1);
                        aux.verts.Add(GetIndex(x, y + 1, ny) + 1);
                        aux.verts.Add(GetIndex(x - 1, y + 1, ny) + 1);
                        faces.Add(aux);
                        y++;
                    }

            }
            ready = true;
        }

        private int[] matrixIndex(int indx, int cols)
        {
            return new[] {indx/cols,indx % cols};
        }

        private int GetIndex(int gx, int gy,int cols)
        {
            return gx * cols + gy;
        }
    }
}
```

6.3.2.4 GlForm

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using OpenTK;
using OpenTK.Graphics.OpenGL;

namespace surfaceScan
{
    public class glForm
    {
        modelCoord VRP = new modelCoord(0,0,0);
        double angleX = 50;
        double angleY = 20;
        double angleZ = 0;
        double wminx, wminy, wmaxx, wmaxy;
        double anterior, posterior;
        double dist = 80;
        int zoom = 1;

        modelo mdl;

        public glForm(modelo m)
        {
            mdl = m;
```

```
        }

        public void Init()
        {
            GL.ClearColor(System.Drawing.Color.Black);
            GL.Enable(EnableCap.AutoNormal);
            GL.PolygonMode(MaterialFace.FrontAndBack, PolygonMode.Line);
        }

        public void computeDefaultCamara()
        {
            double radio = 200;
            if (mdl.grafMod.bounding != null)
            {
                radio = mdl.grafMod.bounding.getRadio();
                dist = radio + 5;
                anterior = dist - radio;
                posterior = 2 * radio + dist;
            }

            wmaxx = radio;
            wmaxy = radio;
            wminy = -radio;
            wminx = -radio;
        }

        private void SetProjection(int w,int h)
        {
            GL.MatrixMode(MatrixMode.Projection);
            GL.LoadIdentity();
            if (mdl.Name == "" || mdl.Name == "ERROR")
            {
                GL.Ortho(-200 * zoom / 2, 200 * zoom / 2, -200 * zoom / 2, 200 * zoom /
2, -200, 200); // Bottom-left corner pixel has coordinate (0, 0)
            }
            else
            {
                GL.Ortho(wminx * zoom, wmaxx * zoom, wminy * zoom, wmaxy * zoom,
anterior, posterior);
            }
            GL.Viewport(0, 0, w, h); // Use all of the glControl painting area
        }

        private void SetModelView()
        {
            GL.MatrixMode(MatrixMode.Modelview);
            OpenTK.Matrix4d modelview = new Matrix4d();
            modelview = Matrix4d.Identity;

            Matrix4d trans1 = Matrix4d.CreateTranslation(0,0,dist);
            Matrix4d rotz = Matrix4d.CreateRotationZ(-angleZ);
            Matrix4d rotx = Matrix4d.CreateRotationX(angleX);
            Matrix4d roty = Matrix4d.CreateRotationY(-angleY);
            Matrix4d trans2 = Matrix4d.CreateTranslation(-VRP.X, -VRP.Y, -VRP.Z);
            modelview = Matrix4d.Mult(modelview, trans1);
            modelview = Matrix4d.Mult(modelview, rotz);
            modelview = Matrix4d.Mult(modelview, rotx);
            modelview = Matrix4d.Mult(modelview, roty);
```

```
            modelview = Matrix4d.Mult(modelview, trans2);

            GL.LoadMatrix(ref modelview);
        }

        public void paint(int width,int height)
        {
            GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);
            GL.LoadIdentity();

            SetProjection(width, height);
            SetModelView();

            GL.Begin(BeginMode.Lines);
            GL.Color3(System.Drawing.Color.Red); GL.Vertex3(0, 0, 0); GL.Vertex3(600, 0,
0);
            GL.Color3(System.Drawing.Color.Green); GL.Vertex3(0, 0, 0); GL.Vertex3(0,
600, 0);
            GL.Color3(System.Drawing.Color.Blue); GL.Vertex3(0, 0, 0); GL.Vertex3(0, 0,
400);
            GL.End();
            GL.Enable(EnableCap.Light0);
            GL.Color3(System.Drawing.Color.Yellow);
            if (mdl.grafMod.Ready)
            {
                mdl.grafMod.render();
            }
        }

        public void drawPoints()
        {
            modelo mdl = modelo.Instance;
            if (mdl.grafMod != null)
            {
                GL.Begin(BeginMode.Points);
                foreach (modelCoord m in mdl.grafMod.vertices)
                {
                    GL.Vertex3(m.X, m.Y, m.Z);
                }
                GL.End();
            }
        }

        public void Zoom(int z)
        {
            if (z < 0) zoom = 1 / Math.Abs(z);
            else if (z == 0) zoom = 1;
            else zoom = z;
        }

        public void Rotate(string axis, int dg)
        {
            switch (axis)
            {
                case "x":
                    angleX = dg;
                    break;
                case "y":
```

```
                    angleY = dg;
                    break;
                case "z":
                    angleZ = dg;
                    break;
            }
        }
    }
}
```

6.3.2.5 Eventos

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace surfaceScan
{
    /// <summary>
    /// EventArgs para pasar errores entre los componentes.
    /// </summary>
    public class errorEventArg : EventArgs
    {
        public errorEventArg(string s)
        {
            msg = s;
        }

        private string msg;

        public string Message
        {
            get { return msg; }
            set { msg = value; }
        }
    }

    public class rowScanEventArgs : EventArgs
    {
        public rowScanEventArgs(List<double> m)
        {
            measures = new List<double>(m);
        }

        private List<double> measures;

        public List<double> Measures
        {
            get { return new List<double>(measures); }
            set { measures = new List<double>(value); }
        }
    }

    /// <summary>
    /// EventArgs para pasar el directorio a el sistema.
```

```
/// </summary>
public class scaraReadyDirEventArg : EventArgs
{
    public scaraReadyDirEventArg(string s)
    {
        dir = s;
    }

    private string dir;

    public string Directory
    {
        get { return dir; }
        set { dir = value; }
    }
}

public class scaraTerminalEventArgs : EventArgs
{
    public scaraTerminalEventArgs(string text)
    {
        this.text = text;
    }

    private string text;

    public string Text
    {
        get { return text; }
        set { text = value; }
    }
}

/// <summary>
/// EventArgs para pasar una muestra del sensor laser a el sistema, cuando
/// este funciona en modo automatico
/// </summary>
public class newSampleEventArgs : EventArgs
{
    public newSampleEventArgs(double m)
    {
        sample = m;
    }

    private double sample;

    public double Sample
    {
        get { return sample; }
        set { sample = value; }
    }
}

public class newModelPointEventArgs : EventArgs
{
    public newModelPointEventArgs(modelCoord mC)
    {
        point = new modelCoord(mC.X, mC.Y, mC.Z);
```

```
        }

        private modelCoord point;

        public modelCoord Point
        {
            get { return point; }
        }
    }
}
```

6.3.2.6 Controlador interfaz visualizador

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using OpenTK;
using OpenTK.Graphics.OpenGL;
using System.Threading;

namespace surfaceScan
{
    public partial class visualizadorForm : Form
    {

        escaner escan;
        glForm glModel;

        bool loaded = false;
        bool modelScaned = false;

        public visualizadorForm()
        {
            try
            {
                InitializeComponent();
                initGrid();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void initGrid()
        {
            grid = new DataGridView();
            grid.Columns.Add("x", " X ");
            grid.Columns.Add("y", " Y ");
            grid.Columns.Add("z", " Z ");
        }
```

```
        private void initGlcontrol()
        {
            //glControl1.Enabled = false;
        }

        /// <summary>
        /// Visualizacion de un objeto ya escaneado, no permite el reescaneado de zonas
concretas.
        /// </summary>
        /// <param name="path"></param>
        public visualizadorForm(string path)
        {
            InitializeComponent();
            modelo mdl = modelo.Instance;
            glModel = new glForm(mdl);
            mdl.Name = mdl.grafMod.readSimpleObj(path);
            if (mdl.Name != "ERROR")
            {
                textBox1.Text = mdl.Name;
                textBox1.ReadOnly = true;
                foreach (modelCoord modC in mdl.grafMod.vertices)
                {
                    string auxX, auxY, auxZ;
                    auxX = modC.X.ToString();
                    auxY = modC.Y.ToString();
                    auxZ = modC.Z.ToString();
                    grid.Rows.Add(new string[] { auxX, auxY, auxZ });
                }
                glModel.computeDefaultCamara();
                glControl1.Update();
            }
        }


        public visualizadorForm(sistema sistem,string name, List<coordinates> area,
double precision, int tipo, double zH ,double offset)
        {
            InitializeComponent();
            initGlcontrol();
            textBox1.Text = name;
            modelo mdl = modelo.Instance;
            mdl.Name = name;

            escan = new escaner(sistem);
            escan.escanEndEvent += new EventHandler(escan_escanEndEvent);
            escan.newPoint += new escaner.newPointEventHandler(escan_newPoint);
            escan.escan(area, precision, tipo, offset);
        }

        private delegate void escanNewPoint(object sender, newModelPointEventArgs a);
        public void escan_newPoint(object sender, newModelPointEventArgs a)
        {
            if (this.grid.InvokeRequired) this.Invoke(new escanNewPoint(escan_newPoint),
this, a);
            else
            {
                grid.Rows.Add(a.Point.X, a.Point.Y, a.Point.Z);
                glControl1.Refresh();
```

```
            }
        }

        private delegate void endEscan(object sender, EventArgs e);
        private void escan_escanEndEvent(object sender, EventArgs e)
        {
            if (glControl1.InvokeRequired)
            {
                this.Invoke(new endEscan(escan_escanEndEvent),this,e );
            }
            else
            {
                modelo mdl = modelo.Instance;
                mdl.grafMod.name = mdl.Name;
                mdl.createFaces();
                glModel = new glForm(mdl);
                glModel.computeDefaultCamara();
                glControl1.Refresh();
            }
        }

        private delegate void glControl1_RefreshDL(object sender, EventArgs a);
        public void glControl1_Refresh(object sender, EventArgs e)
        {
            if (glControl1.InvokeRequired)
            {
                this.Invoke(new glControl1_RefreshDL(glControl1_Refresh), this, new
EventArgs());
            }
            else
            {
                glControl1.Refresh();
            }
        }

        private void button5_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void glControl1_Load(object sender, EventArgs e)
        {
            loaded = true;
            glModel.Init();
        }

        private void glControl1_Paint(object sender, PaintEventArgs e)
        {
            if (!loaded)
                return;

            glModel.paint(glControl1.Width,glControl1.Height);
            glControl1.SwapBuffers();
        }


        private void trackBar1_ValueChanged(object sender, EventArgs e)
        {
```

```
    glModel.Zoom(trackBar1.Value);
    glControl1.Refresh();
}

private void trackBar2_ValueChanged(object sender, EventArgs e)
{
    glModel.Rotate("x", trackBar2.Value);
    glControl1.Refresh();
}

private void trackBar3_ValueChanged(object sender, EventArgs e)
{
    glModel.Rotate("y", trackBar3.Value);
    glControl1.Refresh();
}

private void trackBar4_ValueChanged(object sender, EventArgs e)
{
    glModel.Rotate("z", trackBar4.Value);
    glControl1.Refresh();
}

private void svebutton_Click(object sender, EventArgs e)
{
    modelo mdl = modelo.Instance;
    if (mdl.grafMod.Ready)
    {
        saveFileDialog1.Filter = "Modelos 3D(*.obj)|*.obj";
        DialogResult res = saveFileDialog1.ShowDialog();
        if (res == System.Windows.Forms.DialogResult.OK)
        {
            mdl.save(saveFileDialog1.FileName);
        }
    }
}

private void visualizadorForm_FormClosing(object sender, FormClosingEventArgs e)
{
    this.Dispose();
}

private void button1_Click(object sender, EventArgs e)
{
    openFileDialog1.Filter = "Modelos 3D(*.obj)|*.obj";
    DialogResult res = openFileDialog1.ShowDialog();
    if (res == System.Windows.Forms.DialogResult.OK)
    {

        modelo mdl = modelo.Instance;
        mdl = mdl.newInstance;
        glModel = new glForm(mdl);
        mdl.Name = mdl.grafMod.readSimpleObj(openFileDialog1.FileName);
        if (mdl.Name != "ERROR")
        {
            textBox1.Text = mdl.Name;
            textBox1.ReadOnly = true;
            foreach (modelCoord modC in mdl.grafMod.vertices)
            {
```

```
                    string auxX, auxY, auxZ;
                    auxX = modC.X.ToString();
                    auxY = modC.Y.ToString();
                    auxZ = modC.Z.ToString();
                    grid.Rows.Add(new string[] { auxX, auxY, auxZ });
                }
                glModel.computeDefaultCamara();
                glControl1.Update();
            }
        }
    }
}
```