industriales
etsii UPCT

Universidad Politécnica de Cartagena

Ingeniería Industrial

Intensificación en sistemas eléctricos

Departamento de Ingeniería de Sistemas y Automática

Año académico: 2014/2015

Diseño de trayectorias del efector final de un robot manipulador para la evitación de obstáculos

PROYECTO FIN DE CARRERA

Autor: José Antonio Torres Sánchez

Directores: Jorge Feliu Batlle / José Luis Muñoz Lozano

Cartagena, 9 de diciembre de 2015

PREAMBULO

El actual proyecto se ha desarrollado en el laboratorio de I+D del Departamento de Ingeniería de Sistemas y Automática de la Universidad Politécnica de Cartagena, al que queremos agradecer el apoyo y la cesión de su instalaciones, especialmente a Jorge Feliu y a José Luis Muñoz por haberse ofrecido a dirigir nuestro proyecto.

También queremos dar las gracias a Carlos Díaz por su ayuda y paciencia, lo que nos ha resultado clave para llevar a buen puerto todo lo que hemos realizado. También a Pablo Martínez por estar siempre dispuesto a echar una mano y prestarnos el equipo de grabación.

El objetivo que se va a perseguir en los próximos capítulos es la planificación de trayectorias a seguir por un brazo robótico para conseguir la evitación de los obstáculos que se encuentre para llegar a una posición final.

Este proyecto se ha llevado a cabo de forma conjunta con el proyecto de Lidia Sánchez Martínez en el que se muestra como reconocer objetos y obstáculos a través del sensor Sick LMS 200 cuyas características se especifican en el Capítulo 4.

Mostraremos como conseguir que el extremo del brazo se desplace desde un punto inicial hasta un punto final, evitando los objetos en su camino detectados por el sensor, todo ello programado en C++ empleando *Visual Studio 2010 Ultimate*. Indicaremos todos los pasos a seguir para conseguir dicho objetivo con el fin de que, cualquier otro estudiante o persona con limitados conocimientos de programación y automática pueda reproducir rápidamente y sin ningún problema todo lo que hemos conseguido.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1:

Motivaciones

1. MOTIVACIONES

Hay grandes dudas sobre cómo se desarrollarán a corto, medio y largo plazo los posibles caminos que pueden tomar la evolución la industria, el comercio y la búsqueda de nuevas fuentes y recursos energéticos. Tampoco se ponen de acuerdo los expertos en la forma de salir de la actual crisis económica, ni siquiera en cuando estaremos preparados para dar el salto definitivo a las energías renovables. Sin embargo, de lo que no hay duda pase lo que pase en el futuro es del papel primordial y absolutamente necesario de la robótica, la automatización industrial y la ingeniería de control.

Cuesta imaginarse cualquier empresa que no pueda resultar ampliamente beneficiada económicamente de la automatización de sus procesos. Es por ello que, sin duda alguna, siempre serán valorados conocimientos de programación y de ingeniería de control, independientemente del resto de requisitos que se precisen.

Asimismo, una de las grandes ambiciones de la Humanidad desde el origen de los tiempos es conocer el universo que nos rodea, más allá de nuestro planeta y del Sistema Solar. Las grandes empresas espaciales, como la N.A.S.A. o la agencia Espacial Europea (E.S.A.) no cesan en sus intentos por llegar más lejos posible y descubrir el origen del agua y de la vida. Todo ello no sería posible sin el desarrollo de la Ingeniería de Control.

Todo lo anteriormente descrito, sin duda, hace que desarrollar un proyecto que permita profundizar en los conocimientos sobre esta materia sea una experiencia muy atractiva para cualquier estudiante. Todo ello unido a la magnífica labor docente llevada a cabo por el departamento en asignaturas como “Teoría de Sistemas”, “Ingeniería de control” o “Robótica Industrial” me han animado a dar el paso definitivo a realizar este proyecto.

CAPÍTULO 2:

Introducción

2. INTRODUCCIÓN

2.1. ESTADO DEL ARTE

Hay multitud de trabajos, especialmente aquellos que implican un esfuerzo físico excesivo o bien son extremadamente monótonos o peligrosos, que a las personas no les gusta o no están dispuestas a hacer.

Es aquí donde la evolución de la robótica juega un papel primordial. Además, la velocidad y precisión de los equipos automatizados hacen de ellos el trabajador más eficiente con el que ningún otro trabajador humano podría competir. Además, parece obvio que los máquinas no necesitan descansar (salvo en determinadas circunstancias especiales, como de mantenimiento), comer, un salario o una zona segura para trabajar.

2.1.1. Robots

Podemos definir un robot como un conjunto de dispositivos que reciben información (sensores) y que, por medio de la interfaz adecuada, emplean dichos datos para realizar un movimiento de salida (evitar obstáculos, sujeción de objetos, etc.) bien a través de sistema operativo externo conectado con el robot y con los sensores, o bien a través de microprocesadores implementados en el propio robot.

2.1.2. Cinemática del robot

Estudio de su movimiento con respecto a un sistema de referencia. Hay dos tipos de análisis cinemáticas posibles:

- Análisis cinemático directo: Determinar la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas de referencia, conocidos los ángulos de las articulaciones y los parámetros geométricos de los elementos del robot.
- Problema cinemático inverso: Determinar la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas.

2.1.3. Jacobiano

Matemáticamente las ecuaciones cinemáticas directas definen una función entre el espacio cartesiano de posiciones y orientaciones y el espacio de las articulaciones. Las relaciones de velocidad se determinan por el Jacobiano de esta función.

El Jacobiano es una matriz compuesta por las derivadas de una función escalar. Es imprescindible su uso en el análisis y control del movimiento de un robot (planificación y ejecución de trayectorias suaves, determinación de configuraciones singulares, ejecución de movimientos coordinados, derivación de ecuaciones dinámicas).

En muchos casos y sistemas operativos modernos las librerías encargadas de crear la trayectoria de cada articulación para alcanzar la posición deseada vienen preconfiguradas, por lo que se facilita su uso en aquellas personas con limitado o nulo conocimiento de programación.

2.1.4. Articulaciones y grados de libertad

Con la misma morfología que un brazo humano, un brazo robótico industrial está formado por una serie de partes rígidas unidas por articulaciones. A esta unión se le denomina cadena.

Al inicio de esta cadena se le denomina base. En el caso que nos ocupa (brazo robótico) esta base será fija (*fixed*), aunque también podría no serlo si estamos considerando un robot móvil.

En el extremo opuesto a la base se acopla la herramienta de trabajo (desde una mano para el agarre hasta cualquier otro utensilio para pintar, atornillar, etc.).

En muchos casos, las articulaciones permiten que entre las partes que unen (también llamadas ejes), se pueda producir un movimiento de desplazamiento, de giro o una combinación de ambos.

Con tres traslaciones según el respectivo eje X, Y o Z y tres giros o rotaciones (yaw, pitch, roll) relacionadas con estos mismos ejes, podemos posicionar cualquier elemento, objeto u herramienta en el espacio. Generalmente los robots consiguen el posicionado por medio de sus tres primeras articulaciones a partir de la base y la orientación de su elemento terminal o herramienta con el resto de articulaciones. No es necesario que un robot tenga los 6 GDL para todas las aplicaciones, hay robots con solo 3 GDL.

Por otro lado, también se habla de robots con más de 6 articulaciones y GDL, que permiten aumentar la accesibilidad a ciertas zonas de trabajo; en este caso y para un solo brazo-robot, no se tienen más de 6 GDL, pues alguna de las articulaciones o ejes proporcionan falsos GDL que son repetidos de los proporcionados por otras articulaciones.

2.2. UN POCO DE HISTORIA...

Desde la Antigüedad, los hombres han intentado construir máquinas automáticas que faciliten su trabajo, hagan más cómoda su existencia, satisfagan su curiosidad y su afán de aprender e investigar, o simplemente les sirvan de entretenimiento.

Ya en la Antigua Grecia se construyeron ingenios de funcionamiento automático a los que llamaron autómatas; posteriormente en la Edad Media y en el Renacimiento se siguieron fabricando diversos autómatas, entre ellos el Gallo de Estrasburgo (1230) y el León Animado de Leonardo Da Vinci.

Durante los siglos XVII y XVIII se crearon ingenios mecánicos de mayor complejidad que tenían alguna de las características de los robots actuales. Jacques de Vaucanson (1709-1782) construyó varios autómatas, uno de los más conocidos es un pato mecánico, que bebe, come, grazna, chapotea en el agua y digiere su comida “como un pato verdadero”; estos primeros autómatas estaban destinados fundamentalmente a ser exhibidos en las ferias y servir de entretenimiento en las Cortes y entre la nobleza.

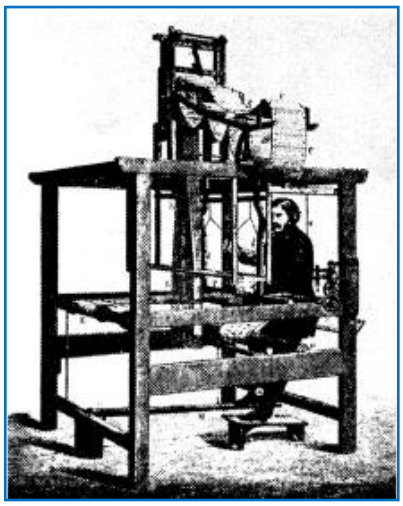


Ilustración 2.1. Telar de J. Jacquard

A finales del siglo XVIII y también a principios del XIX se desarrollaron algunas máquinas para empleo en la industria textil, entre las que ya había algún telar en el que mediante el uso de tarjetas perforadas se podía elegir el tipo de tela a tejer, este hito, constituyó uno de los primeros precedentes históricos de las máquinas programadas por control numérico.

Un poco más tarde que en la industria textil, se incorporan los automatismos en las industrias mineras y metalúrgicas, es la época de las máquinas de vapor; James Watt (1736-1819) contribuyó decisivamente al desarrollo de estas máquinas e inventó el llamado regulador de Watt, que es un regulador centrifugo de acción proporcional; con él nació el concepto de realimentación y la regulación automática, que es una de las bases de los robots industriales actuales, pues entre otros elementos, incorporan diversos sensores y reguladores PID.

La palabra robot se empleó por primera vez en 1920 en una obra de teatro llamada *Robots Universales Rossum* escrita por el dramaturgo checo Karel Capek, se deriva de la palabra checa "robotnik" y significa, siervo, servidor o trabajador forzado. Es en el siglo XX cuando se empieza a hablar de robots y se produce el desarrollo de estos, que va ligado con el desarrollo de los microprocesadores. En la tabla siguiente se citan algunos hechos destacables con sus fechas aproximadas:

1954	A partir de esta fecha, el estadounidense George Devol, comienza la construcción de un brazo articulado que realiza una secuencia de movimientos programables por medio de computador; se considera que este "brazo" es el primer robot industrial.
1956	Devol conoció a Joseph Engelberger y juntos fundaron en 1960 la empresa *Unimation* dedicada a la fabricación de robots.

1961	Se realizan pruebas de un robot *Unimate* accionado hidráulicamente, en un proceso de fundición en molde en General Motors.
1968	Kawasaki se une a *Unimation* y comienza la fabricación y el empleo de robots industriales en Japón. En este año *General Motors*, emplea baterías de robots en el proceso de fabricación de las carrocerías de los coches
1973	La empresa sueca ASEA, fabrica el primer robot completamente eléctrico, es el tipo de accionamiento que ha acabado imponiéndose, debido a los avances registrados en el control de motores eléctricos.
1974	Se introduce el primer robot industrial en España. También es el año en el que se comienza a usar el lenguaje de programación AL, del que derivarán otros de uso posterior como el VAL (*Victor´s Assembly Languaje*) de los robots PUMA, implementado en 1975 por Victor Scheinman, que junto a Devol y Engelberger, son pioneros en la robótica industrial.
1978	Comienza a emplearse el robot PUMA (*Programmable Universal Machine for Assambly*) de *Unimatión*, que es uno de los modelos que más se ha usado, su diseño de "brazo" multiarticulado es la base de la mayoría de los robots actuales.
1981	Comienza la comercialización del robot tipo SCARA (*Selective Compliance Arm for Robotic Assambly*) en Japón.
1987	Se constituye la Federación Internacional de Robótica con sede en Estocolmo.

Ilustración 2.2.Cronología de los avances modernos de la robótica

Durante estos años, se sientan las bases de la robótica industrial, posteriormente irá aumentando el empleo y la sofisticación de los robots con el aumento de las prestaciones de los microprocesadores y las posibilidades de la informática.

CAPÍTULO 3:

Microsoft Visual Studio 2010 Ultimate

3. MICROSOFT VISUAL STUDIO 2010 ULTIMATE

Como se ha dicho anteriormente, para llevar a cabo la programación necesaria en este proyecto se ha empleado este software, cuyas características se exponen a continuación:

3.1. DESCRIPCIÓN

Microsoft Visual Studio 2010 Ultimate es un exhaustivo paquete de herramientas de administración del ciclo de vida de las aplicaciones para equipos. Con este paquete es posible garantizar la calidad de los resultados, desde el diseño hasta la implementación. Tanto para soluciones nuevas como para mejorar las aplicaciones ya existentes, *Visual Studio 2010 Ultimate* resulta una herramienta de gran utilidad gracias a que admite un número cada vez mayor de plataformas y tecnologías (incluídos los sistemas informáticos en cloud y en paralelo).

3.2. VENTAJAS

Son muchas las ventajas del empleo de este software, enumeramos algunas de ellas:

- **Permite una actitud proactiva, así como la administración ágil de un proyecto**

Visual Studio 2010 Ultimate está optimizado para el proceso de desarrollo iterativo actual con características que ayudan a mantener la productividad y a reaccionar frente a posibles riesgos antes de que se produzcan. Permite supervisar el estado del proyecto mediante informes que se generan automáticamente. Además, es posible administrar la capacidad del proyecto con datos históricos y documentos de planificación basados en *Microsoft Excel*.

- **Características de *Visual Studio 2010 Ultimate***

Microsoft Visual Studio 2010 Ultimate incluye potentes herramientas que simplifican todo el proceso de desarrollo de aplicaciones, de principio a fin. Los equipos pueden observar una mayor productividad y ahorro de costes al utilizar características de colaboración avanzadas, así como herramientas de pruebas y depuración integradas que le ayudarán a crear siempre un código de gran calidad.

- **Administración del ciclo de vida de las aplicaciones (ALM)**

La creación de aplicaciones de éxito requiere un proceso de ejecución uniforme que beneficie a todos los componentes del equipo. Las herramientas de ALM integradas en *Visual Studio 2010 Ultimate* contribuyen a que las organizaciones colaboren y se comuniquen de forma efectiva en todos los niveles, y a que se hagan una idea precisa del estado real del proyecto, lo que garantiza que se ofrezcan soluciones de gran calidad al tiempo que se reducen los costos.

- **Depuración y diagnóstico**

Visual Studio 2010 Ultimate presenta *IntelliTrace*, una valiosa característica de depuración que hace que el argumento "no reproducible" sea cosa del pasado. Los evaluadores pueden archivar errores enriquecidos y modificables para que los desarrolladores puedan reproducir siempre el error del que se informe en el estado en el que se encontró. Otras características incluyen análisis de código estático, métricas de código y creación de perfiles.

- **Herramientas de prueba**

Visual Studio 2010 Ultimate incorpora multitud de herramientas avanzadas de pruebas para ayudar a garantizar la calidad del código en todo momento. Pruebas de IU codificadas, que automatizan la realización de pruebas de la interfaz de usuario en aplicaciones basadas en web y en Windows®, así como de pruebas manuales, *Test Professional*, pruebas de rendimiento de web, pruebas de carga, cobertura de código y otras características completas que no se encuentran en otras ediciones de *Visual Studio*.

- **Arquitectura y modelado**

El Explorador de arquitectura de *Visual Studio 2010 Ultimate* ayuda a entender los activos de código existentes y otras interdependencias. Los diagramas por capas ayudan a garantizar el cumplimiento de la arquitectura y permiten validar artefactos de código con respecto al diagrama. Además, *Visual Studio 2010 Ultimate* admite los cinco diagramas de UML más comunes que conviven junto con su código.

- **Desarrollo de bases de datos**

El desarrollo de bases de datos requiere el mismo cuidado y atención que el desarrollo de aplicaciones. *Visual Studio 2010 Ultimate* es consciente de ello y

proporciona potentes herramientas de implementación y administración de cambios que garantizan que la base de datos y la aplicación estén siempre sincronizadas.

- **Entorno de desarrollo integrado**

Multitud de características personalizables como, por ejemplo, compatibilidad con varios monitores. Es posible dar rienda suelta a la creatividad utilizando los diseñadores visuales para mejorar las últimas plataformas, incluido *Windows 7*.

- **Compatibilidad con la plataforma de desarrollo**

Tanto para crear soluciones nuevas como para mejorar las aplicaciones ya existentes, *Visual Studio 2010 Ultimate* permite hacer realidad dicha idea en una gran variedad de plataformas, entre las que se incluyen *Windows, Windows Server*, *Web*, *Cloud*, *Office* y *SharePoint*, entre otras, todo en un único entorno de desarrollo integrado.

- **Team Foundation Server**

Team Foundation Server (TFS) es la plataforma de colaboración sobre la que se asienta la solución de administración de ciclo de vida de aplicaciones de *Microsoft*. TFS automatiza y simplifica el proceso de entrega de software, y proporciona rastreabilidad completa y la posibilidad de comprobar en tiempo real el estado de los proyectos (para todos los miembros del equipo) con potentes herramientas de elaboración de informes y paneles.

- **Lab Management**

Visual Studio 2010 Ultimate ofrece un conjunto completo de características de laboratorio de pruebas, incluido el aprovisionamiento de entornos a partir de plantillas, la configuración y el desmontaje de entornos virtuales y entornos de puntos de comprobación. (*Lab Management* estará disponible como candidato a la versión comercial como RTM y se distribuirá posteriormente.)

- **Suscripción a MSDN**

Visual Studio 2010 Ultimate con MSDN es la oferta más completa para los desarrolladores. Además de todas las características incluidas en *Visual Studio 2010 Professional* con MSDN y *Visual Studio 2010 Premium* con MSDN, *Ultimate* con

MSDN incluye más horas de uso de Azure, acceso no V*isual Studio a Team Foundation Server* a través de *Teamprise* y software de administración de pruebas y laboratorio.

- **Comprobar el código usando la automatización de IU**

Las pruebas automatizadas que controlan la aplicación a través de la interfaz de usuario se conocen como *pruebas de IU codificadas* (CUIT). Estas pruebas incluyen una comprobación funcional de los controles de la interfaz de usuario. Permiten comprobar si toda la aplicación, incluida la interfaz de usuario, funciona correctamente. Las pruebas de IU codificadas son especialmente útiles donde haya una validación o cualquier otra lógica en la interfaz de usuario, por ejemplo, en una página web. También se suelen usar para automatizar una prueba manual existente.

Como se muestra en la ilustración siguiente, una experiencia típica de desarrollo podría ser aquella donde, inicialmente, el usuario se limite a compilar la aplicación (F5) y a hacer clic en los controles de la interfaz de usuario a fin de comprobar que todo funciona correctamente. Después, se puede decidir crear una prueba codificada de forma que no sea necesario seguir probando la aplicación manualmente. Dependiendo de la funcionalidad concreta que se prueba en la aplicación, se puede escribir código para una prueba funcional o bien una prueba de integración que puede que incluya o no la realización de pruebas en el nivel de interfaz de usuario. Si simplemente se desea tener acceso directamente a alguna lógica de negocios, se puede codificar una prueba unitaria. Sin embargo, en algunas circunstancias, puede ser beneficioso incluir pruebas de los diversos controles de IU en la aplicación. Una prueba de IU codificada puede automatizar el escenario inicial (F5), comprobando que la renovación de código no afecte a la funcionalidad de la aplicación.

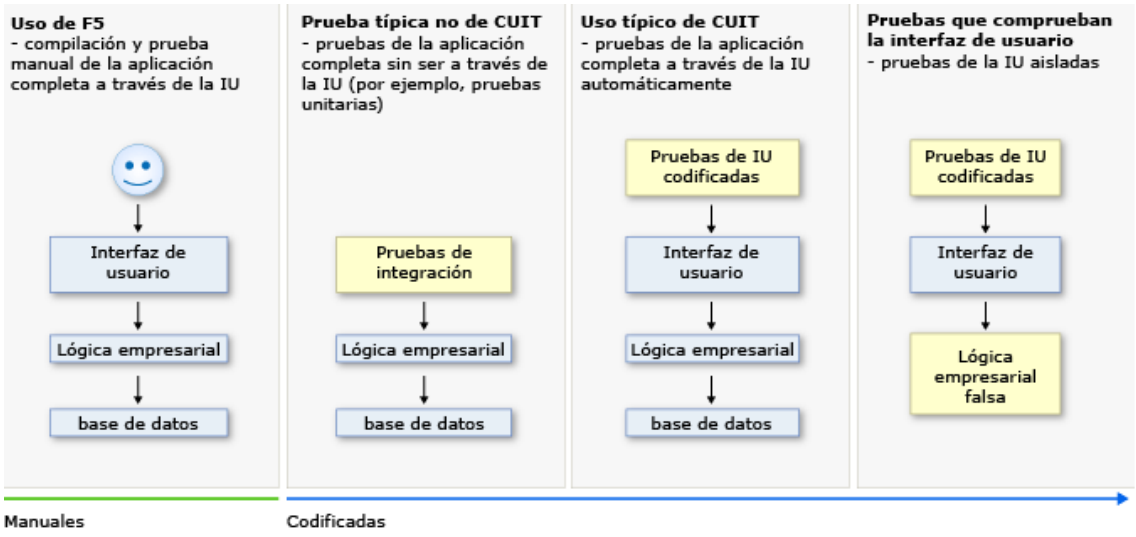


Ilustración 3.1. Pruebas durante el desarrollo de la aplicación

3.3. MICROSOFT FOUNDATION CLASSES (MFC)

Dentro de las posibilidades en cuanto a la selección de la interfaz para trabajar en C++, seleccionamos la descrita a continuación, la más ampliamente utilizada en la actualidad y con expectativas de que siga siendo así en el futuro. En este apartado nos referiremos a un conjunto de clases interconectadas por múltiples relaciones de herencia, que proveen un acceso más sencillo a las API de Windows. Fueron introducidas por Microsoft en 1992 y desde entonces fueron apareciendo nuevas versiones con las actualizaciones del entorno de programación Visual C++, gracias a las cuales éste se convierte en un generador de programas C++ para Windows. Tiene una gran complejidad añadida debido a la necesidad de que el programador ahora no sólo debe controlar C/C++, sino que además debe conocer las clases de la MFC para poder utilizar su potencia. Con el paso del tiempo *Microsoft Foundation Classes* se ha convertido en la implementación estándar de la industria para la creación de aplicaciones gráficas en plataformas PC. A pesar de tener sus limitaciones, su adopción demuestra los beneficios de productividad de la reutilización de marcos comunes para desarrollar aplicaciones gráficas para negocios.

3.3.1. Características

MFC proporciona C++ para Windows macros de tratamiento de mensajes (a través de mapas de mensajes), las excepciones en tiempo de ejecución e identificación del tipo RTTI, la serialización y la creación de instancias de clases dinámicas. Las macros para manejo de mensajes dirigidos a reducir el consumo de memoria, evitando el uso gratuito de tablas virtuales y también para proporcionar una estructura más concreta para diversos Visual C++ -suministrado herramientas para editar y manipular el código sin necesidad de analizar el lenguaje completo. Las macros de tratamiento de mensajes reemplazado el mecanismo de función virtual proporcionada por el C++.

- ***Usos***

El *Microsoft Foundation Class* (MFC) de la biblioteca ofrece un ejemplo bien conocido de un software eficaz marco. El MFC es una biblioteca de clases C++ que proporciona una interfaz para la programación de Windows y al mismo tiempo encapsula el nivel inferior de la API Win32. Proporciona una gran cantidad de funcionalidades que se encuentran en Aplicaciones de Windows, como la gestión de documentos y la gestión de los distintos puntos de vista sobre los datos del documento, y a su vez proporciona una interfaz orientada a objetos que solucionan las complejas tareas que involucran la comunicación a través redes, el acceso a la base de datos y gestión de documentos compuestos. Las aplicaciones de Windows se construyen mediante la especialización de los componentes que se encuentran en el marco de trabajo de MFC, como la Clases C View y C Document, para cumplir con los requisitos de la aplicación.

3.3.2. Tipos de aplicaciones:

Es posible crear tres tipos de aplicaciones por medio de MFC.

- Single-document interface (SDI): cada documento viene asociado con una vista simple, en una única ventana, aunque en ocasiones sería deseable tener la capacidad de cambiar la vista actual del documento por otra nueva. single-document interface (SDI)-
- Multiple-document interface (MDI) permite crear aplicaciones en las que sea posible mostrar múltiples documentos simultáneamente, cada uno de ellos en su

propia ventana. A menudo se ofrece un menú con submenús para cambiar entre ventanas y documentos.

- Dialog-based: de corta duración, las aplicaciones para tareas simples se pueden implementar en los cuadros de diálogo. Usaremos esta interfaz al ser la más apropiada para el fin que buscamos.
- Multiple top-level documents: permite abrir muchas partes del documento, cada una con su propio menú e iconos. También se pueden cerrar individualmente, pero si se selecciona la opción de salir desde el menú principal la aplicación Cierra todas sus partes.

3.4. CREACIÓN DE NUESTRTO PROYECTO

- Después de instalarla, abrimos nuestra versión de *Microsoft Visual Studio 2010 Ultimate.*
- En el menú principal seleccionamos File > New > Project

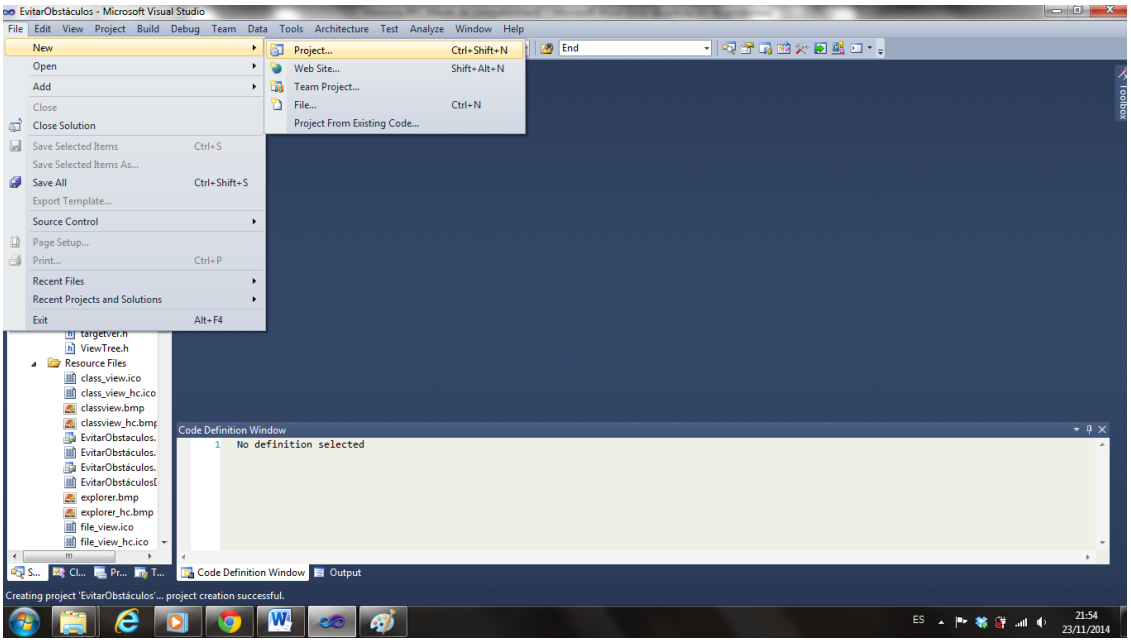


Ilustración 3.2. Nuevo proyecto de Microsoft Visual Studio.

- A continuación: Visual C++ > MFC > MFC Application.

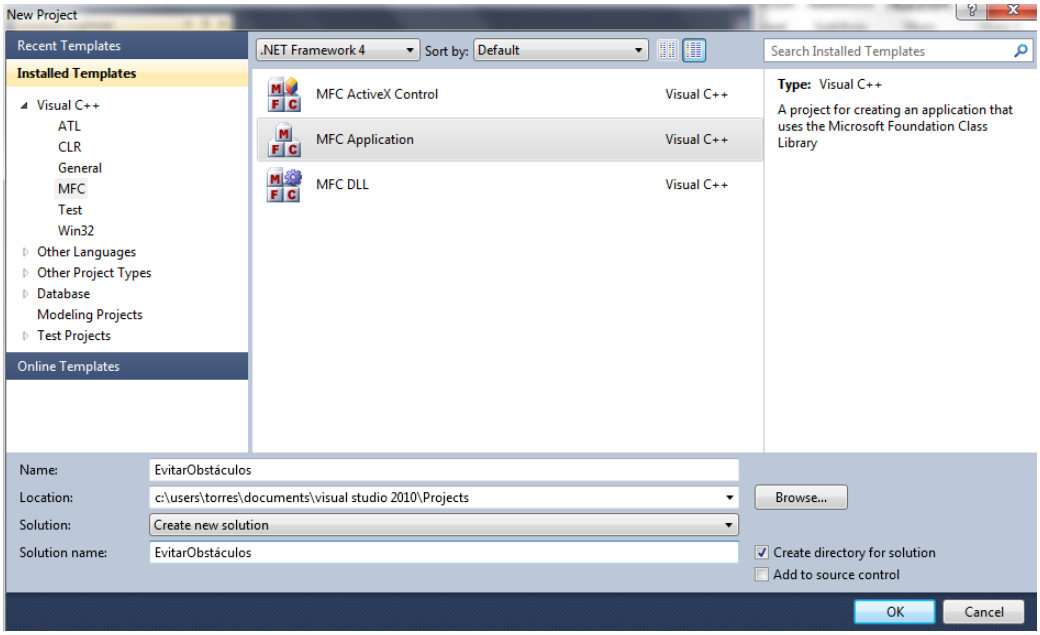


Ilustración 3.3. Selección de tipo de aplicación y nombre.

- Seleccionamos el nombre del proyecto, sin espacios ni caracteres especiales, así como la carpeta donde queremos localizar el archivo.

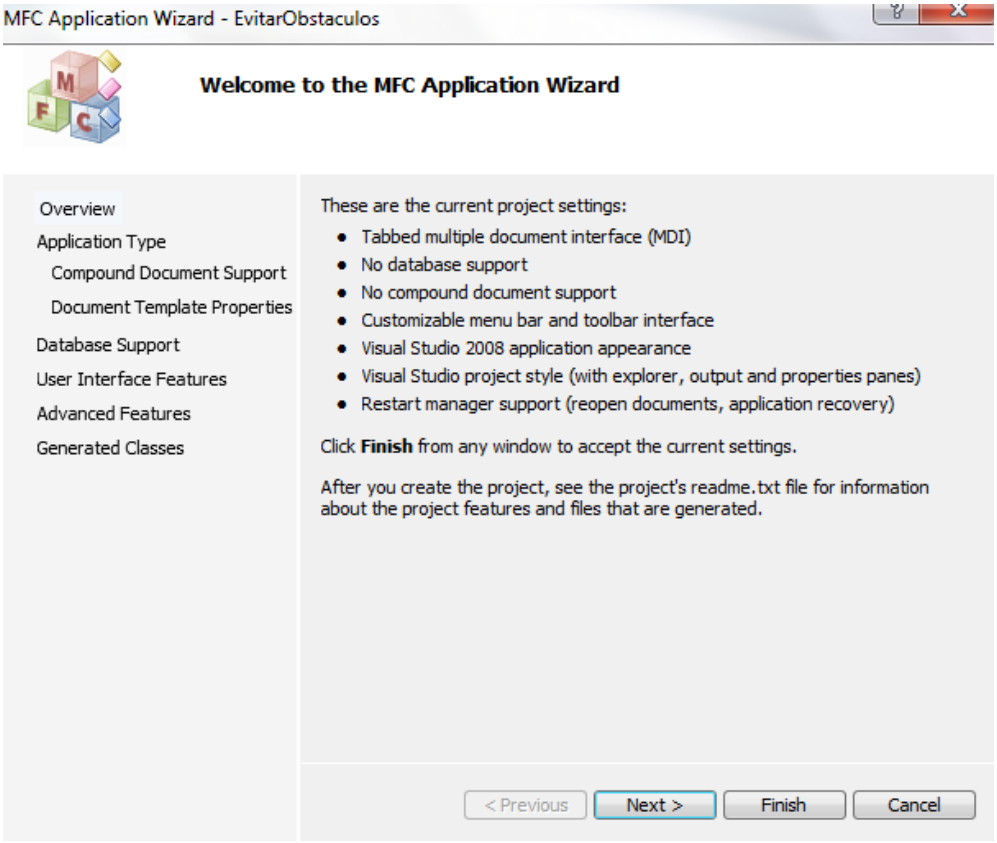


Ilustración 3.4. Selección de parámetros I.

- A continuación nos aparece una ventana donde se nos indica las características por defecto del proyecto actual. Podemos confirmarlas pulsando "Finish" o bien modificarlas pulsando "Next". En nuestro caso vamos a modificar algunos de estos parámetros.

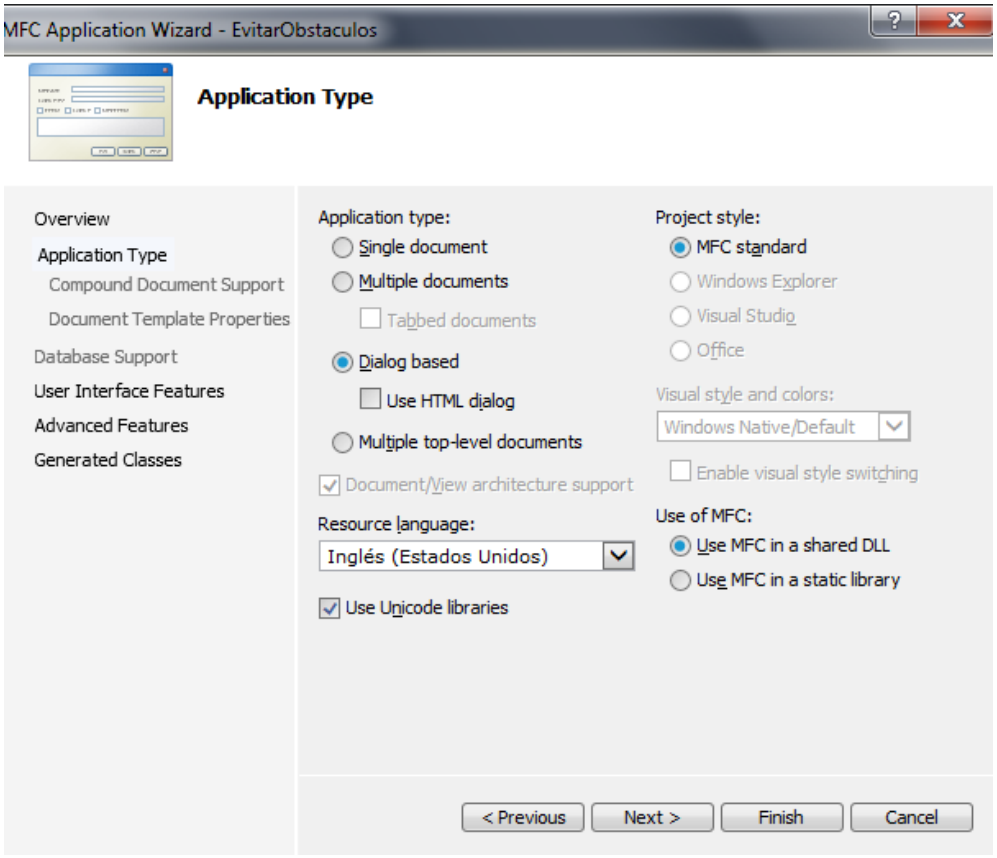


Ilustración 3.5. Selección de parámetros II.

- Seleccionamos la aplicación de tipo cuadros de diálogo (Application Type > Dialog based) por los motivos comentados en el apartado 3.3 de este capítulo.

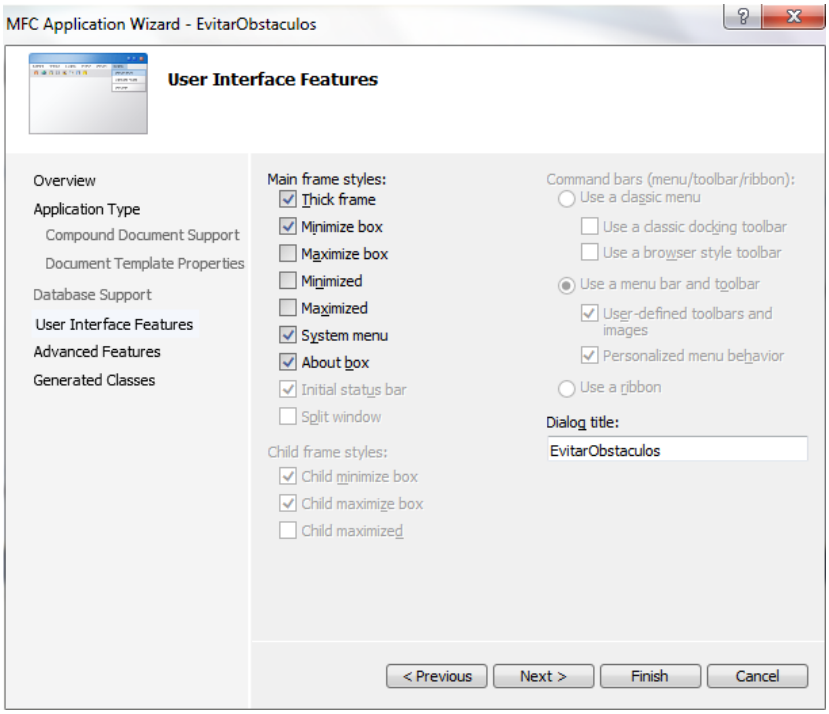


Ilustración 3.6. Selección de parámetros III.

- A continuación se nos solicita el estilo que queremos dar al borde externo de la interfaz gráfica donde vamos a ejecutar el código. Además de "Thick frame", "System menu" y "About box" que vienen por defecto, añadimos por comodidad "Minimize box" para poder minimar dicha ventana.

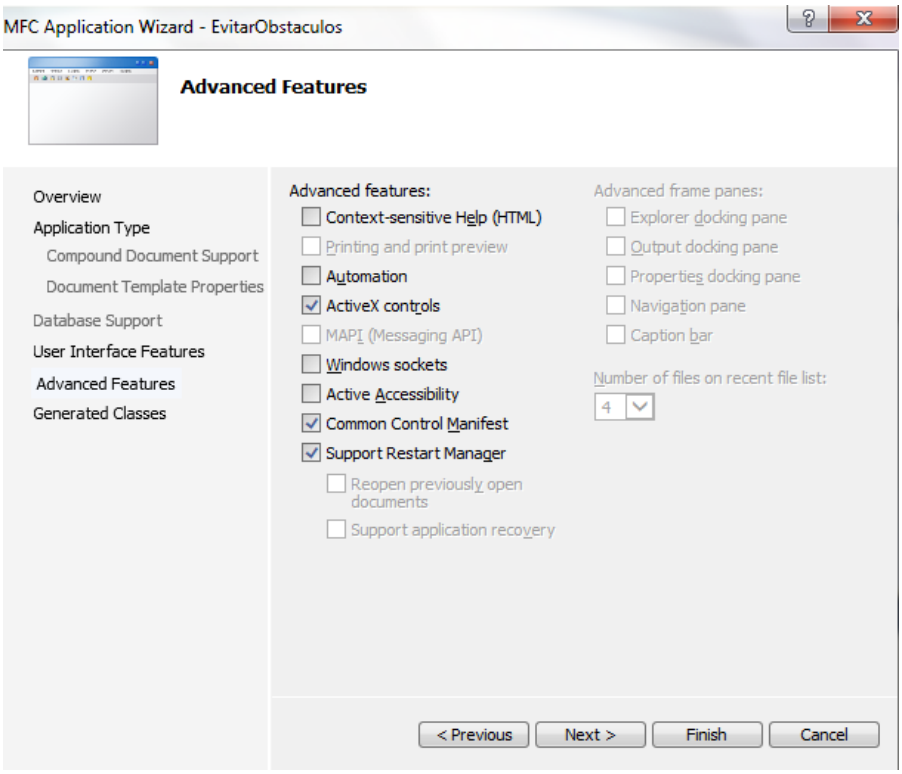


Ilustración 3.7. Selección de parámetros IV.

- En esta ventana se nos permite modificar algunas características avanzadas. No será necesario modificar lo que aparece por defecto.

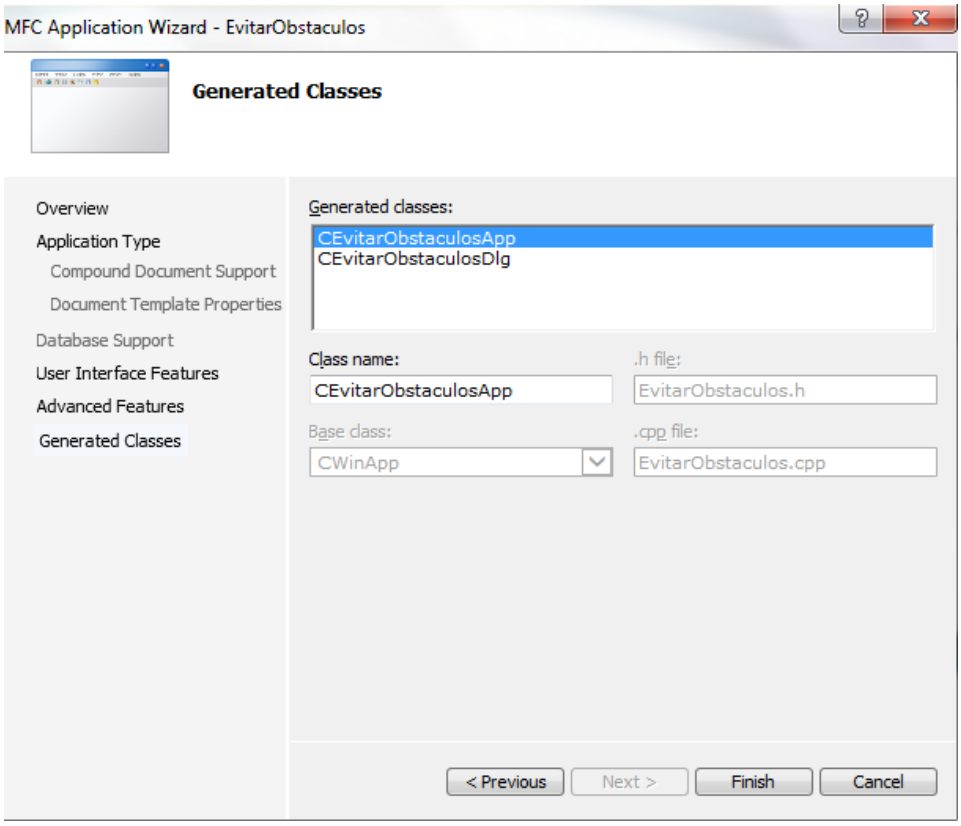


Ilustración 3.8. Selección de parámetros V.

- Igualmente, en esta última ventana dejaremos los parámetros que aparecen por defecto.

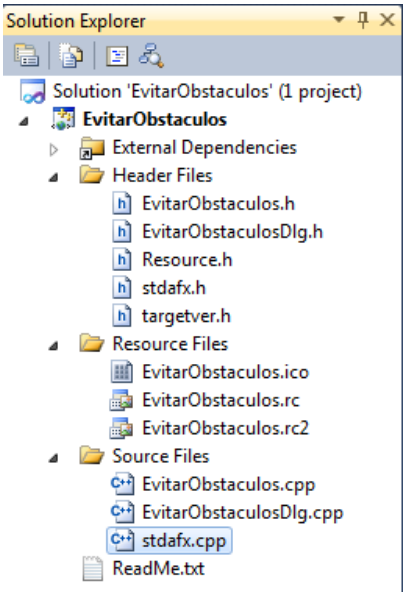


Ilustración 3.9. Distribución del proyecto.

3.5. REQUISITOS DEL SISTEMA

Requisitos de software

Visual Studio 2010 puede instalarse en los sistemas operativos siguientes:
Windows XP (x86) con Service Pack 3 – todas las ediciones, excepto Starter Edition
Windows Vista (x86 y x64) con Service Pack 1 – todas las ediciones, excepto Starter Edition
Windows 7 (x86 y x64)
Windows Server 2003 (x86 y x64) con Service Pack 2
Windows Server 2003 R2 (x86 y x64)
Windows Server 2008 (x86 y x64) con Service Pack 2
Windows Server 2008 R2 (x64)

Arquitecturas compatibles:

32 bits (x86)
64 bits (x64)

Requisitos de hardware

Equipo con un procesador de al menos 1,6 GHz
1024 MB de RAM
Espacio disponible en disco duro de 3 GB
Unidad de disco duro de 5.400 RPM
Tarjeta de vídeo compatible con DirectX 9 con una resolución de 1280 x 1024 o superior
Unidad de DVD-ROM

-Ver más aquí: http://www.identi.li/index.php?topic=178232#sthash.ehn9s0PM.dpuf

CAPÍTULO 4:

Descripción del Hardware utilizado

4. DESCRIPCIÓN DEL HARDWARE UTILIZADO.

En este capítulo presentaremos las características del Hardware que hemos empleado para llevar a cabo el proyecto descrito en los sucesivos capítulos.

4.1. SICK LMS 200.

4.1.1. Descripción

Dispositivo que permite medir distancias en una ángulo de 180º. Se ha colocado enfrente del robot y se comunica por el puerto serie con la CPU del mismo. Su funcionamiento se basa en un mecanismo de espejo rotatorio, que va dirigiendo un pulso láser. Las medidas se calculan por tiempo de vuelo, en tiempos inferiores a 70ms, y da medidas correctas sobre casi todo tipo de superficies. Más información en: *https://www.mysick.com*

Ilustración 4.1. Escáner Sick LMS 200.

4.1.2. Características

Features	
Field of application:	Indoor
Type:	Short Range
Light source:	Infrared (905 nm)
Laser class:	1 (EN/IEC 60825-1), eye-safe
Field of view:	180 °
Scanning frequency:	75 Hz
Angular resolution:	0.25 ° 0.5 ° 1 °
Heating:	Optional via external heating plate
Operating range:	0 m ... 80 m
Max. range with 10 % reflectivity:	10 m
Fog correction:	no
MTBF:	80,000 h

Ilustración 4.2. Características Sick LMS 200.

4.1.3. Funcionamiento

Es muy importante detenerse un momento en este punto para percatarse del error sistemática (+/- 15 mm) y del error estadístico (+/- 5 mm) de las mediciones proporcionadas por el sensor. Esto ocasionará, si estamos interactuando con objetos pequeños y a distancia pequeña, una fuente de error. Por ello, es posible que, a pesar de ser el código correcto, pueda generarse un comportamiento no deseado. La única solución es emplear un margen de maniobra mayor al previsto inicialmente, o bien alejar el escáner y emplear objetos de mayores dimensiones. Si se quiere verificar la exactitud del código y no perder la cabeza con este error no controlable, se recomiendo comentar en un primer momento las líneas que proporcionan las medidas SDIST [0], SDIST [1] y SDIST [2] y sustituirlo por la distancia real medida manualmente del objeto al origen del sensor. No será necesario modificar en ningún momento los valores de SANG [i], pues los ángulos recogidos no están sometidos a ningún error que se necesario considerar y podrán ser tomados como los valores reales.

Performance	
Response time:	≥ 13 ms
Detectable object shape:	Almost any
Systematic error:	± 15 mm
Statistical error:	± 5 mm
Integrated application:	Field evaluation
Number of field sets:	2 field tripples (6 fields)
Simultaneous processing cases:	1 (3 fields)

Ilustración 4.3. Funcionamiento.

4.1.4. Mecánica / electrónica

El sensor deberá ser alimentado mediante una fuente de tensión externa a 24 V DC y se alimentará a 0,6 A. Sabremos que está perfectamente preparado para la toma de datos cuando únicamente permanezca una luz verde fija en el mismo.

Mechanics/electronics	
Electrical connection:	2 Interface plug with 9 pin D-Sub socket (solder connection)
Operating voltage:	≤ 24 V DC± 15 %
Power consumption:	30 W
Housing:	Die-cast aluminum
Housing color:	light blue (RAL 5012)
Enclosure rating:	IP 65
Protection class:	II (VDE 0106/IEC 1010-1) [1]
Weight:	4.5 kg
Dimensions:	156 mm x 155 mm x 210 mm

[1] Insulated

Ilustración 4.4. Mecánica y electrónica del escáner.

4.1.5. Dimensiones

Mostramos en el siguiente esquema cuales son las dimensiones del escáner que estamos empleando:

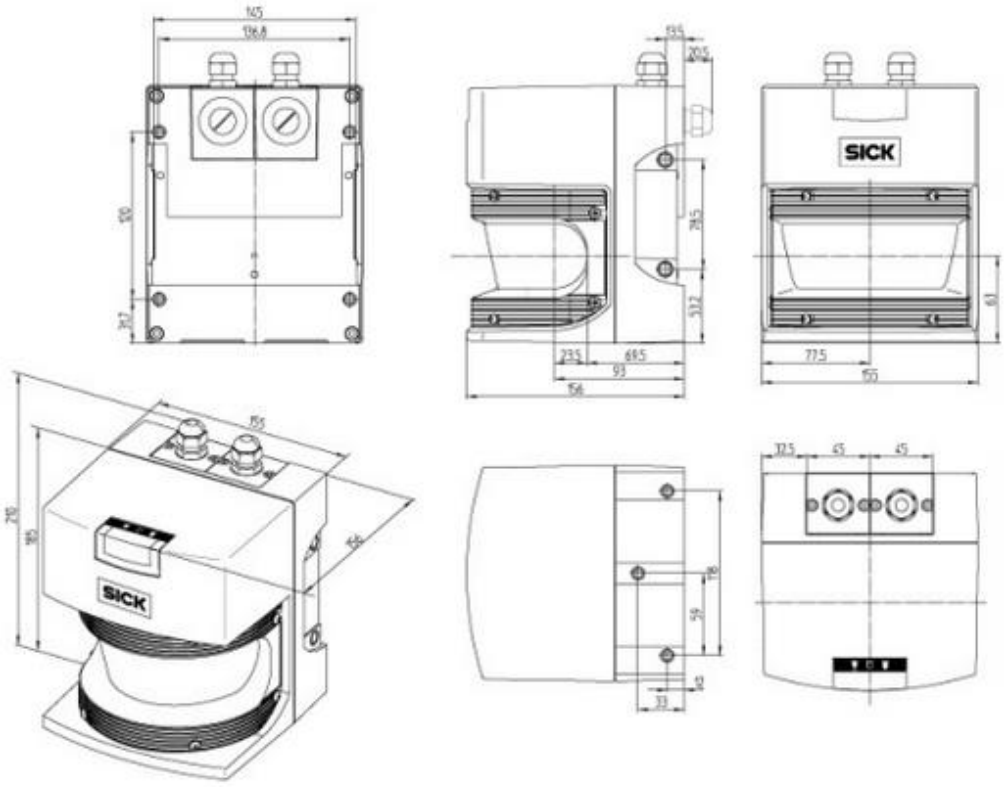


Ilustración 4.5. Dimensiones y topología del escáner.

4.1.6. Diagrama de rango de operación

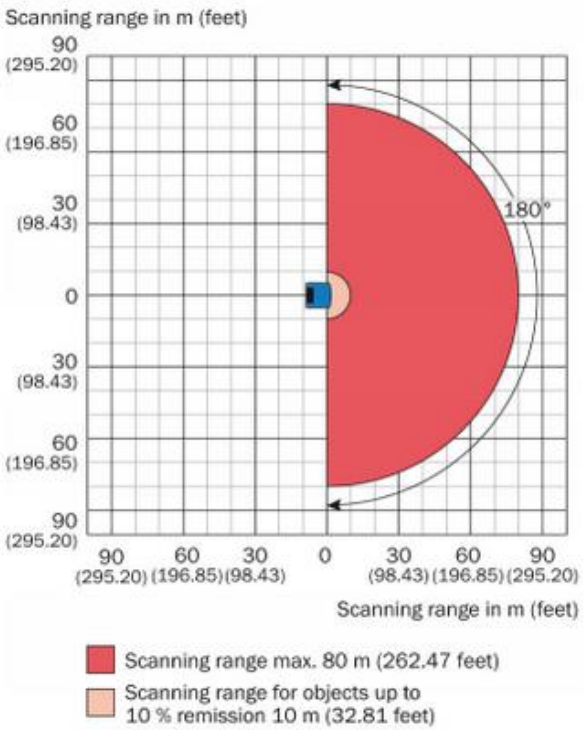


Ilustración 4.6. Rango de operación Sick LMS 200.

4.1.7. Adaptador de puerto serie a USB

Empleamos el siguiente adaptador de puerto serie a USB. En función de en qué puerto sea conectado, será necesario cambiar el nombre del puerto dentro de nuestro código. No hacerlo implicará la imposibilidad de obtener los datos del escáner.

Ilustración 4.7. Adaptador de puerto Serie a USB

Mostramos como ha sido conectado nuestro escáner al PC, si volvemos a conectarlo en el puerto USB mostrado no habrá que modificar el código en ningún punto.

Ilustración 4.8. Adaptador en el puerto COM 5

4.2. ROBOTNIK MODULAR ARM

Para nuestro desarrollo emplearemos el brazo robótico modular de Robotnik cuyas principales características describiremos a continuación.

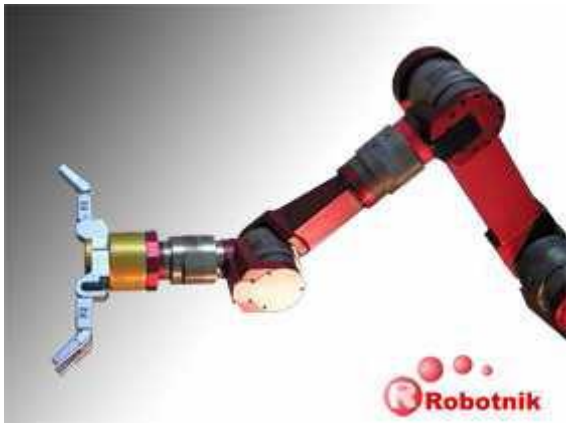


Ilustración 4.9. Brazo robótico modular.

4.2.1. Descripción

El Brazo Modular de Robotnik Automation es un brazo sobre la base de los módulos PowerCube Schunk. Estos módulos incluyen engranajes reductores de la velocidad proporcionada por el motor, la etapa de potencia y el controlador, por lo que el brazo resultante no necesita un armario de control externo. En su lugar, la conexión externa del brazo se reduce a sólo 24 VDC y comunicación a través del bus CAN.

4.2.2. Componentes

El brazo empleado dispone de piezas o enlaces independientes que, una vez ensamblados, formarán el conjunto deseado. Mostramos cada una de esas piezas.

- Enlace 1:

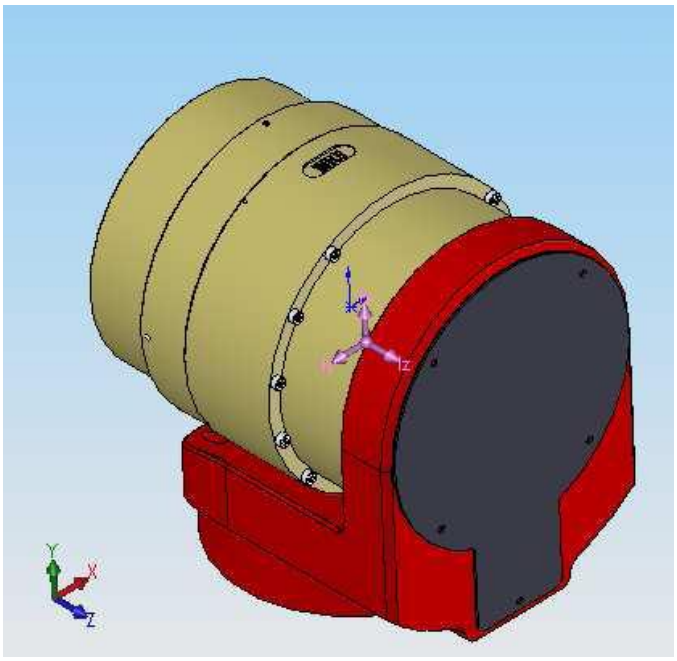


Ilustración 4.10. Enlace 1 del robot.

- Enlace 2:

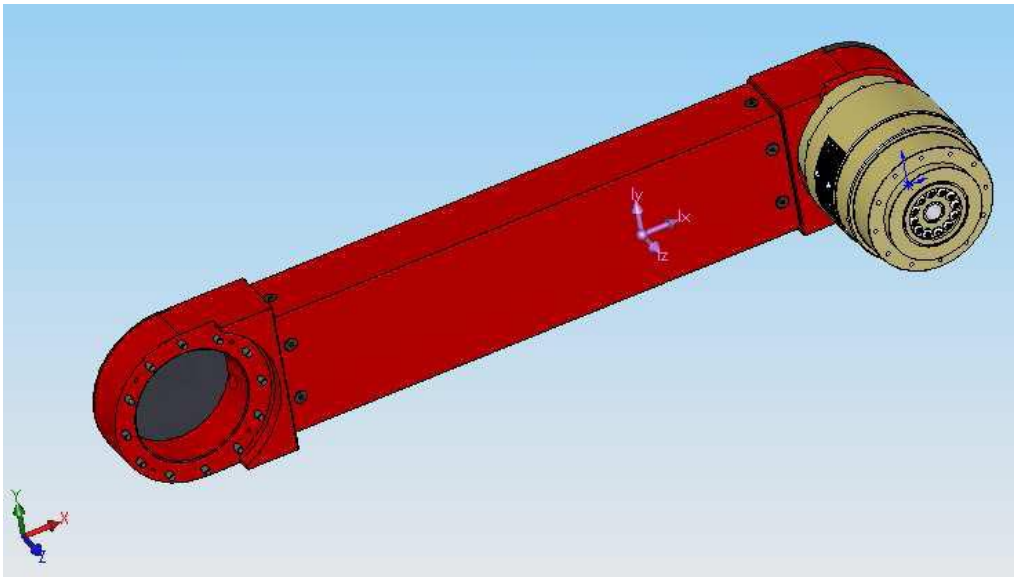


Ilustración 4.11. Enlace 2 del robot.

- Enlace 3:

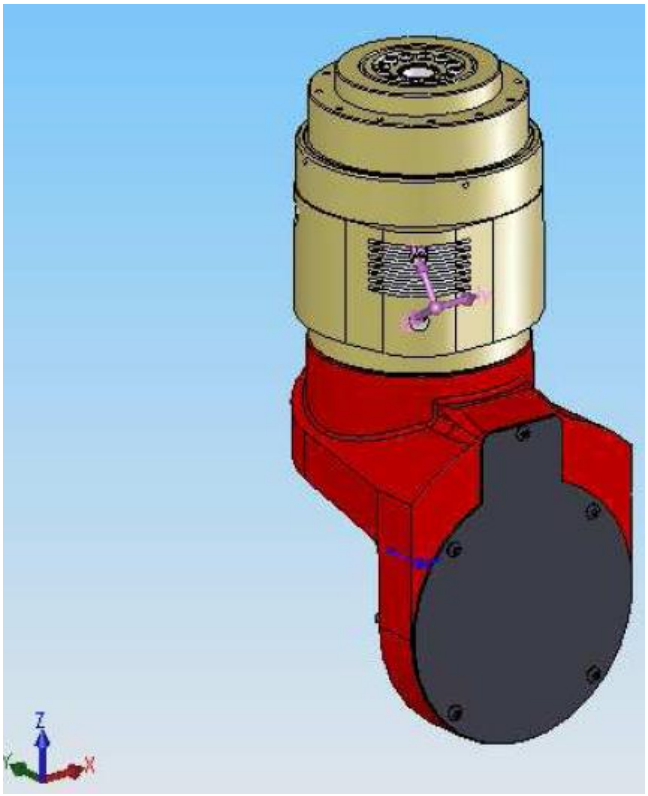


Ilustración 4.12. Enlace 3 del robot.

- Enlace 4:

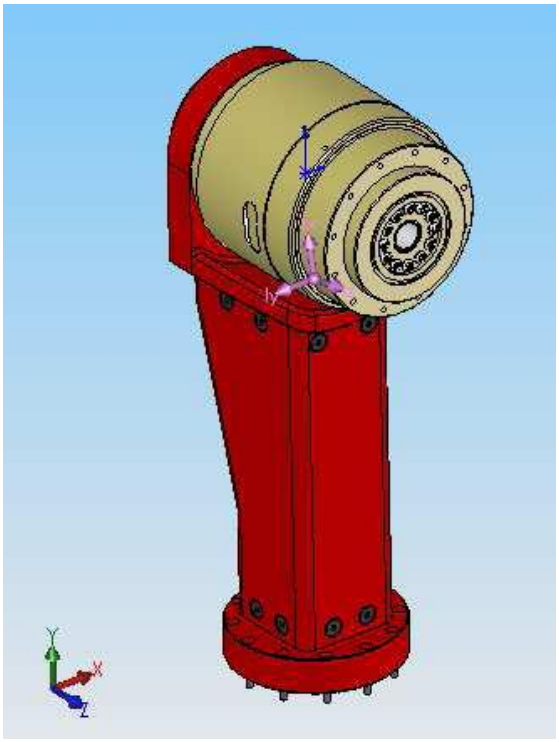


Ilustración 4.13. Enlace 4 del robot.

- Enlace 5:

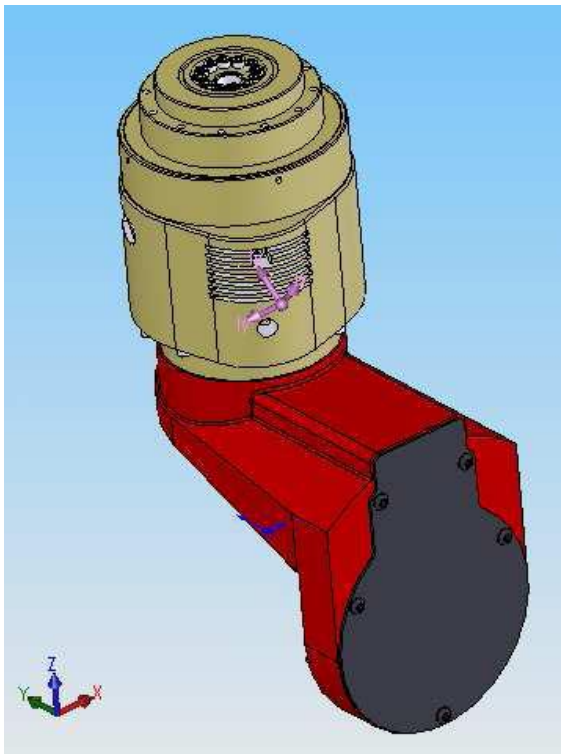


Ilustración 4.14. Enlace 5 del robot.

Además, es posible colocar una pieza que sería considerada enlace 6 en el extremo que será la encargada de interactuar con el entorno, una pinza, una mano robótica, etc.

Mostramos a continuación una tabla resumen con los centros de masa o COM (center of mass) y parámetros de inercia. Los ejes de los motores 1, 4 y 6 están alineados, mientras que los ejes 2,3 y 5 son paralelos.

Element	COM	Inertia Parameters
Link 1	Cx = -0,00057 [m] Cy = -0,01133 [m] Cz = 0,00878 [m]	Ixx = 0 Iyy = 0,000030941 [kg*m^2] Izz = 0 Ixy = 0 Ixz = 0 Iyz = 0
Link 2	Cx = 0,18128 [m] Cy = -0,00123 [m] Cz = -0,07017 [m]	Ixx = 0,00609233218 [kg*m^2] Iyy = 0,1845680479 [kg*m^2] Izz = 0,18838664807 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Link 3	Cx = -0,00087 [m] Cy = -0,01087 [m] Cz = 0,10241 [m]	Ixx = 0,00069941799 [kg*m^2] Iyy = 0,00071684059 [kg*m^2] Izz = 0,00053893461 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Link 4	Cx = -0,00041 [m] Cy = -0,04893 [m] Cz = 0,00485 [m]	Ixx = 0,00653606965 [kg*m^2] Iyy = 0,00097427479 [kg*m^2] Izz = 0,00637720193 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Link 5	Cx = -0,00011 [m] Cy = -0,01824 [m] Cz = 0,09177 [m]	Ixx = 0,00083022261 [kg*m^2] Iyy = 0,00064595927 [kg*m^2] Izz = 0,00071374229 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0

Ilustración 4.15. Tabla resumen COM y parámetros de inercia.

4.2.3. Seguridad

El Robotnik Modular Arm es un poderoso robot capaz de moverse a velocidades de alrededor de 1 m/s. Debe concienciarse a toda persona de los principios de seguridad que deben seguirse para trabajar con él:

- Es imprescindible mantener el pulsador de parada de emergencia a mano para poder ser empleado en caso de movimiento inesperado del brazo. Es necesario comprobar regularmente que dicho pulsador funciona adecuadamente.

Ilustración 4.16. Seta de emergencia del robot.

- No rociar agua o aceite en el robot, caja de operación, o el poder espinal. El contacto con agua o aceite puede causar descargas eléctricas o mal funcionamiento de la unidad.
- Confirme que el robot de alimentación está correctamente conectado a tierra antes de usar. Una conexión a tierra insuficiente puede causar descargas eléctricas, fuego, mal funcionamiento o averías.
- No intentar desamblar o modificar el robot sin la presencia de personal autorizado.
- No utilice la unidad en lugares expuestos a gases inflamables o corrosivos. Gas filtrado acumulado alrededor de la unidad puede provocar un incendio o explosión.

- Instalar el robot en un lugar que pueda soportar su peso y condiciones mientras se ejecuta. Una superficie inestable puede causar que la unidad se caiga, vuelque o averías, lo que podría provocar lesiones al operador.
- Conecte el cable de alimentación firmemente a la toma de pared. La inserción incompleta en la toma de corriente hace que la conexión se caliente y puede provocar un incendio.
- Compruebe que el enchufe no está cubierto de polvo.
- Asegúrese de apagar la fuente de alimentación antes de conectar el cable de alimentación espinal con el robot.
- Apague la unidad antes de insertar y extraer cables

4.3. OBJETOS EMPLEADOS

Para el desarrollo del proyecto emplearemos los siguientes 3 objetos.

- Prisma de base hexagonal.
- Cilindro cortado por un plano inclinado.
- Pirámide de base hexagonal.

Todos ellos tienen una base de radio (o apotema) de 5 cm. Esta información será imprescindible a la hora de diseñar el código del que hablaremos en el próximo capítulo.

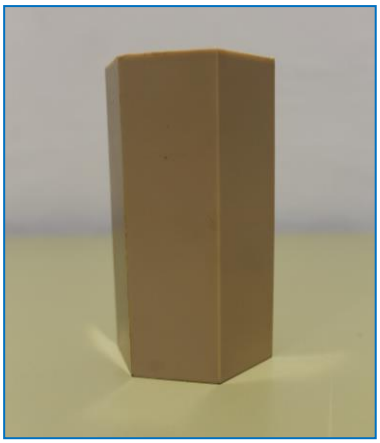


Ilustración 4.17. Alzado Objeto 1

Ilustración 4.18. Planta Objeto 1

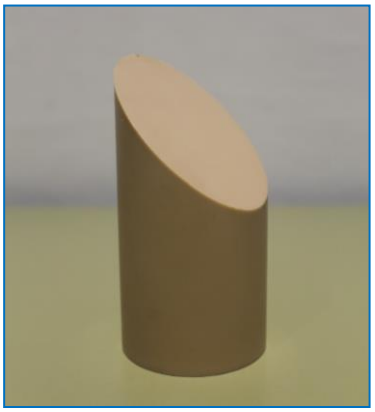


Ilustración 4.19. Alzado Objeto 2.

Ilustración 4.20. Planta Objeto 2.

Ilustración 4.21. Alzado Objeto 3.

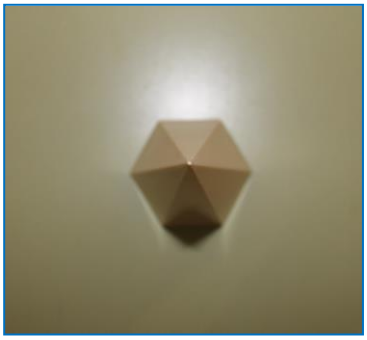


Ilustración 4.22. Planta Objeto 3

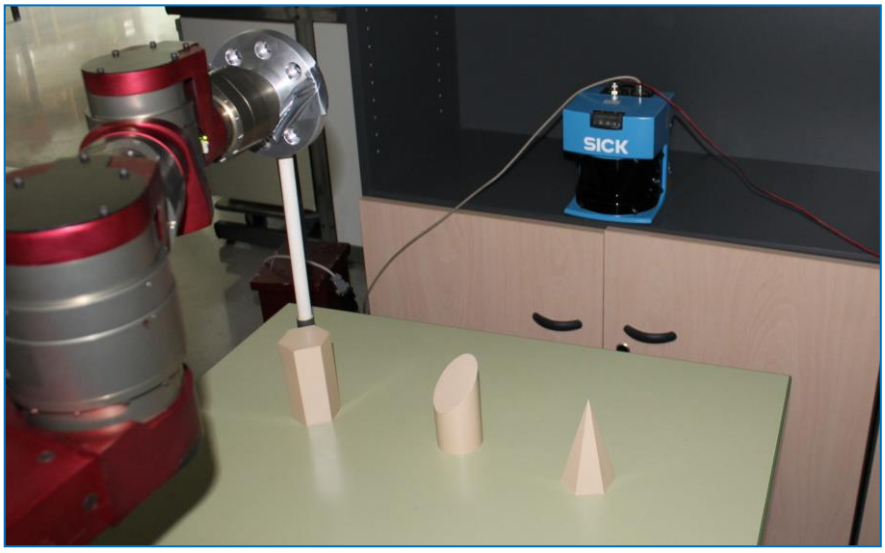


Ilustración 4.23. Vista general del escenario de trabajo.

CAPÍTULO 5:

Desarrollo del Proyecto

5. DESARROLLO DEL PROYECTO

A continuación vamos a explicar paso por paso como hemos desarrollado el actual Proyecto, tratando de ser lo más claro posible a fin de poder ser lo más útil posible de cara a futuros proyectos o trabajos de investigación.

5.1. LIBRERÍAS EMPLEADAS

Como en todo proyecto será necesario emplear una serie de librerías, algunas ya pertenecientes al propio lenguaje de programación e incluidos en *Microsoft Visual Studio 2010 Ultimate*, otras pertenecientes al propio software del brazo robótico y otras elaboradas por nosotros mismos. Es importante destacar en su definición, especialmente si no se está acostumbrado a trabajar en C++. Las librerías internas a Visual Studio y que no es necesaria añadir al proyecto se definen entre los signos < y >. Las incluidas por el programador en archivos de cabecera locales (.h o header) deben ser incluidas entre "comillas". Aquí las tenemos:

```
5  #include "stdafx.h"
6  #include "DemoRobotnik2.h"
7  #include "DemoRobotnik2Dlg.h"
8  #include "afxdialogex.h"
9  #include <math.h>
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdlib.h>
13 #include <conio.h>
```

Ilustración 5.1. Librerías empleadas.

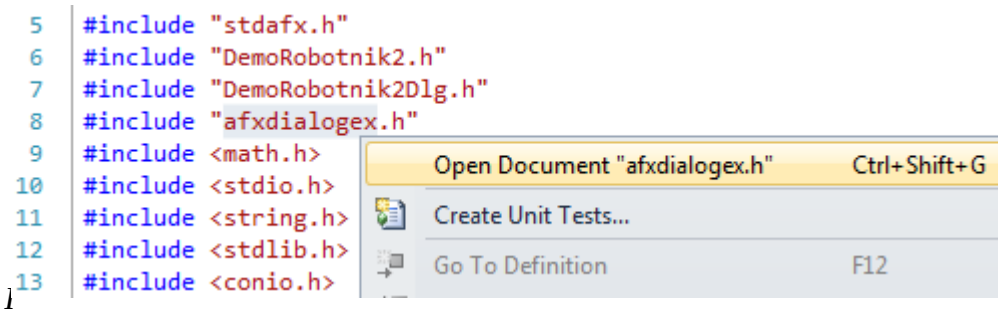
- "stdafx.h": archivos de inclusión para archivos de inclusión estándar del sistema, o archivos de inclusión específicos del proyecto utilizados frecuentemente, pero cambiados rara vez. Aquí estarán definidas a su vez otras librerías, la mayoría internas y ya incluidas en el programa necesarias para su correcto funcionamiento y también otras dos que merecen un tratamiento especial, específicamente de nuestro proyecto y que trataremos independientemente:

  #include "m5apiw32.h"

  #include "Prueba.h"

- "DemoRobotnik2.h" y "DemoRobotnik2Dlg.h": archivos de encabezado del programa y cuadros de diálogo.

- "afxdialogex.h": parte de la librería de Microsoft Foundation Classes C++ (MFC) y que sirve de apoyo a la misma. Al ser parte de MFC no se define con los signos < >. Es posible obtener más información sobre la misma pulsando el botón derecho sobre la misma y abrir el documento:



lustración 5.2. Análisis de librerías.

- El resto son librerías internas de C++ y no es posible verificar el contenido, simplemente añadirlas y emplear las funciones que aportan. Por ejemlo, la librería stdio.h permite acciones como mostrar en pantalla (printf) o solicitar datos al usuario (scanf); la librería math.h permite realizar operaciones matemáticas de mayor nivel y la librería string.h permite emplear cadenas de caracteres.

5.2. DEFINICIÓN DE VARIABLES

Es necesario definir todas las variables que vamos a emplear. Debido a las innumerables modificaciones realizadas en el código han tenido que ser reescritos en múltiples ocasiones. Más adelante explicaremos el fin de cada una de ellas:

```
234  void HiloPrincipal (LPVOID lpvParamHilo)
235  {
236      int i;
237      float pos1, pos2, pos3, pos4, pos5, pos6;
238      float Rpos = 3.141592 / 180.0; // Factor de conversión a radianes
239      float vel  = 5 * Rpos; // Rad/s
240      float acel = 1 * Rpos; // Rad/s2
241      float Pos0[6] = {85*Rpos, 20*Rpos, -5*Rpos, 1*Rpos, 0, 0};
242      float Distancia1,Distancia2,Distancia3,Angulo1,Angulo2,Angulo3; // Definimos variables de entrada del usuario
243      float x1,x2,x3,y1,y2,y3,xr1,xr2,xr3,yr1,yr2,yr3,xp1,xp2,xp3,xp4,xp5,xp6,yp1,yp2,yp3,yp4,yp5,yp6,Alfa1,Alfa2,Alfa3,Alfa4,Alfa5,Alfa6;
244      float Alfa7,Alfa8,Alfa9, Alfa10,Alfa11,Alfa12,Alfa13,Alfa14, Beta1,Beta2,Beta3,Beta4,Beta5,Beta6,Beta7,Beta8,Beta9,Beta10,Beta11,Beta12;
245      float Beta13,Beta14, Gamma1, Gamma2, Gamma3, Gamma4, Gamma5, Gamma6, Gamma7,Gamma8,Gamma9,Gamma10,Gamma11,Gamma12,Gamma13,Gamma14;
246      float Hipotenusa1, Hipotenusa2, Hipotenusa3, Hipotenusa4, Hipotenusa5, Hipotenusa6, Hipotenusa7, Hipotenusa8, Hipotenusa9,Hipotenusa10;
247      float Hipotenusa11, Hipotenusa12, Hipotenusa13, Hipotenusa14,AngBrazo1,AngBrazo2,AngBrazo3,AngBrazo4,AngBrazo5,AngBrazo6, AngBrazo7;
248      float AngBrazo8, AngBrazo9, AngBrazo10, AngBrazo11, AngBrazo12,AngBrazo13, AngBrazo14,xp7,xp8,xp9,xp10,xp11,xp12,xp13,xp14,yp7,yp8,yp9;
249      float yp10,yp11,yp12,yp13,yp14,AngAntBrazo1,AngAntBrazo2,AngAntBrazo3,AngAntBrazo4,AngAntBrazo5,AngAntBrazo6,AngAntBrazo7,AngAntBrazo8;
250      float AngAntBrazo9,AngAntBrazo10,AngAntBrazo11,AngAntBrazo12,AngAntBrazo13,AngAntBrazo14,LongBrazo,LongAntBrazo,tpaso;
251      unsigned long state3 = 0, state5 = 0;
252      int m_dev=0;
253      float x4b,x8b,y4b,y8b,Alfa4b,Alfa8b,Beta4b,Beta8b,Gamma4b,Gamma8b,AngBrazo4b,AngBrazo8b,AngAntBrazo4b,AngAntBrazo8b,xp4b,xp8b,yp4b,yp8b;
254      float Hipotenusa4b,Hipotenusa8b;
```

Ilustración 5.3. Definición de variables.

5.3. OBTENCIÓN DE COORDENADAS DE OBJETOS

5.3.1. Solicitud al usuario

Inicialmente se planteó dar la opción al programa de solicitar al usuario la posición de los objetos para poder realizar los movimientos evitando los obstáculos. En la función basado en diálogos realizada ("dialog based") no es posible emplear los comandos printf y scanf para realizar esto (sólo pueden ejecutarse en documentos que muestren los resultados secuencialmente por consola o "single document") y habría que emplear cuadros de texto (TextBox) para realizar esta petición. No obstante, mostramos la parte del código que ha quedado comentada y en desuso, además de por todo lo anterior, por introducir un proceso lo más automatizado posible.

```
259 /*
260 printf("\nIntroduzca las coordenadas polares del punto medio del primer objeto respecto al sensor");
261 printf("\nDistancia1: ");
262 scanf_s("%i",&Distancia1);
263 printf("\nAngulo1: ");
264 scanf_s("%i",&Angulo1);
265
266 printf("\nIntroduzca las coordenadas polares del punto medio del segundo objeto respecto al sensor")
267 printf("\nDistancia2: ");
268 scanf_s("%i",&Distancia2);
269 printf("\nAngulo2: ");
270 scanf_s("%i",&Angulo2);
271
272 printf("\nIntroduzca las coordenadas polares del punto medio del tercer objeto respecto al sensor");
273 printf("\nDistancia3: ");
274 scanf_s("%i",&Distancia3);
275 printf("\nAngulo3: ");
276 scanf_s("%i",&Angulo3);
277 */
```

Ilustración 5.4. Solicitud de datos al usuario.

5.3.2. Recepción de datos del sensor

Como se ha dicho anteriormente, el trabajo y recogida de datos del sensor se ha desarrollado paralelamente en el Proyecto Fin de Carrera de Lidia Sánchez Martínez; no obstante, y al haberse desarrollado de forma paralela y ser imprescindible para el presente proyecto vamos a exponer los apartados clave del mismo.

Para interactuar con el sensor se ha desarrollado un programa tomando como base el proporcionado por SICK. Originalmente este programa estaba orientado a tomar las lecturas y generar un fichero Excel con ellas. Puesto que nuestro objeto era emplear dichos datos para automatizar los movimientos del robot se han prescindido de estas funciones. No las hemos borrado por si pudieran ser de interés de cara a desarrollos futuros. Mostramos un ejemplo:

```
1109  /*
1110  void write_excel_file(const char * filename,LMSAPI_LASER_DATA * data)
1111  {
1112      FILE * fp = fopen(filename,"w");
1113
1114      //escribir encabezado  del archivo de excel
1115      fprintf(fp,"<?xml version=\"1.0\"?>\n");
1116      fprintf(fp,"<?mso-application progid=\"Excel.Sheet\"?>\n");
1117      fprintf(fp,"<?mso-application progid=\"Excel.Sheet\"?>\n");
1118      fprintf(fp,"<Workbook xmlns=\"urn:schemas-microsoft-com:office:spreadsheet\" \n");
1119      fprintf(fp,"xmlns:o=\"urn:schemas-microsoft-com:office:office\" \n");
1120      fprintf(fp,"xmlns:x=\"urn:schemas-microsoft-com:office:excel\" \n");
1121      fprintf(fp,"xmlns:ss=\"urn:schemas-microsoft-com:office:spreadsheet\" \n");
1122      fprintf(fp,"xmlns:html=\"http://www.w3.org/TR/REC-html40\">\n");
1123      fprintf(fp,"<DocumentProperties xmlns=\"urn:schemas-microsoft-com:office:office\">\n");
1124      fprintf(fp,"</DocumentProperties> \n");
1125      fprintf(fp,"<Worksheet ss:Name=\"Mediciones_LMS200\">\n");
1126      //Encabezado de tabla
1127      fprintf(fp,"<Table>\n");
```

Ilustración 5.5. Exportar datos a Excel

5.3.2.1. *Abrir el puerto de comunicación con el sensor*

Lo primero será abrir el puerto serie para comunicarnos con el sensor, en nuestro caso será el COM 5 mediante la función lmsapi_serial_open_port. Mostramos a continuación una parte de la misma.

```
1222 int lmsapi_serial_open_port(  int nPort, int nBaud,char cParity, int nBits, int nStopBits) //<-- COPIADO A .H
1223 {
1224     _lmsapi_init_serial (); // <-- COPIADO
1225
1226     if (m_bOpened) return( 1 );
1227
1228     wchar_t szPort[15];
1229     wchar_t szComParams[50];
1230
1231     DCB dcb;
1232
1233     wsprintf ( szPort, TEXT("COM%d"), nPort );
1234     m_hIDComDev = CreateFile ( TEXT("COM5"), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, NULL );
1235     if( m_hIDComDev == NULL ) return( 0 );
1236
1237     memset( &m_OverlappedRead, 0, sizeof( OVERLAPPED ) );
1238     memset( &m_OverlappedWrite, 0, sizeof( OVERLAPPED ) );
1239
1240     COMMTIMEOUTS CommTimeOuts;
```

Ilustración 5.6. Apertura del puerto serie de comunicación con escáner.

Para ello comprobamos que lo primero que hace es llamar a la función _lmsapi_init_serial. Esta nos dará información sobre el estado del puerto, y en caso de estar cerrado se procederá a abrirlo.

5.3.2.2. Intercambio de datos con el sensor.

De la misma manera que es necesario preguntar para obtener la respuesta a de alguien sobre cualquier situación, es necesario solicitar al sensor los datos que queremos para ser capaces de interpretarlos y obtener la respuesta que buscamos. Para ello, es necesario enviar un código en hexadecimal a través de nuestro programa e interpretar la respuesta obtenida. Los caracteres específicos para establecer esta comunicación se establecen en los manuales proporcionados en la bibliografía y que se adjuntan en el cd de este proyecto. Se envía un datagrama, y cuando este ha sido enviado satisfactoriamente recibe el mensaje de respuesta del sensor en un buffer de recepción.

Description	STX	Address	Length		Command	Data		Checksum	
						Data	LMS status		
Byte position	1	2	3	4	5	6 to 729	730	731	732
Hex. value	02	80	D6	02	B0	724 bytes	10	15	D4

Table 7-29: Complete telegram for response B0h (*Table 7-24, page 49*)

Ilustración 5.7. Datagrama Sick LMS 200.

Pasamos a describir los parámetros usados en este punto, también comentados en el código para servir de ayuda a la hora de enfrentarse a los varios miles de líneas que han resultado del código.

Connection: Debe ser una conexión abierta a un puerto serial.

Packet: Datagrama a enviar

Len: largo del datagrama a enviar.

Outpacket: Datagrama de respuesta

Outlen: tamaño máximo del datagrama de respuesta.

expected_response: Código de respuesta esperado en la recepción del mensaje. Si se obtiene un código de respuesta diferente, se interpretará como una respuesta incorrecta.

\return: un entero positivo si el comando fue procesado con éxito, si no entregara un número negativo que indica un error.

La función _lmsapi_read_measurement_data que mostramos a continuación realiza la lectura de medidas del sensor. Para ello será necesario definir el vector raw_data de tipo unit8_t que albergará las mediciones en bruto obtenidas del sensor. Asimismo definimos el vector packet [20] que tendremos que analizar con calma, pues contendrá no sólo las peticiones al sensor láser, sino también la confirmación de lectura, mediciones y otras variables basura que en principio no serán de interés. Para ello, deberemos analizar detenidamente los manuales adjuntos a la memoria.

La introducción del comando packet[0] = 0x30; será la orden emitida al sensor para que nos proporcione las mediciones.

```
1646 int _lmsapi_read_measurement_data(LMSAPI_CONNECTION * connection,uint16_t *data, size_t datalen)
1647 {
1648     uint8_t raw_data[2048];
1649     //Solicitar datos con el comando 30
1650     uint8_t packet[20];
1651     packet[0] = 0x30; /* Solicitar datos medidos */
1652     // Obtener el muestreo completo, con angulo de apertura 90 o 180
1653     packet[1] = 0x1;
1654
1655     int len =  lmsapi_send_command(connection,packet,2,raw_data, sizeof(raw_data),0xB0);
1656
1657     if (len <1)
1658     {
1659         RETURN_ERROR(-1, "Telegrama vacio");
1660     }
1661
1662     int count = 0;
1663     // Procesar los datos
1664     if (raw_data[0] == 0xB0)//Leyendo medición del rango completo (100 o 180)
1665     {
```

Ilustración 5.8. Solicitud de datos al escáner.

```
1666            // Determinar el numero de valores medidos
1667            //
1668            //int units = raw_data[2] >> 6;
1669            count = (int) raw_data[2] | ((int) (raw_data[3] & 0x3F) << 8);
1670            assert((size_t) count <= datalen);
1671            connection->scan_min_segment = 0;
1672            connection->scan_max_segment = count-1;
1673            // Saltarse la info de status
1674            // to remove packet header.
1675            //
1676            int i;
1677            int j;
1678
1679            for (i = 2; i < count/2-1; i++)
1680            {
1681                int src = 2 * i + 3;
1682                data[i] = raw_data[src + 0] | (raw_data[src + 1] << 8);
1683            }
1684        }
1685        else
1686        {
1687            RETURN_ERROR(-1, "Paquete desconocido");
1688        }
1689        return count;
1690    }
```

Ilustración 5.9. Recuperación de datos del escáner.

Como vemos anteriormente, en la línea 1655 se define la variable len, que como define la longitud del datagrama a enviar, y tomará el valor de retorno (siempre un entero) de la función lmsapi_send_command.

Vemos como el valor “retlen” local a la función se transforma posteriormente en la variable “len” definida.

```
1598 int lmsapi_send_command(LMSAPI_CONNECTION * connection, uint8_t*packet, int len, uint8_t*outpacket, int maxoutlen, int expected_response)
1599 {
1600     int retlen = 0;
1601     int tries;
1602     // limpiar el buffer
1603     lmsapi_serial_read_data_waiting();
1604     for (tries = 0; tries < LMSAPI_MAX_TRIES; tries++)
1605     {
1606         if (lmsapi_write_to_laser (connection,packet, len) < 1)
1607         {
1608             RETURN_ERROR (-1, "Error al solicitar medición");
1609         }
1610         lmsapi_sleep (LMSAPI_READING_TIME_INTERVAL);
1611         // Obtener la respuesta del laser. (Acknowledgment)
1612         retlen = lmsapi_read_from_laser (connection, outpacket, maxoutlen);
1613
1614         if (retlen == 0)
1615         {
1616             printf ("\n Intento fallido de lectura");
1617         }
1618         else if (retlen < 0)
1619         {
1620             RETURN_ERROR(retlen, "Error de lectura en el puerto");
1621         }
1622         else if (outpacket[0] == NACK)
```

Ilustración 5.10. Recuperar la longitud del datagrama a enviar.

```
1622            else if (outpacket[0] == NACK)
1623            {
1624                RETURN_ERROR(-1, "Peticición denegada");
1625            }
1626            else if (outpacket[0] != expected_response)
1627            {
1628                RETURN_ERROR(-1, "Respuesta equivocada del sensor.");
1629            }
1630            else
1631            {
1632                //operacion exitosa.
1633                break;
1634            }
1635        }
1636        if (tries>=LMSAPI_MAX_TRIES)
1637        {
1638            RETURN_ERROR(-2, "Tiempo excedido de espera. El sensor no responde.");
1639        }
1640        return retlen;
1641    }
```

Ilustración 5.11. Mensajes de respuesta equivocada del sensor.

Como vemos, el programa devuelve múltiples mensajes de error en caso de que se detecte alguna anomía en el funcionamiento del mismo. Todo está perfectamente comentado en el código que se encuentre en el cd del presente proyecto, de modo que no profundizaremos más en ese ámbito.

5.3.2.3. Función deteccionobjetos

Nuestro escáner realizará un barrido de 180º y obtendrá para cada uno de ellos (estamos empleando resolución de 1º) la distancia a la que se encuentra el objeto. Sin embargo, de esos 180 datos que obtendremos únicamente nos interesa quedarnos con 3 (la posición de los 3 objetos que vamos a emplear) y desechar el resto. Para ello se hace imprescindible establecer un filtro a partir de la siguiente función.

Para comprender la función es necesario tener en cuenta que i corresponde a la posición angular en grados; data[i] a la distancia en mm del objeto en el ángulo i; j corresponde al número de objetos y k al número de datos recogidos para un mismo objeto (dicho de otra manera, al número de grados que ocupa el objeto).

```
1692 int deteccionobjetos (float*data, int count, float*SDIST, float*SANG, int objetosmaximo)
1693 {
1694     int i, j, k;
1695     for (i=0; i<181; i++)
1696     {
1697         if (data[i] > 1000 || i < 70 || i > 135)
1698         {
1699         data[i] = 8000;
1700         }
1701     }
1702     j = 0;
1703     for ( i=0; i < 181;  )
1704     {
1705         SDIST[j] = 0;
1706         SANG [j] = 0;
1707         while (data[i] == 8000)
1708         {
1709             i++;
1710         }
1711         k = 0;
1712         while (data[i] < 1000)
1713         {
1714             SDIST[j] = SDIST[j] + data[i];
1715             SANG [j] = SANG [j] + i;
1716             i++;
1717             k++;
1718         }
1719         SDIST [j] = SDIST [j] / k ;
1720         SANG [j] = SANG [j] / k;
1721         j++;
1722     }
1723 return j;
1724 }
```

Ilustración 5.12. Función deteccionobjetos.

- En el primer bucle for asignamos una distancia de 8000 mm a todos los objetos que estén a más de 1 m (consiguiendo filtrar tanto la base del robot como el trípode y la cámara de video empleada) y los que estén en una posición angular menor de 70º y mayor de 135º (filtrando también también la estantería y el cable que va al ordenador, cuyas datos tampoco nos interesan a pesar de ser menores de 1m.
- En el segundo bucle for vamos haciendo un barrido mientras los objetos estén a 8000 mm (datos basura) y nos detendremos cuando nos encontremos un dato válido (inferior a 1000 mm o podríamos haber definido como distinto a 8000 mm pues sería equivalente). En ese momento vamos acumulando todos los valores de distancias en el vector SDIST [j] y de ángulos en el vector SANG [j].
- Para el primer objeto (j=0):
    - SDIST [j] = SDIST [0] = SDIST [0] + data (i)
    - SANG [j] = SANG [0] = SANG [0] + i

- Añadimos una unidad al ángulo i para verificar si el objeto j = 0 también ocupa esta posición. También añadimos una unidad a la variable k, la cual deberá ser inicializada a 0 cada vez que volvamos a ver un dato a 8000 (sólo así podremos reiniciar el contador para ver cuantas posiciones angulares respecto al escáner).
- Finalmente, cuando dejemos de observar un data[i] < 1000, esto implicará que ya se han acabado las mediciones que correspondían a ese objeto.
- En ese momento, dividimos todo lo acumulado entre el número de mediciones consideradas k. Y obtendremos SDIST[0] y SANG [0] como la distancia y el ángulo a la que se encuentran el primer objeto. Entonces se añadirá una unidad a la variable j y se buscará el siguiente objeto sobre la mesa.

NOTA IMPORTANTE: los datos obtenidos no se corresponden exactamente con el centro geométrico de las figuras al ser una media entre los puntos que ve el sensor, y no tener en cuenta la profundidad de los objetos. Sería necesario, al menos en caso de emplear este mismo escáner, realizar un algoritmo que tuviera en cuenta las dimensiones de cada objeto (no podríamos establecer un código generalista como el que hemos obtenido nosotros), así como identificarlos. Ello se escapa de los objetivos de este proyecto. En cualquier caso, las posiciones obtenidas por este procedimiento serán más que suficientes para nuestro desarrollo.

5.3.3. Datos obtenidos del sensor como punto de partida.

Tras definir la variable j y el número máximo de objetos que tomaremos como 10 (aunque sólo consideraremos 3 objetos sobre la mesa) será necesario definir los vectores SDIT y SANG que guardarán las coordenadas polares de los objetos. También definiremos una estructura de medición que llamaremos “mediciones” en la línea 280. Se recomienda entrar en estas funciones del código para ver cómo se han creado.

A continuación se llama a la función deteccionobjetos que hemos comentado anteriormente y que está definida más avanzado el código que obtener las medidas deseadas. Es necesario dividir el rango entre 1000, pues las medidas del escáner

vendrán en mm, mientras que en el código de interacción con el robot que trataremos posteriormente estaremos empleando medidas en m.

```
278  int j, objetosmaximo=10;
279  float SDIST[10], SANG [10];
280  LMSAPI_LASER_DATA  * mediciones = lmsapi_laser_data_create();
281  pDlg->m_count=lmsapi_request_measurement(pDlg->miconexion, mediciones, 1.0f);
282
283  deteccionobjetos (mediciones->distances, pDlg->m_count, SDIST, SANG, objetosmaximo);
284
285  Distancia1 = SDIST [0]/1000;
286  Angulo1 = SANG [0];
287  Distancia2 = SDIST [1]/1000;
288  Angulo2 = SANG [1];
289  Distancia3 = SDIST [2]/1000;
290  Angulo3 = SANG [2];
```

Ilustración 5.13. Definición de estructura de medición y posición objetos.

5.4. TRANSFORMACIÓN DE COORDENADAS

Ya hemos sido capaces de obtener las coordenadas polares de los objetos respecto al sensor. Ahora es momento de referirlas al origen de coordenadas del brazo robótico, al ser este el punto de partida para generar los movimientos de cada una de las articulaciones que nos permitirán generar las trayectorias necesarias para nuestro proyecto. Para ello, hemos seguido los siguientes pasos:

- Una vez asignados los valores de Distancia1, Angulo1, Distancia2, Angulo2, Distancia3 y Angulo3 como las coordenadas polares de los 3 objetos respecto al escáner, debemos transformamos las coordenadas polares a cartesianas respecto al mismo origen. Puesto que las funciones trigonométricas son referidas a radianes y nosotros las tenemos en grados, debemos multiplicar nuestros ángulos por *pi/180*. Olvidar esto implicará un funcionamiento anómalo.

```
293  //coord cartesianas respecto sensor primer objeto
294  x1 = Distancia1*cos(Angulo1*pi/180);
295  y1 = Distancia1*sin(Angulo1*pi/180);
296
297  //coord cartesianas respecto sensor segundo objeto
298  x2 = Distancia2*cos(Angulo2*pi/180);
299  y2 = Distancia2*sin(Angulo2*pi/180);
300
301  //coord cartesianas respecto sensor tercer objeto
302  x3 = Distancia3*cos(Angulo3*pi/180);
303  y3 = Distancia3*sin(Angulo3*pi/180);
```

Ilustración 5.14. Obtención de coordenadas cartesianas respecto sensor.

- Sin dejar de trabajar en coordenadas cartesianas, debemos referir dicho origen al origen del brazo robótico. Como podemos ver en la siguiente perspectiva desde el punto de vista del origen de coordenadas del brazo robótico, tanto robot como escáner láser se encuentran alineados. De modo que el eje X será el mismo pero con una rotación de 180º debido a que ambos tienen el eje X positivo hacía su derecha y están uno frente al otro. Por ejemplo, el prisma de base hexagonal de la figura está colocado a X = + 0,268 m respecto al escáner, lo que implicará decir que se encuentra a Xr = - 0,268 m respecto al brazo.

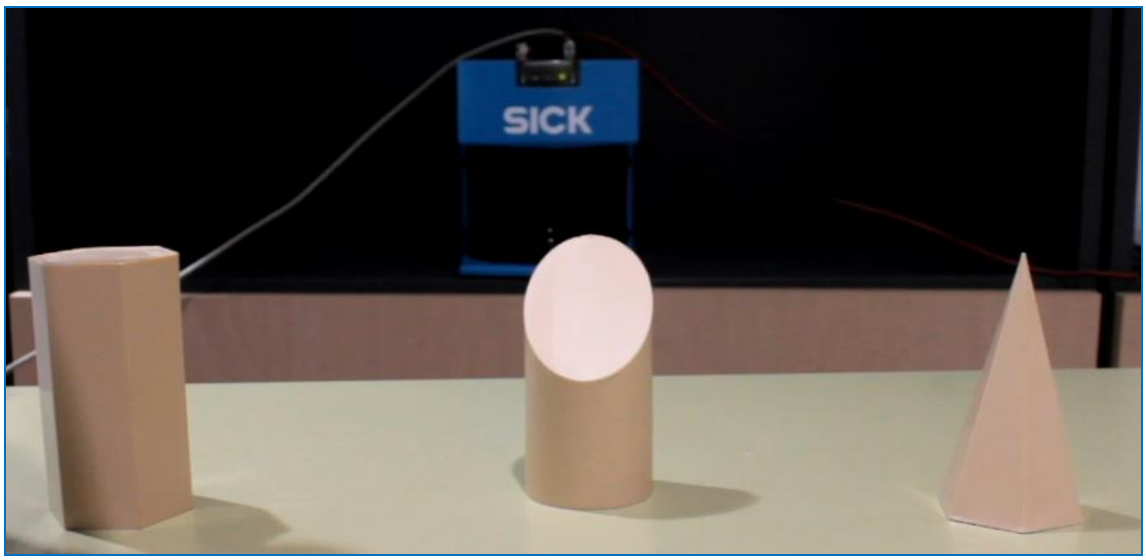


Ilustración 5.15 Escena sensor y objetos.

En cuanto a la coordenada Y, al estar colocados ambos dispositivos perpendiculares entre sí, no tendremos que emplear matrices de rotación para referir ambos orígenes de coordenadas. Dicho eje tendrá la misma orientación; sin embargo, al estar separados ambos dispositivos una distancia de 1,080 m, deberá restarse dicho valor al proporcionado por el escáner. Siguiendo con el ejemplo del prisma de base hexagonal de la figura anterior, si está situado en una posición Y = + 0,671 m respecto al escáser, ello implicará una posición respecto al robot de Yr = 1,080 – 0,671 = 0,409 m.

Con todo lo anterior, la transformación del sistema de referencia quedará en nuestro programa de la siguiente manera:

```
305 // TRANSFORMACION DEL SISTEMA DE REFERENCIA
306
307 //coord cartesiana respecto robot objeto 1
308 xr1 = x1*(-1);
309 yr1 = 1.08 - y1; // 1.08 m distancia sensor al robot
310
311 //coord cartesiana respecto robot objeto 2
312 xr2 = x2*(-1);
313 yr2 = 1.08 - y2; // 1.08 m distancia sensor al robot
314
315 //coord cartesiana respecto robot objeto 3
316 xr3 = x3*(-1);
317 yr3 = 1.08 - y3; // 1.08 m distancia sensor al robot
```

Ilustración 5.16. Transformación al sistema de referencia del robot.

5.5. SELECCIÓN DE LA TAREA A REALIZAR

Vamos a diseñar dos posibles trayectorias que deberá seguir nuestro brazo robótico. Para ello definimos la variable trayectoria, la cual podrá tener dos posibles valores:

a) Trayectoria en zig-zag: en este caso asignaremos a la variable el valor 1. El puntero que hemos atado con cinta americana al extremo del brazo robótico deberá llegar desde la posición inicial a la posición final de mesa evitando los obstáculos que se encuentre generando una trayectoria en forma de zigzag. Además, en caso de que algunos objetos estén tan cerca que sea físicamente imposible atravesarlos por su punto medio, se generará un mensaje en nuestro programa: "Objetos demasiado juntos. Buscando trayectoria alternativa." Se considerarán los objetos como si fueran un todo y seguirá la trayectoria en zig-zag.

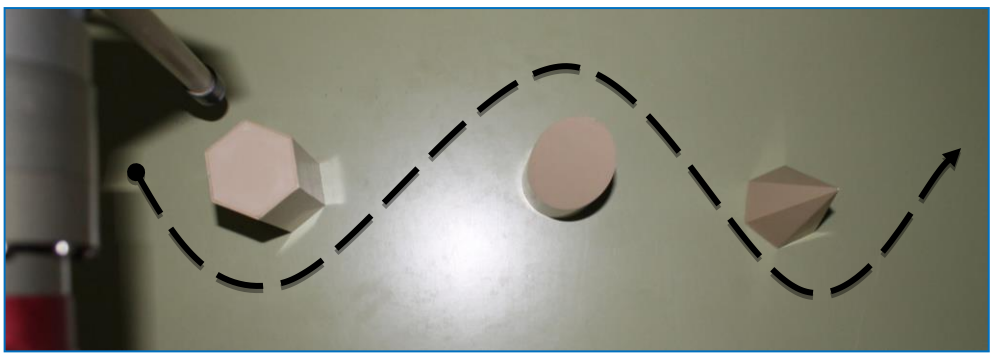


Ilustración 5.17. Trayectoria zig-zag en situación normal.

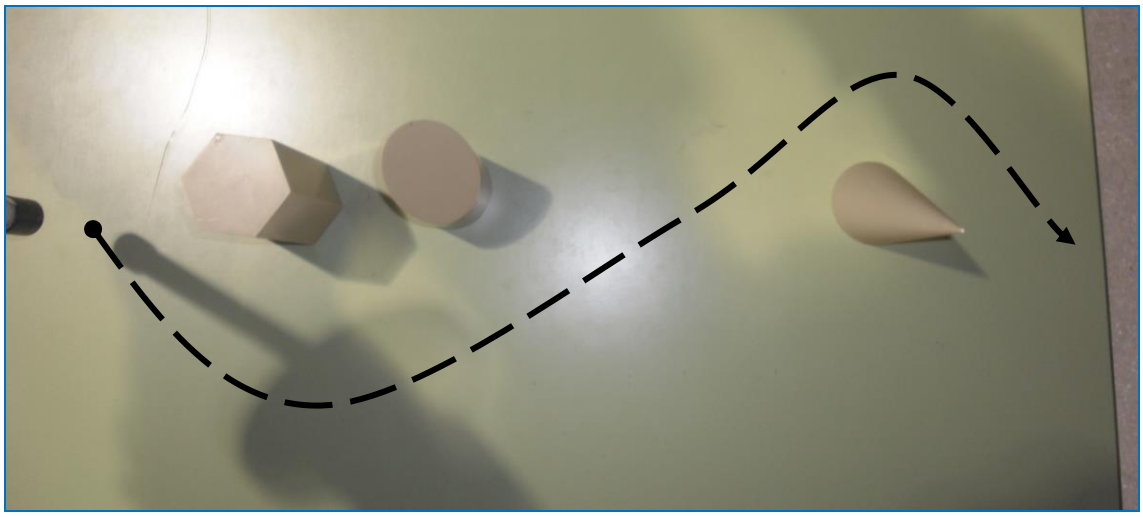


Ilustración 5.18. Trayectoria en zig-zag objetos 1 y 2 muy juntos.

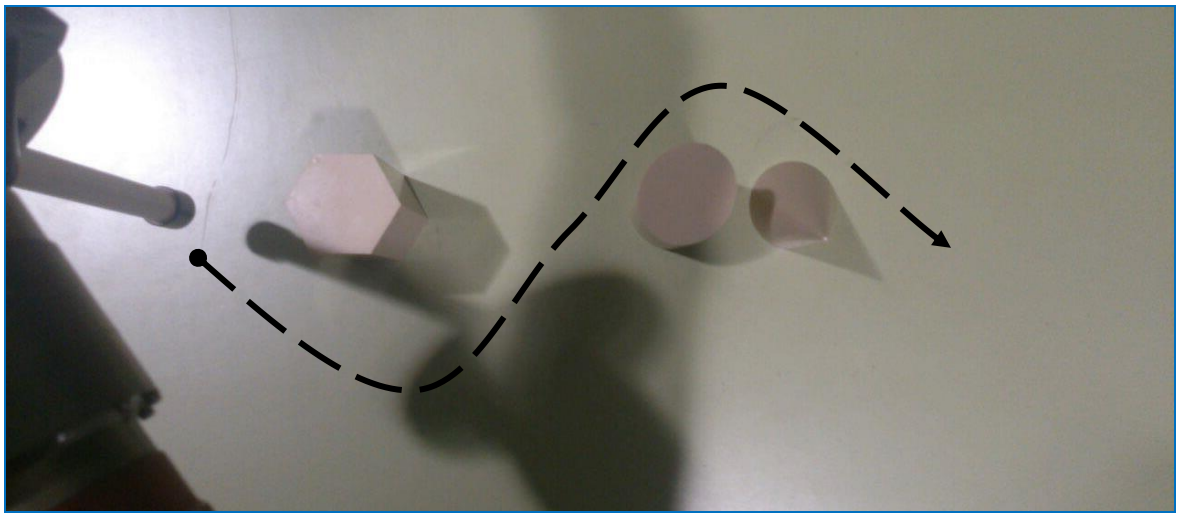


Ilustración 5.19. Trayectoria en zig-zag objetos 2 y 3 muy juntos.

b) Trayectoria circular: en este caso el puntero rodeará por completo cada uno de los objetos antes de pasar al siguiente objeto, hasta alcanzar el punto final. Para simplificar el código y el razonamiento no consideraremos aquí el caso de que los objetos estén demasiado juntos.

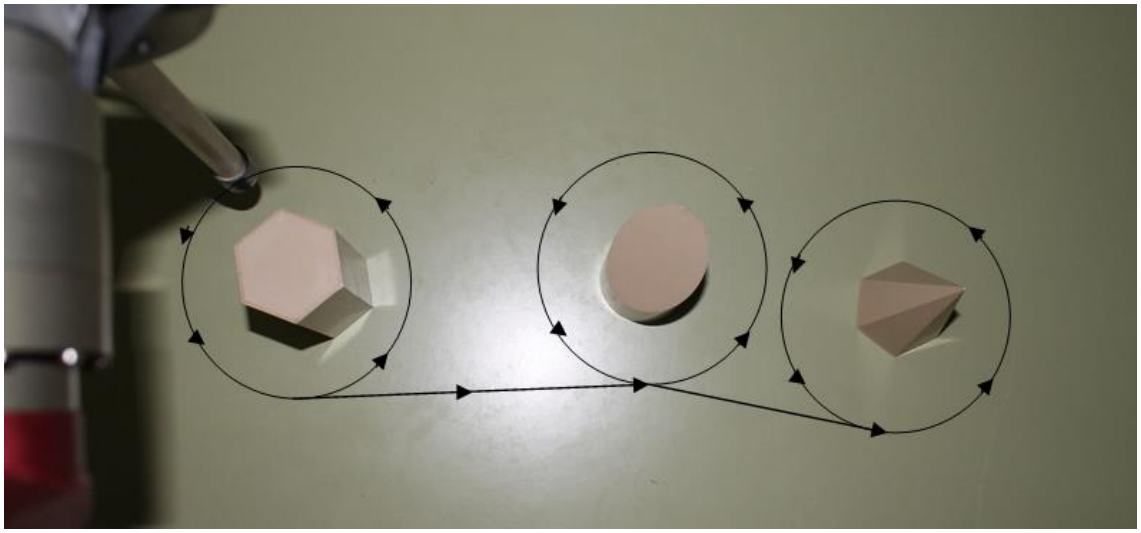


Ilustración 5.20. Trayectoria circular.

5.5.1. Trayectoria en zig-zag

Como hemos comentado anteriormente, en esta función el robot irá desde una posición inicial variable hasta una posición final dada, evitando todos los objetos que se encuentre.

5.5.1.1. *Habilitar la ejecución de esta variante del código*

Lo primero será indicar al programa que ejecute esta secuencia, para ello debemos habilitar la línea que nos permitirá entrar posteriormente en ella.

```
318   int trayectoria;
319   trayectoria = 1; // Trayectoria en zig-zag
320   //trayectoria = 2; // Trayectoria circular
```

Ilustración 5.21. Selección secuencia trayectoria en zig-zag

5.5.1.2. Planificación de los puntos de paso.

Ahora tendremos que establecer los puntos por los que queremos que pase el robot, para ello debemos tener en cuenta las limitaciones del mismo, así como la forma en que recibimos los datos del escáner.

Vamos a establecer que el robot esquive el primer objeto por la parte más cercana al origen del robot, el segundo por la parte más lejana y nuevamente el tercero

por la parte más cercana. Por último se tomará una posición final a 5 cm a la derecha del último objeto, para volver a la posición de partida, donde concluirá la ejecución.

Como hemos comentado, es extremadamente importante tener en cuenta las limitaciones de los datos recibidos por el escáner. Según los manuales adjuntos, hay un error sistemático de 15 mm en las medidas recibidas, ello puedo suponer que al intentar moverse a los puntos de paso, si estos quedan desplazados en el error admisible, choque con alguno de los objetos. Se recomienda no perder la cabeza ante esto situación y analizar las medidas recibidas por el sensor, intentando moverse con un margen mayor o modificando los propuestos en este proyecto para aumentar la seguridad de las operaciones. Debido al bloqueo del módulo 2, el cual no ha podido ser subsanado, nos hemos visto limitados al empleo de dos únicas articulaciones, por lo que también es fácil encontrarse con que de forma fortuita se cree un punto de paso imposible para las dimensiones del robot o que suponga moverse el módulo 3 una cantidad de mayor a 135º o menor de – 135º; o bien el módulo 5 por encima de +91º o por debajo de – 91º. En el primer caso todos los módulos quedarán en la posición de origen por defecto, mientras que en el segundo quedarán detenidos en la posición máxima admisible, por lo que en cualquiera de los dos casos la respuesta no será la esperada.

Los puntos de paso que estableceremos son:

- Punto de paso 1: misma coordenada x del objeto 1 y reduciremos la coordenada y en 9 cm.
- Punto de paso 2: punto de medio del objeto 1 y 2.
- Punto de paso 3: misma coordenada x del objeto 2 y aumentaremos la coordenada y en 7 cm.
- Punto de paso 4: punto medio de los objetos 2 y 3.
- Punto de paso 5: misma coordenada x que el objeto 3 y la coordenada y será reducida en 9 cm.
- Punto de paso 6: la misma coordenada y que el objeto 3 y la coordenada x será ampliada en 5 cm.

Asimismo, como variables de partida para la cinemética inversa que describiremos a continuación, será necesario definir los ángulos “Alfai” como la

posición de cada uno de los puntos de paso i respecto al origen del robot, y la distancia "Hipotenusai" como aquella que separa el punto de paso i del origen del robot. Mostramos a continuación la trigonometría básica necesaria.

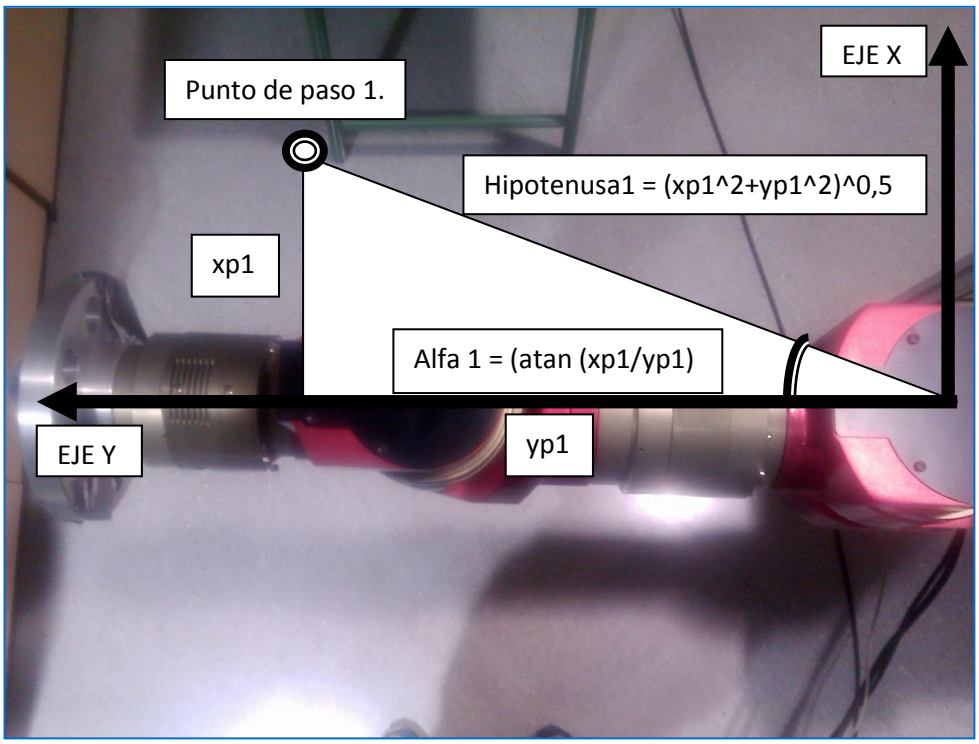


Ilustración 5.22. Transformación al sistema de referencia del robot

```
319  int trayectoria;
320 trayectoria = 1; // Evitar obstaculos
321  //trayectoria = 2; // Analizar objetos
322
323  if (trayectoria == 1 )
324  {
325
326  // PUNTO DE PASO 1 por detrás objeto 1
327  xp1 = xr1;
328  yp1 = yr1-0.09;
329  Hipotenusa1 = pow(pow(xp1, 2)+pow(yp1, 2), 0.5f);
330  Alfa1 = atan2(xp1,yp1)*180/pi;
```

Ilustración 5.23. Cálculo del punto de paso 1.

Debido a la incompatibilidad de tipos de datos, es necesario pasar al 0,5 a float añadiendo una f inmediatamente después del valor.

Definiremos las variables AngBrazoi como la posición angular que debe tomar el módulo 3 para alcanzar la posición objetivo i. De la misma manera con AngAntBrazoi para el módulo 5. Dichos valores se calculan siguiendo los algoritmos de cinemática inversa descritos en el siguiente capítulo.

5.5.1.3. Ejecución de los movimientos

Para llevar a cabo los movimientos será necesario emplear la librería m5apiw32.h suministrada por robotnik cuyas funciones se definen en el Anexo1. Realizaremos asimismo las siguientes consideraciones:

- No vamos a considerar el movimiento del robot a una posición de partida, pues supondremos que el robot ya está inicialmente en dicha posición y no tendrá que ser desplazado a la misma.
- Definiremos un tiempo entre instrucciones tpaso = 5000 s. Es necesario llevar cuidado con su elección y puede que tenga que ser corregido localmente para evitar un funcionamiento anómalo. (Ver errores frecuentes)
- Es posible que inicialmente se encuentren bloqueadas las articulaciones de nuestro brazo. Por ello se hace necesario resetearlas para ponerlas en funcionamiento y no encontrar funcionamientos anómalos durante la ejecución. Para ahorrar el elevado espacio en memoria que supondría resetear siempre de forma sistemática dichos módulos optamos por emplear una de las funciones de la librería de robotnik: *PCube_getModuleState.* Dicho valor devuelve el estado del módulo que se le solicite y, si dicho estado indica que el módulo está bloqueado, se procede a habilitarlo. En caso de estar ya funcional, se omite esta operación.
- La función Pcube_resetModule realiza la acción de reiniciar el módulo y dejarlo en marcha para el funcionamiento.
- Inicialmente se optó por emplear la función *Pcube_moveRamp,* la cual necesita como datos de entrada la posición objeto, asi como la velocidad y aceleración deseadas. Al iniciar la ejecución se mostraba un caja de texto con el siguiente mensaje: "*Pulse cuando haya acabado la instrucción*". En el momento de pulsar

se ejecutaba la secuencia correspondiente al siguiente punto de paso. Sin embargo esto suponía unas limitaciones bastante obvias, como la que supone tener a un operario atento a la secuencia. Es por ello por lo que se optó por emplear la función PCube_moveStep, dicha función únicamente necesita la posición angular objetiva y el tiempo deseado para llegar hasta ella, calculándose automáticamente la velocidad y aceleración necesarias para tal fin.

- Sabiendo el tiempo que se va a tardar en llegar a la posición objetivo es posible introducir un retraso para ejecutar la siguiente instrucción que coincida con el tiempo de paso que va a tardar nuestro robot en alcanzar la posición. Mediante la función Sleep (tpaso); podemos conseguir esto. Ahora habremos conseguido un programa mucho más automatizado e interesante.
- Es muy importante recordar que las instrucciones que mandamos ejecutar deben ir en radianes, por lo que hay que transformarlas al recibirlas en grados.
- Ponemos como ejemplo como ejecutar el punto de paso 1, siendo exactamente igual para el resto de puntos.

```
427  tpaso=5000;
428  //POS1
429  if  ( xr2-xr1 > 0.080  ||  yr2-yr1 > 0.080 )
430  {
431
432            if (!pDlg->m_parar)   // Articulación 3
433              {
434              PCube_getModuleState (m_dev, 3, &state3);
435                  if(state3=8206)
436                      {
437                          PCube_resetModule (pDlg->m_dev, 3);
438                      }
439                          PCube_moveStep (pDlg->m_dev, 3, AngBrazo1*pi/180 , tpaso);
440              }
441
442             if (!pDlg->m_parar)  // Articulación 5
443              {
444                  PCube_getModuleState (m_dev, 5, &state5);
445                      if(state5=8206)
446                          {
447                              PCube_resetModule (pDlg->m_dev, 5);
448                          }
449                  PCube_moveStep (pDlg->m_dev, 5, AngAntBrazo1*pi/180, tpaso);
450              }
451          Sleep (tpaso);
452  }
```

Ilustración 5.24. Ejecución punto de paso 1.

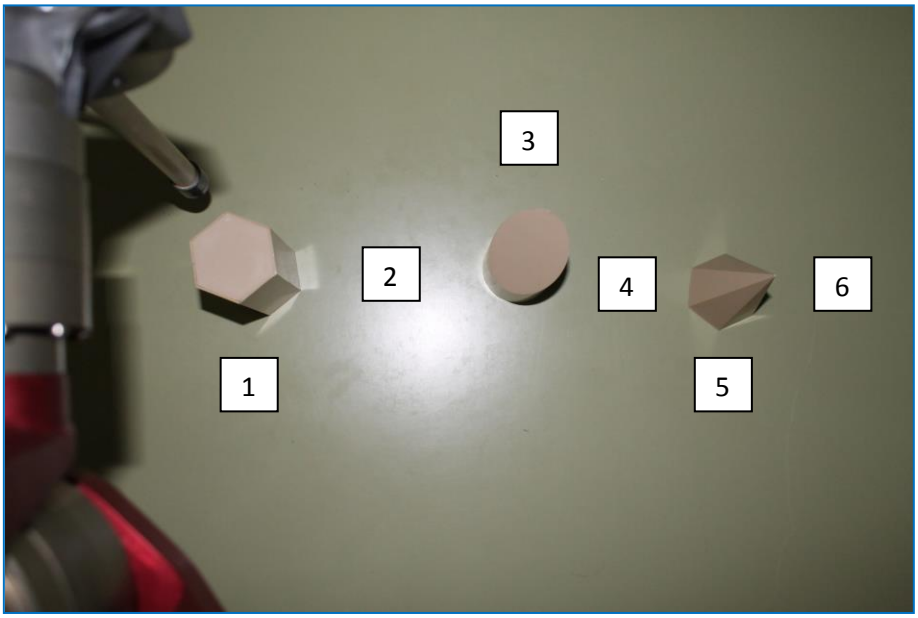


Ilustración 5.25. Puntos de paso situación normal en zig-zag.

5.5.1.4. Problemática de objetos demasiado juntos.

Según lo visto hasta ahora deberíamos plantearnos inmediatamente qué ocurriría si dos de los objetos estuvieran tan cercanos que a la hora de ejecutar los puntos de paso 2 y 4 (puntos medios entre los objetos 1 y 2, y 2 y 3 respectivamente) no hubiera espacio suficiente para pasar entre ellos. La respuesta parece tan obvia como la pregunta. Se produciría una colisión entre el robot y los objetos. Sin embargo, lo que podría ser anecdótico dentro del laboratorio podría suponer unas consecuencias catastróficas si eso ocurriera en una situación ingenieril real. Es por ello que nada debe dejarse a la suerte o a la casualidad y trataremos a continuación como evitar esta situación.

Lo primero que debemos hacer tanto en el punto de paso 2 como en el punto de paso 4 (las únicas dos situaciones en las que puede haber problemas pues no consideraremos que para una misma posición angular respecto al escáner puedan haber varios objetos a distinta distancia del mismo por el único motivo que no podría ser detectado por el sensor y por lo tanto no implementado en nuestro programa) es establecer una condición a partir de la cual ejecuta la secuencia normal si la distancia entre los objetos es suficiente para pasar entre ellos. Si dicha distancia no es suficiente

se ejecuta una trayectoria alternativa que implicará rodear ambos objetos como si fueran un todo. Para ello empleamos un OR lógico (representado por el símbolo "||"). Ello implicará que si se cumple cualquiera de las dos condiciones se ejecutan las instrucciones indicadas.

Hemos establecido un margen de 8 cm. Ha sido necesaria la secuencia lógica indicada para evitar que interprete que no puede pasar entre dos objetos colocados muy cercanos según el eje X pero muy lejanos según el eje Y. En dicho caso se ejecutará la secuencia normal vista anteriormente.

En caso de que no se cumple ninguna se pasa automáticamente a las instrucciones indicadas en el *else*. Ello querrá decir que la diferencia de posición es menor de 7 cm tanto según el eje X como el eje Y.

5.5.1.4.1. Objetos 1 y 2 muy cercanos.

Los puntos 1 y 2 sólo se ejecutarán si los objetos 1 y 2 están lo suficientemente separados entre sí (en nuestro ejemplo establecemos 8 cm); en caso contrario, de la posición inicial pasaremos al punto 3 y continuaremos el resto del código según lo previsto. Ello se consigue con los condiciones mostrados, si no se cumplen los requisitos de distancia se omitirá la parte del código que ejecuta los dos primeros puntos de paso.

```
453      //POS 2
454      if  ( xr2-xr1 > 0.080  ||  yr2-yr1 > 0.080 )
455          {
456        if (!pDlg->m_parar)   // Articulación 3
457              {
458                  PCube_getModuleState (m_dev, 3, &state3);
459                      if(state3=8206)
460                          {
461                              PCube_resetModule (pDlg->m_dev, 3);
462                          }
463                  PCube_moveStep (pDlg->m_dev, 3, AngBrazo2*pi/180 , tpaso);
464              }
465        if (!pDlg->m_parar)   // Articulación 5
466              {
467                  PCube_getModuleState (m_dev, 5, &state5);
468                      if(state5=8206)
469                          {
470                              PCube_resetModule (pDlg->m_dev, 5);
471                          }
472                  PCube_moveStep (pDlg->m_dev, 5, AngAntBrazo2*pi/180, tpaso);
473              }
474       Sleep (tpaso);
475      }
```

Ilustración 5.26. Ejecución punto de paso 2.

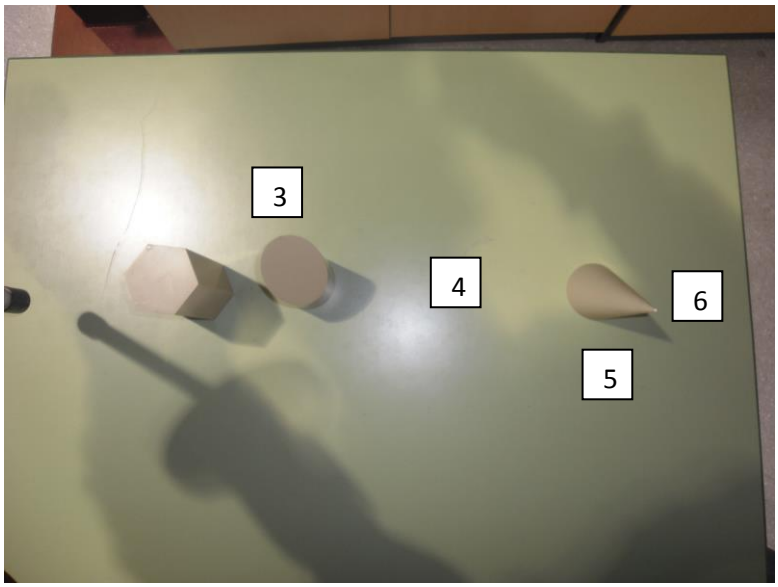


Ilustración 5.27. Secuencia puntos de paso (objetos 1 y 2 muy cercanos)

5.5.1.4.2. *Objetos 2 y 3 muy cercanos.*

Exactamente ocurrirá lo mismo con el punto de paso 4, con la particularidad de que, ahora, pasaremos del punto de paso 3 al punto final 6.

```
503   //POS 4
504       if  ( abs(xr3-xr2) > 0.080  ||  abs(yr3-yr2) > 0.080 )
505       {
506         if (!pDlg->m_parar)   // Articulación 3
507               {
508                 PCube_getModuleState (m_dev, 3, &state3);
509                   if(state3=8206)
510                       {
511                           PCube_resetModule (pDlg->m_dev, 3);
512                       }
513                  PCube_moveStep (pDlg->m_dev, 3, AngBrazo4*pi/180 , tpaso);
514               }
515
516         if (!pDlg->m_parar)   // Articulación 5
517               {
518                   PCube_getModuleState (m_dev, 5, &state5);
519                       if(state5=8206)
520                           {
521                               PCube_resetModule (pDlg->m_dev, 5);
522                           }
523                   PCube_moveStep (pDlg->m_dev, 5, AngAntBrazo4*pi/180, tpaso);
524               }
525        Sleep (tpaso);
526       }
```

Ilustración 5.28. Secuencia punto de paso 4.

```
528         //POS 5
529         if  ( abs(xr3-xr2) > 0.080  ||  abs(yr3-yr2) > 0.080 )
530         {
531            if (!pDlg->m_parar)    // Articulación 3
532               {
533                   PCube_getModuleState (m_dev, 3, &state3);
534                       if(state3=8206)
535                           {
536                               PCube_resetModule (pDlg->m_dev, 3);
537                           }
538                   PCube_moveStep (pDlg->m_dev, 3, AngBrazo5*pi/180 , tpaso);
539               }
540            if (!pDlg->m_parar)    // Articulación 5
541               {
542                   PCube_getModuleState (m_dev, 5, &state5);
543                       if(state5=8206)
544                           {
545                               PCube_resetModule (pDlg->m_dev, 5);
546                           }
547                   PCube_moveStep (pDlg->m_dev, 5, AngAntBrazo5*pi/180, tpaso);
548               }
549   Sleep (tpaso);
550         }
```

Ilustración 5.29. Secuencia punto de paso 5.

Como comprobamos en las imágenes anteriores, si los objetos 2 y 3 están excesivamente cerca se omitirá pasar a través de ellos, así como por el punto de referencia 5, evitando así su colisión.

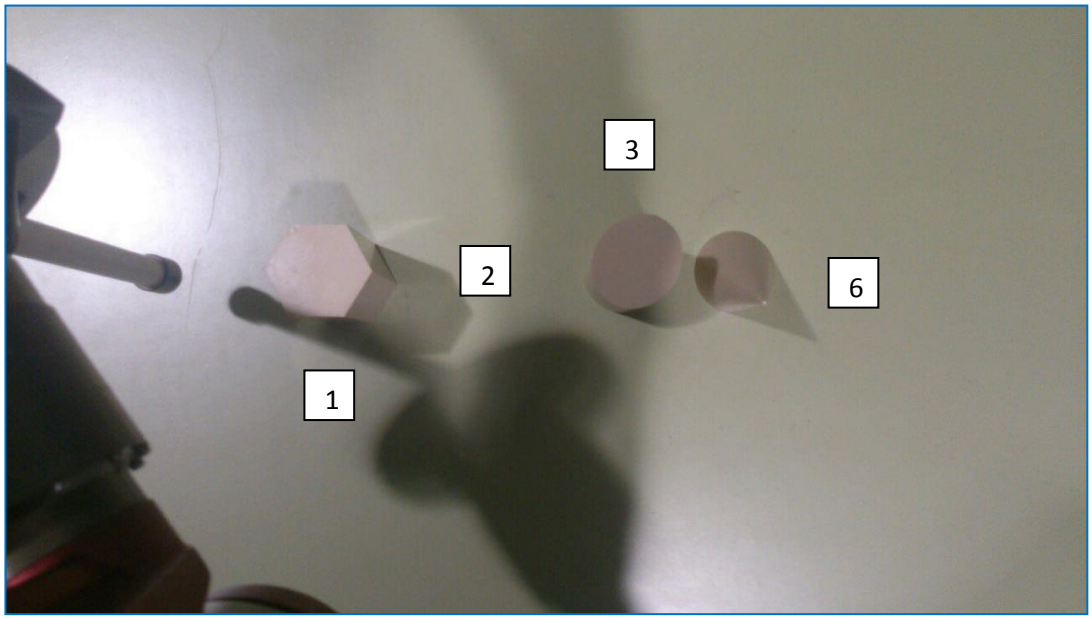


Ilustración 5.30. Secuencia puntos de paso (objetos 2 y 3 muy cercanos)

5.5.2. Trayectoria circular

Como hemos explicado anteriormente tras inicializar esta opción el puntero rodeará completamente los objetos antes de pasar al siguiente.

5.5.2.1. Habilitar la ejecución de esta variante del código

De la misma manera que para la secuencia de evitar obstáculos, lo primero será habilitar la entrada del programa en las funciones deseadas.

```
318   int trayectoria;
319   //trayectoria = 1; // Trayectoria en zig-zag
320   trayectoria = 2; // Trayectoria circular
```

Ilustración 5.31. Selección secuencia Trayectoria circular.

5.5.2.2. Planificación de los puntos de paso

A continuación calculamos los puntos de paso para realizar las trayectorias que permiten rodear cada uno de los objetos hasta llegar a la posición final.

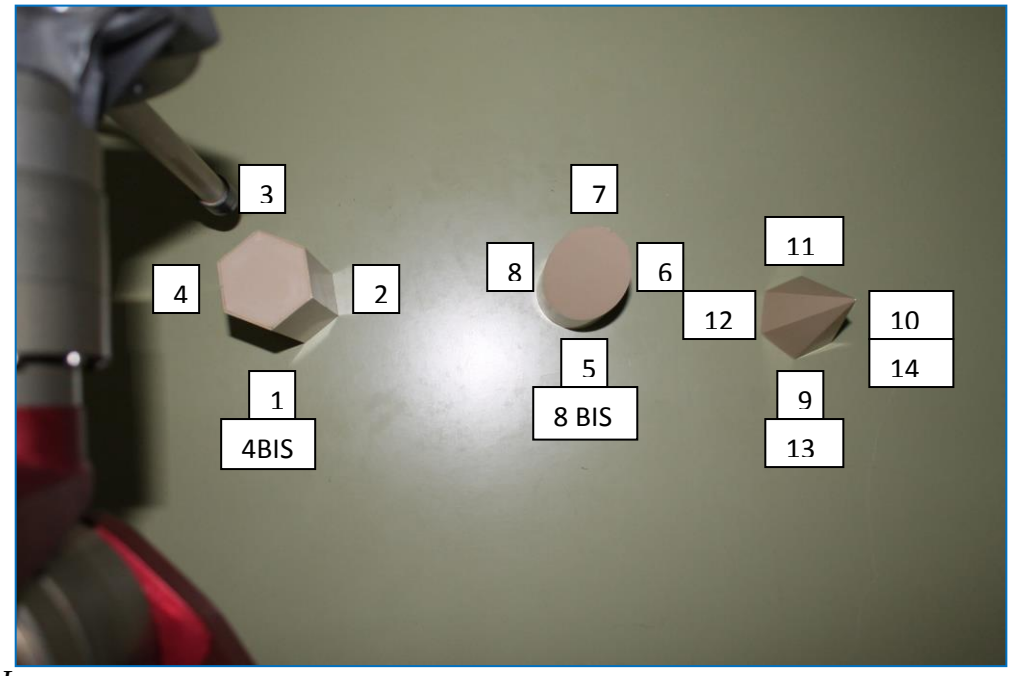


Ilustración 5.32. Puntos de paso para la Trayectoria circular.

Tras finalizar el código proponemos la mejora de introducir los puntos 4BIS y 8BIS antes de pasar al siguiente objeto, de otra manera es frecuente que el robot choque con el objeto que acaba de analizar antes de pasar al siguiente objeto.

La ejecución de los movimientos, intervalos de Sleep y la cinemática inversa será exactamente igual que en el caso anterior por lo que no lo repetiremos en este punto. En el mismo código puede verse paso por paso esta secuencia del código, así como en los vídeos que se adjuntan.

CAPÍTULO 6:

Cálculos de la cinemática

6. CÁLCULOS DE LA CINEMÁTICA

Una vez que hemos automatizado la obtención de la posición de cada uno de los puntos a los que debe ir nuestro brazo robótico para cumplir su objetivo nos encontramos con un nuevo problema: la obtención de cuantos grados debe moverse cada uno de los módulos del robot para alcanzar dicha posición. A este proceso se le llama cinemática inversa y su resolución es mucho más compleja que en el caso de la cinemática directa, así como exponencialmente más laboriosa cuanto mayor sea el número de grados de libertad.

Hemos desarrollado un algoritmo para su cálculo de manera simple. Al estar trabajando en un plano y tener únicamente dos grados de libertad, tendremos 2! = 2 posiciones posibles, lo que suele llamarse posición "codo arriba" o "codo abajo".

6.1. DEFINICIONES Y OBJETIVOS

Disponemos de los siguientes datos:

- Longitud del brazo = 0,275 m.
- Longitud del antebrazo = 0,2 m.
- Eje X
- Eje Y

A partir de ellos, vamos a averiguar cuantos grados deben moverse los módulos 3 (brazo) y 5(antebrazo). Es importante recordar las limitaciones de movimiento de cada uno de ellos para no chocar con otras partes del robot.

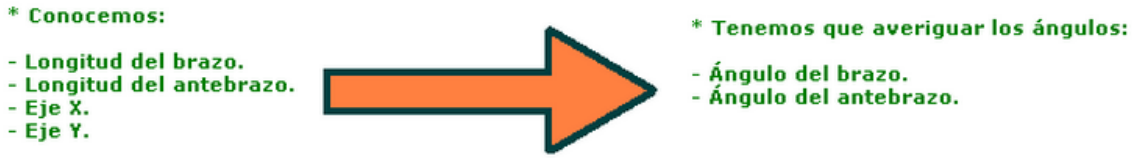


Ilustración 6.1. Objetivos de la cinemática inversa

Los sentidos positivos de los ejes se definen positivos según se muestran en la siguiente figura.

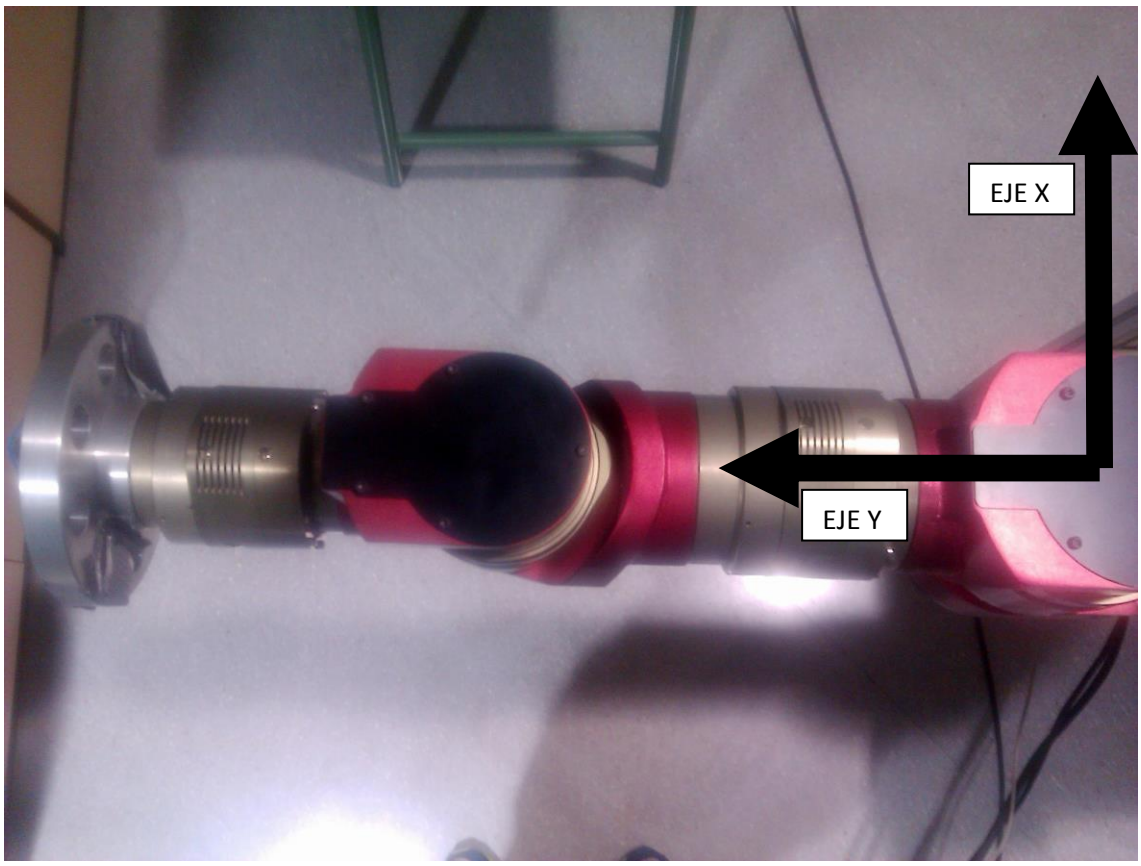


Ilustración 6.2. Ejes del robot.

6.2. CÁLCULOS

En este punto seguiremos llamando Alfai a la posición del punto de paso i respecto al origen del robot. Además, definiremos Betai como la orientación del brazo (extremidad que une los módulos 3 y 5) respecto a la posición del objeto y Gammai como la orientación de la prolongación del antebrazo respecto al brazo.

En la siguiente figura se muestra gráficamente todo lo comentado, así como la resolución analítica de estos valores.

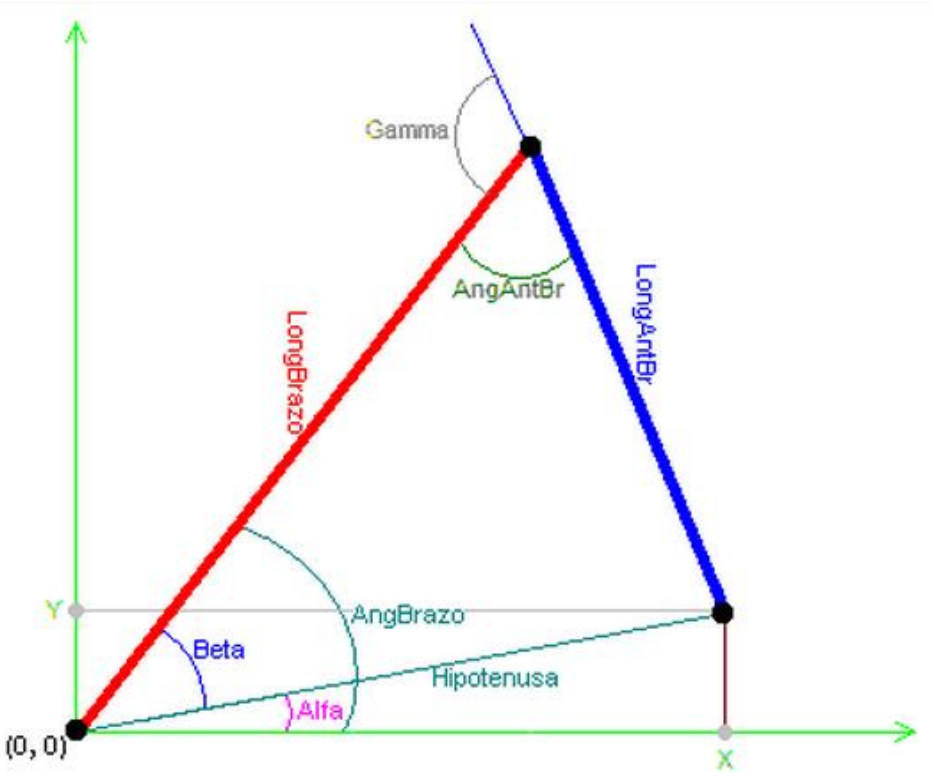


Ilustración 6.3. Representación de la cinemática inversa

Hipotenusa = sqr((X^2)+(Y^2))

Alfa = Atan2(Y, X)

Beta = Acos((LongBrazo^2 - LongAntBr^2 + Hipotenusa^2) / (2 * LongBrazo * Hipotenusa))

AngBrazo = Alfa + Beta

Gamma = Acos ((LongBrazo^2 + LongAntBr^2 - Hipotenusa^2) / (2 * LongBrazo * LongAntBr))

AngAntBr = Gamma - 180

Ilustración 6.4. Solución a la cinemática inversa

Mostramos a modo de ejemplo cual sería la resolución en nuestro código de la cinemática inversa del primer punto de paso. Nos mostramos los cálculos del resto de puntos de paso, pues pueden calcularse de la misma manera, además de estar todos incluidos en el código funcional adjunto al CD que acompaña a este proyecto.

```
332 // CINEMATICA INVERSA Punto de paso 1
333
334 LongBrazo = 0.275; //m
335 LongAntBrazo = 0.2; //m
336 Beta1 = acos((pow(LongBrazo, 2)-pow(LongAntBrazo, 2)+pow(Hipotenusa1, 2))/(2*LongBrazo*Hipotenusa1))*180/pi;
337 AngBrazo1 = (Alfa1+Beta1)-5;
338 Gamma1 = acos((pow(LongBrazo, 2)+pow(LongAntBrazo, 2)-pow(Hipotenusa1, 2))/(2*LongBrazo*LongAntBrazo))*180/pi;
339 AngAntBrazo1 = (Gamma1-180);
```

Ilustración 6.5. Programación de la cinemática inversa.

NOTA IMPORTANTE: En el módulo 3, la posición de referencia no es 0º. Dicho eje se encuentra desplazado +5 grados de la que debería ser la posición neutra. Esto puede ser debido a una mala orientación a la hora de anclar el brazo robótico a la base fija. Por ello, a la hora de calcular el valor AngBrazoi deberemos restar 5 grados al ángulo obtenido (línea 337 del gráfico anterior).

CAPÍTULO 7:

Errores frecuentes

7. ERRORES FRECUENTES

a) De forma puntual podemos encontrarnos con la sorpresa de que algunos módulos queden bloqueados al ejecutar la instrucción demandada, a pesar de haberse comprobado su estado y haber sido reseteados dentro del propio código. De forma empírica hemos encontrado que esto ocurre al establecer tpaso < 5000 s en ciertas ocasiones. Por ejemplo, con un tpaso = 3000 s puede comprobarse que a la hora de ejecutarse el punto de paso 6, el módulo 5 queda bloqueado y no varía respecto a la posición anterior, mientras que el módulo 3 si realiza su movimiento con normalidad. Por ello, en el código final y con objeto de no que se tardara tanto en realizar los movimientos se optó por establecer un tpaso = 3000 excepto en el punto de paso 6, que localmente se estableció un tpaso = 5000. Sería interesante de cara a desarrollos futuros encontrar la verdadera causa de este problema y como subsanarlo.

b) Es necesario tener mucho cuidado en las conversiones de grados a radianes y viceversa. A pesar de que en el *PowerCube Software* introducimos en grados la posición final de nuestros módulos a modo de ejemplo, en C++ empleando la librería m5apiw32.h debemos introducirlos en radianes. Debemos entonces multiplicar *pi/180. Obviamente la diferencia entre olvidar este matiz es de grandes dimensiones, y la posición objetivo pasaría a estar en una situación muy distinta. Si además hemos pedido que vaya a esa posición en tan solo 3000 ms se puede producir un movimiento a gran velocidad que podría provocar algún accidente. Por ello se recomienda nuevamente poner en marcha el robot sin presencia humana dentro de su zona de influencia y disponer siempre a mano de l

a
```
PCube_moveStep (pDlg->m_dev, 3, AngBrazo1*pi/180 , tpaso);
```

seta de emergencia.

c) También es frecuente dejar olvidar el puntero del robot en una posición tan próximo a la zona de detección de objetos que el propio escáner lo detecte como

tal. Entonces se obtendrá un funcionamiento anómalo. Se recomienda detener el programa y alejar el puntero para evitar esto. Para optimizar el tiempo y analizar las posibles fuentes de error se recomienda enormemente el empleo de puntos de ruptura o *Breakpoints*. Se colocan simplemente pulsando en el margen izquierdo del código hasta obtener un punto rojo como el de la figura. El código se detendrá en su ejecución al llegar a él y podremos seguir ejecutándolo línea a línea pulsando F10 (recomendable y para nosotros absolutamente imprescindible en la detección de errores) o bien pulsando la flecha verde de ejecución, en cuyo caso continuará hasta finalizar la ejecución de todo el código o encontrar otro Breakpoint. En caso de llegar a la llamada a una función podemos entrar en ella pulsando F11.

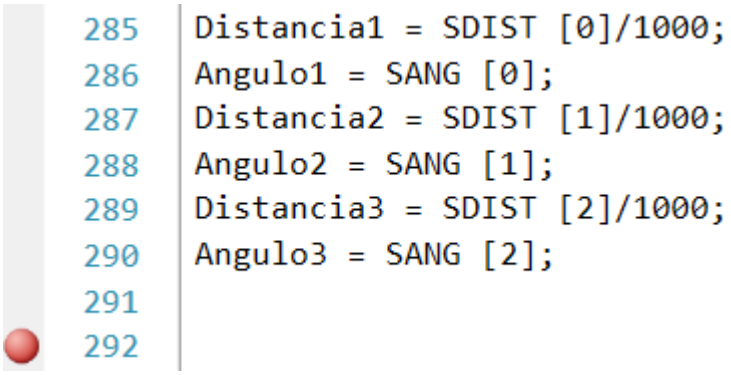


Ilustración 7.1. Empleo de los Breakpoints.

No hemos colocado el Breakpoint anterior en esa posición por casualidad. En el momento que el código se detenga ahí podremos ver el valor de Distancia1 y Angulo1 con sólo mantener el ratón sobre él. Si dichos valores no son coherentes se recomienda ver paso a paso pulsando F10 donde está el error.

d) En caso de colocar los objetos excesivamente cerca, es posible que la resolución angular del escáner interprete que son un único objeto y no sea capaz de encontrar un punto entre ambos a una distancia muy superior. Esto puede detectarse automáticamente viendo los valores de los ángulos donde existen objetos. Si el último dato (Angulo 3) es muy superior a lo esperado debemos plantearnos esta posibilidad. Separando sensiblemente los objetos conseguiremos superar el problema.

CAPÍTULO 8:

Desarrollos futuros

8. DESARROLLOS FUTUROS

La consecución del actual proyecto abre múltiples frentes de cara a desarrollos futuros con el mismo. Para ello, lo primero debería ser arreglar el módulo 2 por la empresa suministradora y volver a tenerlo completamente operativo.

Una vez conseguido esto, podría estudiarse implementar algoritmos de cinemática inversa más complejos para conseguir un movimiento más flexible y la realización del seguimiento de trayectorias con una zona de influencia mayor al del actual proyecto.

De la misma manera, aumentar (siempre cuidadosa y gradualmente) la velocidad de los módulos para alcanzar las trayectorias propuestas daría sin ninguna duda una mayor visibilidad y dinamismo al proyecto con un esfuerzo relativamente sencillo.

También sería muy interesante el trazado de un algoritmo de trazado de trayectorias tridimensional que permitiera sacar el máximo partido al brazo robótico. En este caso, al usar todos los módulos del mismo se recomienda tener especial cuidado en materia de seguridad. Por ejemplo, añadir manualmente una pequeña pala o recogedora que permitiera recoger objetos o materiales a granel y elevarlos a una superficie de mayor cota podría resultar muy atractivo.

Asimismo, realizar la interacción con objetos en movimientos para tareas de domótica e interacción de con humanos en tiempo real supondría, sin ninguna duda, un gran avance respecto a los objetivos iniciales del proyecto.

CAPÍTULO 9:

Otras alternativas

9. OTRAS ALTERNATIVAS

9.1. ROS

9.1.1. ROS Hydro

A día de hoy, ROS Hydro es la versión estable más reciente compatible con los paquetes que nos interesa para poder desarrollar nuestro proyecto.

La citada versión de ROS sólo acepta las siguientes versiones de UBUNTU:

Ubuntu 12.04 (Precise)

Ubuntu 12.10 (Quantal)

Ubuntu 13.04 (Raring)

A continuación indicaremos como instalarlo:

Paso 1:

Lo primero es configurar los repositorios de Ubuntu para permitir “restringido”, “universo” y “multiverso”.

Nota: En Ubuntu 9.04 (Jaunty) y versiones posteriores, los repositorios principal, universo, restringido y multiverso están habilitados por defecto. No obstante se recomienda comprobarlo para prevenir futuros errores. by default.

Existen una gran cantidad de programas de Ubuntu disponibles para satisfacer las necesidades de los usuarios. Muchos de estos programas se almacenan en archivos de software comúnmente conocidos como repositorios. Esto hace que sea muy fácil instalar nuevo software en Ubuntu utilizando una conexión a Internet, a la vez que proporciona un alto nivel de seguridad, mucho mayor que otros sistemas operativos como Windows, ya que cada programa disponible en los repositorios está probado y construido específicamente para cada versión de Ubuntu.

Los repositorios de software de Ubuntu se organizan en cuatro áreas separadas o "componentes", de acuerdo con el nivel de apoyo ofrecido por Ubuntu y si el programa en cuestión cumple con el Software Libre, filosofía de Ubuntu.

Los componentes del repositorio son:

Principal - con apoyo oficial del software.

Restringido - software que no está disponible bajo una licencia completamente libre compatibles.

Universo - Comunidad mantiene el software, el software es decir, no con apoyo oficial.

Multiverso - El software que no es libre.

Los CDs de instalación de Ubuntu contienen software de los componentes "restringidos" y "Principal". Una vez que el sistema está al tanto de los lugares de Internet para estos repositorios, muchos más programas estarán disponibles para la instalación. Mediante el uso de las herramientas de gestión de paquetes de software ya instalados en el sistema, es posible buscar, instalar y actualizar cualquier pieza de software directamente a través de Internet, sin la necesidad del CD.

AÑADIR REPOSITORIOS EN UBUNTU:

Lo descrito a continuación modifica el archivo de configuración de repositorios de software con sede en:

```
/etc/apt/sources.list
```

Para las últimas versiones de Ubuntu, la forma más fácil es ir a través de la "Ubuntu Software Center". Abra el centro de software, a continuación, en el menú Edición, seleccione "Orígenes del Software". Es necesario introducir la contraseña del equipo para cambiar la configuración en esta ventana.

Para versiones anteriores de Ubuntu, hay varias opciones:

Menú principal: Sistema> Administración> Orígenes del Software.

Synaptic: Sistema> Administración> Gestor de paquetes Synaptic: >> Configuración >> Repositorios.

Menú Principal: Ubuntu Software Center: >> Editar, Orígenes del Software.

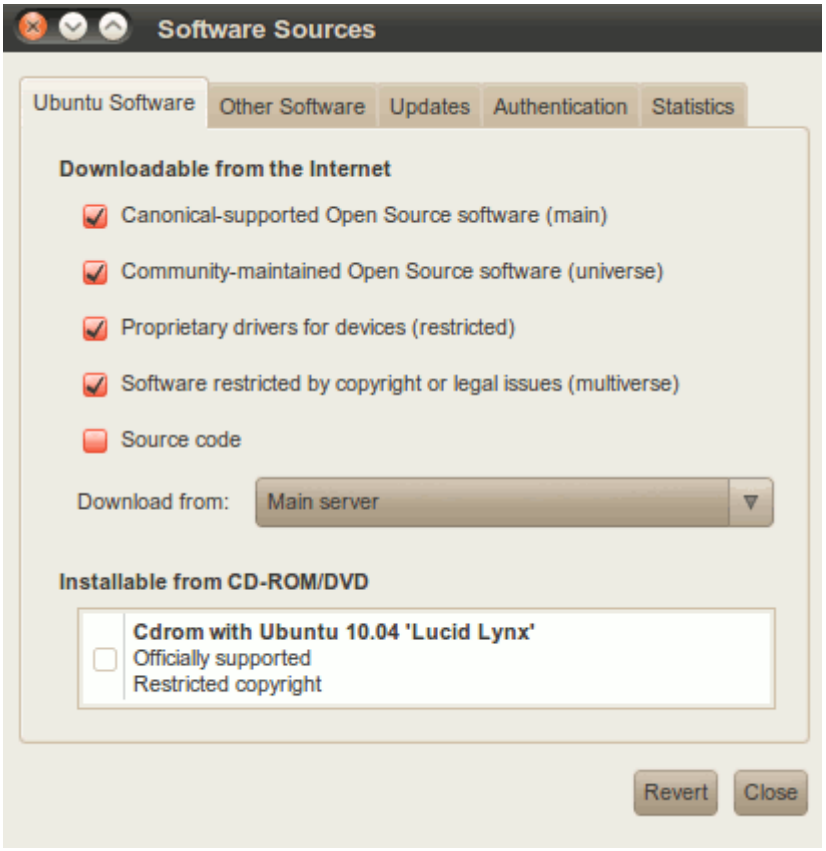


Ilustración 9.1. Verificación orígenes software.

Paso 2:

Configurar el sistema para aceptar paquetes de nuestra fuente. ROS Hydro únicamente soporta paquetes "Precise" (Ubuntu 12.04); "Quantal" (Ubuntu 12.10) y "Raring" (Ubuntu 13.04). Para ello es necesario activar el comando como "superusuario" (sudo), lo cual ejecuta el código como administrador del sistema, por lo que se nos pedirá la contraseña del equipo.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu XXX main" > /etc/apt/sources.list.d/ros-latest.list'
```

En nuestro caso, XXX tomará el valor Precise por estar trabajando en la versión de Ubuntu 12.04, en caso de trabajar con Ubuntu 12.10 o 13.04 habría que sustituirlo por Quantal o Raring respectivamente.

Paso 3:

Configurar las claves:

```
wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add -
```

Paso 4: Instalación.

Lo primero es asegurarse de que nuestro índice de paquetes está actualizado:

```
sudo apt-get update
```

Hay varias formas posibles de instalar los paquetes de ROS.

- Instalación completa (recomendada). Instala todos los paquetes disponibles:

```
sudo apt-get install ros-hydro-desktop-full
```

- Instalación parcial: ROS, rqt, rviz, y librerías genéricas.

```
sudo apt-get install ros-hydro-desktop
```

- Instalación ROS-Base. (Bare Bones). Instala la base de ROS junto con las librerías de comunicación. No instala herramientas GUI.

```
sudo apt-get install ros-hydro-ros-base
```

Las herramientas GUI (en inglés Graphical User Interface) sirven para automatizar el proceso de pruebas de software con interfaces gráficas de usuario. Consiste entonces en la aplicación de una forma en la cual un usuario puede interactuar con el hardware, en nuestro caso el robot, realizando distintas funcionalidades en forma intuitiva y dinámica. Es por ello imprescindible su instalación.

- Es posible instalar independientemente los paquetes deseados:

```
sudo apt-get install ros-hydro-PACKAGE
```

Por ejemplo:

```
sudo apt-get install ros-hydro-slam-gmapping
```

Es posible encontrar los paquetes disponibles usando:

```
apt-cache search ros-hydro
```

Paso 5: Iniciar rosdep

Antes de iniciar ROS, será necesario iniciar rosdep. Este paquete permite instalar fácilmente las dependencias del sistema para las fuentes que se deseen compilar y es necesario para algunos componentes del núcleo de ROS.

```
sudo rosdep init
rosdep update
```

Paso 6: Configuración del entorno

Un interprete de comandos es un programa que funciona como interfaz de usuario con el sistema operativo. Es el encargado de traducir los comandos de los usuarios a instrucciones que el sistema operativo pueda entender y viceversa, traducir el resultado devuelto por el sistema operativo a un lenguaje que los usuarios podamos entender.

acrónimo de *Bourne-Again Shell*

```
echo "source /opt/ros/hydro/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Para cambiar la versión de ROS que estamos empleando:

```
source /opt/ros/hydro/setup.bash
```

Paso 7: Obtener *rosinstall*

rosinstall es una herramienta de línea de comandos que se utiliza con frecuencia en ROS y que se distribuye por separado. Permite descargar fácilmente muchos árboles de código fuente para paquetes de ROS con un comando.

```
sudo apt-get install python-rosinstall
```

9.2. MOVE IT !

9.2.1. Introducción

MoveIt! es la interfaz empleada por ROS para la manipulación móvil, que incorpora los últimos avances en la planificación de movimiento, manipulación, percepción 3D, cinemática, el control y la navegación. Proporciona una plataforma fácil de usar para el desarrollo de aplicaciones de robótica avanzada, la evaluación de nuevos diseños de robots y productos de construcción robótica integrada por industriales, comerciales, de I + D y otros dominios.

9.2.2. Instalación del paquete

Actualmente Moveit! Esta desarrollado para trabajar en Ubuntu y con las versiones ROS Hydro y Groovy. Se espera que pronto pueda estar disponibles para ROS Indigo.

Si se está trabajando con Ubuntu 13.4 32 bits aparece un error con el paquete Eigen 3.1.2. Desde la propia firma de ROS comunican que no esperan que vaya a ser arreglado, por lo que recomiendan actualizar el sistema a la misma versión de 64 bits.

Veamos a continuación como instalar Moveit! En su versión estable más reciente (Hydro)

Paso 1:

Moveit! Puede ser instalado directamente como un conjunto de paquetes en Ubuntu. Para ello:

sudo apt-get install ros-hydro-moveit-full

Paso 2:

Moveit! Ofrece instalar, a modo de ejemplo, los paquetes relacionados con el robot PR2 que podrán usarse a modo de prueba para comprender mejor el funcionamiento de ROS. Aunque no sea necesario se recomienda su instalación:

sudo apt-get install ros-hydro-moveit-full-pr2

Paso 3:

Lo siguiente será configurar el entorno de trabajo. Por defecto el espacio de trabajo desde donde podremos cargar los paquetes de ROS Hydro será:

source /opt/ros/hydro/setup.bash

En caso no cargar el espacio de trabajo donde tengamos instalados los paquetes, aparece un error por pantalla.

9.2.3. Errores aparecidos

Si se está trabajando con Ubuntu 13.4 32 bits aparece un error con el paquete Eigen 3.1.2. Desde la propia firma de ROS comunican que no esperan que vaya a ser arreglado, por lo que recomiendan actualizar el sistema a la misma versión de 64 bits. Igualmente aparece este error en la versión 12.4 (la empleada hasta el momento). Por lo que se recomienda actualizarla a la versión 13.4 64 bit.

CAPÍTULO 10:

Bibliografía

10. BIBLIOGRAFIA

http://www.linux-es.org/

https://wiki.citius.usc.es/inv:por-clasificar:ros

Lewis, Frank L. Abdallah, C. T. Dawson, D. M. (1993). *Control of robot manipulators*

New York : Macmillan

Mark Spong F.L. Lewis C.T. Abdallah (1993). *Robot Control. Dynamic, Motion Planning and Analysis.* New York, IEEE

Rembold, Ulrichn (1990). *Robot technology and applications* . New York : Marcel Dekker.

http://msdn.microsoft.com/es-es/library/dd286726.aspx

«Microsoft Security Bulletin MS11-025 - Important : Vulnerability in Microsoft Foundation Class (MFC) Library Could Allow Remote Code Execution (2500212)». Microsoft.com.

«Microsoft Security Bulletin MS09-035 - Moderate : Vulnerabilities in Visual Studio Active Template Library Could Allow Remote Code Execution (969706)». Microsoft.com.

http://www.microsoft.com/security/atl.aspx (enlace roto disponible en Internet Archive; véase el historial y la última versión).

«Download Microsoft Visual Studio 2005 Service Pack 1 MFC Security Update from Official Microsoft Download Center»

«Visual C++ - Exploring New C++ and MFC Features in Visual Studio 2010».

http://www.youtube.com/watch?v=FF51TOje8N4

Lewis, Frank L. Abdallah, C. T. Dawson, D. M. (1993). Control of robot manipulators

Somasegar, S. (2014-07-08). "Announcing the Visual Studio 2013 Release Candidate". Microsoft. Retrieved 2013-09-14.

http://www.etitudela.com/profesores/rpm/rpm/downloads/robotica.pdf

http://sisbib.unmsm.edu.pe/bibvirtual/publicaciones/indata/v04_n1/actualidad.htm

Mark Spong F.L. Lewis C.T. Abdallah (1993). Robot Control. Dynamic, Motion Planning and Analysis. New York, IEEE

Rembold, Ulrichn (1990). Robot technology and applications . New York : Marcel Dekker

http://msdn.microsoft.com/es-es/library/d06h2x6e.aspx

http://www.microsoft.com/es-es/download/details.aspx?id=23691

http://www.es.schunk.com/schunk/index.html?lngCode=ES&country=ESP

http://www.robotnik.es/

ROBOTNIK_MODULAR_ARM-Installation and System Configuration Manual

ROBOTNIK_MODULAR_ARM-Robot Dynamics Manual

ROBOTNIK_MODULAR_ARM-Robot kinematics and control

ROBOTNIK_MODULAR_ARM-System Architecture Manual

ROBOTNIK_MODULAR_ARM-System Startup Manual

http://freespace.virgin.net/hugo.elias/models/m_ik2.htm

http://proyectosroboticos.wordpress.com/tag/cinematica-inversa

http://proton.ucting.udg.mx/materias/robotica/r166/r78/r78.htm

www.esi2.us.es/~vivas/ayr2iaei/DET_PLAN_CAMINOS.pdf

http://msdn.microsoft.com/es-es/library/60k1461a.aspx

http://wiki.ros.org/

https://wiki.citius.usc.es/inv:por-clasificar:ros

http://robolab.dieecs.com/equipos/medios

https://www.mysick.com/

Anexo I:

Librería m5apiw32

ANEXO I: LIBRERÍA M5APIW32

A continuación mostramos las funciones que nos permite emplear la librería proporcionada por robotnik. Aunque hay multitud de ellas, todas pueden llegar a resultar útiles según los objetos a perseguir.

Apertura y cierre de la interfaz de comunicación	Función	Descripción
	PCube_openDevice	Abre la interfaz especificada por InitString. Devuelve un ID válido.
	PCube_closeDevice	Cierra la interfaz especificada.

Funciones administrativas	Función	Descripción
	PCube_getModuleIdMap	Recupera el número de módulos que hay en el bus. Al mismo tiempo visualiza la dirección física de los módulos en ID lógicas, estas son guardadas en un array en orden ascendente.
	PCube_updateModuleIdMap	Actualiza el mapa de módulos
	PCube_getModuleCount	Devuelve el número de módulos conectados.
	PCube_getModuleType	Asocia cada módulo a un tipo de mecanismo. Mecanismo de rotación: TYPEID_MOD_ROTARY = 0x0F Mecanismo lineal: TYPEID_MOD_LINEAR = 0xF0
	PCube_getModuleVersion	Recupera la versión del sistema operativo del módulo, el resultado está expresado en hexadecimal.
	PCube_getModuleSerialNo	Recupera el número de serie de un módulo.
	PCube_getDllVersion	Devuelve la versión del DLL utilizado.

	PCube_configFromFile	Permite configurar el sistema a partir de un archivo Ini.
	PCube_serveWatchdogAll	Refresca el perro guardián de todos los módulos. Sólo es válido para CAN.
	PCube_getDefSetup	Devuelve la configuración por defecto de un módulo. El resultado es una palabra de instalación
	PCube_getDefBaudRate	Recupera el valor por defecto de la velocidad de comunicación o ancho de banda. Sólo valido para RS232 y CAN. El resultado está expresado entre 0 y5.
	PCube_setBaudRateAll	Ajusta la velocidad de comunicación a la especificada. Sólo válido para bus CAN.
	PCube_getDefGearRatio	Devuelev el valor de la relación de transmisión por defecto.
	PCube_getDefLinearRatio	Devuelve el factor de conversión por defecto de un movimiento rotatorio a lineal.
	PCube_getDefCurRatio	Devuelve el factor por defecto para la conversión de corriente en dígitos a Amperios.
	PCube_getDefBrakeTimeOut	Recupera el retraso por defecto entre el final del movimiento y el descanso.
	PCube_getDefIncPerTurn	Devuelve el valor por defecto del número de incrementos por rotación del motor.

Recuperación de posición	Función	Descripción
	PCube_getPos	Devuelve la posición actual del módulo especificado por las ID del mecanismo y del módulo. El resultado es expresado en radianes.
	PCube_getPosInc	Devuelve la posición actual del módulo especificando por las ID del mecanismo y del módulo. El resultado es expresado en incrementos.

Recuperación de velocidad	Función	Descripción
	PCube_getVel	Devuelve la velocidad actual asociada al mecanismo y al módulo especificado. El resultado es expresado en radianes/s.
	PCube_getVelInc	Devuelve la velocidad actual del mecanismo asociado a las ID especificadas. El resultado es expresado en incrementos/s.
	PCube_getIPolVel	Devuelve la velocidad interpolada del mecanismo asociado a las ID especificadas. El resultado es expresado en radianes/s.

Recuperación de estado del módulo	Función	Descripción
	PCube_getModuleState	Devuelve el estado actual del módulo asociado a las ID especificadas. El resultado es una palabra de estado.
	PCube_getStateDioPos	Devuelve una combinación de información acerca del estado del módulo, posición y estado digital.

Recuperación de posición síncrona	Función	Descripción
	PCube_savePosAll	Fuerza a todos los módulos conectados a guardar la posición actual.
	PCube_getSavePos	Recupera la posición almacenada con la función PCube_savePosAll de un módulo.

Configuración	Función	Descripción
	PCube_getDefConfig	Recuperación la configuración por defecto del módulo asociado a las ID especificadas. El resultado es una palabra de configuración.
	PCube_getConfig	Recupera la configuración actual del módulo asociado a las ID especificadas.
	PCube_setConfig	Ajusta la configuración del módulo a la deseada a través de una nueva palabra de configuración.

Coeficientes del lazo PID	Función	Descripción
	PCube_getDefA0	Devuelve el valor por defecto del parámetro A0.
	PCube_getA0	Devuelve el valor actual del parámetro A0.
	PCube_setA0	Ajusta el valor de A0 al valor deseado (1..12).
	PCube_getDefC0	Devuelve el valor por defecto del parámetro C0.
	PCube_getC0	Devuelve el valor actual del parámetro C0.
	PCube_setC0	Ajusta el valor de C0 al valor deseado (12..64).
	PCube_getDefDamp	Devuelve el valor por defecto del coeficiente de amortiguamiento del sistema.
	PCube_getDamp	Devuelve el valor actual del coeficiente de amortiguamiento del sistema.
	PCube_setDamp	Ajusta el valor del coeficiente de amortiguamiento al valor deseado (1..4).
	PCube_recalcPIDParams	Actualiza el lazo PID y crea nuevos coeficientes válidos.

Rango de posición: posición mínima	Función	Descripción
	PCube_getDefMinPos	Devuelve la posición mínima por defecto del módulo especificado. El resultado está expresado en radianes.
	PCube_getMinPos	Devuelve la posición mínima del módulo especificado. El resultado está expresado en radianes.
	PCube_getMinPosInc	Devuelve la posición mínima del módulo especificado. El resultado está expresado en incrementos.
	PCube_setMinPos	Ajusta el valor actual de posición mínima del módulo especificado. El valor ha de ser en radianes o metros.

Rango de operación: posición mínima	Función	Descripción
	PCube_setMinPosInc	Ajusta el valor actual de posición mínima del módulo especificado. El valor está expresado en incrementos.

Rango de operación: posición máxima	Función	Descripción
	PCube_getDefMaxPos	Devuelve el valor de posición máxima por defecto del módulo especificado. El resultado está expresado en radianes.
	PCube_getMaxPos	Devuelve la posición máxima del módulo especificado. El resultado está expresado en radianes o metros.
	PCube_getMaxPosInc	Devuelve la posición máxima del módulo especificado. El resultado está expresado en incrementos.
	PCube_setMaxPos	Ajusta el valor de posición máxima del módulo especificado. El valor ha de ser expresado en radianes o metros.
	PCube_setMaxPosInc	Ajusta el valor de posición máxima del módulo especificado. El valor ha de ser expresado en incrementos.

Velocidad máxima	Función	Descripción
	PCube_getDefMaxVel	Devuelve la velocidad máxima por defecto del módulo especificado. La velocidad está expresada en rad/s o m/s.
	PCube_getMaxVel	Devuelve la velocidad máxima del módulo especificado. El resultado está expresado en rad/s o m/s.
	PCube_getMaxVelInc	Devuelve la velocidad máxima del módulo especificado. El resultado está expresado en incrementos/s.
	PCube_setMaxVel	Ajusta la velocidad máxima del módulo especificado. El valor ha de ser expresado en rad/s o m/s.
	PCube_setMaxVelInc	Ajusta la velocidad máxima del módulo especificado. El valor ha de ser expresado en incrementos/s.

Aceleración máxima	Función	Descripción
	PCube_getDefMaxAcc	Devuelve la aceleración máxima por defecto del módulo especificado. El resultado está expresado en rad/s² o m/s².
	PCube_getMaxAcc	Devuelve la aceleración máxima del módulo especificado. El resultado está expresado en rad/s² o m/s².
	PCube_getMaxAccInc	Devuelve la aceleración máxima del módulo especificado. El resultado está expresado en Incrementos/s².
	PCube_setMaxAcc	Ajusta la aceleración máxima del módulo especificado. El valor ha de ser expresado en rad/s² o m/s².

Parada rápida	Función	Descripción
	PCube_haltModule	Provoca una parada rápida del módulo especificado.
	PCube_haltAll	Provoca una parada rápida de todos los módulos.

Reinicio del módulo	Función	Descripción
	PCube_resetModule	Reinicia el módulo especificado.
	PCube_resetAll	Reinicia todos los módulos conectados al bus.

Movimiento en modo rampa con posición especificada	Función	Descripción
	PCube_movePos	Comienza un movimiento en modo rampa del módulo especificado a la posición deseada. La posición deseada está expresada en radianes o metros. La velocidad y aceleración deben de ser ajustadas las funciones de ajuste de velocidad de rampa y de aceleración: PCube_setRampVel y PCube_setRampAcc resp. PCube_setRampVelInc y PCube_setRampAccInc.
	PCube_movePosInc	Comienza un movimiento en modo rampa del módulo especificado a la posición deseada. La posición ha de ser expresada en incrementos.
	PCube_setMaxAccInc	Ajusta la aceleración máxima del módulo especificado. El valor ha de ser expresado en Incrementos/s².

Movimiento en modo rampa con posición, velocidad y aceleración especificada	Función	Descripción
	PCube_moveRamp	Comienza un movimiento en modo rampa del módulo especificado. La posición objetivo es dada en radianes o metros. La velocidad ha de ser dada en rad/s o m/s. La aceleración ha de ser dada en rad/s² o m/s².
	PCube_moveRampInc	Comienza un movimiento en modo rampa del módulo especificado. La posición es dada en incrementos, la velocidad en Incrementos/s y la aceleración en Incrementos/s².
	PCube_moveRampExtended	Comienza un movimiento en modo rampa del módulo especificado. La posición es dada en incrementos, la velocidad en incrementos/s y la aceleración en Incrementos/s². Devuelve el estado, posición actual y estado digital.

Movimiento de velocidad constante	Función	Descripción
	PCube_moveVel	Comienza un movimiento de velocidad constante, expresada en rad/s o m/s.
	PCube_moveVelInc	Comienza un movimiento de velocidad constante, expresada en incrementos/s.
	PCube_moveVelExtended	Comienza un movimiento de velocidad constante, expresada en rad/s o m/s. Devuelve el estado del módulo, posición actual y estado digital.

Movimiento con posición y tiempo específicos.	Función	Descripción
	PCube_moveStep	Comienza un movimiento a la posición deseada, especificada en radianes o metros y el tiempo en ms.
	PCube_moveStepInc	Comienza un movimiento a la posición deseada, especificada en incrementos y el tiempo en ms.
	PCube_moveStepExtended	Comienza un movimiento a la posición deseada, especificada en radianes o metros y el tiempo en ms. Devuelve el estado del módulo, posición actual y estado digital.