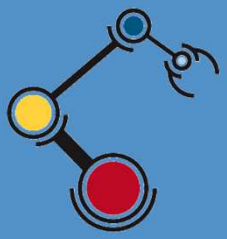
Escuela Politécnica Superior

Control de Fuerza de un Mitsubishi PA10 para almacenamiento de posiciones

Master Universitario en Automática y Robótica



Trabajo Fin de Máster

Autor:
Joaquín Berenguer Berenguer

Tutor/es:
Gabriel J. García Gómez

Julio 2014

Índice de Contenidos

I. Justificación y Objetivos

El objetivo de este proyecto es la programación de una aplicación que permita almacenar posiciones del extremo del robot Mitsubishi PA10 desplazando manualmente el extremo del robot. Para ello se hará uso del sensor de fuerza que tiene instalado el Mitsubishi PA10 en su extremo.

Se implementará un control de impedancia y se programará el interfaz que permita fácilmente almacenar la posición deseada.

Posteriormente se empleará el control de reproducción del PA10 para reproducir trayectorias enseñadas mediante este método.

Hasta ahora el PA10 del Laboratorio, cuando se quería mover el extremo del robot, se debía enviar el movimiento a realizar de forma articular o cartesiana, para cada movimiento de una trayectoria determinada, este sistema daba como resultado que el tiempo de prácticas o de proyecto, se perdía en realizar dichos movimientos, en vez del proyecto que se quería realizar.

Este proyecto pretende, disminuir de forma considerable el tiempo dedicado al posicionamiento del Robot, y dedicar el esfuerzo a lo que realmente se quiere realizar con el PA10.

En nuestro caso, utilizaremos el Sensor de Fuerza que tiene el Robot en su extremo para mover el extremo del Robot a una posición determinada, registrar la posición dentro de una trayectoria seleccionada previamente, y seguir moviendo el Robot, a posiciones diferentes dentro de la Trayectoria definida, de forma que una vez completadas todas la posiciones que queramos en cuestión de minutos, podamos reproducir la Trayectoria completa, definiendo entre una posición y la siguiente, la velocidad y fuerza a aplicar.

II. Agradecimientos

A mi tutor Gabriel J. García Gómez, por su apoyo en la dirección correcta y ánimo para conseguir el objetivo.

También a Santiago T. Puente Méndez, por su ayuda en la comprensión tanto del JR3 como del PA10.

III. Dedicatoria

A mis padres.

IV. Citas

Para quién no sabe dónde ir, cualquier viento es favorable. (Anónimo)

El único que no se equivoca, es el que no hace nada. (Aristóteles)

Aprende del ayer, vive el presente, y debes tener esperanza en el futuro. (Einstein)

V. Índice de Ilustraciones

1. Introducción

Este proyecto consiste en desarrollar una aplicación en VC++ 2010, usando MFC´s con un Menú, que permitirá cambiar la forma en la que se ha estado utilizando el PA10 hasta el momento, permitiendo crear posiciones de cualquier tarea que queramos realizar de forma amena y rápida, para poder dedicar la parte más importante del proyecto a lo que queramos realizar y no al posicionamiento del Robot.

Por otro lado, también se va a utilizar la combinación de posicionamiento del PA10 con la fuerza a aplicar por el extremo del Robot en la tarea que se quiera realizar, para demostrar este punto, se realizará después del posicionamiento, una aplicación práctica de Lijado.

El posicionamiento del extremo del Robot se realizará ejerciendo una pequeña fuerza sobre el extremo del Robot, esta fuerza será medida por el Sensor JR3, que nos enviará una matriz de 6 elementos (Fx, Fy, Fz, Mx, My, Mz), permitiendo interpretar a la aplicación desarrollada, el movimiento que queremos realizar. El PA10 para poder realizar el movimiento, se pondrá en modo Control de Velocidad, y a partir de aquí para que los movimientos sean estables se ha aplicado un Control de Impedancia, que impone un comportamiento dinámico deseado a la interacción entre el extremo del Robot y el entorno, siendo el entorno en este caso, la fuerza que aplicamos en el extremo del PA10.

El rendimiento deseado se especifica a través de la dinámica generalizada de la impedancia, simulando un conjunto completo de ecuaciones de masa-resorte-amortiguador.

Se ha implementado en C++, un PID que impone un comportamiento adecuado a cada aplicación, variando de forma flexible las constantes de Masa, Resorte y Amortiguación.

La forma de implementación, ya que tenemos 3 matrices de 6x6, ha sido imponer matrices diagonales donde las constantes de traslación son iguales en los tres ejes, y las constantes de Rotación son iguales en las tres orientaciones. Se ha creado una pantalla donde de forma fácil se pueden modificar dichas constantes, para cada tipo de aplicación.

1.1 Marco Teórico

El Control de Fuerza, es la base de este proyecto. Los Robots Industriales habitualmente siguen trayectorias sin tener en cuenta el control de fuerza, y este punto el que tiene más importancia en este proyecto.

Todo el sistema está basado en la fuerza aplicada en el extremo del PA10, hasta ahora el PA10, se manejaba desde programas que movían el extremo del Robot, desde programas que directamente mandaban al PA10 posiciones cartesianas o articulares. Que además de ser muy tediosas de programar, no tenían en cuenta la fuerza aplicada en la posición donde se colocaba el extremo del Robot.

En este proyecto se basa precisamente en estos dos puntos : Facilidad de uso en el movimiento del extremo del Robot, utilizando directamente nuestra mano para mover el extremo del robot, y una vez encontrada la posición donde queremos el extremo del robot, fijar la fuerza que debe aplicarse en cada posición.

El futuro de los Robots, pasará sin ninguna duda por el control de fuerza como parte básica de los robots, no se entenderá que un Robot no tenga en cuenta la fuerza aplicada en cada movimiento, ya sean estos Robots : Industriales o de Servicio.

1.2 Objetivos

Mover el Robot con la mano tanto para hacer una traslación, como solo para realizar la orientación del extremo del PA10. Aplicando muy poca fuerza, pero manteniendo la estabilidad del sistema mediante el Control de Impedancia implementado.

Mover el extremo del PA10, sería inútil, si no utilizamos una aplicación que nos facilite su uso, para ello se ha creado una aplicación con VC++ 2010 MFC con Menús, que facilita mucho la tarea que se está haciendo, de forma que cuando hemos movido el extremo del PA10, a una posición que nos interesa, pulsamos un botón de la pantalla correspondiente, e insertamos dicha posición con todas sus características, incluyendo un comentario de dicha posición, en una Base de Datos, llamada SQLITE que almacena dicha posición dentro de una trayectoria definida anteriormente, de forma que cada posición definida, forma parte de la trayectoria, así, la reproducción de la trayectoria será fácil de ejecutar posteriormente.

Cada Trayectoria y sus Posiciones, pueden verse, modificarse, y borrarse posteriormente de forma ágil, mediante las pantallas realizadas dentro de la aplicación, así mismo usando la herramienta sqlite3, puede manejarse la base datos, y usando SQL, gestionar la BD, en el apartado Modelo de Base de Datos, se podrá comprender como se puede realizar este punto.

Para realizar una aplicación práctica, una vez tengamos la aplicación de mover el extremo del robot, y hayamos insertado sus posiciones en la BD; modificaremos las posiciones, para realizar la aplicación de Lijado, de forma que además de moverse de una posición a otra, aplique una fuerza determinada, a una velocidad determinada.

Un problema importante encontrado, ha sido la compensación del Peso de la Herramienta, que en cada aplicación queramos utilizar. El Peso de la Herramienta genera una fuerza en el extremo del Robot, que depende de la posición del extremo, esta fuerza es medida por el Sensor de Fuerza, y enmascara la fuerza que aplicamos en el extremo del PA10 con la mano, con la fuerza aplicada por el Peso de la Herramienta, desglosar ambas fuerzas no ha sido tarea fácil, y se ha añadido un punto especial para explicar cómo se ha solventado este problema.

1.3 Metodología

Se ha implementado la aplicación, utilizando la Metodología en Espiral, me ha parecido que siendo el autor el único componente del equipo informático, y teniendo muchas dudas sobre la forma de implementación, ya que no disponía anteriormente de conocimientos, ni experiencia, sobre el Sensor de Fuerza, el PA10, el control de impedancia, los PID, etc… había mucha incertidumbre, como para usar una metodología diferente.

Tenía que realizar muchas pruebas antes de decidir por donde tenía que realizar la aplicación.

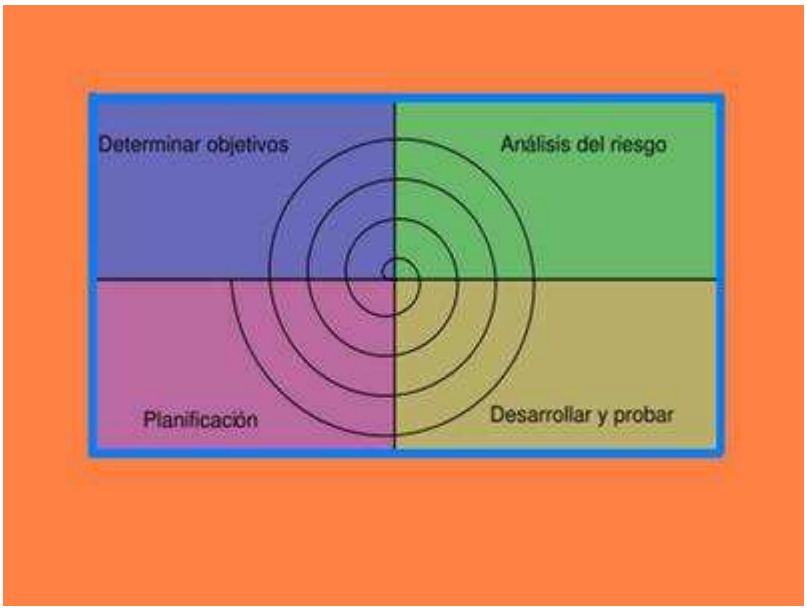


Ilustración 1-1Metodología en Espiral

Así que ha sido una manera continua de determinar objetivos, ver riesgos, desarrollar y probar, y volver a planificar, con los errores o impedimentos encontrados.

2. Fundamentos Técnicos

En este capítulo repasaremos las herramientas utilizadas para el desarrollo del proyecto, muy importante para poder pasar después en el siguiente capítulo, a desarrollar la aplicación.

Nos centraremos en la parte utilizada dentro del proyecto de las herramientas utilizadas, y no globalmente, como se utiliza dicha herramienta, para ello se propondrá diferente bibliografía, que permitirá profundizar en el uso de dichas herramientas para aplicaciones diferentes.

2.1 Sensor de Fuerza JR3

El Sensor de Fuerza de la empresa JR3, basado en placas PCI conectadas al bus del PC, utilizando procesadores de señales digitales (DSP), puede proporcionar datos desacoplados y filtrados a una frecuencia de 8KHz, por canal.

La placa contiene los circuitos necesarios para recibir la señal digital en serie en forma de flujo de datos desde el sensor que se encuentra en el extremo del PA10. Al mismo tiempo proporciona el voltaje y corriente necesaria por el Sensor de Fuerza desde la misma placa. Esto se realiza con un cable de 6 hilos, desde la placa PCI conectada al PC, hasta el extremo del PA10, la señal diferencial, permite distancias suficientes desde el PC hasta el extremo del Robot.

Con lo que recibimos exactamente el flujo de Fuerzas y Momentos, en una matriz de 6 elementos: Fx, Fy, Fz, Mx, My y Mz, desde el extremo del PA10.

Dentro de los diferentes modelos que la empresa JR3, proporciona, el que está instalado es el modelo: 67M25A3, cuya especificación más importante es su diámetro, ya que todos los modelos utilizan este parámetro como el principal, en este caso es de 67mm, el siguiente parámetro importante es su espesor, que es de 25mm. Dentro de los modelos 67M25A3, existen 4 submodelos que hacen referencia a la fuerza máxima que pueden soportar, en nuestro caso el sensor utilizado es de 200N, más que suficiente, porque el PA10, solo puede manejar 100N.
El Material de construcción es el Aluminio, con un peso de 175gr. Su precisión es del 1% para todos los ejes. La tabla que representa las características del sensor para los 6 elementos de la matriz son los siguientes:

	Fx	Fy	Fz	Mx	My	Mz
Rango de Medida (N)	+/-200	+/-200	+/-400	+/-12	+/-12	+/-12
Resolución Digital (N)	0,050	0,050	0,1	0,0032	0,0032	0,0032
Rigidez por Eje (N/m)	13 10^6	13 10^6	130.10^6	53000	53000	15000
Sobrecarga en un Eje (N)	930	930	3870	58	58	48
Sobrecarga Multi-Eje A (N)	1.200	1.200	3870	79	79	48
Sobrecarga Multi-Eje B (N)	970	970	3870	215	215	48

Ilustración 2Tabla de Características del JR3

Ilustración 3Sensor Fuerza JR3

Desde la Aplicación vemos el sensor desde un driver instalado en el PC, dentro del Bus PCI, que tenemos que enlazar al compilar nuestro programa.

Lo primeros que tenemos que hacer es inicializar la librería que nos ofrece el driver, de forma que seleccionemos el modelo de sensor y su situación dentro de la placa PCI, ya que puede contener varios sensores, como se puede observar en el siguiente dibujo :

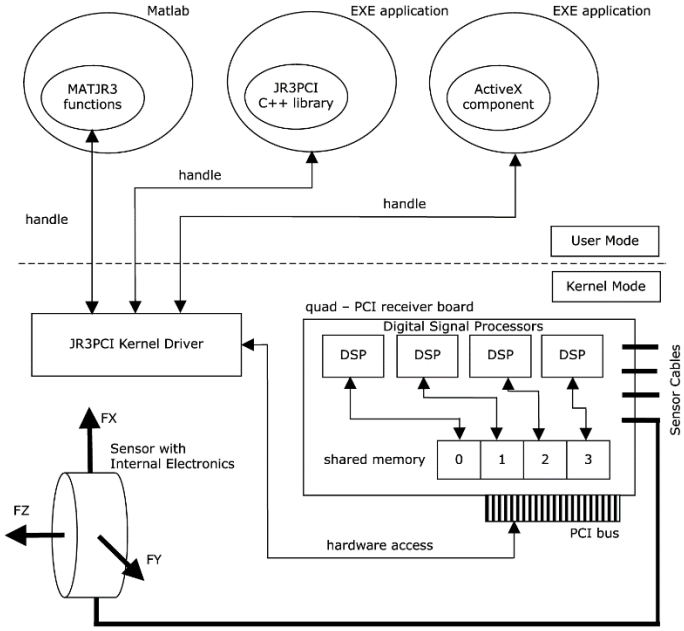


Ilustración 4-Driver JR3

Se ha hecho coincidir por Software, que los ejes y orientaciones coincidan con el PA10, para que cuando hagamos fuerza sobre el extremo, tengamos la sensación que empujamos en la dirección y sentido correctos.

También se establecido una forma por la que podemos indicar al software que queremos hacer una traslación o una orientación, y que habitualmente cuando estamos lejos del objetivo queremos realizar una traslación y cuando estamos cerca solo una orientación, esto podemos manejarlo empujando muy cerca del propio sensor, para realizar una

traslación, ya que en ese caso los momentos involucrados son muy bajos y podemos despreciarlos por software.

De la misma forma cuando realizamos una fuerza en el extremo de la herramienta, la parte más alejada del sensor, los momentos son relativamente mayores que las fuerzas, y despreciamos la fuerza, quedándonos solamente con los momentos leídos, así podemos controlar fácilmente los movimientos queremos realizar, esto se hace con la función, LeerSensorData, que nos devuelve una matriz llamada fuerzas[0], que después de haber inicializado el sensor con: InitLib, nos permite conocer las fuerzas y momentos aplicados, en Newton metro.

Esta función nos devuelve el valor en Nm, pero lo que leemos directamente del sensor es un valor digital entre 0 y 16384, esta función se convierte en Nm, de acuerdo con el máximo valor que soporta el Sensor utilizado, este valor se puede modificar desde las Constantes de la Aplicación, la función base utilizada de la librería del Sensor es:

```
read_ftdata(Filtro, cjr3.nump);
```

Esta función del driver del sensor, tiene implicaciones con la forma de funcionar del Sensor, como decíamos anteriormente el sensor puede leer a un máximo de 8KHz, pero los valores leídos, pueden tener distorsiones sino se filtran adecuadamente, por ello el propio Sensor proporciona filtros paso bajo, cada divisor por 4, de forma que disponemos de filtros que podemos utilizar para 2000Hz, 500Hz, 125Hz, 31,25Hz, 7,8125Hz, así el usuario puede utilizar el filtro que mejor se adapte a sus necesidades, en nuestro caso la parte crítica la tenemos en el máximo tiempo que el PA10, puede funcionar en modo Control de Velocidad, y como tenemos que realizar muchos cálculos entre la lectura del sensor y la siguiente vez que llamamos a la función del PA10 del control de velocidad, el compromiso se ha llegado en 32msg, que es la frecuencia de 31,25Hz, que nos devuelve una señal lo suficientemente estable desde el JR3, y es suficiente para la Aplicación, de control de fuerza.

El Segundo parámetro, es importante porque todas las funciones de lectura del Sensor, vienen referidas a él, y se trata del número de procesador, esto es así porque la placa DSP, puede tener varios procesadores, para varios sensores, en la placa utilizada, para el proyecto, tiene 2 procesadores, de los que usamos el segundo, identificado por 1.

2.2 PA10

Decir algo diferente sobre el PA10 a estas Alturas, después de tantos años de ser utilizado en el Laboratorio, por muchas personas, sería un poco insensato.

Pero si me gustaría además de resumir su funcionamiento, y utilización para el proyecto, la actualidad en la que se encuentra su funcionamiento.

Quizás la utilización en cuanto al Control de Velocidad y Control en Tiempo Real, es lo que hace diferente a este proyecto respecto del uso por otros proyectos, y por ello hablaremos un poco más de estos modos de funcionamiento.

En general debemos decir que es un Robot de 7 grados de libertad, y en la figura siguiente, puede apreciarse su estructura:

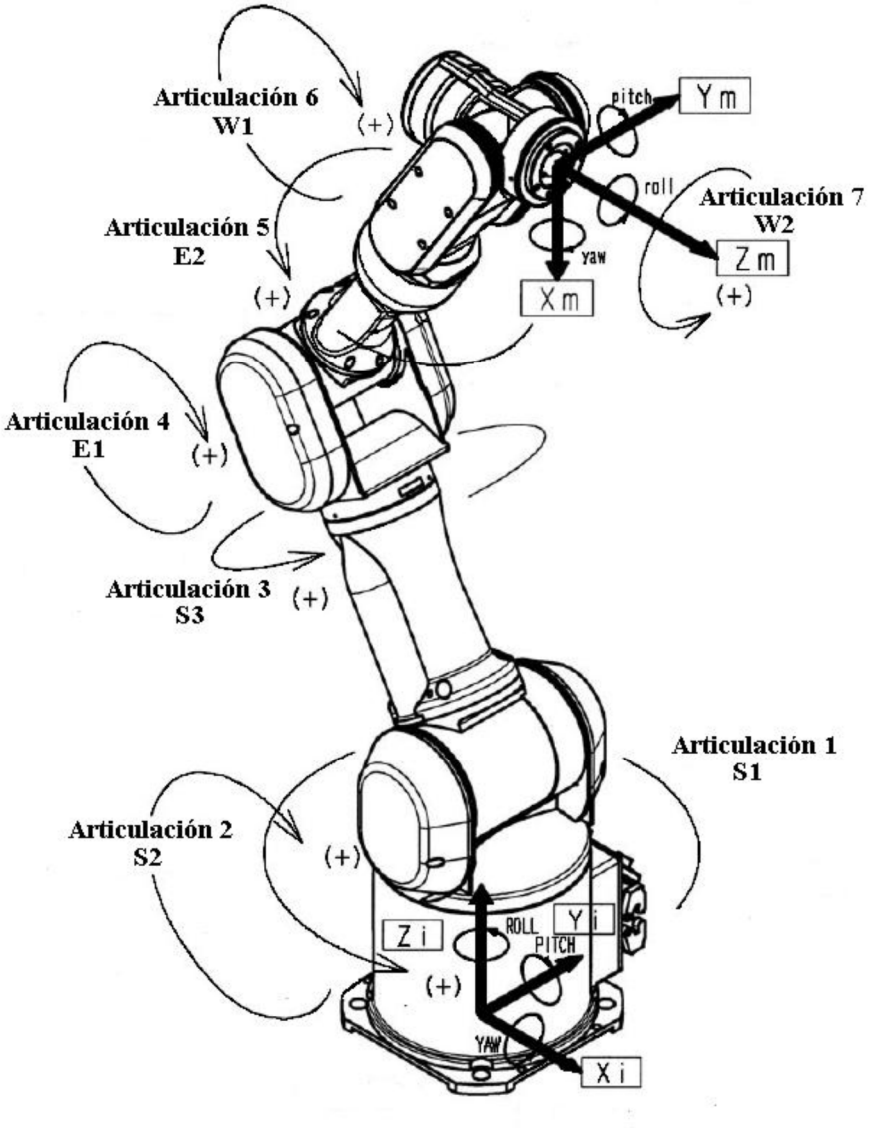


Ilustración 5- PA10

Para nuestro proyecto, el PA10 se va a utilizar dentro de un bucle con un Timer, mediante el modo de Control de Velocidad, este modo se caracteriza porque una vez inicializado el modo, en nuestro caso siempre nos basamos en posición y orientación del extremo del robot, identificado por xyzypr, en minúsculas, otro modo sería respecto de la base identificado por XYZYPR, y los intermedios solo posición o solo orientación, respecto al extremo o la base.

La razón de elegir con respecto al extremo, es porque tal como se demuestra en la sección de Control de Impedancia, las formulas cuando se aplican con respecto al extremo del Robot, son independientes de la estructura del Robot.

El modo de Control de Impedancia, obliga a refrescar la velocidad, cada 200msg, y por ello todo el proceso de cálculo, debe quedar dentro de dicho tiempo, el timer actualmente se ha fijado en 50msg, que en este momento es suficiente, pero que si hace falta extenderlo, es fácilmente ampliable.

El único caso que tenemos que tener en cuenta para no cumplir las especificaciones del Control de Velocidad, es la inserción en BD, de forma masiva, cuando queramos hacer esto utilizaremos una matriz en memoria donde vayamos insertando las posiciones articulares, para que una vez terminada la trayectoria se inserten todas las posiciones en la BD en una sola transacción.

Es interesante de todas formas tener la cinemática directa del PA10, para más claridad:

Los parámetros DH del PA10 son:

	Parte	**Link i**	**ai-1**	**αi-1**	**Di**	**θi**
Hombro	S1	1	0	0	d1	**θ1**
“	S2	2	0	-90º	0	**θ2**
“	S3	3	0	90º	d3	**θ3**
Codo	E1	4	0	-90º	0	**θ4**
“	E2	5	0	90º	d5	**θ5**
Muñeca	W1	6	0	-90º	0	**θ6**
“	W2	7	0	90º	d7	**θ7**

Como ya sabemos solo tenemos que aplicar la formulación:

$$^{i}T_{i+1} = Rot(z_{i-1}, \theta_i)Tras(z_{i-1}, d_i)Tras(x_i, a_i)Rot(x_i, \alpha_i)$$

$$^{i}\mathbf{T}_{\mathbf{i+1}} = \begin{pmatrix} cos\theta_i & -sen\theta_i cos\alpha_i & sen\theta_i sen\alpha_i & a_i cos\theta_i \\ sen\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sen\alpha_i & a_i sen\theta_i \\ 0 & sen\alpha_i & cos\alpha_i & d_i \end{pmatrix}$$

Ilustración 6Matriz Transformación de DH

Los valores Di, son: d1 = 317 mm., d3 = 450 mm., d5 = 480 mm., d7 = 70 mm.
Aplicando las transformaciones de DH, siguiendo el método descrito en la formula anterior tenemos que:

$$^{0}\mathbf{T}_{1} = \begin{pmatrix} cos\theta_1 & -sen\theta_1 & 0 & 0 \\ sen\theta_1 & cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad ^{1}\mathbf{T}_{2} = \begin{pmatrix} cos\theta_2 & 0 & -sen\theta_2 & 0 \\ sen\theta_2 & 0 & cos\theta_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^{2}\mathbf{T}_{3} = \begin{pmatrix} cos\theta_3 & 0 & sen\theta_3 & 0 \\ sen\theta_3 & 0 & -cos\theta_2 & 0 \\ 0 & 1 & 0 & d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad ^{3}\mathbf{T}_{4} = \begin{pmatrix} cos\theta_4 & 0 & -sen\theta_4 & 0 \\ sen\theta_4 & 0 & cos\theta_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^{4}\mathbf{T}_{5} = \begin{pmatrix} cos\theta_5 & 0 & sen\theta_5 & 0 \\ sen\theta_5 & 0 & -cos\theta_5 & 0 \\ 0 & 1 & 0 & d_5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad ^{5}\mathbf{T}_{6} = \begin{pmatrix} cos\theta_6 & 0 & -sen\theta_6 & 0 \\ sen\theta_6 & 0 & cos\theta_6 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^{6}\mathbf{T}_{7} = \begin{pmatrix} cos\theta_7 & 0 & sen\theta_7 & 0 \\ sen\theta_7 & 0 & -cos\theta_7 & 0 \\ 0 & 1 & 0 & d_7 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$^{0}\mathbf{T}_{7} = {}^{0}\mathbf{T}_{1}\,{}^{1}\mathbf{T}_{2}\,{}^{2}\mathbf{T}_{3}\,{}^{3}\mathbf{T}_{4}\,{}^{4}\mathbf{T}_{5}\,{}^{5}\mathbf{T}_{6}\,{}^{6}\mathbf{T}_{7}$$

Un punto viene dado por p = (x, y, z, α, β, γ)

x = (−(((c1c2c3 − s1s3)c4 − c1s2s4)c5 − (−(−c1c2s3 − s1c3))s5)s6 +
(−(c1c2c3 − s1s3)s4 − c1s2c4)c6)d7 + (−(c1c2c3 − s1s3)s4 − c1s2c4)d5 −
c1s2d3

y = (−(((s1c2c3 + c1s3)c4 − s1s2s4)c5 − (−(−s1c2s3 + c1c3))s5)s6 +
(−(s1c2c3 + c1s3)s4 − s1s2c4)c6)d7 + (−(s1c2c3 + c1s3)s4 − s1s2c4)d5 −
s1s2d3

z = (−((s2c3c4 + c2s4)c5 − s2s3s5)s6 + (−s2c3s4 + c2c4)c6)d7 +
(−s2c3s4 + c2c4)d5 + c2d3

α = atan2(b, a) ,
β = atan2(d, e) ,
γ = atan2 (−c, a / cos(α))

a = ((((c1c2c3 − s1s3)c4 − c1s2s4)c5 + (−c1c2s3 −s1c3)s5)c6 –

(−(−(c1c2c3 − s1s3)s4 − c1s2c4))s6)c7 +
(−((c1c2c3 − s1s3)c4 − c1s2s4)s5 + (−c1c2s3 − s1c3)c5)s7

b = ((((s1c2c3 + c1s3)c4 − s1s2s4)c5 + (−s1c2s3 + c1c3)s5)c6 –
(−(−(s1c2c3 + c1s3)s4 − s1s2c4))s6)c7 +
(−((s1c2c3 + c1s3)c4 − s1s2s4)s5 + (−s1c2s3 + c1c3)c5)s7

c = (((−s2c3c4 − c2s4)c5 + s2s3s5)c6 –
(−(s2c3s4−c2c4))s6)c7 + (−(−s2c3c4 − c2s4)s5 + s2s3c5)s7

d = −(((−s2c3c4 − c2s4)c5 + s2s3s5)c6 − (−(s2c3s4 − c2c4))s6)s7 +
(−(−s2c3c4 − c2s4)s5 + s2s3c5)c7

e = −(−((−s2c3c4 − c2s4)c5 + s2s3s5)s6 − (−(s2c3s4 − c2c4))c6)

Donde los términos ci y si corresponden a cos(qi) y sen(qi) con i = 1, 2, . . . , 7, respectivamente. O sea a las articulaciones q1, q2, ... , q7.

La relación entre las velocidades articulares $\dot{q}$ y las velocidades operacionales $\dot{x}$ está dada por el jacobiano analítico J(q) ∈ IR6×n, de modo que x’ = J(q) q’

La Cinemática Inversa de la velocidad es:

q’ = J(q)x’ + [I − J(q)’J(q)]z

J(q)’ -> Seudo Inversa Derecha del Jacobiano J(q)

Sin embargo, no disponemos de conocimiento técnico de los parámetros dinámicos de sus eslabones, ni de sus características no lineales de fricción en sus articulaciones, por todo ello hace que el desarrollo de un modelo dinámico preciso sea extremadamente difícil. Esto nos impide calcular, con respecto a la Base, los movimientos del extremo, y por ello hemos realizado, los cálculos con respecto al extremo.

En cuanto al desarrollo de la aplicación, el PA10, se controla con funciones, que residen en la librería papci.lib, y esta debe utilizarse obligatoriamente desde C, así que este es el lenguaje utilizado, lo mismo sucedía para el Sensor de Fuerza JR3, cuya librería se llama : jr3pci_lib.lib, que se incluye al linkar la aplicación. El include jr3pci_ft.h, debe ponerse para completar la estructura de la Aplicación.

Para utilización del PA10 y del Sensor de Fuerza JR3, debemos incluir los siguientes define:

#include <pa.h>
#include <paerr.h>
#include <jr3pci_ft.h>

Y linkar la siguientes librerías: papci.lib y jr3pci_ft.h

Las llamadas a cualquier función de estas librerías, se ha embebido dentro de las funciones de la aplicación para uniformidad con el resto de funciones, fácil

modificación e integración dentro del sistema de mensajes centralizados de la aplicación.

Tanto para el JR3 como para el PA10, se ha definido una clase, que permite utilizar sus funciones y sus variables desde cualquier parte de la aplicación, así mismo las funciones de la aplicación que llaman a funciones del JR3 o PA10, dejan sus resultados en variables específicas de la clases definidas, a continuación podemos ver la definición de las clases enunciadas:

```
class GestionPA10 {
public:
        bool simulacion;
        int Timer;
        ANGLE angulo1;
        ERR error;
        MATRIX matriz1;
        INT32 valor1, valor1Act, valor1Ant;
        REAL32 vectorP[3], vectorR[3], vectorPAct[3], vectorPAnt[3], vectorRAct[3];
        REAL32 vectorRAnt[3]; //vectores de posición y orientación
        REAL32 valorf1;
        REAL32 CVel[7], CVelAct[7], CVelAnt[7];
        REAL32 VelOne[7]; //para fijar las velocidades de cada articulación
        REAL32 VVelocidad[1]; //para velocidades solo de posición y rotacion
        int InitLib(int);
        int CerrarLib(int);
        int Home(void);
        int Escape(void);
        int Seguridad(void);
        int Velocidad(REAL32, int);
        int LecturaArticulaciones(void);
        int LecturaContadorInterno(void);
        int LecturaPosicionExtremo(void);
        int LecturaOrientacionExtremo(void);
        int LecturaPosOriExtremo(void);
        int ControlCartesianoAbs(int);
        int ControlCartesianoXYZ(REAL32 , REAL32 , REAL32 );
        int ControlCartesianoYPR(REAL32 , REAL32 , REAL32 );
        int ControlCartesianoxyz(REAL32 , REAL32 , REAL32 );
        int ControlCartesianoypr(REAL32 , REAL32 , REAL32 );
        int ControlArticular(UINT32 ejes);
        int LeeMCC(void);
        int LeeVersion(void);
        int Ocupado(void);
        int ControlVelocidadInit(void);
        int Rearme(void);
private:
        void Control_Errores(int , int );
};

class GestionJR3 {
```

```
public:
        short nump;
        bool principio;
        bool simulacion;
        double dsp_ver;
        short copyright;
        short countx;
        short offsets;
        short errorc;
        short softwarevn;
        short softwared;
        short softwarey;
        short serialn;
        short modeln;
        short units;
        short cald;
        short caly;
        short vecta;
        short thickness;
        short channels;
        short bits;
        short retjr3;
        short filtro;
        float fuerzas[6], fuerzasAnt[6], fuerzasAct[6], maxfuerzasNm[6];
        six_axis_array fescalaset, frecminescala, frecmaxescala, fdefaultescala,
                offset1, offset2;
        force_array fLectAct, fLectAnt;
        force_array fescalaget, fa1, fa2, fescala;
                //son 8 enteros : Fuerzas : fx, fy, fz, Momentos : mx, my,mz, y dos
                //magnitudes de vector de F y de M,
        // Magnitud de los Vectores : v1 (para fuerzas), v2 (para momentos)
        //warning_bits SaturacionJR3; //xx_near_sat, 6 valores fx, fy,fz, mx, my,
        //mz, que dice la posible saturación
        f_m_saturation fmsat;
        float escalaNm[6], medida1Nm[6], medida2Nm[6];
        force_sensor_data DatosJR3;
        void VerificarEstado(void);
        int LeerVariablesJR3(void);
        void LeerSensorData(short);
        void LeerFullScales(short);
        short SetFullScales(void);
        short Ocupado(void);
        void CerrarLib(void);
        short InitLib(short);
        short SetOffset(short);
        short LeerPicos(short);
        short LeerWarnings(void);
};
```

2.3.1 Desplazamiento del Extremo del Robot

Desplazar el extremo del robot entre la configuración A y la configuración B en un tiempo determinado. Se trabaja sobre el espacio articular. Problema a resolver: Que el extremo del robot se desplace entre dos puntos y que siga una determinada trayectoria. Tenemos una posición origen y una posición deseada:

$$(x_o, y_o, z_o, \alpha_o, \beta_o, \gamma_o)$$

$$(x_d, y_d, z_d, \alpha_d, \beta_d, \gamma_d)$$

Para ello tenemos que planificar la trayectoria e imponer el tiempo que debemos tardar en el desplazamiento. Las posiciones de las articulaciones son muy importantes, teniendo en cuenta la posición origen, y la destino, porque posiblemente pueda haber varias soluciones, o ninguna, y debamos decidir cuál es la mejor, teniendo en cuenta, la rapidez, comodidad de la postura del robot, para que las articulaciones sufran lo menos posible, al mismo tiempo que debemos decidir las velocidades en cada punto, por el que pasemos. Debemos conocer la trayectoria analítica en el espacio cartesiano, que es la evolución de cada coordenada cartesiana en función del tiempo:

$$x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t)$$

Para ello se muestrean los puntos por los pasaría la trayectoria para obtener la trayectoria definitiva. Se emplea a continuación la cinemática inversa, para obtener las posiciones articulares para cada punto por el que debamos pasar. Con ello se obtienen en función del tiempo las posiciones articulares:

$$q_1(t), q_2(t), q_3(t), q_4(t), q_5(t), q_6(t), q_7(t)$$

2.3 Control de Impedancia

2.3.2 Introducción al Control de Fuerza

Para describir la interacción entre el Robot PA10 y su entorno tenemos que representar las velocidades lineales y angulares instantáneas en el extremo del robot y las fuerzas y momentos instantáneas que actúan en el extremo del robot, que vienen dadas por las ecuaciones:

$$\xi = \begin{bmatrix} v^T \\ w^T \end{bmatrix}$$

$$F = \begin{bmatrix} f^T \\ n^T \end{bmatrix}$$

v^T son las velocidades lineales, W^T las velocidades angulares, F^T las fuerzas y n^T los momentos

Los vectores de los elementos son de 6 dimensiones cada uno de ellos, porque cada uno de ellos hace referencia a posición (x, y, z) y a orientación (α, β, γ).

Denominaremos Espacios de Movimiento y Fuerza, a ***M y F***

ξ se llama Twists (Giros) y **F** Wrenches (llave, herramienta, …). en cualquier caso son la velocidad y la fuerza.

Si ***M*** = {e1, e2, e3, e4, e5, e6, e7}

y ***F*** = {f1, f2, f3, f4, f5, f6}

Nos interesa que los vectores sean recíprocos:

$$\xi^T F = v^T f + w^T n = 0$$

Nos interesa que sean recíprocos porque el producto permanece invariable con respecto a un cambio lineal de la base de un sistema de coordenadas a otro.

2.3.3 Operador Impedancia

La impedancia Z(s) es el ratio de la transformada de Laplace entre la Fuerza y la Velocidad:

$$Z(s) = F(s) / V(s)$$

Si suponemos un sistema de masa-muelle-amortiguador descrito por la ecuación diferencial:

$$M\ddot{x} + B\dot{x} + Kx = F$$

Donde M es la Masa, B el Muelle y K el amortiguador

Tomando la transformada de Laplace

$$Z(s) = F(s)/V(s) = Ms + B + K/s = (Ms2 + Bs + K)/s$$

La Clasificación de Operadores Impedancia

El control de posición sería complicado en un entorno muy rígido como sucede en la limpieza de cristales de una ventana.

Similarmente el control de fuerza sería difícil en un entorno muy blando.

Para clasificar tomemos de nuevo Z(s)

Sistema Inercial si y solo si, |Z(0)| = 0

Sistema Resistivo si y solo si, |Z(0)| = B, para B entre 0 e infinito

Sistema Capacitivo si y solo si, |Z(0)| = infinito

Como las tareas a realizar, para control de fuerzas, son relativas al extremo, es lógico que el algoritmo de control se derive directamente de la tarea en el espacio (task space) que en el espacio de las articulaciones del robot.

Cuando el manipulador está en contacto con el entorno las ecuaciones de dinámica deben modificarse para incluir el momento de reacción: J Fe correspondiente a la fuerza del extremo, donde J es el Jacobiano del manipulador,

$$M(q)\ddot{q} + C(q,\dot{q}) + g(q) + J^T(q)F_e = u$$
$$u = M(q)a_q + C(q,\dot{q}) + g(q) + J^T(q)a^f$$

Donde aq y af son lazos externos de control con unidades de aceleración y fuerza. Utilizando las relaciones de joint space y task space:

$$\ddot{x} = J(q)\ddot{q} + \dot{J}(q)\dot{q}$$
$$a_x = J(q)a_q + \dot{J}(q)\dot{q}$$
$$\ddot{x} = a_x + W(q)(F_e - a_f)$$
$$W(q) = J(q)M^{-1}(q)J^T(q)$$
$$\ddot{x} = a_x$$

W(q) es el tensor de movilidad, conceptualmente tendremos ventaja, separando el control de posición y de fuerza, asumiendo que ax es una función solo de la posición y velocidad, y que af es solo función de la fuerza. Simplificando tendremos que af = Fe

Asumimos también que J(q) y W(q) son invertibles.

2.3.4 Control Impedancia Básico

Comenzando con el sistema uní-dimensional, del ejemplo de la figura siguiente:

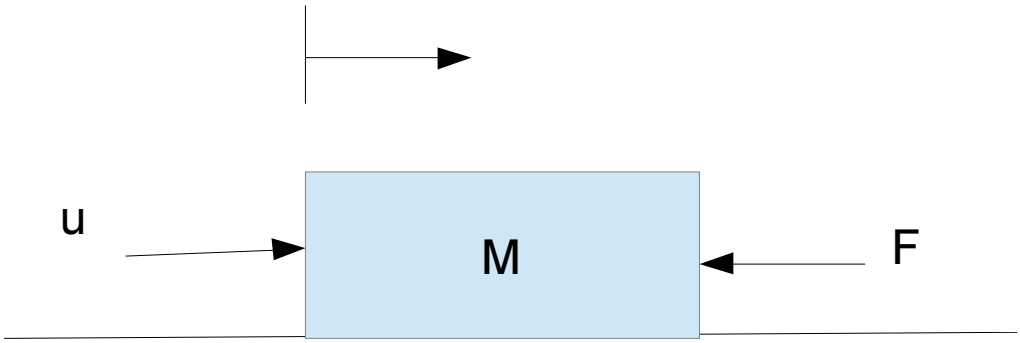


Ilustración 2-7Sistema Uní-dimensional

Su ecuación del movimiento es :

$$M\ddot{x} = u - F$$

Con u=0, el sistema aparece como pura inercia con masa M. Pero suponemos que se elige una fuerza de realimentación u = -mF. Entonces el sistema en bucle cerrado queda como sigue:

$$M\ddot{x} = u - F$$
$$\frac{M}{1+m}\ddot{x} = -F$$

Con lo que vemos que la fuerza de realimentación aparece como un cambio en la inercia aparente del sistema. La idea detrás del control de impedancia es regular la inercia, amortiguación y rigidez aparente del sistema, a través de la fuerza de sobrealimentación.

$$a_x = \ddot{x}^d - M_d^{-1}(B_d\dot{x}_e + K_d x_e + F)$$
$$M_d\ddot{x}_e + B_d\dot{x}_e + K_d x_e = -F$$
$$x_e(t) = x(t) - x^d(t)$$

F es la fuerza medida del entorno, del sensor de fuerza.

Md, Bd y Kd son matrices de 6x6 especificando respectivamente la inercia, amortiguación y rigidez deseadas. Xd es la trayectoria deseada y Xe es el error cometido en la trayectoria.

Cuando F=0, la trayectoria deseada coincide y el error es nulo.

Control de Híbrido de Impedancia

El control de impedancia es independiente de la dinámica del entorno. Es razonable pensar que incorporando el modelo de dinámica del entorno los resultados serán mejores. Por ejemplo, controlar la impedancia mientras simultáneamente regulamos la posición o la fuerza, lo que no es posible solamente controlando la impedancia. Ze Impedancia del entorno es fija y conocida Zr es la impedancia del robot y determinada por la impedancia de entrada. Con esto tenemos que:

1.- Si la impedancia del entorno Ze(s) es capacitiva, usaremos Norton (impedancia en paralelo), en caso contrario, usaremos Thevenin (impedancia en serie).

2.- Se elige una impedancia del robot Zr(s) y representarla como dual a la impedancia del entorno Ze. Esto es si Ze(s) era impedancia paralelo, Zr(s) será serie, y viceversa.

3.- Mezclar el robot y el entorno en un punto, y diseñar el bucle externo de control ax para alcanzar la impedancia deseada del robot mientras tracking (rastreo, …) una referencia de la posición o la fuerza.

Ejemplo de Entorno Capacitivo:

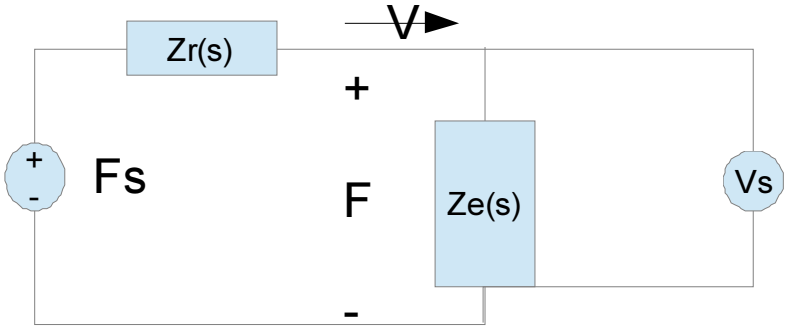


Ilustración 2-8Entorno Capacitivo

Si suponemos que Vs = 0, entonces suponemos que no hay ruido del entorno, que Fs representa, la fuerza de referencia:

$$F/Fs = Ze(s) / (Ze(s) + Zr(s))$$

La fuerza de error es a una Fs = Fd/s, es = -Zr(0)/(Zr(0) + Ze(0)) = 0

Como Ze(0) = infinito, porque es un entorno Capacitivo y Zr es distinto de 0, para cumplir con la regla establecida.

Como consecuencia, podemos tracear un valor fuerza de referencia constante, mientras simultáneamente tenemos un Zr para el robot.

Para ello tenemos que diseñar un bucle externo de control, usando realimentación posición, velocidad y fuerza. Lo que impone limitaciones a Zr. La inversa de Zr es de grado 1. El termino del bucle externo ax, los escogemos como:

$$Z_r(s) = M_c s + Z_{rem}(s)$$

$$a_x = \frac{-1}{M_c}\ddot{x} + \frac{1}{m_c}(F_s - F)$$

$$a_x = \ddot{x}$$

$$Z_r(s)\dot{x} = Fs - F$$

Por tanto en un entorno Capacitivo, podemos tener una fuerza de realimentación que puede ser usada para regular la fuerza de contacto y especificar la impedancia del robot deseada.

Ejemplo de Entorno Inercial

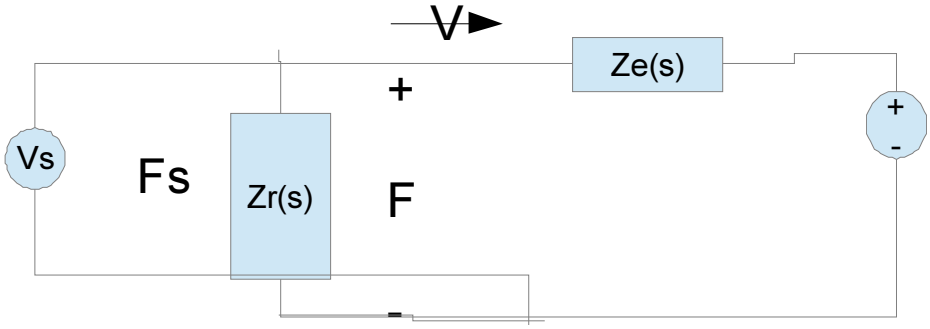


Ilustración 2-9Entorno Inercial

2.3.5 Control de Impedancia y Fuerza

Una de las características que define el control de fuerza, es la fuerza de contacto en el extremo del robot, que es donde está el sensor de fuerza, JR3. Que representa la capacidad de interacción entre el robot y su entorno. Control de Impedancia y Control Híbrido de Fuerza/Movimiento son los objetivos que perseguimos. Durante la interacción del manipulador con el entorno, este sufre impedimentos en el trazado de trayectorias, restringiendo el movimiento. Por ello debemos tener una idea lo más exacta posible de las **fuerzas de contacto** con las que el manipulador se encuentre.

Hay dos categorías, Control de fuerza indirecto utilizando Control de Impedancia, en el cual se consigue el control de fuerza vía el control de movimiento, sin un cierre explícito del bucle de control de fuerza. La segunda categoría es la de Control de Fuerza Directa, utilizando Control Híbrido de Fuerza/Movimiento.

1.- Control por Cumplimiento, Obediencia, Obligación (Compliance Control)

Consideramos un sistema donde tenemos un esquema de control de posición cuando aparecen fuerzas de contacto. Para ello utilizamos Esquemas de Control de Espacio

Operacional. Considerando el modelo dinámico del manipulador, puede escribirse como:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F\dot{q} + g(q) = u - J^T(q)h_e$$

Donde He es el vector de fuerzas de contacto ejercido por el extremo del robot en el entorno. Es razonable predecir que si He es distinto de 0, el esquema de control, basado en control de movimiento solamente fallará para llegar a la posición deseada Xd, por tanto si tenemos que

$$\tilde{x} = x_d - x_e$$
$$J_a^T(q)K_p\tilde{x} = J^T(q)h_e$$

Xe es la posición del extremo del manipulador, y la segunda ecuación sucede en el equilibrio, suponiendo un Jacobiano completo tenemos:

$$\tilde{x} = K_P^{-1}T_A^T(x_e)h_e = K_P^{-1}h_A$$

donde Ha es el vector equivalente de fuerzas.

De la formula anterior se verifica que en el equilibrio el manipulador bajo acción de un control de posición, se comporta como un amortiguador generalizado en el espacio operacional cumpliendo KP-1 respecto de la fuerza equivalente Ha.

Si suponemos que Kp es una matriz diagonal, y que tenemos la matriz de Transformación del manipulador. Vemos que no depende la configuración del manipulador, si no que depende de la orientación del extremo del robot en el momento de la interacción a través de la matriz de transformación T. Escribiendo la formula anterior, de la siguiente forma:

$$h_A = K_P\tilde{x}$$

donde Kp representa la matriz de rigidez, de acuerdo con el vector de fuerzas equivalente Ha. Esto se denomina cumplimiento activo.

Consideramos ahora la interacción del manipulador bajo la acción del control dinámico inverso en el espacio operacional.

$$\ddot{q} = y - B^{-1}(q)J^T(q)h_e$$
$$y = J_A^{-1}(q)M_d{-}1(M_d\ddot{x}_d n + K_D\dot{\tilde{x}} + K_P\tilde{x} - M_d\dot{J}_A(q,\dot{q})\dot{q})$$
$$M_d\ddot{\tilde{x}} + K_D\dot{\tilde{x}} + K_P\tilde{x} = M_dB_A^{-1}h_A$$
$$B_A(q) = J_A^{-T}(q)B(q)J_A^{-1}(q)$$

Ba(q) es la matriz de inercia del manipulador en el espacio operacional,

La tercera expresión anterior establece la relación a través de la impedancia mecánica generalizada entre el vector de fuerzas y el vector de desplazamientos en el espacio operacional.

Esta impedancia puede atribuirse a un sistema mecánico caracterizado por una matriz de masas Md, una matriz de amortiguación Kd y una matriz de rigidez Kp, que pueden usarse para especificar el comportamiento dinámico a lo largo de las direcciones espacio operacional.

La presencia de Ba hace que el sistema este acoplado.

Si queremos mantener la linealidad y el desacoplamiento durante la interacción con el entorno, es necesario medir la fuerza de contacto.

Y si suponemos esto, y no tenemos error en la medida de la fuerza de contacto:

$$M_d\ddot{\tilde{x}} + K_D\dot{\tilde{x}} + K_P\tilde{x} = h_A$$

La impedancia depende de la orientación del extremo del manipulador a través de la matriz de transformación T. Lo que hace que la selección de los parámetros de la matriz sea difícil.

Para evitar esto, es suficiente rediseñar el control de la entrada '**y**' como una función de error del espacio operacional.

$$\tilde{x} = -\begin{bmatrix} o_{d,e}^d \\ \phi_{d,e} \end{bmatrix}$$

$$\dot{\tilde{x}} = -J_{Ad}(q,\tilde{x})\dot{q} + b(\tilde{x}, R_d, \dot{o_d}, w_d)$$

$$b(\tilde{x}, R_d, \dot{o_d}, w_d) = \begin{bmatrix} R_d^T\dot{o_d} + S(w_d^d)o_{d,e}^d \\ T^{-1}(\phi_{d,e})w_d^d \end{bmatrix}$$

$$\ddot{\tilde{x}} = -J_{Ad}\ddot{q} - \dot{J_A}d + \dot{b}$$

$$M_d\ddot{\tilde{x}} + K_D\dot{\tilde{x}} + K_P\tilde{x} = h_e^d$$

Donde todos los vectores están referidos al sistema de coordenadas deseado. Representa una impedancia lineal en cuanto a la fuerza del vector Hed, **y es independiente de la configuración del manipulador.**

El esquema de bloques que representa el control de impedancia es:

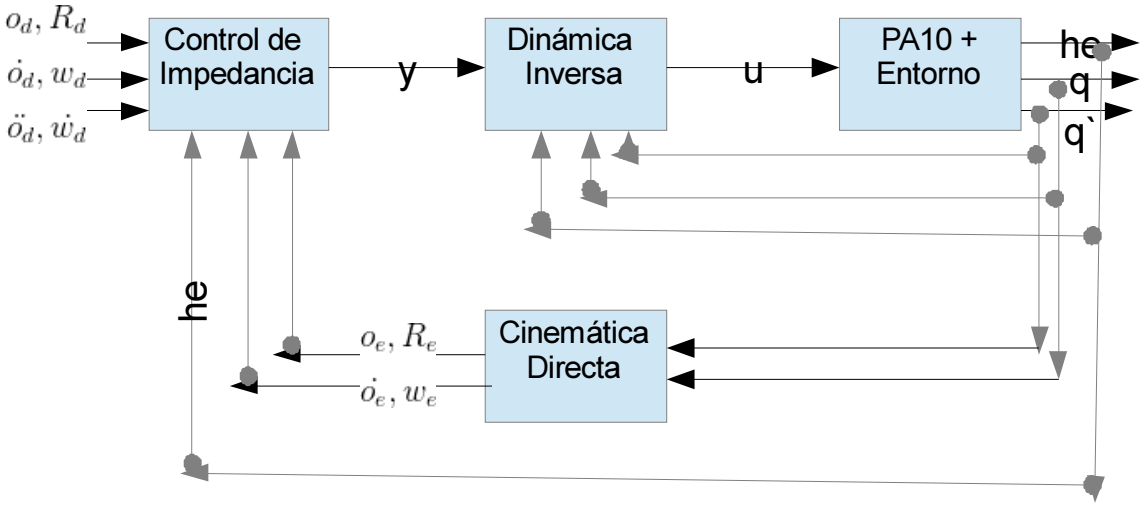
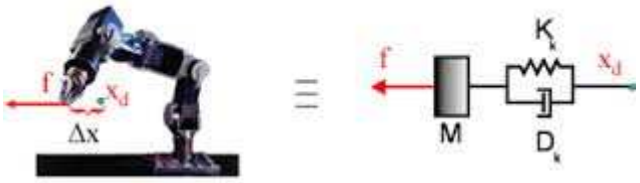


Ilustración 2-10Control de Impedancia

Algunos ejemplos de simulación se pueden apreciar en la figuras siguientes :



$$f = M\Delta\ddot{x} + D_k\Delta\dot{x} + K_k\Delta x$$

Ilustración 2-11Sistema Masa-Resorte-Amortiguación

2.4 PID

Como continuación a la parte teórica de Control de Impedancia, y teniendo como resultado un sistema de masa-resorte-amortiguador. Se ha implementado un PID, que resuelva el problema del control de impedancia, la implementación se basa en las constantes definidas en el punto anterior, y que se implementan en la Base de Datos, dentro de la tabla de Constantes, para poder ser utilizado en cualquier circunstancia.

Las constantes definidas son: Inercia, Amortiguación y Rigidez, tanto para la posición como para la orientación, obteniendo una matriz diagonal de 6 elementos.

En cada bucle, del timer, se recalcula, el PID para adaptarse a los transitorios generados.

La implementación en C++, para la parte Proporcional, Integral y Derivativa.

Para cada elemento de la matriz diagonal, es decir se hace 6 veces, 3 para posición y 3 para orientación se realiza lo siguiente:

```
mp10.vectorPAnt[i1] = mp10.vectorPAct[i1];
mp10.vectorPAct[i1] = mp10.vectorP[i1];
error[i1] = setpoint[i1] - mp10.vectorPAct[i1] ;
errorsum[i1] += error[i1];
derror[i1] = mp10.vectorPAct[i1] - mp10.vectorPAnt[i1] ;
proporcional[i1] = Kp[i1] * error[i1];
derivativa[i1] = Kd[i1] * derror[i1];
integral[i1] = Ki[i1] * errorsum[i1];
salida[i1] = proporcional[i1] + derivativa[i1] + integral[i1];
setpoint[i1] = salida[i1];
```

Se almacenan los valores anteriores, para su uso posterior.

El Error de la parte Proporcional del Control, es la diferencia valor donde queremos ir (setpoint), con el lugar donde estamos actualmente, leído del extremo del PA10.

El Error Derivativo, es la diferencia entre el valor actual leído del extremo del PA10, con el valor anterior, es decir de la posición donde nos encontrábamos en el bucle anterior del Timer.

El Error Integral, es el sumatorio, de todos los errores Proporcionales descritos anteriormente.

El Error del PID, es la suma de los tres errores anteriores. Y se modifica la salida para que sea el siguiente lugar donde queremos ir, que se modificará en el bucle, con el incremento del desplazamiento.

2.5 Software

Este nuevo apartado, describe el software utilizado, es importante destacar que el software utilizado, es software disponible en el laboratorio, o de código abierto como la Base de Datos utilizada, por lo que cualquier modificación puede realizarse sin ningún problema.

El software utilizado, intensivamente ha sido Visual C++ 2010, que es el que tenemos disponible en el Laboratorio.

Las librerías utilizadas, son las del JR3 y del PA10, disponibles en el Laboratorio.

Para el Desarrollo se ha utilizado la Base de Datos, SQLITE, porque es gratuita, es fácil de mover ya que solo necesita un fichero, y aunque para inserciones puede ser lenta, en lecturas, es bastante rápida, que lo que nos interesa para el proyecto.

2.5.1 VC++

Se ha utilizado Visual C++ 2010, de Microsoft, de forma estándar.

Además se ha utilizado MFC´s, con implementación de un Menú, para que sea más fácil la utilización de la Aplicación.

Inicialmente, se intentó utilizar Visual C++ con CLI, para que fuera una versión más moderna, con widgets mejorados y mejores. Pero después de realizar parte de la Aplicación, las librerías del JR3, no compilaban, con lo que se optó por la solución actual de MFC´s con Menú.

2.5.2 MFC

La utilización de MFC´s con Menú, permite una implementación, estructurada, y fácilmente entendible por un desarrollador que quiera modificar la aplicación, ya que cada función se ha implementado como una Pantalla nueva, que realiza dicha función.

Las Pantallas en la mayoría de las ocasiones, acceden a la Base de Datos, y por ello se ha implementado unas funciones específicas para cada Tabla, de la BD, dichas funciones se han implementado de forma general fuera de la Aplicación MFC, y dentro de una Librería.

Así mismo en dicha Librería, también se encuentra toda la Gestión del PA10 y del JR3.

2.5.3 Base de Datos SQLITE

La Base de Datos, utilizada es SQLITE (http://www.sqlite.org), en su página web, se puede encontrar todo lo necesario para su utilización desde C++, como desde otros lenguajes, como Python, que también ha sido utilizado para la realización de Listados, en el Proyecto.

También se puede utilizar una herramienta sqlite3, que permite utilizar la BD; directamente desde un terminal de Windows.

La BD; también se puede utilizar en Linux, siendo completamente transparente su uso.

SQLITE, está contenida, en un solo fichero, que para este proyecto se llama: Jr3Pa10.db, se abre dicho fichero dentro de la aplicación con la llamada:

```
sqlite3 *db;

rc = sqlite3_open("../../FichBase/Jr3Pa10.db", &db);
```

No necesita configuración, directamente utilizable con sentencias SQL, una creada la BD, esta se crea simplemente con: sqlite3 Jr3Pa10.db, que crea un fichero vacío.

Es una BD transaccional de acuerdo con los estándares SQL.

Es la más utilizada en el mundo, y sus fuentes son de dominio público.

Se puede descargar desde: http://www.sqlite.org/download.html

Y su documentación desde: http://www.sqlite.org/docs.html

Se puede verificar su licencia de uso desde: http://www.sqlite.org/copyright.html

Por la importancia que ha tenido para el Proyecto el API de SQLITE para C++, vamos a realizar una breve descripción, del uso de la BD, desde Visual C++ 2010.

Inicialmente necesitamos conocer dos objetos importantes, que son la realización de la conexión, esto se hace al arrancar la aplicación, igual que se hace con las librerías del JR3 y del PA10, y se mantiene abierta hasta que cerramos la aplicación, igual que se hace con las librerías del JR3 y del PA10.

Cada vez que queremos enviar una sentencia SQL a la BD, tenemos que preparar la sentencia mediante el objeto:

```
int sqlite3_prepare_v2(
  sqlite3 *db,            /* Database handle */
  const char *zSql,       /* SQL statement, UTF-8 encoded */
  int nByte,              /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const char **pzTail     /* OUT: Pointer to unused portion of zSql */
```

Donde lo más destacable es la sentencia SQL que se le pasa como parámetro en la variable zSql.

Esta función crea el objeto que permitirá manejar los datos que la BD, nos devuelva, o queramos enviar a la BD.

Una vez hemos creado el objeto, podremos hacer un bind o enlace de las variables de C++, para ser usadas dentro de la aplicación.

A partir de aquí, ya podemos leer de la BD, con la sentencia: sqlite3_step, que nos devuelve la siguiente fila en el caso de una select, o ejecuta directamente un insert, update o delete, que le hayamos pasado al prepare en zSql.

Para terminar el uso de la base de datos, dentro del programa C++,

Hace falta describir como recibimos una fila desde la base de datos, cuando hacemos una select y como enviarnos a la base de datos, datos de la aplicación cuando hacemos cualquier llamada a sqlite.

Para ello nos basamos en la definición de una clase de una tabla cualquier de la base de datos:

```
class Trayectoria {
public:
        int puntero_lista;
        int lista_codigos[100];
        int codigo;
        int Herramienta;
        int TrayectoriaSesion;
        int TReal;
        BOOL Rep_Tiempo_Real;
        const unsigned char *descripcion;
        const unsigned char *fecha;
        const unsigned char *alumnos;
        char fec1[20];
        char desctra[100];
        char alumc[100];
        int LeerTrayectoria(int Codigo);
        int BuscarTrayectoria(int);
        int InsertaTrayectoria(void);
        int ActTrayectoria(void);
public:
```

```
        void RegistroTrayectoria(sqlite3_stmt *cursor1){
                codigo = sqlite3_column_int(cursor1, 0);
                descripcion = sqlite3_column_text(cursor1, 1);
                fecha = sqlite3_column_text(cursor1, 2);
                alumnos = sqlite3_column_text(cursor1, 3);
                Herramienta = sqlite3_column_int(cursor1, 4);
                TReal = sqlite3_column_int(cursor1, 5);
                ConvTexto();
        }
public:
        void ConvTexto(void){
                sprintf_s(fec1, "%s", fecha);
                sprintf_s(desctra, "%s", descripcion);
                sprintf_s(alumc, "%s", alumnos);
        }
public:
        void TrayectoBind(sqlite3_stmt *cursor1){
                sqlite3_bind_text(cursor1, 1, (char *) desctra, -1,  SQLITE_STATIC);
                sqlite3_bind_text(cursor1, 2, (char *) fec1, -1,  SQLITE_STATIC);
                sqlite3_bind_text(cursor1, 3, (char *) alumc, -1,  SQLITE_STATIC);
                sqlite3_bind_int(cursor1, 4, Herramienta);
                sqlite3_bind_int(cursor1, 5, TReal);
                sqlite3_bind_int(cursor1, 6, codigo);

        }
};
```

Como vemos en esta definición de clase, utilizamos la sentencia: sqlite3_bind_... para pasarle datos de la aplicación a la BD, y poder hacer un insert, update,…

Pero para cada tipo de dato usaremos una sentencia diferente así: sqlite3_bind_text, no sirve para tipos char y date.

Sqlite3_bind_int, lo usaremos para tipos int, y sqlite3_bind_double para float y double.

Para convertir datos que vienen de la BD en variables de la aplicación usaremos las funciones: sqlite3_column_text, para pasar variables char y date, sqlite3_column_int para variables de tipo entero.

Para cada tabla de la BD, se ha definido una clase, de forma que cuando queremos acceder a una determinada tabla, solo utilizamos sentencias SQL, que nos dejan o enviamos desde o hacia variables de la aplicación.

2.6 Peso de la Herramienta

El Peso de la Herramienta, es un punto importante a tener en cuenta, ya que el peso en vacío de la Herramienta, afecta al movimiento que queremos realizar con la mano, ya que ejerce una fuerza sobre el sensor JR3, que hay que compensar conociendo las características de la herramienta.

La solución implementada, ha sido desarrollada creando una tabla en la BD, que almacena para cada Herramienta que se vaya a usar el peso de dicha herramienta para un determinado valor de las articulaciones, almacenando suficientes valores, usando la aplicación, donde se ha implementado una opción llamada definición de Herramienta, se puede obtener un buen número de valores de forma automática, ya que mediante la opción de creación de puntos, el PA10, se va movimiento por los diferentes puntos establecidos y almacenando las posiciones articulares y los valores de la fuerza en vacío o Peso de la herramienta.

Cuando se realice el movimiento real, y se haga fuerza con la mano, se compensará con los valores almacenados, realizando una interpolación, ya que el valor buscado seguramente no estará en los almacenados.

Se ha dado mucha más importancia a las articulaciones: s2, e1 y w1 que son las que más afectan al peso de la herramienta, en nuestro caso además w2 es fijo que por el diseño de la Herramienta de Lijado, se ha fijado en 42 grados, para que la Herramienta diseñada tenga la orientación que nos interesa para realizar el Lijado.

2.7 Diseño Herramienta de Lijado

Para el diseño de la Herramienta de Lijado, se ha utilizado OpenScad, una aplicación Open Source, que funciona tanto en Windows como en Linux, y que puede descargarse desde: http://www.openscad.org.

El diseño se realiza programando en Openscad, y el programa que se ha realizado es el siguiente:

```
//herramienta de Lijado, para poner en el extremo del PA10
ancho = 80;
largo = 110;
alto = 12;
radio_cilindro = 26;
altura_cilindro = 2.5;
ancho_lija = 70;
largo_lija = 100;
tornillo_herramienta = 4.5;
radio_tornillo = 20;
Profundidad_Lija = 6;
alto_tuerca = Profundidad_Lija + 3;
posicion_tornillos = [[radio_tornillo, 0, 0], [0, radio_tornillo, 0],
                [-radio_tornillo, 0, 0], [0, -radio_tornillo, 0]];
posicion_tuercas = [[radio_tornillo, 0, alto-alto_tuerca],
                        [0, radio_tornillo, alto-alto_tuerca],
                        [-radio_tornillo, 0, alto-alto_tuerca],
                        [0, -radio_tornillo, alto-alto_tuerca]];
difference() {
        translate([-ancho/2,-largo/2,0]) {
                color([1,0,0]) cube([ancho,largo,alto]); //cubo externo
        }
        translate([-ancho_lija/2, -largo_lija/2, alto-Profundidad_Lija]) {
                cube([ancho_lija, largo_lija, Profundidad_Lija]); //cubo tamaño de la lija
        }
        for(i1=posicion_tornillos) { //se ponen los tornillos, medidas herramienta PA10
                translate(i1) {
                        cylinder(h=ancho, r=tornillo_herramienta/2);
                }
        }
        for(i1=posicion_tuercas) { //se ponen las tuercas, medidas herramienta PA10
                translate(i1) {
                cylinder(h=altura_cilindro*2, r=tornillo_herramienta);
                }
        }
}
```

A partir de este fuente se ha exportado a formato STL con el propio OpenScad, obteniendo el siguiente, dibujo:

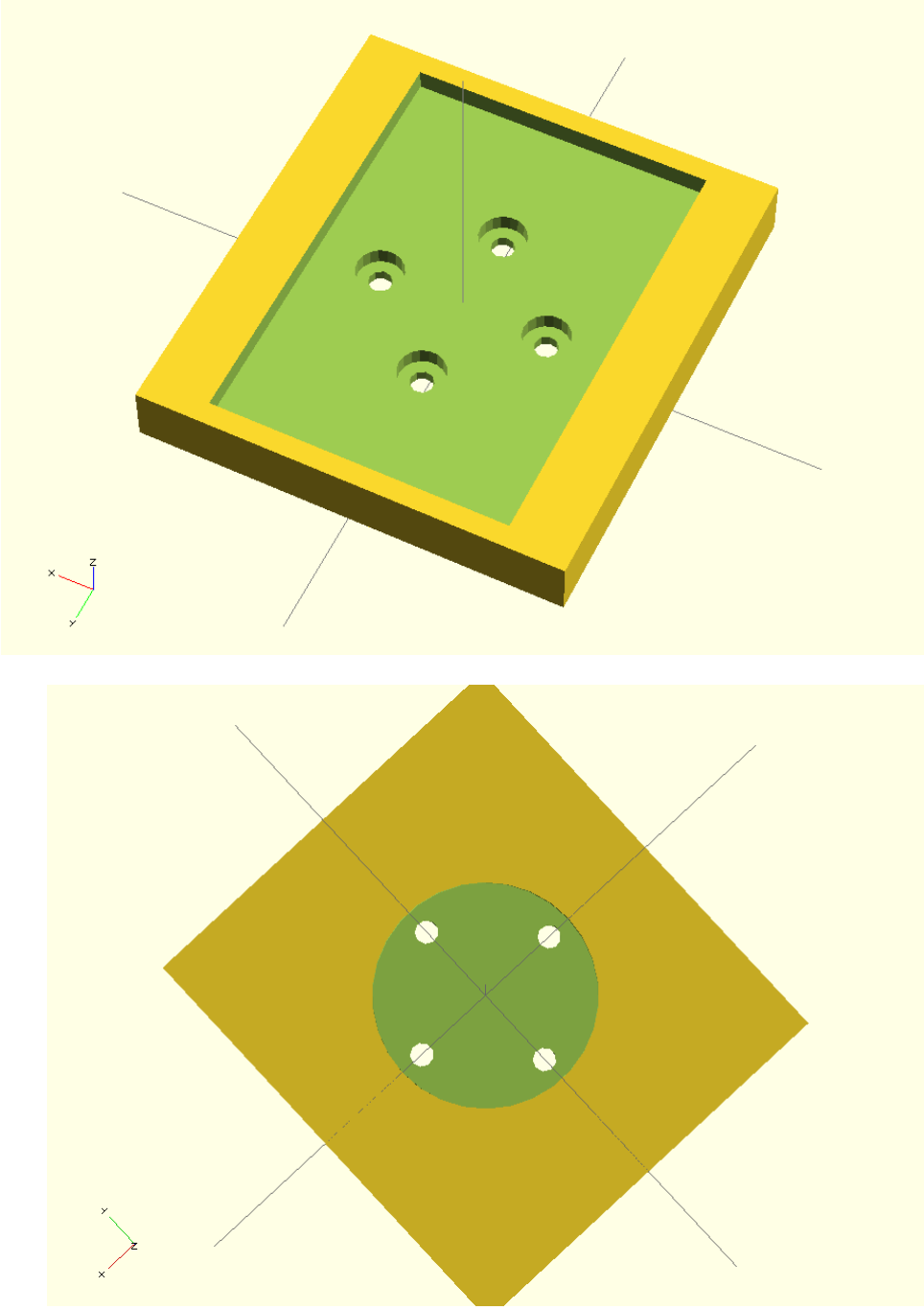


Ilustración 12Herramienta Lijado formato STL

El diseño en formato STL, se ha utilizado para abrirlo con la Aplicación OpenSource: Slic3r, que nos permite generar a partir del formato STL, las sentencias GCode necesarias para enviar a la Impresora 3D, y obtener la Herramienta, que se ha puesto en el extremo del PA10, usando la herramienta de

enganche de Herramientas, propia del PA10, en la figura siguiente puede apreciarse el resultado final.

Ilustración 13Foto Herramienta de Lijado

Se acopla perfectamente a las herramientas utilizadas con el PA10, ya que se ha diseñado expresamente para lo existente.

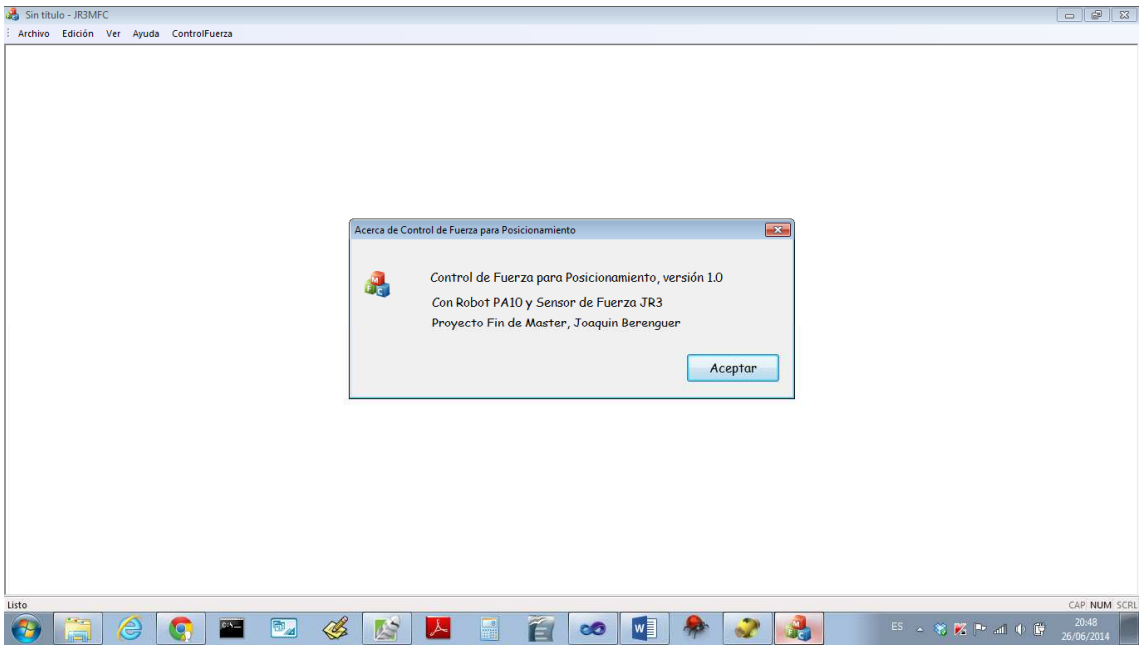
3.- Aplicación

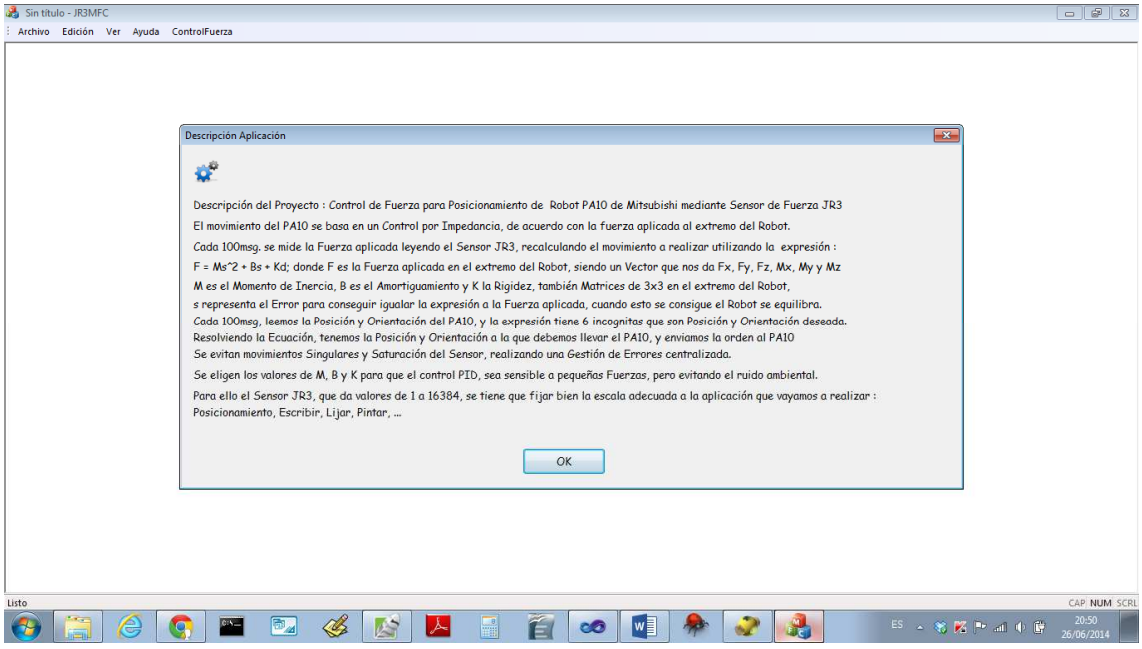
En este apartado vamos a resumir la aplicación desarrollada usando Visual C++,

El esquema de la aplicación se ha realizado mediante openmind.

Cuando se arranca la aplicación entramos en un menú, con las opciones necesarias para el manejo de toda la aplicación, es un menú típico realizado usando las MFC´s, por lo que su manejo es intuitivo, y muy conocido por cualquier usuario, por usarse de forma parecida a cualquier otra aplicación de Windows.

El aspecto que tiene es el siguiente:





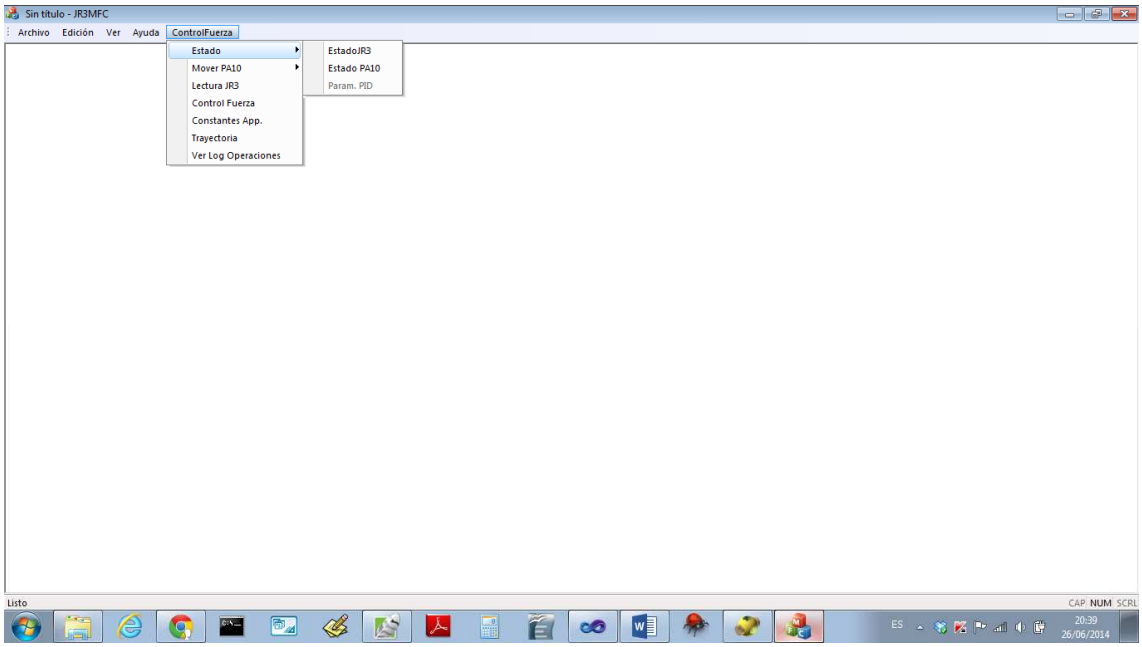


Ilustración 14Menú Aplicación

A continuación iremos describiendo las diferentes opciones de la aplicación y el esquema de la BD, utilizado, ya que sin el esquema de la BD, no sería posible entender el funcionamiento de la Aplicación

3.1 Esquema

3.2 Modelo de Base de Datos

La Herramienta para modelar la BD, se ha realizado con diaw.exe, que se puede descargar gratuitamente. Se adjunta junto con la aplicación el fichero generado, desde diaw, se ha exportado el siguiente diagrama:

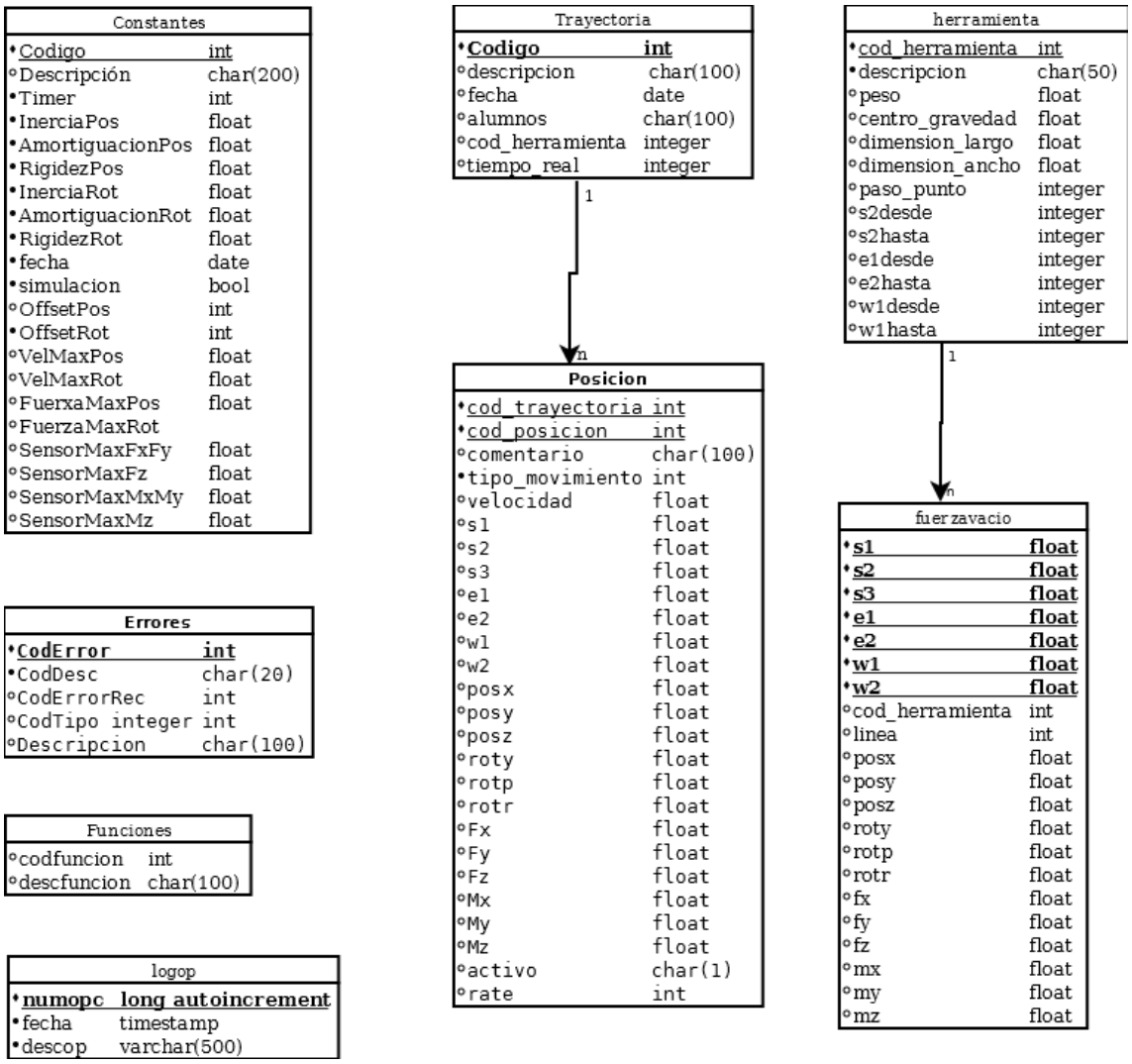


Ilustración 15Modelo de Datos

El modelo de datos en formato SQL, es el siguiente, pero también se adjunta como fichero de la memoria:

CREATE TABLE logop (numopc integer primary key autoincrement, descop varchar(500), fecha timestamp);

CREATE TABLE tabind (nomind varchar (100),tablaid integer, colid integer, posind integer, primary key (tablaid, colid, posind), foreign key (tablaid, colid) references columnas (tablaid, colid));

CREATE TABLE LinMedidasJR3 (CodMedida int, LinMedida int, codEscala int, Fx float, Fy float, Fz float,

Mx float, My float, Mz float, V1 float, V2 float, Sesion int, primary key (CodMedida, LinMedida));

CREATE TABLE reftab (tablaid integer, tablaidref integer, colid integer, colidref integer, posref integer, primary key (tablaid, tablaidref, posref), foreign key (tablaid, colid) references columnas (tablaid, colid));

CREATE TABLE herramienta (cod_herramienta integer, descripcion char(50), peso float, centro_gravedad float, dimension_largo float, dimension_ancho float, paso_punto integer, s2desde integer, s2hasta integer, s3desde integer, s3hasta integer, e1desde integer, e1hasta integer, e2desde integer, e2hasta integer, w1desde integer, w1hasta integer, w2desde integer, w2hasta integer, primary key(cod_herramienta));

CREATE TABLE messages (num_mensaje int, desc_mensaje1 varchar (100) , desc_mensaje2 varchar (100) , desc_mensaje3 varchar (100) , desc_mensaje4 varchar (100) , desc_mensaje5 varchar (100) , primary key (num_mensaje));

CREATE TABLE Errores (CodError integer, CodDesc char(20), CodErrorRec integer, CodTipo integer, Descripcion char(100), primary key (CodErrorRec));

CREATE TABLE fuerzavacio (s1 float, s2 float, s3 float, e1 float, e2 float, w1 float, w2 float, punto char(60), cod_herramienta integer, linea integer,

posx float, posy float, posz float, roty float, rotp float, rotr float,

fx float, fy float, fz float, mx float, my float, mz float, primary key(s1, s2, s3, e1, e2, w1, w2));

CREATE TABLE funciones (codfuncion integer, descfuncion char(100), primary key (codfuncion));

CREATE TABLE posicion (cod_trayectoria integer, cod_posicion integer, tipo_movimiento integer, velocidad float, s1 float, s2 float s3 float,

e1 float, e2 float, w1 float, w2 float, posx float, posy float, posz float, roty float, rotp float, rotr float,

Fx float, Fy float, Fz float, Mx float, My float, Mz float, comentario varchar(100), activo char(1), s3 float, rate int,

primary key (cod_trayectoria, cod_posicion), foreign key (cod_trayectoria) references trayectoria (codigo));

CREATE TABLE Aplicacion (Codigo integer, Timer integer, Inercia float, Amortiguacion float, Rigidez float, fecha1 date, fecha2 date, Descripcion char(200), CoefAmortiguacion float, TEstablcimiento float, FrecNatural float, ParteReal float, ParteImaginaria float, simulacion integer, OffsetPos int, OffsetRot int, InerciaR float, AmortiguacionR float, RigidezR float, VelMaxP float, VelMaxO float, FuerzaMaxP float, FuerzaMaxO float, SensorMaxFxFy integer, SensorMaxFz integer, SensorMaxPar integer, LongitudHerramienta integer);

CREATE TABLE trayectoria (codigo integer, descripcion char(100), fecha date, alumnos char(100), cod_herramienta integer, tiempo_real integer, primary key (codigo));

CREATE TABLE MedidasJR3 (CodMedida int, Descripcion char(50), Fecha date, primary key (CodMedida));

CREATE TABLE dialogo (id integer, descripcion varchar(40), label varchar (40), valor_defecto varchar(40), tipo_dato varchar(30), estilo integer, widget varchar(30), valores_posibles varchar(100), tipo_entrada varchar(20), primary key (id));

CREATE TABLE sesion (codsesion integer, descsesion char(50), comentario char(200), primary key (codsesion));

CREATE TABLE columnas (nomcol varchar(40), descripcion varchar(300), numcol integer, tipocol varchar(30), numerico integer, decimales integer, fecha integer, longitud integer, tipoesp integer, tablaid integer, colid integer, formcol integer, mitipocol char(1), primary key (tablaid, colid) foreign key (tablaid) references tablas (tablaid));

CREATE TABLE tipomov (codmov integer, descmov varchar(40), primary key (codmov));

CREATE TABLE movsesion (codsesion integer, codmov integer, tipomov integer, velocidad float, s1 float, s2 float s3 float, e1 float, e2 float, w1 float, w2 float, posx float, posy float, posz float, roty float, rotp float, rotr float, primary key (codsesion, codmov));

CREATE TABLE tablas (nomtab varchar(40), descripcion varchar(500), tablaid integer, tipo varchar(30), primary key (tablaid));

CREATE TABLE EscalasJR3 (CodEscala int, Fx float, Fy float, Fz float, Mx float, My float, Mz float, Fecha date, Descripcion char(50), primary key (CodEscala));

CREATE TABLE puntosfuerzavacio (combinacion_punto char(60), combinacion_punto_art char(60), cod_herramienta integer, linea integer, posx float, posy float, posz float, roty float, rotp float, rotr float, fx float, fy float, fz float, mx float, my float, mz float, primary key(combinacion_punto));

3.3 Funciones

Para el modelo de funciones, también se ha utilizado diaw., de la misma forma se adjuntará el fichero de diaw, y a continuación se reproduce el diagrama creado:

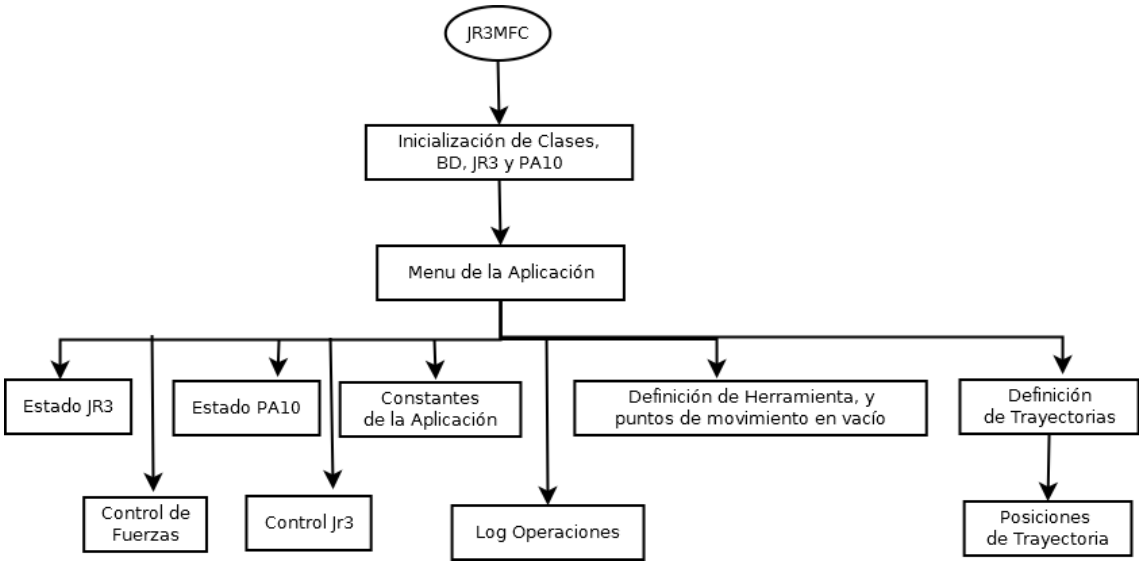
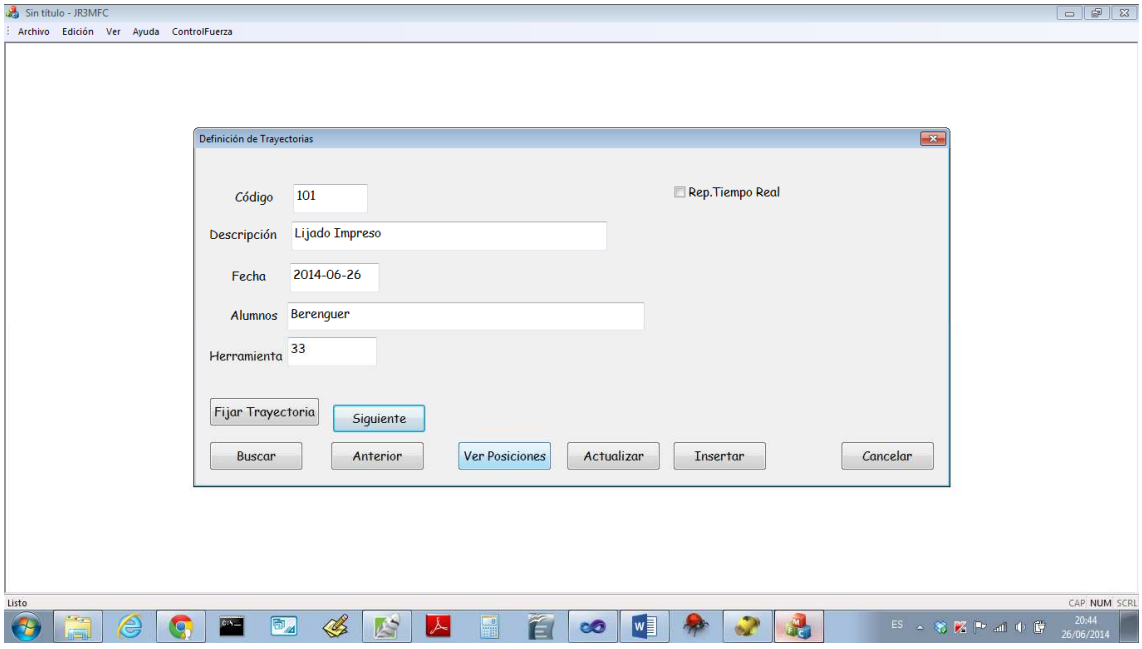
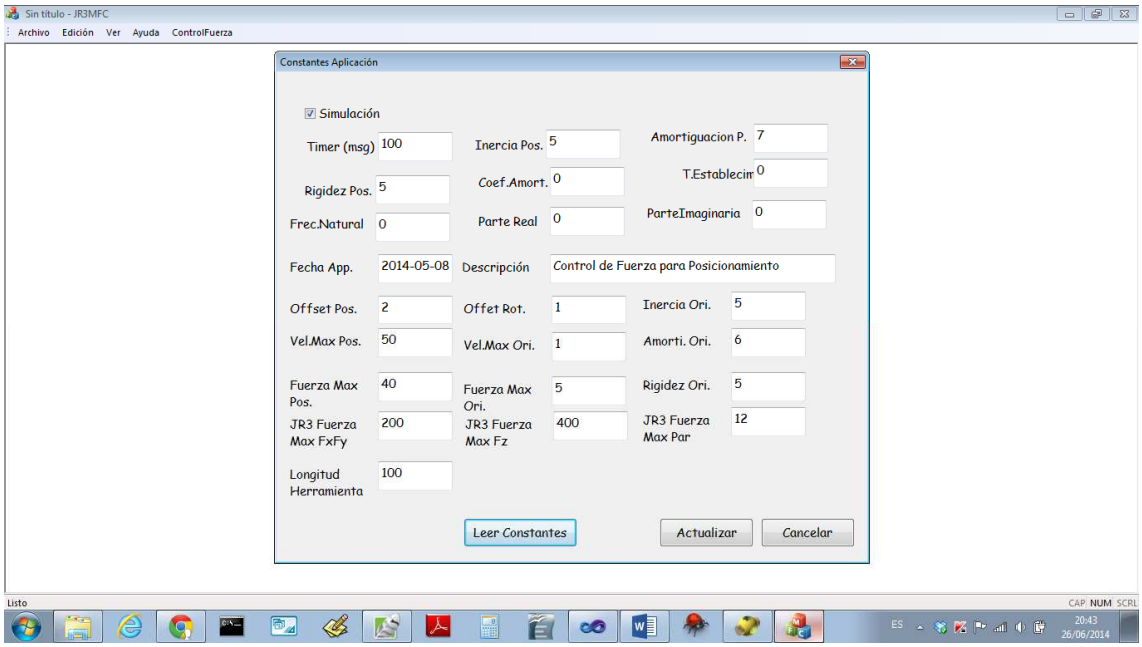
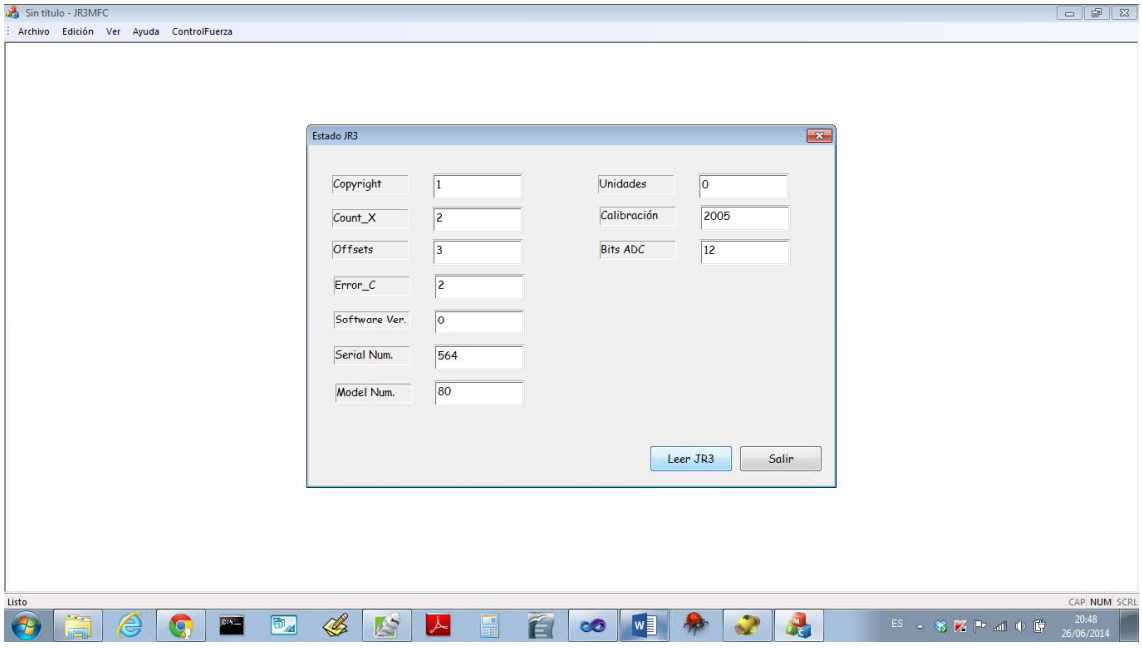


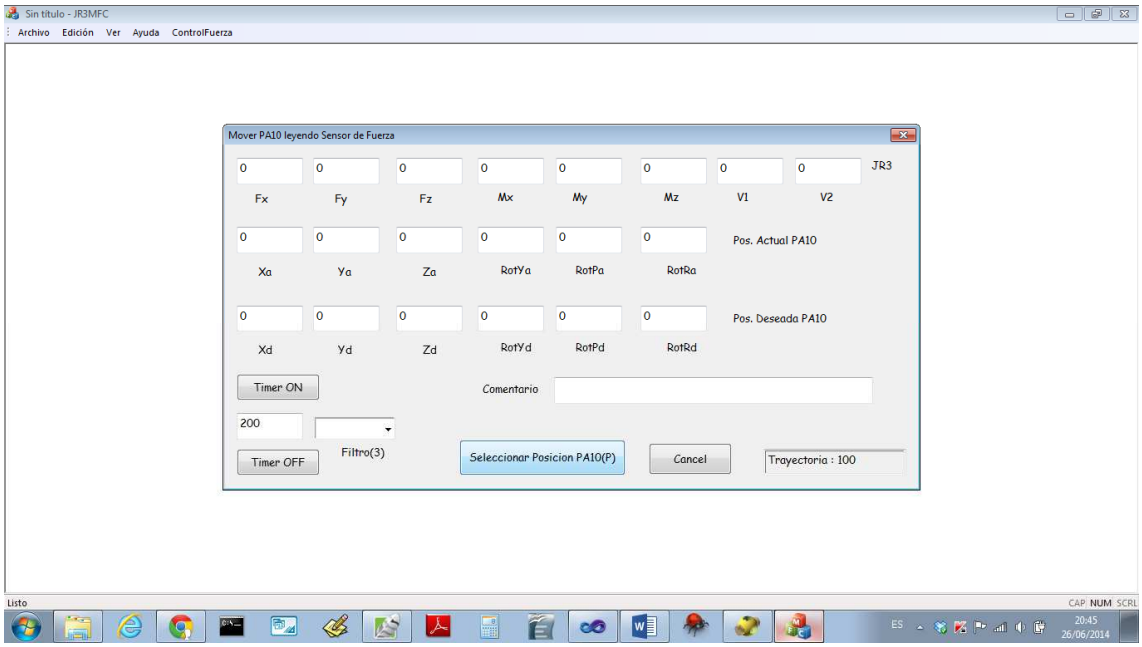
Ilustración 16Menú Aplicación

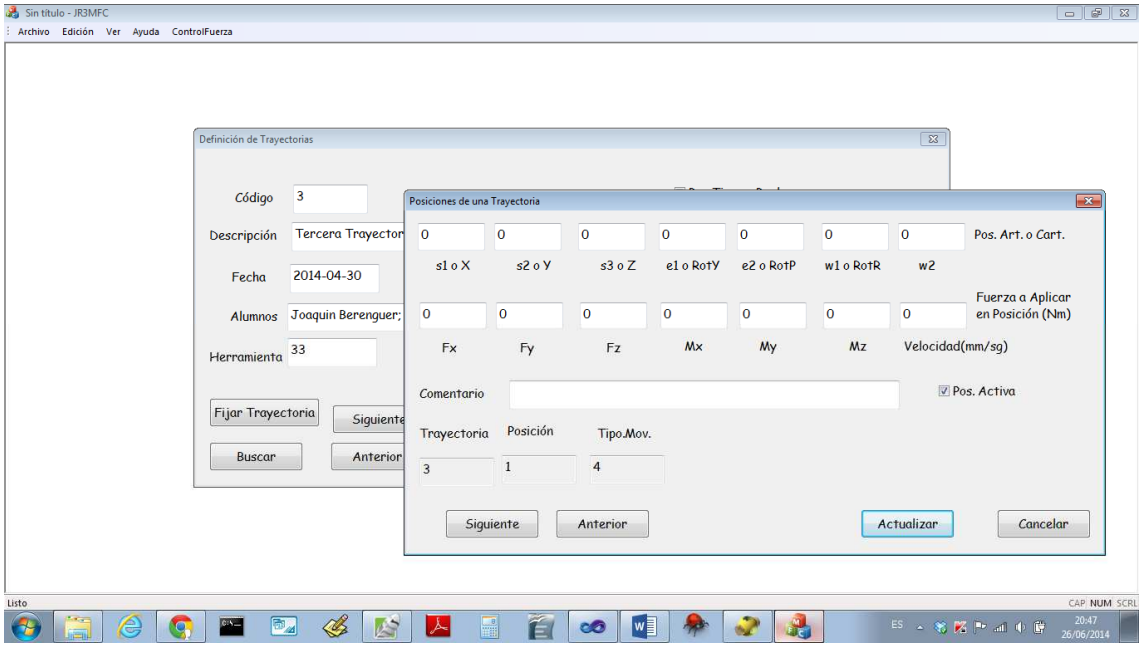
Desde dicho menú se accede a las diferentes Pantallas que se muestran a continuación:

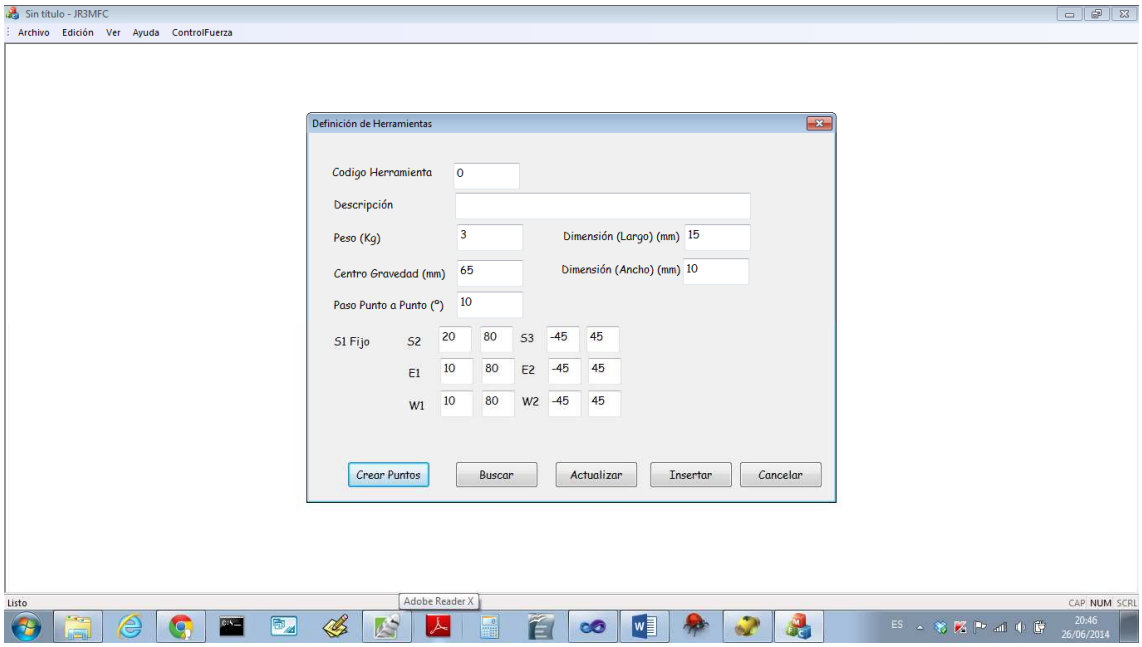


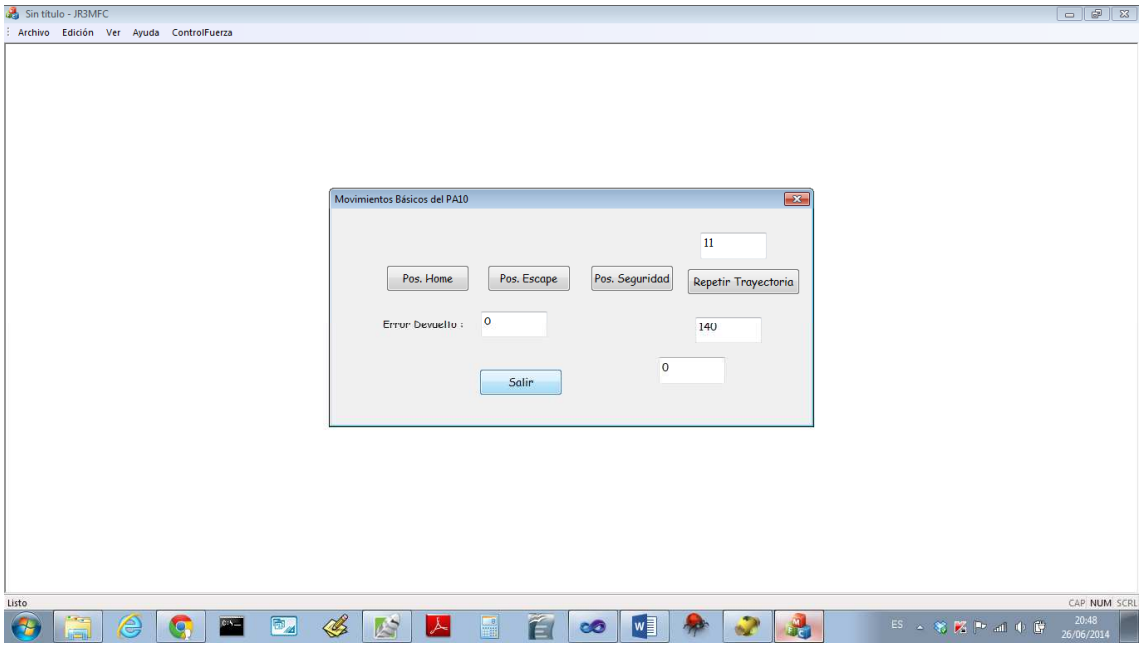

Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Constantes Aplicación
Simulación
Timer (msg) 100
Inercia Pos. 5
Amortiguacion P. 7
Rigidez Pos. 5
Coef.Amort. 0
T.Establecim 0
Frec.Natural 0
Parte Real 0
ParteImaginaria 0
Fecha App. 2014-05-08
Descripción Control de Fuerza para Posicionamiento
Offset Pos. 2
Offet Rot. 1
Inercia Ori. 5
Vel.Max Pos. 50
Vel.Max Ori. 1
Amorti. Ori. 6
Fuerza Max Pos. 40
Fuerza Max Ori. 5
Rigidez Ori. 5
JR3 Fuerza Max FxFy 200
JR3 Fuerza Max Fz 400
JR3 Fuerza Max Par 12
Longitud Herramienta 100
Leer Constantes
Actualizar
Cancelar
Listo

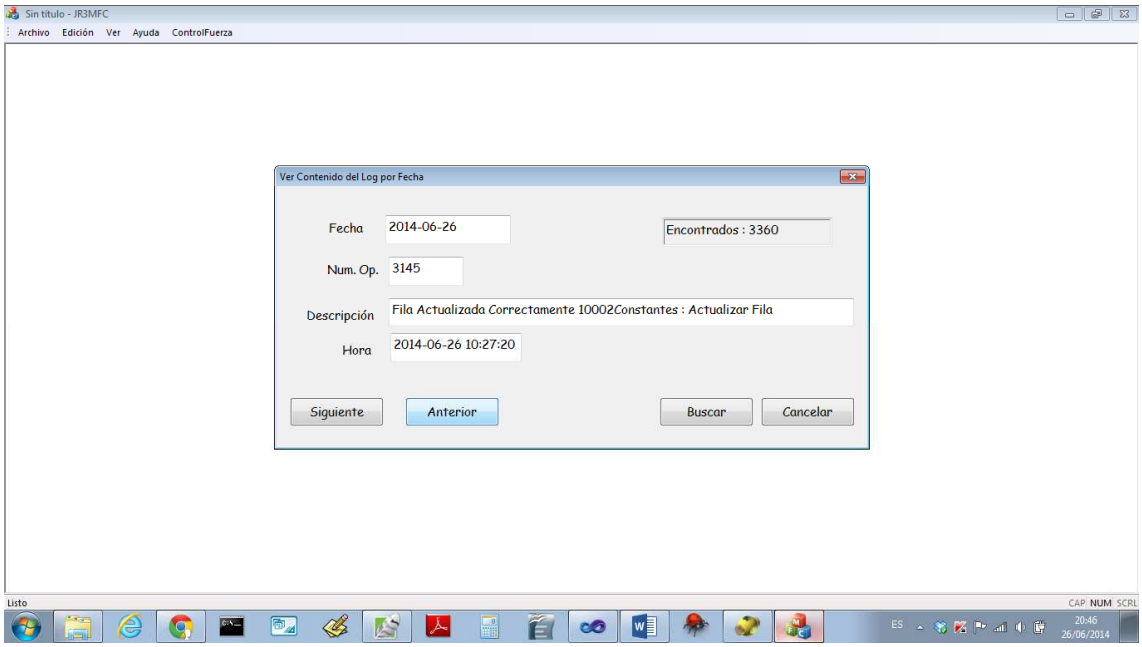

Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Estado JR3
Copyright 1
Unidades 0
Count_X 2
Calibración 2005
Offsets 3
Bits ADC 12
Error_C 2
Software Ver. 0
Serial Num. 564
Model Num. 80
Leer JR3
Salir
Listo

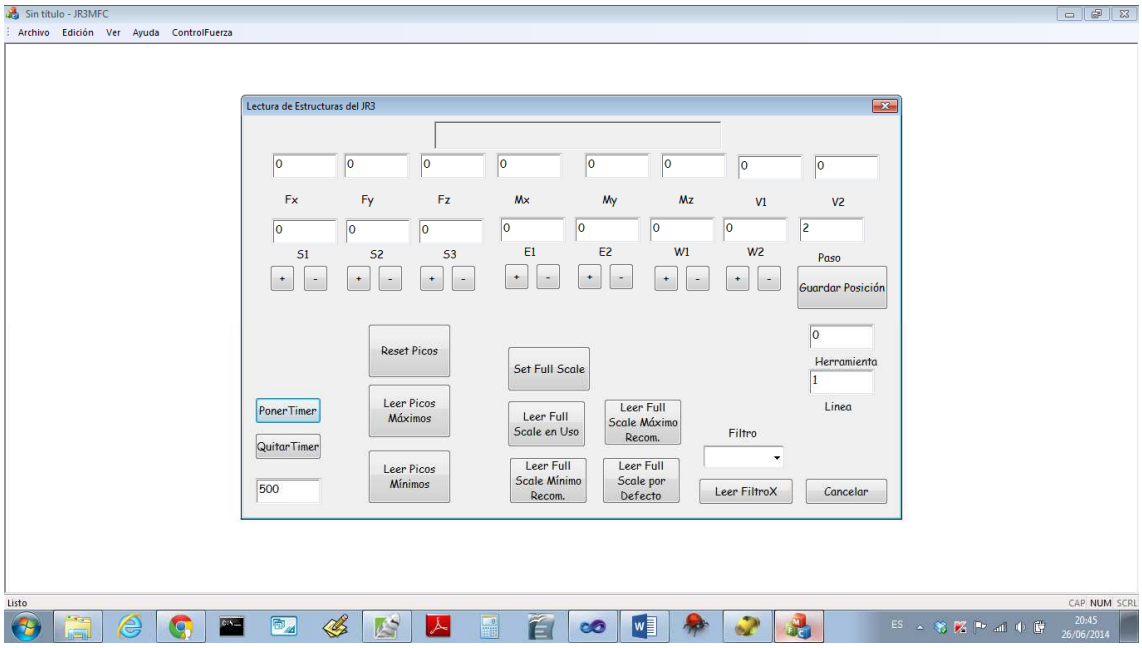

Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Mover PA10 leyendo Sensor de Fuerza
Fx Fy Fz Mx My Mz V1 V2 JR3
Xa Ya Za RotYa RotPa RotRa Pos. Actual PA10
Xd Yd Zd RotYd RotPd RotRd Pos. Deseada PA10
Timer ON
Comentario
200
Timer OFF
Filtro(3)
Seleccionar Posicion PA10(P)
Cancel
Trayectoria : 100
Listo

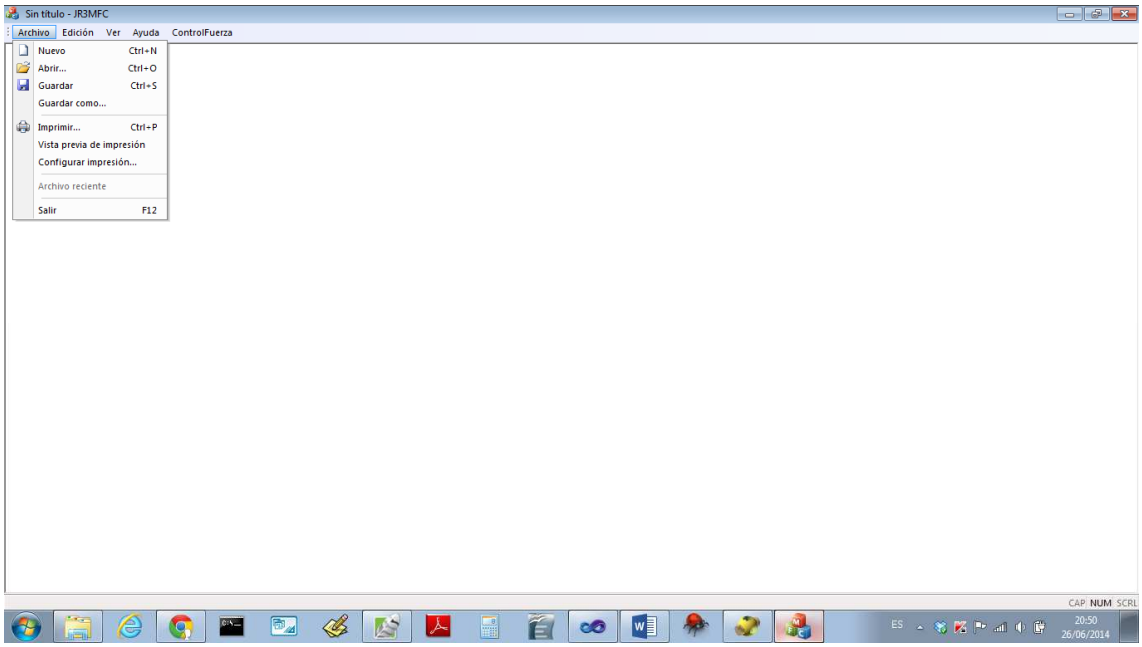

Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Definición de Trayectorias
Código 3
Descripción Tercera Trayector
Fecha 2014-04-30
Alumnos Joaquin Berenguer;
Herramienta 33
Fijar Trayectoria
Siguiente
Buscar
Anterior
Posiciones de una Trayectoria
s1 o X s2 o Y s3 o Z e1 o RotY e2 o RotP w1 o RotR w2 Pos. Art. o Cart.
Fx Fy Fz Mx My Mz Velocidad(mm/sg) Fuerza a Aplicar en Posición (Nm)
Comentario
Pos. Activa
Trayectoria Posición Tipo.Mov.
3 1 4
Siguiente
Anterior
Actualizar
Cancelar
Listo


Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Definición de Herramientas
Codigo Herramienta 0
Descripción
Peso (Kg) 3
Dimensión (Largo) (mm) 15
Centro Gravedad (mm) 65
Dimensión (Ancho) (mm) 10
Paso Punto a Punto (°) 10
S1 Fijo S2 20 80 S3 -45 45
E1 10 80 E2 -45 45
W1 10 80 W2 -45 45
Crear Puntos
Buscar
Actualizar
Insertar
Cancelar
Listo
Adobe Reader X
CAP NUM SCRL
ES
20:46
26/06/2014


Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Movimientos Básicos del PA10
11
Pos. Home
Pos. Escape
Pos. Seguridad
Repetir Trayectoria
Error Devuelto :
0
140
0
Salir
Listo
CAP NUM SCRL
ES
20:48
26/06/2014


Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Ver Contenido del Log por Fecha
Fecha
2014-06-26
Encontrados : 3360
Num. Op.
3145
Descripción
Fila Actualizada Correctamente 10002Constantes : Actualizar Fila
Hora
2014-06-26 10:27:20
Siguiente
Anterior
Buscar
Cancelar
Listo


Sin título - JR3MFC
Archivo Edición Ver Ayuda ControlFuerza
Lectura de Estructuras del JR3
Fx
Fy
Fz
Mx
My
Mz
V1
V2
S1
S2
S3
E1
E2
W1
W2
Paso
Guardar Posición
Herramienta
Linea
Reset Picos
Leer Picos Máximos
Leer Picos Mínimos
Set Full Scale
Leer Full Scale en Uso
Leer Full Scale Mínimo Recom.
Leer Full Scale Máximo Recom.
Leer Full Scale por Defecto
Filtro
Leer FiltroX
Cancelar
PonerTimer
QuitarTimer
500
Listo



Una función muy importante, y que es la base de la aplicación es la de Control de Fuerza y por ello se ha realizado un diagrama con diaw, para su mejor comprensión, se pone en marcha cuando se pulsa el botón de: Poner Timer, en la pantalla de Control de Fuerza, y su diagrama es el siguiente:

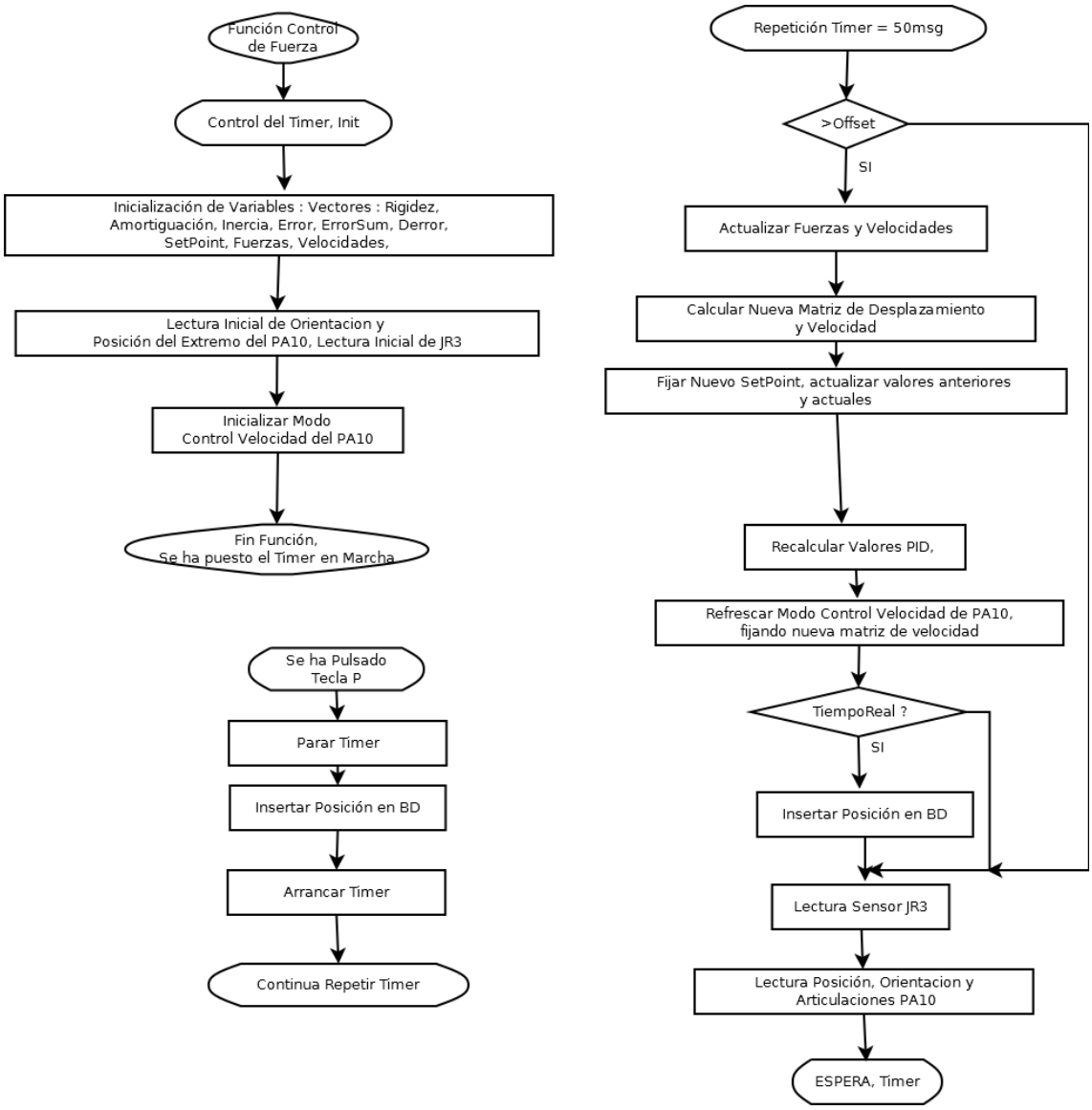


Ilustración 17Función Control de Fuerza

4 Aplicación Práctica: Lijado

Como aplicación de todo lo anterior, hemos elegido el Lijado, por estar directamente relacionado con el Control de Fuerza a aplicar sobre una superficie, y porque se puede reproducir el movimiento que un ser humano debe realizar para lijar.

Estos dos componentes son fundamentales para el Control de Fuerza para Posicionamiento, y por ello se ha realizado.

4.1 Introducción

Para la aplicación práctica de Lijado, se utilizará todo lo expuesto anteriormente, especialmente la función de Control de Fuerza descrito en el apartado de Funciones.

Para ello hemos creado una Herramienta de Lijado, y sus posiciones en vacío para poder compensar su peso en la trayectoria a crear.

Se ha creado una trayectoria, que se usará en el Control de Fuerza, fijando esta para su uso en dicha pantalla.

4.2 Creación de Trayectoria

La Trayectoria se crea con la siguiente pantalla:

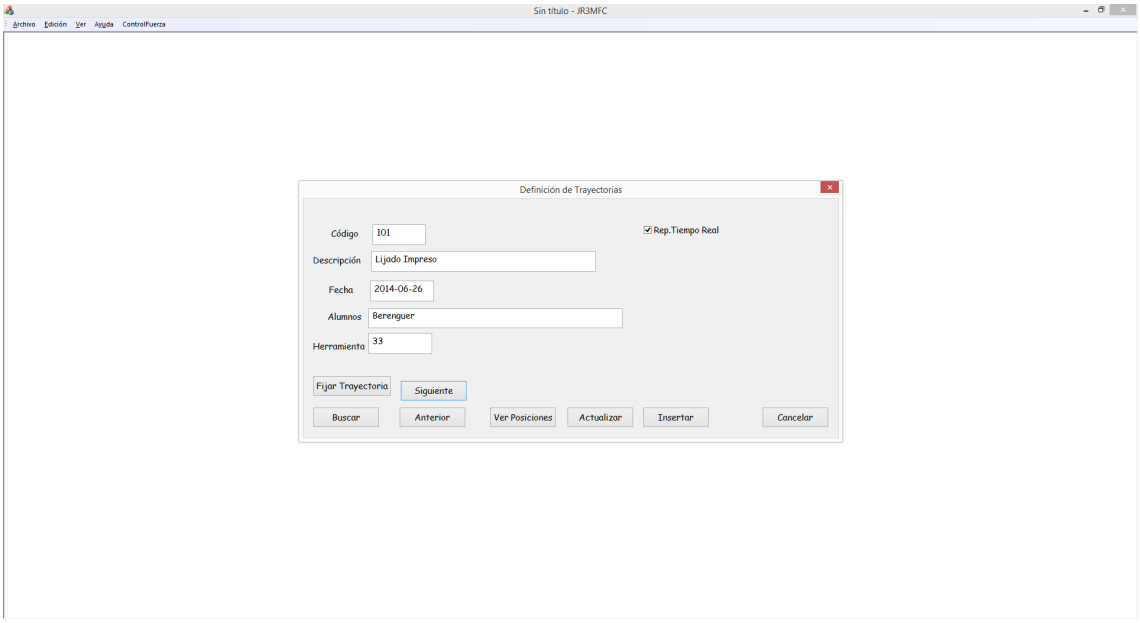


Ilustración 18Creación de Trayectoria de Lijado

Como se puede observar, se relaciona la trayectoria con los Alumnos que realizan dicha Trayectoria, ya que debe servir, para otros fines. También se relaciona, la Trayectoria con la Herramienta a utilizar, ya que se usará para tener en cuenta su peso en vacío, en el momento de la creación de posiciones.

Se observa también en la pantalla que es una trayectoria de tipo Tiempo Real, ya que para reproducir el movimiento se utilizará el modo Tiempo Real del PA10.

4.3 Creación de Posiciones

Cuando se utiliza el Modo Tiempo Real del PA10, en la función de Control de Fuerza descrita en el apartado Funciones, se observa que cada vez que se recalcula la velocidad, se inserta una posición en la tabla de posiciones, esto es cada que se activa el tiempo del Timer, habitualmente en nuestro caso 70msg, por tanto creamos una posición cada 70msg, esto nos permite un movimiento muy sutil y preciso. Mientras vamos moviendo el extremo del PA10, con la mano, se van creando las posiciones, de acuerdo con las fuerzas aplicadas. Para poder realizar este procedimiento, se ha tenido que insertar en memoria, hasta 1000 posiciones, y finalmente en modo batch, pasar dichas posiciones a la BD Sqlite3, ya que en caso contrario, el Modo de Control de Velocidad utilizado para crear las posiciones, se pierde ya que cada 200msg, tenemos que refrescar dicho modo, algo que no podemos hacer en 70msg.

Este punto es muy importante ya que en futuras aplicaciones se demostrará la importancia de este periodo de tiempo, ya que necesitamos un procesador muy potente para poder realizar todos los cálculos dentro del tiempo establecido.

Cuando se Utiliza la Opción de crear Posiciones cuando pulsamos el Botón Seleccionar Posición de la Pantalla de Control de Fuerza, se inserta una nueva fila, con el número de la fila en la Tabla Posición, para después ser reproducido de forma Movimiento Articular, en vez de hacerlo en Tiempo Real. Dependiendo de la Aplicación utilizaremos una forma u otra, pero ambas están disponibles, y esto se hace al crear la trayectoria seleccionando la variable Tiempo Real en dicha Pantalla.

4.4 Fuerzas Aplicadas en cada Posición

Una vez tenemos el movimiento completo, de acuerdo al punto anterior. Podemos reproducirlo mediante la pantalla siguiente:

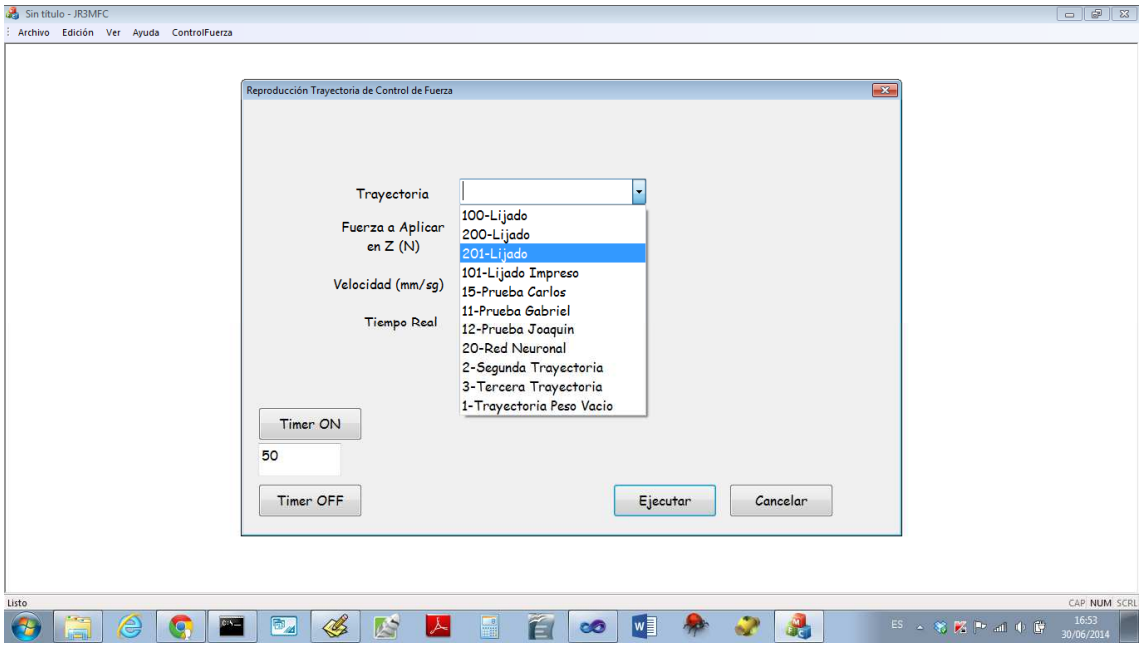


Ilustración 19Reproducción de Trayectoria

Como se observa en la pantalla, solamente tenemos que poner el código de Trayectoria que queramos para que el PA10, reproduzca en Modo Tiempo Real o Movimiento Articular normal, dependiendo la trayectoria deseada, las veces que queramos.

En la imagen siguiente podemos ver como el PA10, se aproxima a la primera posición de la operación de lijado, hasta tener contacto y ejercer una Fuerza indicada en la pantalla, antes de comenzar la reproducción.

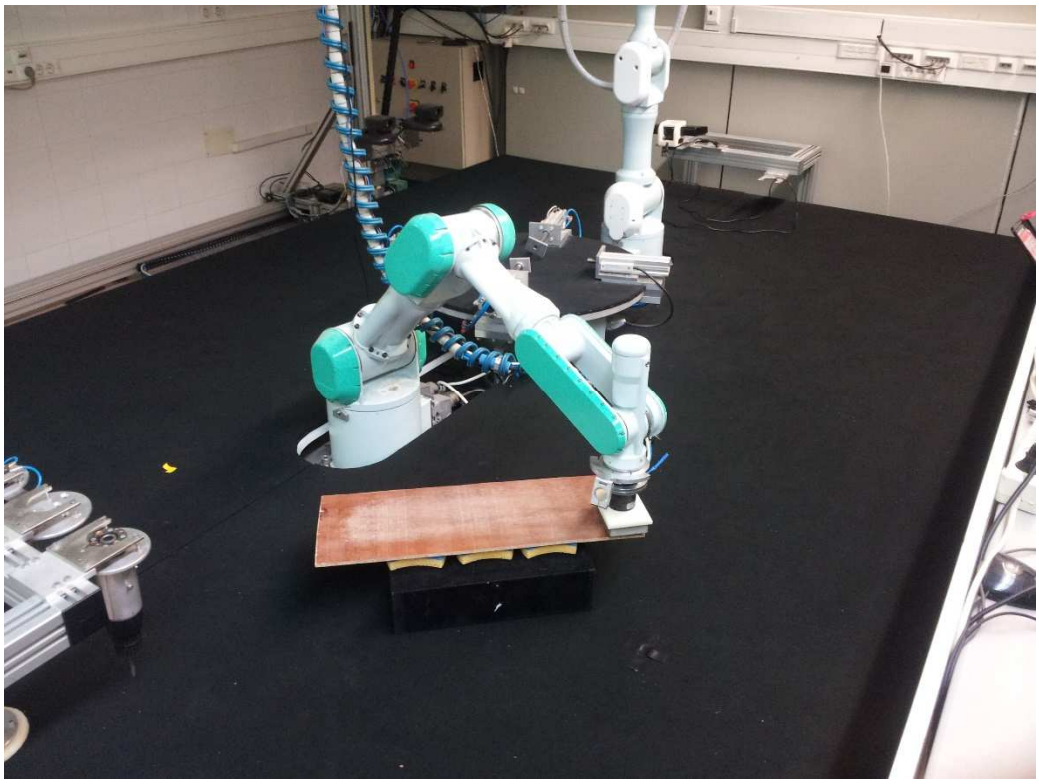


Ilustración 20 Reproducción Trayectoria, Primera Posición

La función de reprducción de trayectoria tiene el siguiente diagrama :

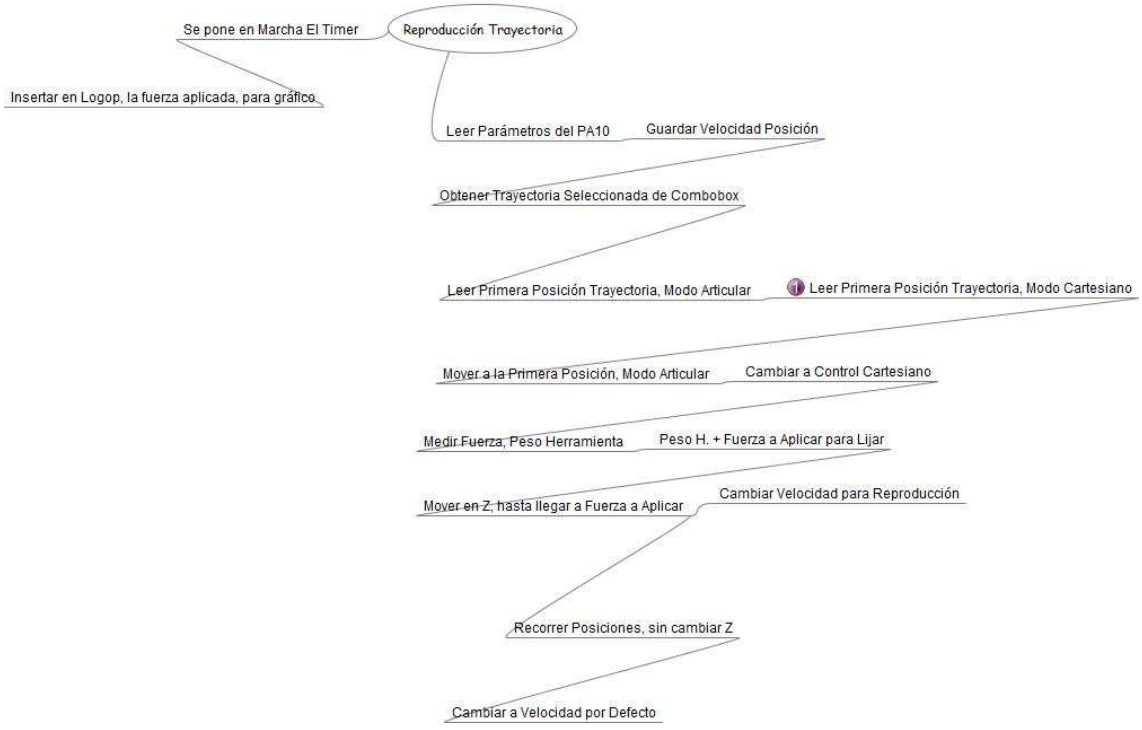


Ilustración 21Diagrama de Reproducción de Trayectoria

.

10. Conclusiones

Se ha desarrollado una segunda aplicación, en la que se ha realizado una trayectoria en modo de control en tiempo real del PA10, para dibujar. En este caso el control tiene que mucho más preciso, y para ello hemos modificado las constantes del Control de Impedancia, para tener menos transitorios, y poder mover el extremo del robot de una manera más precisa ya que las posiciones tienen que ser más precisas. Las constantes de masa-muelle-amortiguador han variado de 4 para el caso de Lijado a 10 para el caso de dibujo, utilizando valores más bajos los transitorios impiden un manejo adecuado, aunque la sensación de movimiento es muy rápido, y para valores más altos de 10, notamos unos movimientos lentos y difíciles de manejar, aunque para dibujo, donde las formas son muy complicadas necesitamos movimientos lentos y precisos, de forma que nuestro control de impedancia permite ambas situaciones.

La herramienta no ha variado, simplemente se ha insertado la punta de un lápiz, en el centro de la lija, aprovechando su esponja interior, en la imagen siguiente se puede apreciar como se ha realizado.

Ilustración 22 Segunda aplicación, Dibujo

El Control de Fuerza, es una parte muy importante de la Robótica, sin tener en cuenta las fuerzas aplicadas y recibidas del entorno de un Robot, es imposible cualquier aplicación práctica medianamente sofisticada.

Con este proyecto hemos rozado la superficie, de todo lo que se puede hacer con el Control de Fuerza, no hemos tenido en cuenta muchas de las nuevas tendencias que

actualmente están saliendo para aplicación en los Robot de Servicio especialmente, donde no podría aplicarse al contacto con el ser humano, sino se dispone de un Control de Fuerza sofisticado.

Las nuevas Manos Robóticas llamadas diestras, tiene como primer cometido el Control de Fuerza sofisticado, con sensores de fuerza repartidos por toda la superficie de la mano, lo que permite conocer la fuerza aplicada y recibida en cada punto de la mano.

En fin, que este proyecto, ha sido la punta del iceberg, de todo lo que se puede hacer con el Control de Fuerza, y espero que sirva de base a otros proyectos donde las aplicaciones puedan ser cada día más sofisticadas.

11. Bibliografía y Referencias

Microsoft Visual C++,MFC´s

Visual C++ 2010, Ivor Horton

Grasping in Robotics, Giuseppe Carbone, Springer

Robot Mechanisms, Jadran Lenarˇciˇc Tadej Bajd , Springer

Robotics, S. G. Tzafestas, Springer

Robotics and Aut. Handbook, Thomas R. Kurfess, CRC Press

SQLITE C Interface, por SQLITE Docs. Website

Robotics, Bruno Siciliano, Springer

Base Force/Torque Sensing for Position based Cartesian Impedance Control, Christian Ott, Articulo IEEE

A Robust Position and Force Control Strategy for 7-DOF Redundant Manipulators, Rajni V. Patel, Articulo IEEE

12. Anexo A

SQLite C Interface

Binding Values To Prepared Statements

int sqlite3_bind_blob(sqlite3_stmt*, int, const void*, int n, void(*)(void*));
int sqlite3_bind_double(sqlite3_stmt*, int, double);
int sqlite3_bind_int(sqlite3_stmt*, int, int);
int sqlite3_bind_int64(sqlite3_stmt*, int, sqlite3_int64);
int sqlite3_bind_null(sqlite3_stmt*, int);
int sqlite3_bind_text(sqlite3_stmt*, int, const char*, int n, void(*)(void*));
int sqlite3_bind_text16(sqlite3_stmt*, int, const void*, int, void(*)(void*));
int sqlite3_bind_value(sqlite3_stmt*, int, const sqlite3_value*);
int sqlite3_bind_zeroblob(sqlite3_stmt*, int, int n);

In the SQL statement text input to sqlite3_prepare_v2() and its variants, literals may be replaced by a parameter that matches one of following templates:

- ?
- ?NNN
- :VVV
- @VVV
- $VVV

In the templates above, NNN represents an integer literal, and VVV represents an alphanumeric identifier. The values of these parameters (also called "host parameter names" or "SQL parameters") can be set using the sqlite3_bind_*() routines defined here.

The first argument to the sqlite3_bind_*() routines is always a pointer to the sqlite3_stmt object returned from sqlite3_prepare_v2() or its variants.

The second argument is the index of the SQL parameter to be set. The leftmost SQL parameter has an index of 1. When the same named SQL parameter is used more than once, second and subsequent occurrences have the same index as the first occurrence. The index for named parameters can be looked up using the sqlite3_bind_parameter_index() API if desired. The index for "?NNN" parameters is the value of NNN. The NNN value must be between 1 and the sqlite3_limit() parameter SQLITE_LIMIT_VARIABLE_NUMBER (default value: 999).

The third argument is the value to bind to the parameter. If the third parameter to sqlite3_bind_text() or sqlite3_bind_text16() or sqlite3_bind_blob() is a NULL pointer then the fourth parameter is ignored and the end result is the same as sqlite3_bind_null().

In those routines that have a fourth argument, its value is the number of bytes in the parameter. To be clear: the value is the number of bytes in the value, not the number of characters. If the fourth parameter to sqlite3_bind_text() or sqlite3_bind_text16() is negative, then the length of the string is the number of bytes up to the first zero terminator. If the fourth parameter to sqlite3_bind_blob() is negative, then the behavior is undefined. If a non-negative fourth parameter is provided to sqlite3_bind_text() or sqlite3_bind_text16() then that parameter must be the byte offset where the NUL terminator would occur assuming the string were NUL terminated. If any NUL characters occur at byte offsets less than the value of the fourth parameter then the resulting string value will contain embedded NULs. The result of expressions involving strings with embedded NULs is undefined.

The fifth argument to sqlite3_bind_blob(), sqlite3_bind_text(), and sqlite3_bind_text16() is a destructor used to dispose of the BLOB or string after SQLite has finished with it. The destructor is called to dispose of the BLOB or string even if the call to sqlite3_bind_blob(), sqlite3_bind_text(), or sqlite3_bind_text16() fails. If the fifth argument is the special value SQLITE_STATIC, then SQLite assumes that the information is in static, unmanaged space and does not need to be freed. If the fifth argument has the value SQLITE_TRANSIENT, then SQLite makes its own private copy of the data immediately, before the sqlite3_bind_*() routine returns.

The sqlite3_bind_zeroblob() routine binds a BLOB of length N that is filled with zeroes. A zeroblob uses a fixed amount of memory (just an integer to hold its size) while it is being processed. Zeroblobs are intended to serve as placeholders for BLOBs whose content is later written using incremental BLOB I/O routines. A negative value for the zeroblob results in a zero-length BLOB.

If any of the sqlite3_bind_*() routines are called with a NULL pointer for the prepared statement or with a prepared statement for which sqlite3_step() has been called more recently than sqlite3_reset(), then the call will return SQLITE_MISUSE. If any sqlite3_bind_() routine is passed a prepared statement that has been finalized, the result is undefined and probably harmful.

Bindings are not cleared by the sqlite3_reset() routine. Unbound parameters are interpreted as NULL.

The sqlite3_bind_* routines return SQLITE_OK on success or an error code if anything goes wrong. SQLITE_RANGE is returned if the parameter index is out of range. SQLITE_NOMEM is returned if malloc() fails.

Result Values From A Query

const void *sqlite3_column_blob(sqlite3_stmt*, int iCol);
int sqlite3_column_bytes(sqlite3_stmt*, int iCol);
int sqlite3_column_bytes16(sqlite3_stmt*, int iCol);
double sqlite3_column_double(sqlite3_stmt*, int iCol);
int sqlite3_column_int(sqlite3_stmt*, int iCol);
sqlite3_int64 sqlite3_column_int64(sqlite3_stmt*, int iCol);
const unsigned char *sqlite3_column_text(sqlite3_stmt*, int iCol);
const void *sqlite3_column_text16(sqlite3_stmt*, int iCol);

int sqlite3_column_type(sqlite3_stmt*, int iCol);
sqlite3_value *sqlite3_column_value(sqlite3_stmt*, int iCol);

These routines form the "result set" interface.

These routines return information about a single column of the current result row of a query. In every case the first argument is a pointer to the prepared statement that is being evaluated (the sqlite3_stmt* that was returned from sqlite3_prepare_v2() or one of its variants) and the second argument is the index of the column for which information should be returned. The leftmost column of the result set has the index 0. The number of columns in the result can be determined using sqlite3_column_count().

If the SQL statement does not currently point to a valid row, or if the column index is out of range, the result is undefined. These routines may only be called when the most recent call to sqlite3_step() has returnedSQLITE_ROW and neither sqlite3_reset() nor sqlite3_finalize() have been called subsequently. If any of these routines are called after sqlite3_reset() or sqlite3_finalize() or after sqlite3_step() has returned something other than SQLITE_ROW, the results are undefined. If sqlite3_step() or sqlite3_reset() or sqlite3_finalize() are called from a different thread while any of these routines are pending, then the results are undefined.

The sqlite3_column_type() routine returns the datatype code for the initial data type of the result column. The returned value is one of SQLITE_INTEGER, SQLITE_FLOAT, SQLITE_TEXT, SQLITE_BLOB, or SQLITE_NULL. The value returned by sqlite3_column_type() is only meaningful if no type conversions have occurred as described below. After a type conversion, the value returned by sqlite3_column_type() is undefined. Future versions of SQLite may change the behavior of sqlite3_column_type() following a type conversion.

If the result is a BLOB or UTF-8 string then the sqlite3_column_bytes() routine returns the number of bytes in that BLOB or string. If the result is a UTF-16 string, then sqlite3_column_bytes() converts the string to UTF-8 and then returns the number of bytes. If the result is a numeric value then sqlite3_column_bytes() uses sqlite3_snprintf() to convert that value to a UTF-8 string and returns the number of bytes in that string. If the result is NULL, then sqlite3_column_bytes() returns zero.

If the result is a BLOB or UTF-16 string then the sqlite3_column_bytes16() routine returns the number of bytes in that BLOB or string. If the result is a UTF-8 string, then sqlite3_column_bytes16() converts the string to UTF-16 and then returns the number of bytes. If the result is a numeric value then sqlite3_column_bytes16() uses sqlite3_snprintf() to convert that value to a UTF-16 string and returns the number of bytes in that string. If the result is NULL, then sqlite3_column_bytes16() returns zero.

The values returned by sqlite3_column_bytes() and sqlite3_column_bytes16() do not include the zero terminators at the end of the string. For clarity: the values returned by sqlite3_column_bytes() andsqlite3_column_bytes16() are the number of bytes in the string, not the number of characters.

Strings returned by sqlite3_column_text() and sqlite3_column_text16(), even empty strings, are always zero-terminated. The return value from sqlite3_column_blob() for a zero-length BLOB is a NULL pointer.

The object returned by sqlite3_column_value() is an unprotected sqlite3_value object. An unprotected sqlite3_value object may only be used with sqlite3_bind_value() and sqlite3_result_value(). If the unprotected sqlite3_value object returned by sqlite3_column_value() is used in any other way, including calls to routines like sqlite3_value_int(), sqlite3_value_text(), or sqlite3_value_bytes(), then the behavior is undefined.

These routines attempt to convert the value where appropriate. For example, if the internal representation is FLOAT and a text result is requested, sqlite3_snprintf() is used internally to perform the conversion automatically. The following table details the conversions that are applied:

Internal Type	Requested Type	Conversion
NULL	INTEGER	Result is 0
NULL	FLOAT	Result is 0.0
NULL	TEXT	Result is a NULL pointer
NULL	BLOB	Result is a NULL pointer
INTEGER	FLOAT	Convert from integer to float
INTEGER	TEXT	ASCII rendering of the integer
INTEGER	BLOB	Same as INTEGER->TEXT
FLOAT	INTEGER	CAST to INTEGER
FLOAT	TEXT	ASCII rendering of the float
FLOAT	BLOB	CAST to BLOB
TEXT	INTEGER	CAST to INTEGER
TEXT	FLOAT	CAST to REAL
TEXT	BLOB	No change
BLOB	INTEGER	CAST to INTEGER

BLOB	FLOAT	CAST to REAL
BLOB	TEXT	Add a zero terminator if needed

The table above makes reference to standard C library functions atoi() and atof(). SQLite does not really use these functions. It has its own equivalent internal routines. The atoi() and atof() names are used in the table for brevity and because they are familiar to most C programmers.

Note that when type conversions occur, pointers returned by prior calls to sqlite3_column_blob(), sqlite3_column_text(), and/or sqlite3_column_text16() may be invalidated. Type conversions and pointer invalidations might occur in the following cases:

- The initial content is a BLOB and sqlite3_column_text() or sqlite3_column_text16() is called. A zero-terminator might need to be added to the string.
- The initial content is UTF-8 text and sqlite3_column_bytes16() or sqlite3_column_text16() is called. The content must be converted to UTF-16.
- The initial content is UTF-16 text and sqlite3_column_bytes() or sqlite3_column_text() is called. The content must be converted to UTF-8.

Conversions between UTF-16be and UTF-16le are always done in place and do not invalidate a prior pointer, though of course the content of the buffer that the prior pointer references will have been modified. Other kinds of conversion are done in place when it is possible, but sometimes they are not possible and in those cases prior pointers are invalidated.

The safest and easiest to remember policy is to invoke these routines in one of the following ways:

- sqlite3_column_text() followed by sqlite3_column_bytes()
- sqlite3_column_blob() followed by sqlite3_column_bytes()
- sqlite3_column_text16() followed by sqlite3_column_bytes16()

In other words, you should call sqlite3_column_text(), sqlite3_column_blob(), or sqlite3_column_text16() first to force the result into the desired format, then invoke sqlite3_column_bytes() or sqlite3_column_bytes16() to find the size of the result. Do not mix calls to sqlite3_column_text() or sqlite3_column_blob() with calls to sqlite3_column_bytes16(), and do not mix calls to sqlite3_column_text16() with calls to sqlite3_column_bytes().

The pointers returned are valid until a type conversion occurs as described above, or until sqlite3_step() or sqlite3_reset() or sqlite3_finalize() is called. The memory space used to hold strings and BLOBs is freed automatically. Do **not** pass the pointers returned from sqlite3_column_blob(), sqlite3_column_text(), etc. into sqlite3_free().

If a memory allocation error occurs during the evaluation of any of these routines, a default value is returned. The default value is either the integer 0, the floating point

number 0.0, or a NULL pointer. Subsequent calls tosqlite3_errcode() will return SQLITE_NOMEM.

Compiling An SQL Statement

```
int sqlite3_prepare(
  sqlite3 *db,           /* Database handle */
  const char *zSql,      /* SQL statement, UTF-8 encoded */
  int nByte,             /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const char **pzTail     /* OUT: Pointer to unused portion of zSql */
);
int sqlite3_prepare_v2(
  sqlite3 *db,           /* Database handle */
  const char *zSql,      /* SQL statement, UTF-8 encoded */
  int nByte,             /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const char **pzTail     /* OUT: Pointer to unused portion of zSql */
);
int sqlite3_prepare16(
  sqlite3 *db,           /* Database handle */
  const void *zSql,      /* SQL statement, UTF-16 encoded */
  int nByte,             /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const void **pzTail     /* OUT: Pointer to unused portion of zSql */
);
int sqlite3_prepare16_v2(
  sqlite3 *db,           /* Database handle */
  const void *zSql,      /* SQL statement, UTF-16 encoded */
  int nByte,             /* Maximum length of zSql in bytes. */
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const void **pzTail     /* OUT: Pointer to unused portion of zSql */
);
```

To execute an SQL query, it must first be compiled into a byte-code program using one of these routines.

The first argument, "db", is a database connection obtained from a prior successful call to sqlite3_open(), sqlite3_open_v2() or sqlite3_open16(). The database connection must not have been closed.

The second argument, "zSql", is the statement to be compiled, encoded as either UTF-8 or UTF-16. The sqlite3_prepare() and sqlite3_prepare_v2() interfaces use UTF-8, and sqlite3_prepare16() and sqlite3_prepare16_v2() use UTF-16.

If the nByte argument is less than zero, then zSql is read up to the first zero terminator. If nByte is non-negative, then it is the maximum number of bytes read from zSql. When nByte is non-negative, the zSql string ends at either the first '\000' or '\u0000' character or the nByte-th byte, whichever comes first. If the caller knows that the supplied string is nul-terminated, then there is a small performance advantage to be gained by passing

an nByte parameter that is equal to the number of bytes in the input string *including* the nul-terminator bytes as this saves SQLite from having to make a copy of the input string.

If pzTail is not NULL then *pzTail is made to point to the first byte past the end of the first SQL statement in zSql. These routines only compile the first statement in zSql, so *pzTail is left pointing to what remains uncompiled.

*ppStmt is left pointing to a compiled prepared statement that can be executed using sqlite3_step(). If there is an error, *ppStmt is set to NULL. If the input text contains no SQL (if the input is an empty string or a comment) then *ppStmt is set to NULL. The calling procedure is responsible for deleting the compiled SQL statement using sqlite3_finalize() after it has finished with it. ppStmt may not be NULL.

On success, the sqlite3_prepare() family of routines return SQLITE_OK; otherwise an error code is returned.

The sqlite3_prepare_v2() and sqlite3_prepare16_v2() interfaces are recommended for all new programs. The two older interfaces are retained for backwards compatibility, but their use is discouraged. In the "v2" interfaces, the prepared statement that is returned (the sqlite3_stmt object) contains a copy of the original SQL text. This causes the sqlite3_step() interface to behave differently in three ways:

1. If the database schema changes, instead of returning SQLITE_SCHEMA as it always used to do, sqlite3_step() will automatically recompile the SQL statement and try to run it again. As many asSQLITE_MAX_SCHEMA_RETRY retries will occur before sqlite3_step() gives up and returns an error.
2. When an error occurs, sqlite3_step() will return one of the detailed error codes or extended error codes. The legacy behavior was that sqlite3_step() would only return a generic SQLITE_ERROR result code and the application would have to make a second call to sqlite3_reset() in order to find the underlying cause of the problem. With the "v2" prepare interfaces, the underlying reason for the error is returned immediately.
3. If the specific value bound to host parameter in the WHERE clause might influence the choice of query plan for a statement, then the statement will be automatically recompiled, as if there had been a schema change, on the first sqlite3_step() call following any change to the bindings of that parameter. The specific value of WHERE-clause parameter might influence the choice of query plan if the parameter is the left-hand side of aLIKE or GLOB operator or if the parameter is compared to an indexed column and the SQLITE_ENABLE_STAT3 compile-time option is enabled.

Evaluate An SQL Statement

int sqlite3_step(sqlite3_stmt*);

After a prepared statement has been prepared using either sqlite3_prepare_v2() or sqlite3_prepare16_v2() or one of the legacy interfaces sqlite3_prepare() or sqlite3_prepare16(), this function must be called one or more times to evaluate the statement.

The details of the behavior of the sqlite3_step() interface depend on whether the statement was prepared using the newer "v2" interface sqlite3_prepare_v2() and sqlite3_prepare16_v2() or the older legacy interfacesqlite3_prepare() and sqlite3_prepare16(). The use of the new "v2" interface is recommended for new applications but the legacy interface will continue to be supported.

In the legacy interface, the return value will be either SQLITE_BUSY, SQLITE_DONE, SQLITE_ROW, SQLITE_ERROR, or SQLITE_MISUSE. With the "v2" interface, any of the other result codes or extended result codes might be returned as well.

SQLITE_BUSY means that the database engine was unable to acquire the database locks it needs to do its job. If the statement is a COMMIT or occurs outside of an explicit transaction, then you can retry the statement. If the statement is not a COMMIT and occurs within an explicit transaction then you should rollback the transaction before continuing.

SQLITE_DONE means that the statement has finished executing successfully. sqlite3_step() should not be called again on this virtual machine without first calling sqlite3_reset() to reset the virtual machine back to its initial state.

If the SQL statement being executed returns any data, then SQLITE_ROW is returned each time a new row of data is ready for processing by the caller. The values may be accessed using the column access functions. sqlite3_step() is called again to retrieve the next row of data.

SQLITE_ERROR means that a run-time error (such as a constraint violation) has occurred. sqlite3_step() should not be called again on the VM. More information may be found by calling sqlite3_errmsg(). With the legacy interface, a more specific error code (for example, SQLITE_INTERRUPT, SQLITE_SCHEMA, SQLITE_CORRUPT, and so forth) can be obtained by calling sqlite3_reset() on the prepared statement. In the "v2" interface, the more specific error code is returned directly by sqlite3_step().

SQLITE_MISUSE means that the this routine was called inappropriately. Perhaps it was called on a prepared statement that has already been finalized or on one that had previously returned SQLITE_ERROR or SQLITE_DONE. Or it could be the case that the same database connection is being used by two or more threads at the same moment in time.

For all versions of SQLite up to and including 3.6.23.1, a call to sqlite3_reset() was required after sqlite3_step() returned anything other than SQLITE_ROW before any subsequent invocation of sqlite3_step(). Failure to reset the prepared statement using sqlite3_reset() would result in an SQLITE_MISUSE return from sqlite3_step(). But after version 3.6.23.1, sqlite3_step() began calling sqlite3_reset() automatically in this circumstance rather than returning SQLITE_MISUSE. This is not considered a compatibility break because any application that ever receives an SQLITE_MISUSE error is broken by definition. The SQLITE_OMIT_AUTORESET compile-time option can be used to restore the legacy behavior.

Destroy A Prepared Statement Object

int sqlite3_finalize(sqlite3_stmt *pStmt);

The sqlite3_finalize() function is called to delete a prepared statement. If the most recent evaluation of the statement encountered no errors or if the statement is never been evaluated, then sqlite3_finalize() returns SQLITE_OK. If the most recent evaluation of statement S failed, then sqlite3_finalize(S) returns the appropriate error code or extended error code.

The sqlite3_finalize(S) routine can be called at any point during the life cycle of prepared statement S: before statement S is ever evaluated, after one or more calls to sqlite3_reset(), or after any call to sqlite3_step()regardless of whether or not the statement has completed execution.

Invoking sqlite3_finalize() on a NULL pointer is a harmless no-op.

The application must finalize every prepared statement in order to avoid resource leaks. It is a grievous error for the application to try to use a prepared statement after it has been finalized. Any use of a prepared statement after it has been finalized can result in undefined and undesirable behavior such as segfaults and heap corruption.