TP 2: Análisis dinámico del Robot M5

Robótica

Prof.: MaS. Ing. Hernan Gianetta

JTP: Ing. Damián Granzella

TP: 2

Alumnos:

Arcusin Leandro

Statello Emiliano

Contenidos

Introducción general

El objetivo de este trabajo es hacer uso de las herramientas de análisis dinámico que tiene la robótica para analizar un diseño de un robot, en el caso particular será el M5 de SkyMec.

El trabajo se divide en 3 partes. La primera es una introducción teórica a las herramientas de análisis dinámico, la segunda es un ejemplo de análisis dinámico resuelto en Matlab con las herramientas del toolbox Corke y la tercera es una implementación del control de motores BLDC hecha en VHDL. Este último punto esta resuelto con herramientas de código abierto y se entrega en este mismo trabajo un pequeño manual dónde se explica como realizar las simulaciones.

Herramientas de análisis dinámico

La dinámica es la rama de la física que estudia los movimientos y las causas que los producen. El objetivo de la dinámica es describir los factores capaces de producir alteraciones de un sistema físico, cuantificarlos y plantear ecuaciones de movimiento o ecuaciones de evolución para dicho sistema. En nuestro caso un robot de 5 GDL.

El objetivo último del análisis dinámico será la generación de las señales de control para los motores que gobiernen nuestro robot.

Estas ecuaciones relacionan matemáticamente la localización del robot definida por sus variables articulares o por las coordenadas de localización de su extremo, y sus derivadas: velocidad y aceleración. Las fuerzas y pares aplicados en las articulaciones (o en el extremo del robot). Los parámetros dimensionales del robot, como longitud, masas e inercias de sus elementos.

Como es de suponer, la complejidad para obtener estas relaciones aumenta conforme aumenta la complejidad del robot. Tanto es así que a veces no se puede obtener una ecuación diferencial de segundo orden que resuelva nuestro modelo sino que se deben apelar al cálculo numérico iterativo para resolverlo.

Parámetros importantes en el análisis dinámico

Centro de Masa

El centro de masa se calcula como:

$$\mathbf{r}_{\mathrm{cm}} = \frac{\int \mathbf{r}\ dm}{\int dm} = \frac{1}{M}\int \mathbf{r}\ dm$$

y es el punto donde la energía potencial se puede considerar cómo:

$$K = \frac{1}{2}mv^2 + K_{rot}$$

Siendo “m” la masa total del cuerpo, “v” la velocidad de traslación del centro de masas y Krot la energía de rotación del cuerpo, expresable en términos de la velocidad angular y el tensor de inercia

Momento Angular

El momento angular es una magnitud física importante porque en muchos sistemas físicos constituye una magnitud conservada, a la cual bajo ciertas condiciones sobre las fuerzas es posible asociarle una ley de conservación.
Se define como:

$$\mathbf{L}_O = \int_V \rho(\mathbf{r}_O \times \mathbf{v}_O) \quad dV$$

Momento de Inercia

Es una medida de la inercia rotacional. Es análogo a la masa de un cuerpo en un sistema traslacional y permite relacionar el momento aplicado a dicho cuerpo con la aceleración angular que va a desarrollar.

$$\mathbf{L} = \mathbf{r} \times \mathbf{p} = \mathbf{r} \times m\mathbf{v}$$

Tensor de Inercia

Es una entidad que caracteriza la inercia rotacional de un sólido rígido. A partir del tensor de inercia, cuyas componentes se calculan:

$$\begin{cases} I_{xx} = \int_M d_x^2 dm = \int_V \rho(y^2+z^2)dxdydz \\ I_{yy} = \int_M d_y^2 dm = \int_V \rho(z^2+x^2)dxdydz \\ I_{zz} = \int_M d_z^2 dm = \int_V \rho(x^2+y^2)dxdydz \end{cases}$$

se puede calcular la energía de rotación:

$$E_{rot} = \frac{1}{2}\begin{pmatrix}\Omega_x & \Omega_y & \Omega_z\end{pmatrix}\begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz}\end{pmatrix}\begin{pmatrix}\Omega_x \\ \Omega_y \\ \Omega_z\end{pmatrix} = \frac{1}{2}\sum_j\sum_k I_{jk}\Omega_j\Omega_k$$

Fuerza Centrípeta

La fuerza centrípeta aparece para producir una aceleración a un cuerpo que está girando. Su expresión es la siguiente:

$$\mathbf{a} = -\frac{v^2}{r}\left(\frac{\mathbf{r}}{r}\right) = -\frac{v^2}{r}\hat{\mathbf{u}}_r = -\omega^2\mathbf{r}$$

Fuerza de Coriolis

La fuerza de Coriolis es una fuerza ficticia que aparece cuando un cuerpo está en movimiento con respecto a un sistema en rotación y se describe su movimiento en ese referencial. La fuerza de Coriolis es diferente de la fuerza centrífuga. La fuerza de Coriolis siempre es perpendicular a la dirección del eje de rotación del sistema y a la dirección del movimiento del cuerpo vista desde el sistema en rotación.

$$\vec{F}_c = 2m\left(\vec{v} \times \vec{\omega}\right),$$

Ecuación del Torque

Esta ecuación es la más importante, y es la que permite relacionar todo lo anterior.

$$\tau = \mathbf{D}\ddot{\mathbf{q}} + \mathbf{H} + \mathbf{C}$$

Y su resolución la haremos a través del toolbox Corke que calcula los valores a través de un método llamado Recursive Newton-Euler.

Análisis dinámico del Robot M5

GHDL

Introducción a GHDL

GHDL es un compilador del lenguaje VHDL.

Toda la información encontrada en este manual fue tomada de http://ghdl.free.fr el sitio oficial del proyecto.

Instalación

Para instalar ghdl en ubuntu ingresamos en una consola:

```
sudo apt-get install ghdl
```

también utilizaremos el GTKWave para visuarlizar las formas de onda generadas

```
sudo apt-get install gtkwave
```

Se recomienda crear una estructura de archivos como la siguiente:

`/directorio_de_trabajo`
`/directorio_de_trabajo/work`: directorio para los archivos de trabajo del ghdl, aquí guarda los productos intermedios de la compilación.
`/directorio_de_trabajo/src`: archivos fuente de nuestro diseño.
`/directorio_de_trabajo/tbench`: archivos de testbench para nuestro diseño.

Comandos de GHDL

La sintáxis para invocar comandos de ghdl es la siguiente:

```
ghdl -comando [--opciones]
```

Análisis

```
ghdl -a [opciones] archivos
```

Compila los archivos indicados y genera un archivo de código objeto por cada uno de ellos.

Elaboración

```
ghdl -e [opciones ] unidad_principal [unidad_secundaria ]
```

Este comando crea un archivo ejecutable (sólo en Linux) conteniendo el código VHDL de los archivos fuente y los archivos de simulación.

Se puede especificar el nombre de la librería de trabajo con la opción
`--work=NOMBRE`
y se puede definir el directorio de trabajo con la opción
`--workdir=DIRECTORIO`

Chequear la Sintaxis

```
ghdl -a [opciones] archivos
```

Chequea la sintaxis de un determinado archivo, sin generar código objeto.

Ejecutar

```
ghdl -r unidad_principal [opciones_de_simulacion]
```
o simplemente
```
./unidad_principal
```

La opción `--vcd =archivo` genera un archivo de salida con las formas de onda.

Importar

```
ghdl -i [opciones] archivos
```

Agrega archivos a la librería de trabajo. En un proyecto con múltiples archivos (fuentes, simulación, funciones) es conveniente utilizar esta opción.

Make

```
ghdl -m [opciones] unidad_principal
```

Analiza automáticamente los archivos desactualizados y elabora un diseño.

Los archivos tiene que haber sido previamente incluidos en la librería de trabajo, ya sea con la función Importar o Analizar.

Ejemplo 1: Hola Mundo

El siguiente ejemplo tiene el objetivo de demostrar el uso de los comandos vistos recientemente. (El programa fue tomado del manual de GHDL, que puede encontrarse en el sitio del proyecto: http://ghdl.free.fr

Como primer paso crearemos la estructura de directorios para trabajar, elegimos el directorio donde queremos guardar el proyecto y creamos la siguiente estructura:

```
/helloworld/src
/helloworld/work
/helloworld/tbench
```

Este último no lo vamos a utilizar en este ejemplo, pero lo creamos igualmente.

Copiamos el siguiente código en un editor de texto y lo guardamos en */helloworld/src* con el nombre *hello.vhdl*

```
-- Hello world program.

-- Defines a design entity, without any ports.
use std.textio.all; --Imports the standard textio package.

entity hello_world is
end hello_world;

architecture behaviour of hello_world is
begin
     process
          variable l : line;
     begin
          write (l, String'("Hello world!"));
          writeline (output, l);
          wait;
     end process;
end behaviour;
```

Análisis

Para analizar el código nos paramos en el directorio /helloworld y en una terminal ingresamos:

```
ghdl -a --workdir=work  ./src/*.vhdl
```

Si todo salió correctamente no deberíamos obtener ninguna respuesta. Sino probablemente haya algún error en el código del archivo o en el comando ingresado.

Entonces si listamos los archivos en /helloworld/work vemos los archivos:

```
hello.o
work-obj93.cf
```

Donde *hello.o* es el código objeto y el archivo *work-obj93.cf* es el archivo de trabajo para éste proyecto. El contenido de este último archivo no es objeto de análisis de este trabajo.

Elaboración

Luego creamos el archivo ejecutable:

```
ghdl -e --workdir=work hello_world
```

Aparece en el directorio principal del proyecto un archivo llamado hello_wolrd, que es ejecutable.
Notamos que el objeto de la compilación se llama **hello_world** en vez de **hello.o** pues el parámetro que recibe el comando ghdl -e es el nombre de la entidad que está compilando, **no del archivo.**

Ejecución

Para ejecutar el ejemplo tipeamos

```
ghdl -r hello_world
```

o simplemente

```
./hello_world
```

Ejemplo 2: Atmel_PWM

En este ejemplo tendremos la oportunidad de ver el uso de GHDL para un proyecto más grande que el anterior, con varios archivos y visualización de de señales.

Los achivos vhdl con los que vamos a trabajar se encuentran en el campus virtual de la materia y son los archivos:

pwm_fpga.vhd

user_pkg_inc_dec.vhd

pwm_pretb.vhd

Dejamos afuera el archivo pwm_posttb.vhd porque en nuestro caso no estamos sintetizando el diseño, sólo simulandolo, de forma que no tiene sentido hacer este análisis.

Estructura de trabajo

El directorio de trabajo sera *pwmrobot*. Alli creamos los directorios *src, tbench y work*.

En el directorio *src* copiamos los archivos *pwm_fpga.vhd* y *user_pkg_inc_dec.vhd*.

En el directorio tbench copiamos el archivo *pwm_pretb.vhd.*

Creación de la librería

En este ejemplo podemos ver que hay una dependencia jerárquica: el módulo de testbench usa una instancia de la entidad pwm_fpga que ni siquiera se encuentra en el mismo archivo.

Para salvar los conflictos que pudiera generar esta jerarquia vamos a *importar* los archivos a una librería de trabajo con el siguiente comando:

```
ghdl -i --workdir=work ./src/*.vhd ./tbench/*.vhd
```

Si listamos el contenido del directorio work, entonces encontraremos el archivo work-obj93.cf pero ningun archivo objeto, pues sólo definimos los archivos con los que vamos a trabajar, pero no los compilamos todavía.

Compilación

Ahora vamos a compilar los archivos y crear un ejecutable con el comando *make*.

La entidad que queremos elaborar (es decir, el nombre de la entidad que define el archivo de testbench) se llama pwm_fpga_test_bench y este será el obejto de nuestra compilación.

El comando es:

```
ghdl -m --ieee=synopsys --workdir=work pwm_fpga_test_bench
```

Dónde la opción --ieee=synopsys indica al GHDL que incluya algunas funciones no estandar de la librería ieee como *'std_logic_arith', 'std_logic_signed', 'std_logic_unsigned', 'std_logic_textio'* [según el manual de GHDL].

Ahora si listamos los archivos veremos que en el directorio principal (pwmrobot) se creó el archivo ejecutable *pwm_fpga_test_bench* y en el directorio */pwmrobot/work* se crearon los archivos objeto (**.o*))para cáda archivo fuente.

Ejecución

Si analizamos el archivo de testbench, podemos notar que el tiempo de ejecución es de 150,1 µs:

```
    sig_reset <= '1';
    wait for 100 ns;
    sig_reset <= '0';
    sig_data_value <= "11000000";
    wait for 50 us;
    sig_data_value <= "10000000";
    wait for 50 us;
    sig_data_value <= "01000000";
    wait for 50 us;
```

Sin embargo, la simulación no es capaz de interpretar esto para finalizar por si misma, sino que tendremos que indicarle el tiempo por el que se debe extender. Esto se hace con la opción *--stop-time=TIEMPO*

Entonces corremos la simulación con:

```
./pwm_fpga_test_bench --stop-time=160us --vcd=signals.vcd
```

La simulación termina con un mensaje:

```
./pwm_fpga_test_bench:info: simulation stopped by --stop-time
```

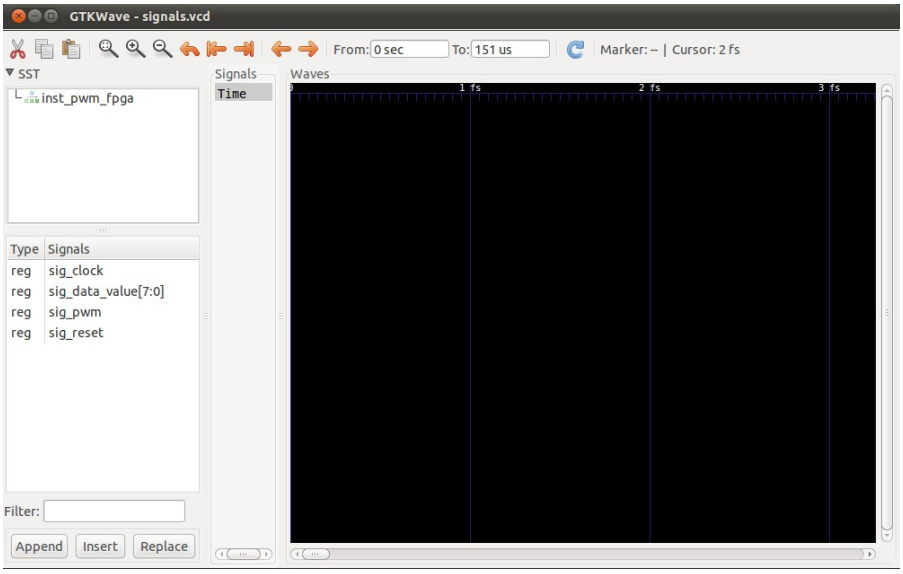
y en el directorio de trabajo se crea el archivo *signals.vcd.*

Visualización de la señales

Para visualizar las señales generadas por la simulación corremos el programa GTKWave

```
gtkwave signals.vcd
```

Vemos una pantalla como la siguiente:



Ninguna señal se muestra, pues tenemos que agregarlas con el botón append que aparece abajo a la izquierda.

Una vez seleccionadas podemos hacer click en el botón de escala “FIT” para visualizar las señales a lo largo de toda la pantalla.

No hacemos una descripción del GTKWave pues su uso es bastante intuitivo y escapa al objetivo de este manual.

Una vez que hayamos agregado las señales y acomodado la escala, podemos analizar el funcionamiento de nuestro modelo: