6to. Congreso Nacional de Mecatrónica, Noviembre 8-10, 2007
Instituto Tecnológico de San Luis Potosí, S.L.P.
Asociación Mexicana de Mecatrónica A.C.

Metodología de diseño para un simulador dedicado a un pendubot

Pablo Sánchez-Sánchez, Fernando Reyes-Cortés, Antonio Michua-Camarillo, W. Fermín Guerrero-Sánchez
Roberto Murueta-Fortiz y Shared Santamaria-Castillo
Universidad Autónoma de Puebla - Robotics Team "oocelo"
lepable@ece.buap.mx, freyes@ece.buap.mx, amc@ece.buap.mx

Abstract - El objetivo principal de este artículo es describir la metodología que se emplea en la elaboración de un simulador utilizando plataformas de diseño asistido por computadora (AutoCAD, 3D StudioMax), lenguajes de programación (Visual C++) y el motor grafico OpenGL. El simulador que se presenta está basado en el prototipo *pendubot* de la empresa *Quancer* .

I. INTRODUCCIÓN

Una plataforma grafica de simulación o *simulador* permite reproducir el comportamiento de un sistema mediante las ecuaciones matemáticas que lo describen, y brindar un ambiente gráfico de interpretación de datos dependientes de los estímulos internos y externos, para que el usuario final entienda dicho comportamiento. Por esta razón los simuladores son herramientas utilizadas en el diseño y creación de prototipos.

La evolución de los simuladores se ha dado en función a los avances en las computadoras y en los lenguajes de programación; los simuladores actualmente son una herramienta poderosa, exacta y amigable en la interacción con el usuario. Es importante recalcar que un simulador es una herramienta que requiere del *modelo dinámico* del sistema y de una *estructura de control* que nos permita realizar correctamente la tarea asignada.

A continuación se enumeran los pasos necesarios para desarrollar un simulador dedicado a un prototipo.

1. Análisis geométrico del sistema (cinemática directa e inversa).
2. Obtención del modelo dinámico aplicando el análisis geométrico.
3. Propuesta y análisis de una estructura de control.
4. Bosquejo del sistema utilizando software de diseño asistido por computadora (AutoCAD).
5. Programación del modelo dinámico y el control en una plataforma de programación (Visual C++).
6. Evaluación del simulador mediante el desarrollo de pruebas controladas.

Partiendo de esto pasos se analiza un pendubot de la empresa *Quanser* y elabora un simulador.

II. SISTEMA DINÁMICO DEL PENDUBOT

Situando al pendubot en un sistema de referencia como se muestra en la figura 1, decimos que l_1 es la longitud del eslabón uno; l_{c_1} y l_{c_2} son las distancias al centro de la masa de los eslabones; q_1 y q_2 son los desplazamientos angulares realizados por los eslabones.

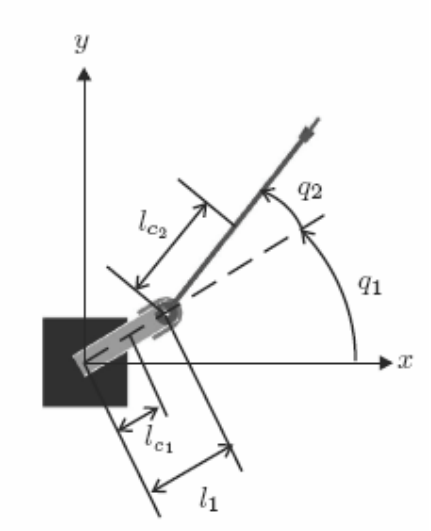


Fig. 1. Pendubot.

Usando la formulación de Euler-Lagrange, encontramos las ecuaciones de movimiento que describen al sistema considerando todos los parámetros involucrados en función al sistema de referencia.

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \tag{1}$$

donde $q, \dot{q}, \ddot{q} \in R^{n\times 1}$ son el vector de posición, velocidad y aceleración articular del robot, respectivamente, $M(q) \in R^{n\times n}$ es la matriz de inercias, $C(q,\dot{q}) \in R^{n\times n}$ es la matriz de Coriolis y fuerza centrípeta, $g(q) \in R^{n\times 1}$ es el par gravitacional, $\tau \in R^{n\times 1}$ es el par aplicado y $\tau_d \in R^{n\times 1}$ representa las perturbaciones externas.

Para simplificar las operaciones algebraicas de la ecuación de movimiento se agruparon siete parámetros quedando sólo cinco.

$$\begin{aligned}\theta_1 &= m_1 l_{c1}^2 + m_2 l_1^2 + I_{c1}\\ \theta_2 &= m_2 l_{c2}^2 + I_{c2}\\ \theta_3 &= m_2 l_1 l_{c2}\\ \theta_4 &= m_1 l_{c1} + m_2 l_1\\ \theta_5 &= m_2 l_{c2}\end{aligned} \tag{2}$$

Las variables utilizadas son las siguientes:

- $m_1 =$ masa total del primer eslabón [kg].
- $l_1 =$ longitud del primer eslabón [m].
- $l_{c1} =$ distancia al centro de masa del primer eslabón [m].
- $I_{c1} =$ momento de inercia del primer eslabón respecto al eje que pasa por su centro de masa [$kg\ m^2$].
- $m_2 =$ masa total del segundo eslabón [kg].
- $l_{c2} =$ distancia al centro de masa del segundo eslabón [m].
- $I_{c2} =$ momento de inercia del segundo eslabón respecto al eje que pasa por su centro de masa [m].
- $g =$ la aceleración de la gravedad [m/s^2].

Con los cuales podemos reescribir los elementos de las matrices $M(q)$, $C(q,\dot{q})$ y $g(q)$ de la siguiente manera:

$$\begin{aligned}m_{11} &= \theta_1 + \theta_2 + 2\theta_3 \cos(q_2)\\ m_{12} &= m_{21} = \theta_2 + \theta_3 \cos(q_2)\\ m_{22} &= \theta_2\\ c_{11} &= -\theta_3 sen(q_2) q_2\\ c_{12} &= -\theta_3 sen(q_2) q_1 - \theta_3 sen(q_2) q_2\\ c_{21} &= \theta_3 sen(q_2) q_1\\ c_{22} &= 0\\ g_1 &= \theta_4 g cos(q_1) + \theta_5 g cos(q_1 + q_2)\\ g_2 &= \theta_5 g cos(q_1 + q_2)\\ detM &= m_{11} m_{22} - m_{21} m_{12}\\ T_1 &= \tau - c_{11}\dot{q}_1 - c_{12}\dot{q}_2 - g_1\\ T_2 &= 0 - c_{21}\dot{q}_1 - c_{22}\dot{q}_2 - g_2\end{aligned} \tag{3}$$

quedando las matrices del sistema en forma simplificada de la siguiente forma:

$$M(q) = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \tag{4}$$

$$C(q,\dot{q}) = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \tag{5}$$

$$g(q) = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} \tag{6}$$

Despejando $\ddot{q}$ de la ecuación (1):

$$\ddot{q} = M(q)^{-1}\left[\tau - C(q,\dot{q})\dot{q} - g(q)\right], \tag{7}$$

obtenemos nuestras variables de estado:

$$\begin{aligned}x_1 &= q_1\\ x_2 &= \dot{q}_1\\ x_3 &= q_2\\ x_4 &= \dot{q}_2\\ \dot{x}_1 &= x_2\\ \dot{x}_2 &= (T_1 m_{22} - T_2 m_{12})/detM\\ \dot{x}_3 &= x_4\\ \dot{x}_4 &= (T_2 m_{11} - T_1 m_{21})/detM\end{aligned} \tag{8}$$

III. DEFINICIÓN DEL CONTROL LOCAL

El siguiente paso es analizar el controlador propuesto comprobando su estabilidad, es importante recalcar que el control se relaciona directamente con el modelo dinámico.

Para la realización de este control linealizamos las ecuaciones de movimiento del pendubot, diseñando un controlador por retroalimentación de estados con el modelo lineal. Para linealizar la planta se utilizaron las *series de Taylor*.

$$f_a(x,u) = f_a(x_r,u_r) + \frac{\partial f}{\partial x}\Big|_{x_r,u_r}(x - x_r) + \frac{\partial f}{\partial u}\Big|_{x_r,u_r}(u - u_r) \tag{9}$$

Para analizar al pendubot consideramos dos puntos de equilibrio: el primero se localiza cuando el brazo se encuentra en la posición hacia *arriba*, figura (2), teniendo como puntos de equilibrio $x_{d1} = \pi/2$, $x_{d3} = 0$ y $u_d = 0$

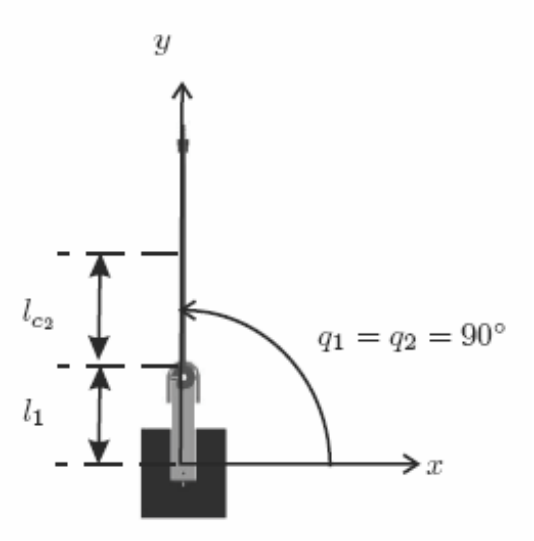
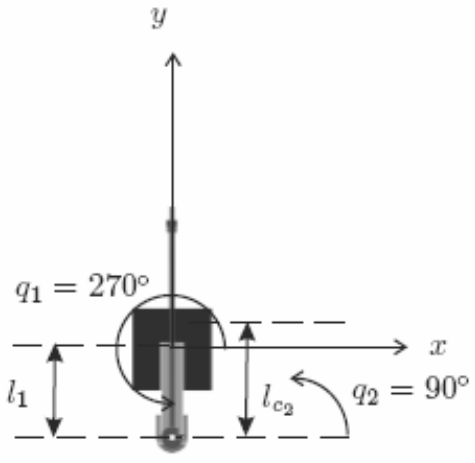


Fig. 2. Péndulo en posición *arriba*.

El segundo punto se encuentra cuando el segundo eslabón queda hacia arriba, a esta posición le llamaremos *media*, figura (3), con los siguientes puntos de equilibrio $x_{d1} = -\pi/2$, $x_{d3} = \pi$ y $u_d = 0$



3. Péndulo posición *media*.

El fabricante proporciona los siguientes parámetros

Tabla 1. Parámetros del fabricante

θ_1	0.0761
θ_2	0.0662
θ_3	0.0316
θ_4	0.9790
θ_5	0.3830

Usando los puntos de equilibrio y los parámetros anteriores, el modelo lineal del péndulo *arriba* y *media* queda descrito de la siguiente forma:

Arriba

$$\dot{x} = Ax + Bu$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0(10) \\ 127.877 & 0 & -29.3637 & 0(11) \\ 0 & 0 & 0 & 1(12) \\ -132.22 & 0 & 100.078 & 0(13) \end{bmatrix} \quad (10)$$

$$B = \begin{bmatrix} 0(15) \\ 16.3891(16) \\ 0(17) \\ -24.2124(18) \end{bmatrix} \quad (11)$$

Media

$$\dot{x} = Ax + Bu$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0(20) \\ -127.877 & 0 & 29.3637 & 0(21) \\ 0 & 0 & 0 & 1(22) \\ 123.534 & 0 & 41.3507 & 0(23) \end{bmatrix} \quad (12)$$

$$B = \begin{bmatrix} 0(25) \\ 16.3891(26) \\ 0(27) \\ -8.5659(28) \end{bmatrix} \quad (13)$$

Las matrices anteriores están en el espacio continuo, pero para nuestro control necesitamos discretizarlas, esto es: $x_{k+1} = Gx_k + Hu_k$ a 0.002 segundos.

III-A. Control de balanceo

Este control esta dividido en dos, el que lo lleva a la posición de equilibrio y el control de balanceo. Para el control de balanceo se utiliza linealización parcial. Entonces si derivamos el control por retroalimentación parcial. Usando la ecuaciones de movimiento del sistema propuestas por Spong y Vidyasagar [1]

$$\tau_1 = m_{11}\ddot{q}_1 + m_{12}\ddot{q}_2 + c_{11}\dot{q}_1 + c_{12}\dot{q}_2 + g_1(q) \tag{14}$$

$$0 = m_{21}\ddot{q}_1 + m_{22}\ddot{q}_2 + c_{21}\dot{q}_1 + g_2(q) \tag{15}$$

Como el eslabón 2 es sub-actuado no se puede linealizar la dinámica sobre ambos grados de libertad. Por lo tanto solo se linealizará el grado de libertad q_1, luego nos ayudamos de un control de lazo abierto que siga la trayectoria dada por el grado de libertad linealizado. Despejando la aceleración angular de (15) obtenemos:

$$\ddot{q}_2 = \frac{-m_{21}\ddot{q}_1 - c_{21}\dot{q}_1 - g_2(q)}{m_{22}} \tag{16}$$

sustituyendo (16) en (14) tenemos

$$\tau_1 = \bar{m}_{11}\ddot{q}_1 + \bar{c}_{11}\dot{q}_1 + \bar{c}_{12}\dot{q}_2 + \bar{g}_1(q) \tag{17}$$

con

$$\bar{m}_{11} = m_{11} - \frac{m_{12}m_{21}}{m_{22}} \tag{18}$$

$$\bar{c}_{11} = c_{11} - \frac{m_{12}c_{21}}{m_{22}} \tag{19}$$

$$\bar{c}_{12} = c_{12} \tag{20}$$

$$\bar{g}_1 = g_1 - \frac{m_{12}g_2}{m_{22}} \tag{21}$$

Con el método de linealización completo, el control de lazo cerrado que linealiza el grado de libertad q_1 se puede definir como:

$$\tau_1 = \bar{m}_{11}v_1 + \bar{c}_{11}\dot{q}_1 + \bar{c}_{12}\dot{q}_2 + \bar{g}_1(q) \tag{22}$$

Ahora si

$$\ddot{q}_1 = v_1 \tag{23}$$

$$-m_{21}v_1 = m_{22}\ddot{q}_2 + c_{21}\dot{q}_1 + g_2(q) \tag{24}$$

Ahora (23) es lineal, entonces se puede implementar un control de lazo cerrado que siga la trayectoria dada por el eslabón uno, por tanto la ecuación no lineal (24) es la respuesta al eslabón dos.

Ésta representa la dinámica interna con respecto a una salida $y = q_1$. El objetivo del control de lazo abierto, es seguir una trayectoria dada por el eslabón uno y al mismo tiempo excitar la dinámica interna para balancearlo y llevarlo a la posición de equilibrio. Se opto por utilizar un PD con retroalimentación directa de aceleración.

$$v_1 = \ddot{q}_1^d + K_d(\dot{q}_1^d - \dot{q}_1) + K_p(q_1^d - q_1) \tag{25}$$

La matriz de ganancias K la podemos construir utilizando el software **Matlab**, para lo cual usamos la instrucción: `K=lqr(A,B,Q,R)` [3] conocemos A y B y proponemos las matrices Q y R.

En nuestro caso utilizamos un control LQR para cada posición, cabe mencionar que dicho control fue discretizado. La matriz de ganancias fue obtenida con un periodo de muestreo de 0.002 segundos.

Para la posición *arriba*:

$$R = [2] \tag{26}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0(43) \\ 0 & 0 & 0 & 0(44) \\ 0 & 0 & 1 & 0(45) \\ 0 & 0 & 0 & 0(46) \end{bmatrix} \tag{27}$$

Se obtuvieron las siguientes ganancias

$$K_{arriba} = \begin{bmatrix} -103.1041 & -17.4983 & -103.4362 & -13.4985 \end{bmatrix} \tag{28}$$

Para la posición *media*:

$$R = [1] \tag{29}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0(2) \\ 0 & 1 & 0 & 0(3) \\ 0 & 0 & 1 & 0(4) \\ 0 & 0 & 0 & 1(5) \end{bmatrix} \tag{30}$$

dándonos una matriz de ganancias

$$K_{media} = \begin{bmatrix} 70.4404 & 8.0650 & 88.9295 & 11.6103 \end{bmatrix} \tag{31}$$

IV. REPRESENTACIÓN GRÁFICA

Después de analizar el sistema y obtener el modelo dinámico y la estructura de control, el siguiente paso consiste en bosquejar el prototipo a través de sus vistas frontal, lateral y superior, las cuales nos sirven para tener una referencia del sistema en el plano (x, y, z). Entre más detallado este el dibujo obtendremos una mejor representación grafica del sistema. El diseño asistido por computador (Computer Aided Design por sus siglas en inglés), es el uso de un amplio rango de herramientas computacionales que se utilizan para la creación de entidades geométricas. La herramienta CAD que se utiliza en el diseño del simulador es *AutoCAD*, programa de diseño para dibujo en 2D y 3D desarrollado por la empresa *Autodesk*. Los dibujos que definen al pendubot (vista frontal, lateral y superior) se encuentran en centímetros.

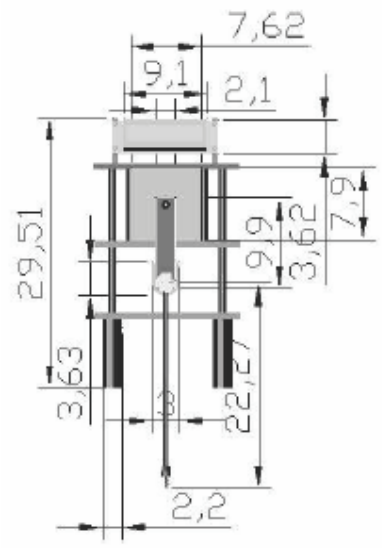


Fig 4. Vista frontal.

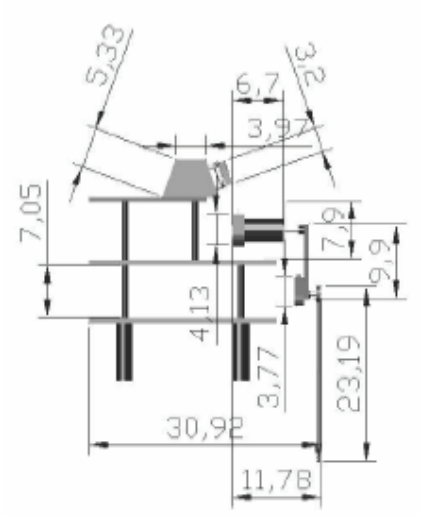


Fig. 5. Vista lateral.

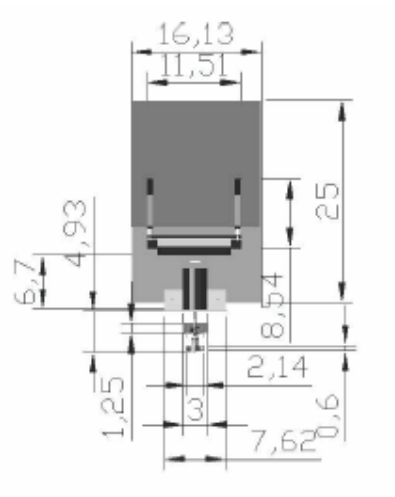


Fig. 6. Vista superior.

Una vez que se bosqueja el prototipo tridimensionalmente se procede a definir el simulador pieza por pieza, ya que el movimiento del prototipo está referenciado a los eslabones. Por tal motivo es necesario identificar *piezas clave* que funcionen como referencia para un movimiento por bloques. Definidas todas las piezas clave se procede a transformarlas en una estructura o fichero de datos que represente mediante una rejilla rectangular de pixeles o puntos de color un formato de imagen (BITMAP, PGM o 3DS).

Al realizar esta transformación, los componentes de la imagen (cabecera, identificador, tamaño y cuerpo de la imagen) se encuentran en un archivo encriptado en código ASCII o binario. En este caso ocupamos el formato 3DS debido que el programa Autocad permite transformar imágenes DWG a 3DS.

V. PROGRAMACIÓN DEL SIMULADOR

El siguiente paso consiste en relacionar mediante un lenguaje de programación las diferentes entidades tanto gráficas como matemáticas para poder describir el comportamiento del prototipo.

V-A. OpenGL

OpenGL es una especificación estándar que define una Interfaz de Programación de Aplicaciones (Application Programming Interface) multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

Fue desarrollada por *Silicon Graphics Inc.* en 1992. Su nombre viene del inglés *Open Graphics Library*, cuya traducción es librería de gráficos libre.

A grandes rasgos, OpenGL es una especificación, es decir, un documento que describe un conjunto de funciones y su comportamiento exacto. A partir de estas especificaciones se crean implementaciones (bibliotecas de funciones creadas para enlazar con las funciones de la especificación OpenGL). La compatibilidad de OpenGL abarca desde Borland C, Visual C, Java, visual Fortran entre otros. [1]

El dibujo en OpenGL se basa en la composición de pequeños elementos, con los que vamos construyendo la escena deseada, estos elementos se llaman *primitivas*.

Todas las primitivas de OpenGL son objetos de una o dos dimensiones, abarcando desde puntos simples a líneas o polígonos complejos. Para definir el área de trabajo en OpenGL hay que imaginar un volumen con perspectiva ortonormal, como se puede observar en la figura (7).

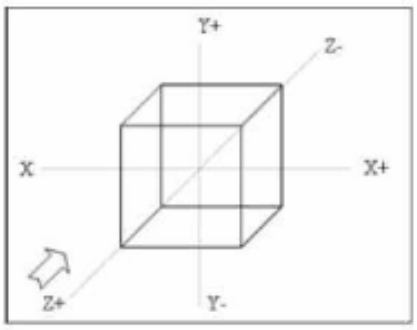


Fig. 7. Definición del espacio tridimencional en OpenGL

Es importante ubicar el sólido, una vez hecho esto, se verifica con *3D studio* el origen del objeto y se procede a crear un nuevo proyecto en Visual C, tomando en cuenta que debe estar incluida la librería *glut.dll* en la carpeta de *system* y *system32* de windows; posteriormente en el proyecto se incorporan las librerías de acuerdo al diagrama a bloque de la figura (8).

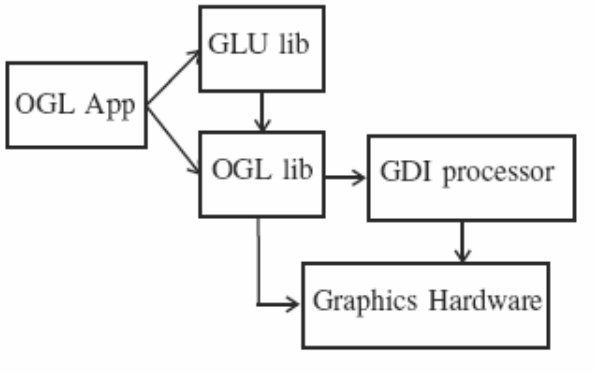


Fig. 8. Procedimiento de cargado de librerías.

Las librerías que se agregan al proyecto son: la *GLU* que contiene funciones gráficas de más alto nivel, que permiten realizar operaciones complejas; y la *GLUT* que es un paquete auxiliar para construir aplicaciones de ventanas, además de incluir algunas primitivas geométricas auxiliares. La interacción de OpenGL con Visual C++ se describe en el siguiente diagrama, figura 9.

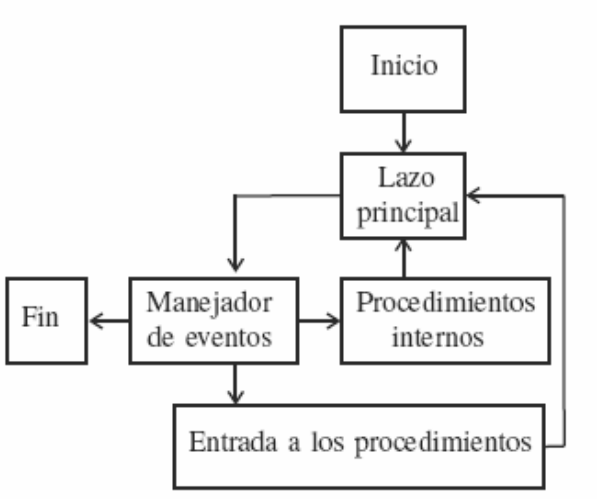


Fig. 9. Diagrama a bloques.

A continuación se enlistan algunas operaciones importantes:

7. Inicializar el modo de visualización.
8. Inicializar el tamaño de la ventana.
9. Inicializar la posición de la ventana.
10. Crear la ventana.
11. Definir el color de fondo.
12. Activar el test del buffer de profundidad.

13. Definir la función para redibujar la ventana.
14. Para generar nuevas ventanas en Open GL se utiliza la instrucción $glviewport(x, y, cx, cy)$ donde (x, y) son la posición de la nueva ventana y (cx, cy) son el tamaño de esta.
15. Finalmente, tenemos que hacer que el bucle de eventos se ponga en marcha.

En OpenGL para manipular imagenes en tres dimensiones se necesitan definir las siguientes operaciones matriciales:

- Rotación: se define mediante la primitiva $glRotatef(tetha, x, y, z)$. Esta función multiplica la matriz actual por una matriz de rotación de tetha grados respecto al eje (x, y, z).

- Translación: Se define mediante la primitiva $glTranslatef(x, y, z)$. Aplica una translación en (x, y, z) sobre la matriz actual.

- Escalado: Se define mediante la primitiva $glScalef(sx, sy, sz)$ Escalado de cada uno de los ejes.

La función $glPushMatrix$ nos permite guardar la matriz actual en la pila, mientras que la función $glPopMatrix$ toma la matriz que se encuentre en el *top* de la pila y la asigna a la matriz actual [2].

Todas estas funciones nos sirven para el tratamiento da la imagen descargada, esta descarga se realiza en una clase donde contendrá variables de tipo puntero las cuales reservaran memoria para que estas sean tratadas.

Esta *clase* debe ser declarada en la cabecera del proyecto principal, las cuales entregaran dos tipos: .cpp y .h; en los archivos .cpp estará todo lo referido a la descarga de imágenes, y en los archivos .h estarán todas las variables que se utilizan para la descarga del archivo 3DS.

V-B. Visual C++

Visual C++ es un lenguaje de programación, esta especialmente diseñado para el desarrollo y depuración de código escrito para las API's de Microsoft Windows, DirectX y la tecnología Microsoft .NET Framework. Existen ciertas herramientas en *Visual C++* para construir un simulador como la *clase*, la cual contiene atributos, métodos, mensajes y eventos. Los pasos para programar un simulador se describen en el diagrama de flujo de la figura 10.

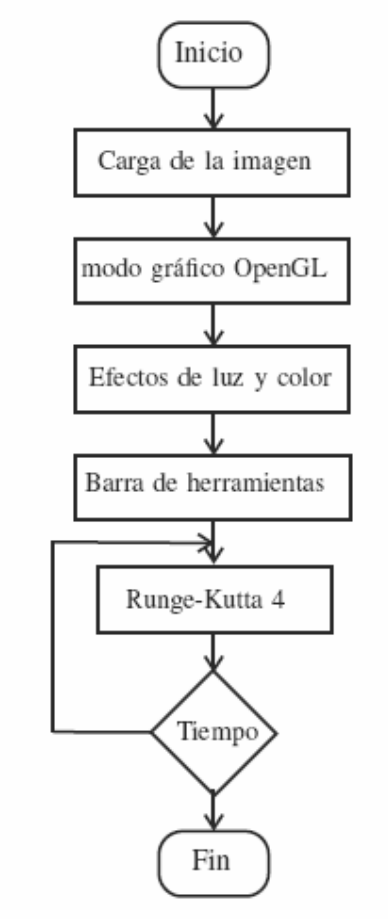


Fig. 10. Diagrama de flujo.

Para comprender lo antes expuesto debemos tener en claro algunas definiciones como:

- Objetos: Un programa orientado a objetos se compone solamente de objetos, donde este es una encapsulación genérica de datos y de los procedimientos para manipularlos.

- Mensajes: Cuando un programa orientado a objetos se ejecuta, los objetos están recibiendo, interpretando y respondiendo a mensajes de otros objetos, lo que origina cambios en el estado del objeto.

- Métodos: Un método se implementa en una clase y determina como tiene que actuar el objeto cuando recibe un mensaje.

- Clases: Una clase se puede considerar como una plantilla para crear objetos de esa clase o tipo; esta describe los métodos y los atributos que definen las características comunes a todos los objetos de esa clase. [3]

Para el proyecto se utiliza el *MFC ClassWizzard* el cual es un asistente que despliega los recursos para poder diseñar el proyecto; generando una clase para la manipulación de los gráficos (clase vista o View); en esta clase incluimos las cabeceras necesarias para el procesado grafico, y los mensajes *OnCreate, OnDraw, OnPaint, OnInitialUpdate, OnSize, OnEraseBkgnd, OnLButtonDown, OnLButtonUp, OnMouseMove*.

V-C. Cargar el Modelo Dinámico

Para cargar el modelo dinámico en Visual C++ es necesario crear una nueva clase, la cual se denomina *modelo dinámico*, en esta clase utilizamos el algoritmo de *Runge Kutta de Cuarto Orden*, el algoritmo se aplica a las formas de Cauchy del pendubot. La figura 11 muestra la pantalla del simulador.

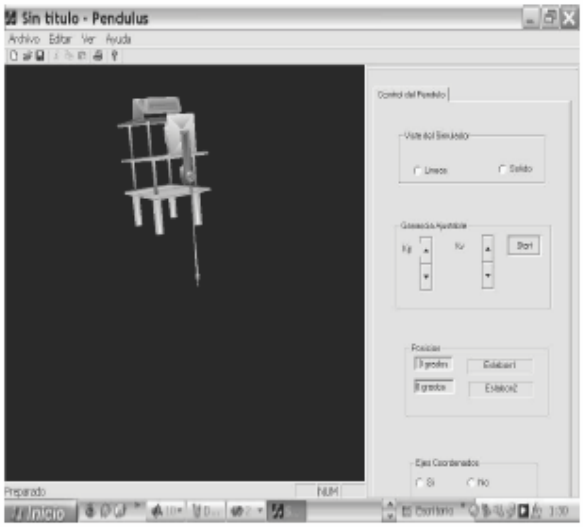


Fig. 11. Simulador.

VI. RESULTADOS

Los resultados obtenidos por el simulador a través de un archivo de datos se ilustra en la figura 9. En la figura 9 el eslabón uno está en $\pi/2$ y el eslabón dos está en una posición 0, donde se realiza el control de balanceo y cuando se estabiliza se inicia el control de equilibrio.

En la figuar 10 el eslabón uno y el dos entran a $\pi/2$, donde la transiente es el control de posición; note que el eslabón dos entrega la energía necesaria para que el eslabón uno entre en equilibrio.

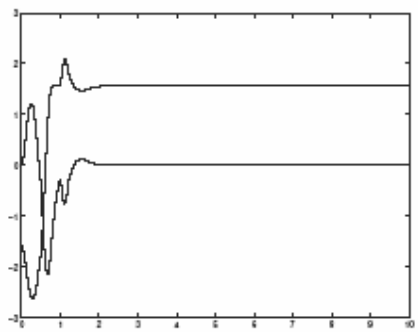


Fig. 12. Comportamiento 1.

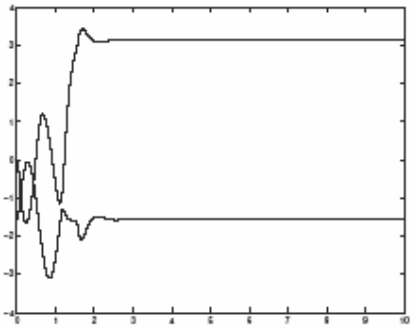


Fig. 13. Comportamiento 2.

VII. CONCLUSIONES

Al probar el método de Rugen Kutta notamos que opero a la perfección después de dos corridas, es decir todavía se esta estudiando nuevas formas de hacer algoritmos numéricos para la solución de las formas de Cauchy del modelo dinámico.

REFERENCIAS

[1] Dorado de la Calle Julian . *Una Aproximación a Open GL*. (Universidad de Cataluña España).
[2] Neider Jackie, .OpenGL Programming Guide (RED BOOK)., *Addison Wesley Publishing Company*, Massachusetts Institute of Technology .
[3] Ceballos Sierra Fco. Javier. *Programación Orientada a Objetos con C++.* México D.F.: Alfaomega Ra-Ma, 1998, Capítulos 3, 6, 10, 17.
[4] Spong,M.W., *Swing Up Control of the Acrobot Using Partial Feedback Linearization*, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1994.