ANEXO B. FUNCIONES DE MATLAB PARA EL PROCESAMIENTO DE SEÑALES.

De las muchas funciones que posee la herramienta computacional Matlab para el procesamiento matemático de señales, solo unas cuantas son usadas en el sistema desarrollado, pero dado el beneficio obtenido mediante su uso es conveniente conocer la forma como operan y la forma como retornan sus valores de salida. Gracias a la función *wavread* es posible convertir los comandos capturados en vectores para realizar las operaciones matemáticas sobre estos mediante el uso de otras funciones. El procesamiento espectral se soporta en la FFT que es realizado por la función *specgram.* Además las funciones anteriores, *norm* y *mean* fueron usadas con éxito en la obtención de los vectores de entrenamiento y simulación, pues sus efectos están orientados al trabajo con vectores y matrices ahorrando el trabajo de implementar los procedimientos que accesan a cada una de sus componentes.

Además de las funciones usadas en el sistema, en la etapa de desarrollo fueron empleadas con éxito otras funciones para la obtención de la transformada rápida de Fourier y su inversa, las cuales son `fft` y `ifft` respectivamente. Esta labor se hizo con el objetivo de comprender que el tratamiento hecho por Matlab genera el mismo resultado del concepto matemático que la define y de esta forma poder abarcar el significado más amplio de `specgram`.

Todas estas funciones están ilustradas mediante los ejemplos que se muestran a continuación. Estos están incluidos en la carpeta ejemplos del CD de la aplicación.

wavread

Lee un archivo de sonido WAVE (wav) de Microsoft.

Sintaxis

```
y = wavread('filename')
[y,Fs,bits] = wavread('filename')
[y,Fs,bits] = wavread('filename',N)
[y,Fs,bits] = wavread('filename',[N1 N2])
[y,Fs,bits] = wavread('filename','size')
```

Descripción

`wavread` soporta datos multicanales, por encima a 16 bits por muestra.

y = wavread('filename') carga un archivo WAVE especificado por la cadena filename, retornando los datos muestreados en y. La extensión .wav es añadida si no es dada. Los valores de amplitud están en el rango [-1,+1].

`[y,Fs,bits] = wavread('filename')` retorna la taza de muestreo `(Fs)` en Hertz y el número de bits por muestra usados para codificar los datos en el archivo.

`[y,Fs,bits] = wavread('filename',N)` retorna sólo las primeras `N` muestras de cada canal en el archivo.

`[y,Fs,bits] = wavread('filename',[N1 N2])` retorna solo las muestras comprendidas entre `N1` y `N2` de cada canal en el archivo.

`siz = wavread('filename','size')` retorna el tamaño de los datos de audio contenidos en el archivo en lugar del dato de audio actual, retornando el vector siz = [muestras canales].

specgram

Análisis frecuencial dependiente del tiempo (espectrograma) de una señal.

Sintaxis

```
B = specgram(a)
B = specgram(a,nfft)
[B,f] = specgram(a,nfft,Fs)
[B,f,t] = specgram(a,nfft,Fs)
B = specgram(a,nfft,Fs,window)
B = specgram(a,nfft,Fs,window,noverlap)
B = specgram(a,f,Fs,window,noverlap)
specgram(a)
```

Descripción

`specgram` calcula la transformada de Fourier discreta en el tiempo de una señal usando una ventana corrediza. El espectrograma es la magnitud de esta función.

`B = specgram(a)` calcula el espectrograma para la señal representada por el vector `a`. Esta sintaxis usa los valores por defecto:

- `nfft = min(256,length(a))` especifica la longitud de FFT que specgram usa. Este valor determina las frecuencias en la cual la transformada de Fourier discreta en el tiempo es calculada, en donde la expresión `nfft = min(256,length(a))` usada por Matlab indica que `nfft` es igual al menor valor entre 256 y el número de elementos de `a`.
- `Fs = 2` es un escalar que especifica las muestras frecuenciales.

- `window = hanning(nfft)` especifica una función de ventana hanning y un número `nfft` de muestras que `specgram` usa en el seccionamiento del vector `a`.
- `noverlap = length(window)/2` es el numero de muestras para la secciones de traslape.

Cualquier argumento que usted omita de la lista de parámetros finales de entrada será reemplazado por los valores de defecto como se muestra mas adelante.

Si el vector `a` es un vector de n componentes reales, `specgram` calcula la transformada de Fourier discreta en el tiempo en frecuencias positivas solamente. Si n es par, specgram retorna nfft/2+1 filas (incluyendo el cero y los términos de la frecuencia de Nyquist) en la matriz `B`. Si n es impar, specgram retorna nfft/2 filas. El número de columnas en la matriz `B` es:

```
k = fix((n-noverlap)/(length(window)-noverlap))
```

Si `a` es un vector de componentes complejas, `specgram` calcula la transformada de Fourier discreta en el tiempo en frecuencias positivas y negativas. En este caso, `B` es una matriz compleja con `nfft` filas. El tiempo aumenta linealmente sobre las columnas de `B` de izquierda a derecha, iniciando con la muestra uno en la columna uno. La frecuencia aumenta linealmente al bajar las filas, iniciando en cero.

`B = specgram(a,nfft)` usa la longitud FFT especificada `nfft` en sus cálculos. `nfft` se especifica como una potencia de 2 para la ejecución del algoritmo de la transformada rápida de Fourier.

`[B,f] = specgram(a,nfft,Fs)` retorna un vector `f` de frecuencias en las cuales la función calcula la transformada de Fourier discreta en el tiempo. `Fs` no tiene efecto en la salida de la matriz `B`, esto es una frecuencia múltiplo escalar.

`[B,f,t] = specgram(a,nfft,Fs)` retorna los vectores de frecuencia y tiempo `f` y `t` respectivamente. `t` es un vector columna de tiempos escalares, con longitud igual al número de columnas de `B`. `t(j)` es el tiempo mas próximo en el cual la *j*-ésima ventana intercepta al vector `a`. `t(1)` es siempre igual a cero.

`B = specgram(a,nfft,Fs,window)` especifica una función de ventana y el número de muestras por sección del vector `a`. Si usted proporciona un escalar para `window`, `specgram` usa una ventana hanning de esa longitud. La longitud de la ventana debe ser menor o igual que `nfft`; `specgram` tiende aumentar las secciones si la longitud de la ventana excede a `nfft`.

`B = specgram(a,nfft,Fs,window,noverlap)` traslapa las secciones del vector a por número de muestras igual a `noverlap`.

Usted puede usar la matriz vacía [ ] para especificar el valor por defecto para cualquier argumento de entrada. Por ejemplo,

```
B = specgram(a,[],10000)
```

es equivalente a:

```
B = specgram(a)
```

Pero con una frecuencia de muestreo de 10,000 Hz en lugar de la de defecto de 2 Hz.

`specgram` sin los argumentos de salida despliega la escala logarítmica del espectrograma en la ventana actual usando:

```
imagesc(t,f,20*log10(abs(b))),axis xy,colormap(jet).
```

El modo `axis xy` despliega el contenido de baja frecuencia de la primera porción de la señal en la esquina inferior izquierda de los ejes. `specgram` usa `Fs` para etiquetar los ejes acorde al tiempo y frecuencia reales.

`B = specgram(a,f,Fs,window,noverlap)` calcula el espectrograma en las frecuencias especificadas en `f`, usando la transformada chirp z (para mas de 20 frecuencias igualmente espaciadas) o una disminución de pendiente de filtro polifase. `f` es un vector de frecuencias en Hertz; este vector debe tener al menos dos elementos.

Algoritmo

`specgram` calcula el espectrograma para una señal dada como sigue:

1. Inicialmente `specgram` separa la señal en secciones traslapando y aplicando la ventana especificada por el parámetro `window` para cada sección.
2. Luego `specgram` calcula la transformada de Fourier discreta en el tiempo de cada sección con una longitud `nfft` para producir una estimación de los términos cortos de frecuencia contenidos en la señal. Estas transformadas constituyen las columnas de `B`. `specgram` tiende a aumentar las secciones si `nfft>length(window)`, así que la cantidad `(length(window)-noverlap)` especifica a cuantas muestras `specgram` corre la ventana.
3. Para entrada real, `specgram` trunca el espectrograma para los primeros nfft/2 + 1 puntos para nfft par y (nfft + 1)/2 para nfft impar.

Ejemplo 1

En el sistema desarrollado el procesamiento espectral es realizado mediante la función `specgram`. Tomemos el comando Luz digitalizado y calculemos su espectrograma con los parámetros que se usaron en el sistema:

```
Y=wavread('LuzVMC.wav')
specgram(Y,256,11025,hamming(256),0)
title('Espectrograma del comando Luz')
```

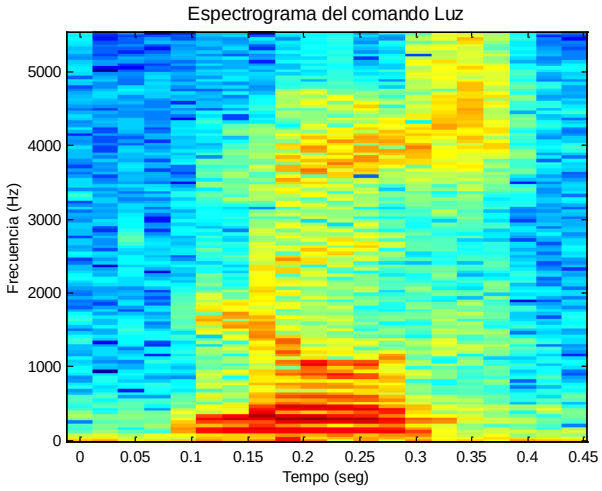

Figura 1. Espectrograma para el comando Luz.

Ejemplo 2

En los cursos sobre teoría de señales se estudio que el pulso rectangular tiene un espectro continuo determinado por la función senc $f\tau$. Calculemos el espectro de esta señal usando `specgram`.

Generemos primero el pulso rectangular de 0.5 mseg. de ancho con una frecuencia de muestreo de 11025 Hz:

```
Fs=11025;
t= 0: 1/Fs: 0.0005; %Define el ancho del pulso
x=square(2*pi*1000*t);
plot(t,x);
title('pulso rectangular')
pause;
```

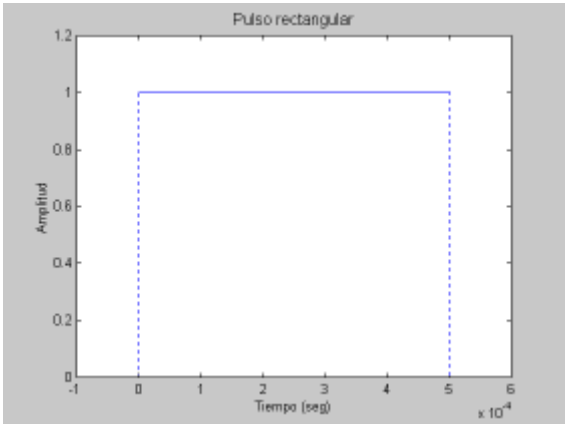

Figura 2. Pulso rectangular

Luego se procede a calcular el espectro para graficar los valores de `B` y `F`.

```
[B,F,t] = specgram(Y,256,[11025],Hamming(256),0);
plot(F,B)
title('Señal senc')
pause;
```

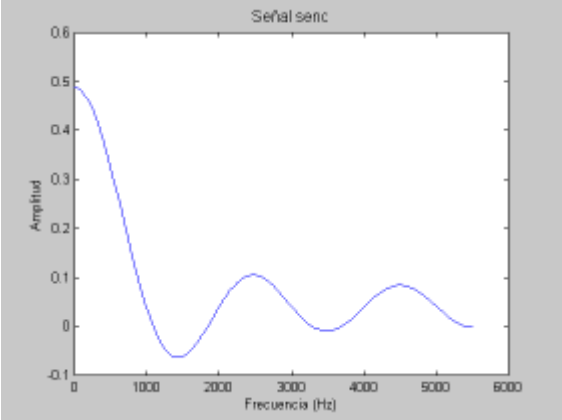

Figura 3. señal obtenida a partir de `specgram`.

Finalmente se obtiene la magnitud para calcular el espectro del pulso rectangular.

```
V=abs(B);
plot(F,V)
title('Espectro del pulso rectangular')
pause;
```

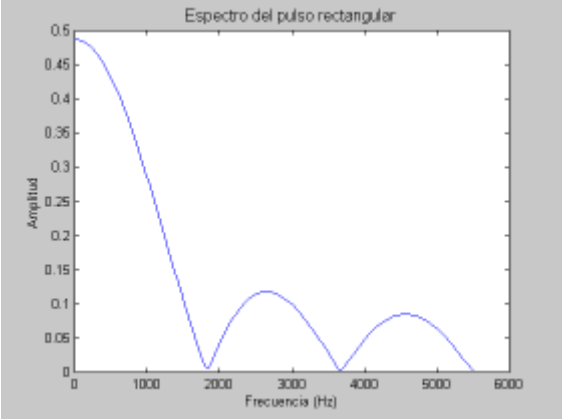

Figura 4. Espectro del pulso rectangular.

Un aspecto importante a tener en cuenta en este ejemplo es el calculo del espectro para las frecuencias positivas, pues `specgram` genera estos resultados debido a que el pulso rectangular esta representado por el vector `Y` de componentes reales.

Diagnósticos

Un mensaje de diagnostico apropiado es desplegado cuando son usados argumentos incorrectos:

- Requires window's length to be no greater than the FFT length. (Requiere que la longitud de la ventana no debe ser más grande que la longitud de FFT).
- Requires `noverlap` to be strictly less than the window length. (Requiere que `noverlap` sea estrictamente menor que la longitud de la ventana).
- Requires positive integer values for `nfft` and `noverlap`. (Requiere valores de enteros positivos para `nfft` y `noverlap`).
- Requires vector input. (Requiere vector de entrada).

norm

Norma de vectores y matrices.

Sintaxis

```
n = norm(A)
n = norm(A,p)
```

Descripción

La *norma* de una matriz es un escalar que proporciona alguna medida de la magnitud de los elementos de una matriz. La función `norm` calcula diversos tipos de norma a una matriz:

`n = norm(A)` retorna el valor singular mas grande de `A`, `max(svd(A))`.

`n = norm(A,p)` retorna diferentes tipo de norma, dependiendo del valor de p:

Si p es...	La función `norm` retorna...
`1`	La norma - 1, o la mayor suma de las columnas de `A`, `max(sum(abs((A)))`.
`2`	El valor singular mas grande, (lo mismo como `norm(A)`).
`inf`	La norma infinita, o la mayor suma de las filas de A, `max(sum(abs(A')))`.
`'fro'`	La norma Frobenius de la matriz `A`, `sqrt(sum(diag(A'*A)))`.

Cuando `A` es un vector, se aplica reglas ligeramente diferentes:

norm(A,p) retorna `sum(abs(A).^`p`)^(1/`p`)`, para cualquier $1 \le p \le \infty$.

`norm(A)` retorna `norm(A,2)`.

`norm(A,inf)` retorna `max(abs(A))`.

`norm(A,-inf)` retorna `min(abs(A))`.

Observación

Para obtener el valor medio cuadrado (RMS) use `norm(A)/sqrt(n)`. Note que `norm(A)`, donde `A` es un vector de n elementos, es la longitud de `A`.

mean

Descripción

Valor medio o promedio de arrays.

Sintaxis

`M = mean(A)` retorna los valores medios de los elementos a lo largo de las diferentes dimensiones de un array.

Si `A` es un vector, `mean(A)` retorna el valor medio de `A`.

Si `A` es una matriz, `mean(A)` trata las columnas de `A` como vectores, retornando un vector fila de valores medios.

Si `A` es un array multidimensional, `mean(A)` trata los valores a lo largo de la primera dimensión como vectores, retornando un array de valores medios.

`M = mean(A,`*`dim`*`)` retorna los valores medios para los elementos a lo largo de la dimensión de `A` especificada por el escalar *`dim`*.

Ejemplos

```
A = [1 2 4 4; 3 4 6 6; 5 6 8 8; 5 6 8 8];
mean(A)
ans =
    3.5000    4.5000    6.5000    6.5000

mean(A,2)
ans =
    2.7500
    4.7500
```

```
    6.7500
    6.7500
```

fft

Transformada rápida de Fourier unidimensional.

Sintaxis

```
y = fft(x)
y = fft(x,n)
```

Descripción

`y = fft(x)` es la transformada discreta de Fourier del vector `x`, calculada con el algoritmo de la transformadas rápida de Fourier (FFT). Si `x` es una matriz, `fft(x)` es la FFT de cada columna de la matriz.

`y = fft(x,n)` es la FFT calculada para `n` puntos en frecuencia. Si la longitud de `x` es menor que `n`, `x` es rellenado con ceros para alargarlo hasta la longitud de `n`. Si la longitud de `n` es mayor que la de `n`, la secuencia `x` es truncada. Cuando `x` es una matriz, la longitud de las columnas son ajustadas de la misma manera.

La función `fft` emplea el algoritmo de la transformada rápida de Fourier de base 2 si la longitud de la secuencia es una potencia de dos, y un algoritmo mas lento de base alternada si no lo es.

X = fft(x) y x = ifft(X) implementan el par de transformada y la transformada inversa para vectores de longitud N. Esta dado por:

$$X(k) = \sum_{j=1}^{N} x(j) w_N^{(j-1)(k-1)}$$

$$x(j) = \frac{1}{N} \sum_{k=1}^{N} X(k) w_N^{-(j-1)(k-1)}$$

donde

$$w_N = e^{-2\pi i / N}$$

es la N - ésima raíz de la unidad.

Algoritmo

Cuando la longitud de la secuencia es una potencia de dos, el algoritmo de la transformada rápida de Fourier de base 2 de alta velocidad es empleado. La rutina FFT de base dos es optimizada para ejecutar una FFT real si la secuencia de entrada es puramente real, de otra manera esta calcula la FFT compleja. Esto causa que la FFT real de potencia de dos sea un 40% mas rápida que una FFT compleja de la misma longitud.

Cuando la longitud de la secuencia no es una potencia de dos exacta, un algoritmo encuentra los factores primos de la longitud de la secuencia y calcula las transformadas de Fourier discretas de base alternada de las secuencias mas cortas.

El tiempo empleado para calcular una FFT varia grandemente dependiendo sobre la longitud de la secuencia. Las FFT‘s de secuencias cuyas longitudes tienen muchos factores primos son calculadas rápidamente; las FFT‘s que tienen pocos no. Secuencias cuyas longitudes son números primos son reducidas a las filas del algoritmo DFT. Por esta razón es mejor quedarse con FFT‘s de

potencia de dos a menos que otras circunstancias dicten que esto no puede ser realizado. Por ejemplo, una maquina calculando una FFT real de 4096 puntos toma 2.1 segundos y 3.7 segundos al calcular una FFT compleja de la misma longitud. Las FFT‘s de secuencias vecinas de longitud 4095 y 4097, sin embargo, toman 7 segundos y 58 segundos respectivamente.

Ejemplo

Calculemos el espectro del pulso rectangular usando `fft`.

```
Fs=10000;
t = 0: 1/Fs: 0.0005; %Define el ancho del pulso
x = square(2*pi*1000*t);
X = fft(x,256);
plot(abs(X));
title('Espectro del pulso rectangular usando fft')
pause;
```

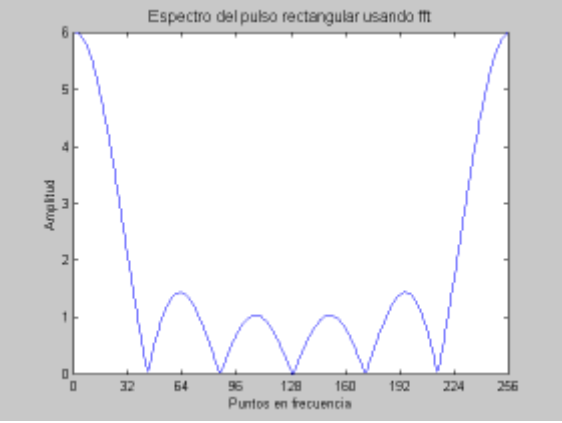

Figura 5. Espectro del pulso rectangular usando `fft`.

A diferencia de `spegram`, `fft` calcula el espectro no solo para las frecuencias positivas, también lo hace para las frecuencias negativas. Esta es la razón por la cual a partir de la muestra 128 el espectro se repite.

ifft

Transformada rápida de Fourier inversa unidimensional.

Sintaxis

```
y = ifft(x)
y = ifft(x,n)
```

Descripción

`ifft(x)` es la transformada rápida inversa de Fourier del vector `x`.

`ifft(x,n)` es la FFT inversa evaluada en `n` puntos.

Algoritmo

El algoritmo para `ifft(x)` es similar al algoritmo para `ifft(x)`, excepto por un cambio de signo y de escala por un factor `n = leng(x)`. Así el tiempo de ejecución es mas rápido cuando `n` es una potencia de dos y mas lento cuando `n` es un número primo grande.

Ejemplo

Para cualquier vector `x`, `ifft(fft(x))` es igual a `x` dentro de un margen de error. Si `x` es real, `ifft(fft(x))` puede tener pequeñas partes imaginarias.

Para ilustrar mejor esta propiedad, calculemos mediante `ifft` la transformada inversa de Fourier del vector `X` obtenido en el ejemplo anterior.

```
y = ifft(X,256);
for   i = 1:1:length(x)
   y1(i)= y(i);
end
plot(t,y1);
title('Transformada inversa de Fourier del vector X usando ifft')
pause;
```

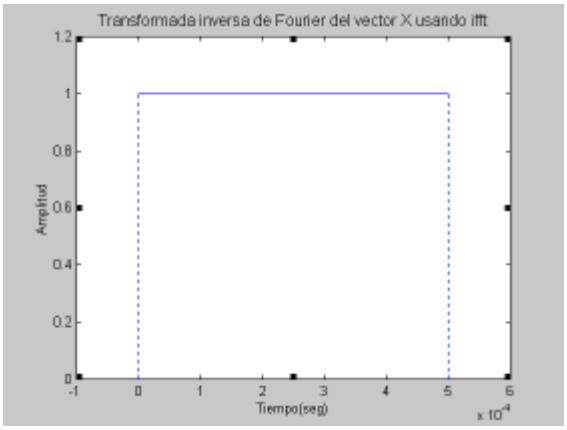

Figura 6. pulso rectangular obtenido mediante `ifft`.

Como era de esperarse, la transformada inversa de la secuencia definida por el vector `X` describe el pulso rectangular original. Un aspecto a tener en cuenta es la correspondencia en el número de puntos en el que se evalúan estas funciones, pues debe ser el mismo para obtener la relación

`x=ifft(fft(x))`. Este es motivo por el cuál el vector `y` es de dimensión 256 y la razón del uso del ciclo `for`, pues se necesita extraer las primeras `length(x)` componentes debido a que solo estas definen el pulso rectangular.

ANEXO C. DESCRIPCIÓN HARDWARE DEL SISTEMA.

Las aplicaciones realizadas por el sistema son en gran parte procesos a nivel de software que después de su ejecución, entregan una respuesta a través del puerto paralelo del PC en que se ejecutan. En este anexo se detalla el hardware encargado de completar dichas aplicaciones, pues se necesita de un medio que adapte la respuesta del sistema a las tareas que finalmente se ejecutan.

El hardware utilizado está representado por un circuito que puede ser empleado para varios propósitos (figura 2). Para la aplicación control proceso, se puede experimentar con motores de corriente continua de mayor tamaño que el empleado en este proyecto, pues esta característica está determinada por la cantidad de corriente que maneja Q2. Para la activación o desactivación de dispositivos se pueden emplear artefactos como una bombilla, un sistema de audio casero etc. cuyo consumo de corriente no exceda la capacidad de Q5.

Descripción del circuito

El circuito consta básicamente de dos etapas como se muestra en la figura 1, una de ellas encargada de controlar al motor compuesta de un manejador de corriente y de un acoplador de pulsos, y la otra encargada de activar o desactivar los dispositivos compuesta de un segundo manejador de corriente y de una interface óptica.

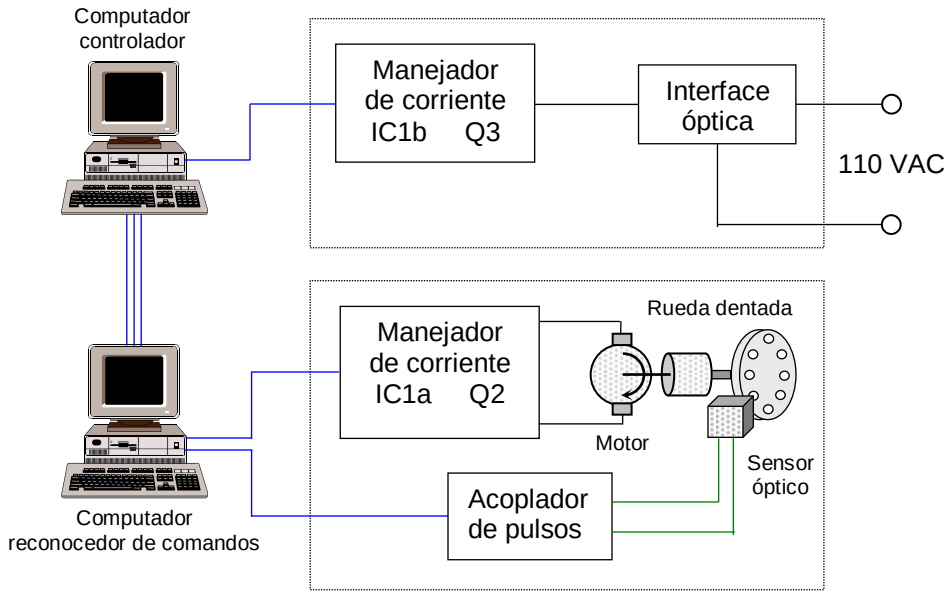

Figura 1. Diagrama de bloques.

Cada bloque está conectado a un PC mediante el puerto paralelo, pues gracias a este medio se envía la respuesta al mundo exterior de los procesos software que se ejecutan en casa uno. El computador controlador a través del pin 16 en la dirección H37A del puerto paralelo envía la señal PWM que controla al motor, y a través del pin 15 en la dirección H378 lee su velocidad. El computador reconocedor de comandos hace uso del pin 16 de su puerto para activar o desactivar los dispositivos. Mediante el uso de los pines del 2 al 9 en la dirección H378 de sus respectivos puertos se comunican los computadores en cuestión, donde el reconocedor de comandos los tiene configurados como salida y el controlador como entrada para que se pueda transmitir el set point.

En la figura 2 se muestra el diagrama esquemático del circuito. A continuación se analiza su funcionamiento y principios de diseño.

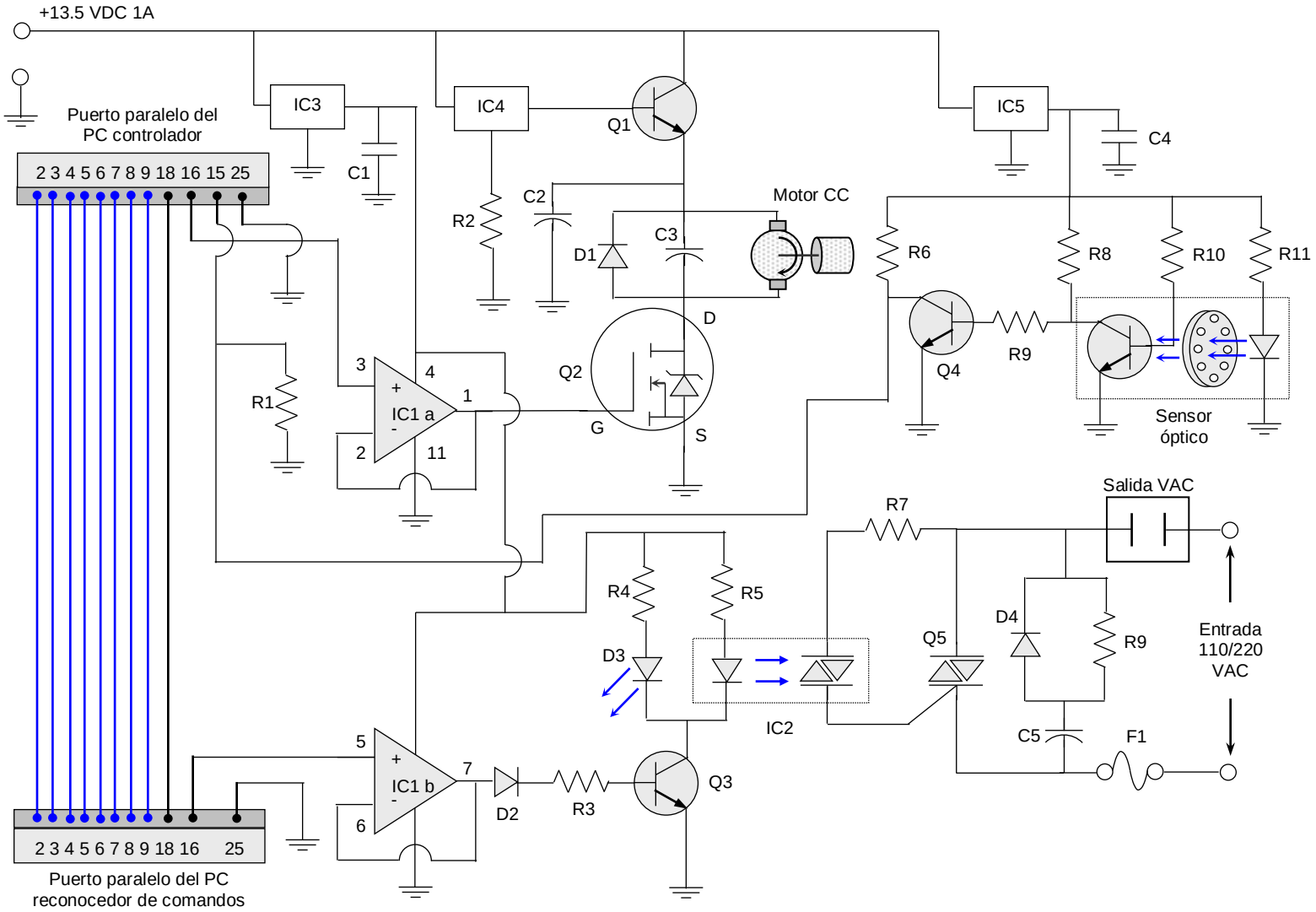

Figura 2. Diagrama esquemático.

Como conocemos, la señal PWM es generada mediante software como resultado de la ejecución del algoritmo de control PI. Esta se obtiene en el pin 16 del puerto y por lo tanto, IC1a es usado para aislar el puerto de la etapa de potencia con el objetivo de protegerlo.

Q2 es el conmutador de potencia. Este recibe el pulso modulado por ancho en su terminal de compuerta (*gate*) y puede conmutar la corriente de la carga entre encendido y apagado a través de la ruta de fuente a drenador (Source - drain).

Cuando Q2 está activado, este provee una ruta hacia tierra para activar al motor. Cuando Q2 está desactivado, la tierra del motor queda aislada.

IC4 y Q1 están dispuestos de tal forma que proveen la alimentación del motor. IC4 entrega un voltaje de referencia de 12 VDC y Q1 es usado para manejar la corriente que este consume. Hay que tener en cuenta que si se desea manejar motores de mayor consumo de voltaje y corriente, se hace necesario modificar esta parte del circuito para poder sostener esta nueva exigencia.

El motor debe tener la fuente de alimentación en su lado positivo todo el tiempo. El capacitor C3 suaviza la forma de onda conmutada y reduce algo de interferencia por radio frecuencia. El diodo D1 es conocido como "diodo volante", el cual protege el circuito contra los voltajes inversos que provienen del motor debido a que este es una carga inductiva.

R8 y R10 son usados para polarizar adecuadamente el fototransistor del sensor óptico y R11 es usado para polarizar el led asociado. A pesar de que los pulsos entregados por este dispositivo tienen la corriente suficiente para activar el puerto, no poseen el voltaje necesario para que sean interpretados. Por este motivo, se hace conveniente adaptar estos niveles de voltaje mediante el uso de Q4.

La función de IC1b es proteger el puerto paralelo al que esta conectado. Al generarse la operación de activar el dispositivo, el pin 16 genera un uno lógico y como consecuencia el voltaje positivo en el pin 7 de IC1b, el diodo D2 conduce. Como resultado, Q3 se satura y polariza el led D3 y el led interno de IC2.

Este led interno emite una luz infrarroja que hace conducir al optotriac (pines 4 y 6); este optotriac puede entregar una corriente de 100 mA, suficiente para disparar el gate del triac Q5, el cual manejara la carga de potencia. Este último puede entregar hasta 15 amperios, suficientes para alimentar los dispositivos domésticos concebidos en esta aplicación como una bombilla de 100 W o un sistema de sonido casero.

El optoacoplador IC2 se utiliza para separar el circuito electrónico de la etapa de potencia con total seguridad y tiene una capacidad de aislamiento de 1 kV. Además, contiene una red interna de detección de cruce por cero de la tensión alterna de la red eléctrica y la lógica para formar un red de enraso de fase, permitiendo la conmutación confiable de cargas inductivas. Sin esta lógica interna, al disparar el triac en cualquier punto de la forma de onda donde el nivel de cambio de voltaje (dv/dt) es muy rápido , circularía una corriente muy elevada a través de la carga, lo cual produciría una gran interferencia electromagnética en radios y otros equipos sensibles.

Lista de materiales

Condensadores

C1, C3, C4 ..... 0.1μF 50V cerámico.

C2 ..... 1μF 100V electrolítico.

C5 ..... 0.47μF 400V.

Resistencias 1/4 w

R1 ..... 15 KΩ.

R2 ..... 330 Ω.

R3, R6 ..... 5.6 KΩ.

R4 ..... 1 KΩ.

R5 ..... 1.5 KΩ.

R7 ..... 330 Ω.1/2 W.

R8 ..... 7.5 KΩ.

R9 ..... 68 Ω 1/2 W.

R10 ..... 100 KΩ.

R11 ..... 270 Ω.

Semiconductores

D1, D4 ..... Diodo rectificador 1N4004.

D2 ..... Diodo de conmutación 1N4148.

D3 ..... Diodo led rojo de 5 mm.

Q1 ..... Transistor NPN TIP31.

Q2 ..... Mosfet de canal N IRFZ44N.

Q3, Q4 ..... Transistores NPN 2N2222.

Q5 ..... Triac Q4015L5.

IC1 ..... Circuito integrado LM 324.

IC2 ..... Optotriac MOC 3031.

IC3, IC4 ..... Reguladores de 12 V LM 7812.

IC5 ..... Regulador de 5V LM 7805.

1 Sensor óptico EE-SX493.

Varios

F1 ..... Fusible de 1 amperio.

1 Tomacorriente.

1 Motor de corriente continua con campo fijo de imán permanente de 1 amperio.

ANEXO D. DISEÑO DETALLADO DEL SISTEMA.

En este apartado se trata el diseño del "Control de Aplicaciones Mediante Comandos Orales Reconocidos por Redes Neuronales". Se realiza el refinamiento de las clases definidas en el análisis y se refina los diagramas de secuencia de mensajes para mayor compresión de las operaciones sobre las clases resultantes.

1. Diseño Detallado

1.1 Refinamiento del Diagrama de Clases

A continuación se procede a refinar las clases definidas en el análisis para obtener una visión mas profunda de los procesos realizados por el sistema.

El sistema básicamente esta compuesto por cuatro partes constitutivas: el establecimiento de la sesión con el usuario, el entrenamiento, la simulación y las aplicaciones. El establecimiento de la sesión con el usuario le asigna a este un campo de trabajo donde es tenido en cuenta su registro para almacenar la configuración de los comandos con los que opera el sistema, las matrices de entrenamiento y las matrices de pesos. El entrenamiento y simulación están orientados a la obtención de las matrices de pesos y simulación de los sistemas neuronales y las aplicaciones son acciones a nivel hardware y software realizadas cuando un comando ha sido reconocido.

Después de considerar estos aspectos, aparecen así las siguientes clases:

- IU_inicio. Interfaz gráfica para el inicio de sesión, para continuar sesión y para los mensajes de confirmación de entrenamiento.

- Validar_ingreso. Valida el ingreso del usuario mediante nombre y clave. Si el usuario trata de ingresar a una sesión con el nombre mal o si se equivoca en la clave, recibe el evento *error* de Registro_usuario y despliega este evento; en ambos casos de error se finaliza la ejecución de Sistema.
- Datos_usuario. Administra lo relacionado con los datos de usuario provenientes de Comunicación_DDE que son objeto de las aplicaciones.
- Enviar_solicitud. Envía las solicitudes de las funciones de Matlab a ejecutar a través de Comuniucación_DDE
- Gestionar_resultado. Gestiona los resultados provenientes de Comunicación_DDE obtenidos desde Matlab a través de un campo de información asociado a cada usuario, el cual incluye los registros del *nombre* y de la *clave.*
- Asocia_nombre. Entrega a sistema la clave asociada al nombre que el usuario configura por primera vez. Si el usuario da mal su nombre, genera un evento de error a sistema.
- Grabadora. Se encarga de grabar las pronunciaciones hechas por el usuario correspondientes a los comandos. Básicamente hace el manejo de la grabadora de sonidos.
- Escucha. Se encarga de escuchar los comandos que pueden ser pronunciados por el usuario para su detección. Básicamente hace la implementación del ciclo de captura automático.
- Puerto. Se encarga de recibir y enviar por el puerto paralelo del PC los datos de entrada y salida del control PI. También se encarga de enviar el dato de salida de la aplicación hardware al dispositivo externo a través de este mismo puerto.
- Datos_control. Clase que implementa el algoritmo de control PI y administra los datos calculados por este algoritmo, para ser enviados al puerto paralelo del PC a través de Puerto.

1.2 Refinamiento de la Secuencia de Mensajes

A continuación se procede a realizar el refinamiento de los diagramas de secuencia de mensajes presentados en la sección de análisis teniendo en cuenta las clases que se derivaron del proceso de refinamiento de clases realizado también en la sección de análisis.

Caso de uso Ingresar al Sistema

En este caso de uso la clase IU_inicio obtiene *nombre* y *clave* de Usuario para invocar la operación validar de la clase Validar_ingreso. La operación validar accede el campo de datos del usuario para obtener nombre y clave para luego compararlos con los datos ingresados al sistema.

El diagrama de secuencia de mensajes se muestra en la figura 1.

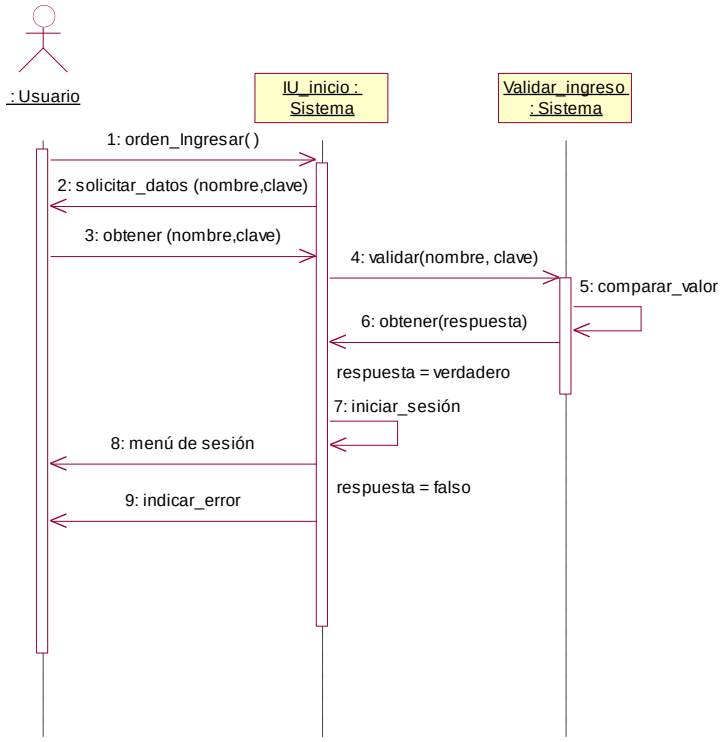

Figura 1. Diagrama de secuencia para el caso de uso Ingresar al Sistema.

Caso de uso Validación de Ingreso

La obtención de *nombre* y *clave* por parte de la clase Registro_usuario provenientes de la clase Validar_ingreso tiene dos variantes. La primera es cuando se trata de un nuevo usuario que ingresa al sistema, el cual es registrado por la operación Registrar_usuario( ) seguido de la confirmación de este evento, invocada por la clase IU_inicio mediante la operación confirmar_usuario. Una vez confirmado el usuario, este procede a configurar los comandos con los que trabajará el sistema. La segunda se presenta cuando ya se ha tenido sesión con el sistema y el usuario desea continuarla, para lo cual la case Asocia_nombre obtiene *nombre asociado a la*

clave para proceder a buscarlo en el campo de datos usando la operación buscar_nombre( ). Si el nombre es erróneo o no existe se notifica este evento a la clase IU_inicio, pero si el nombre es correcto, la clase Validar_ingreso obtiene la clave asociada al nombre usando la operación obtener( ).

La figura 2 muestra la secuencia de mensajes.

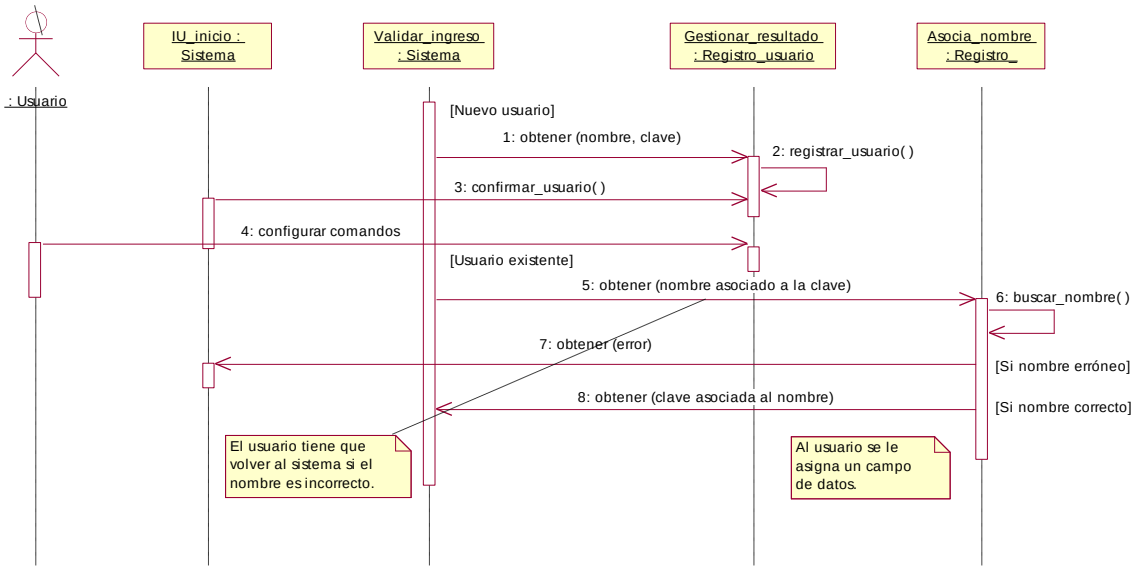

Figura 2. Diagrama de secuencia para el caso de uso Validación de ingreso.

Caso de uso Captura de señal

Como se observa este caso de uso comienza cuando Usuario necesita obtener pronunciaciones para entrenar o simular. En el primer caso se activa la grabadora de sonidos, la cual se encarga de la captura de la pronunciación del comando por parte del usuario (el mensaje *comando* refiere a las pronunciaciones hechas por el usuario) para finalmente guardar el comando_capturado. En el segundo caso antes de la activación del *dispositivo de entrada,* se ejecuta el *ciclo de captura automático* para que la captura de algún comando sea ágil y esté disponible para ser procesado.

El diagrama de secuencia de mensajes de este caso de uso se muestra en la figura 3.

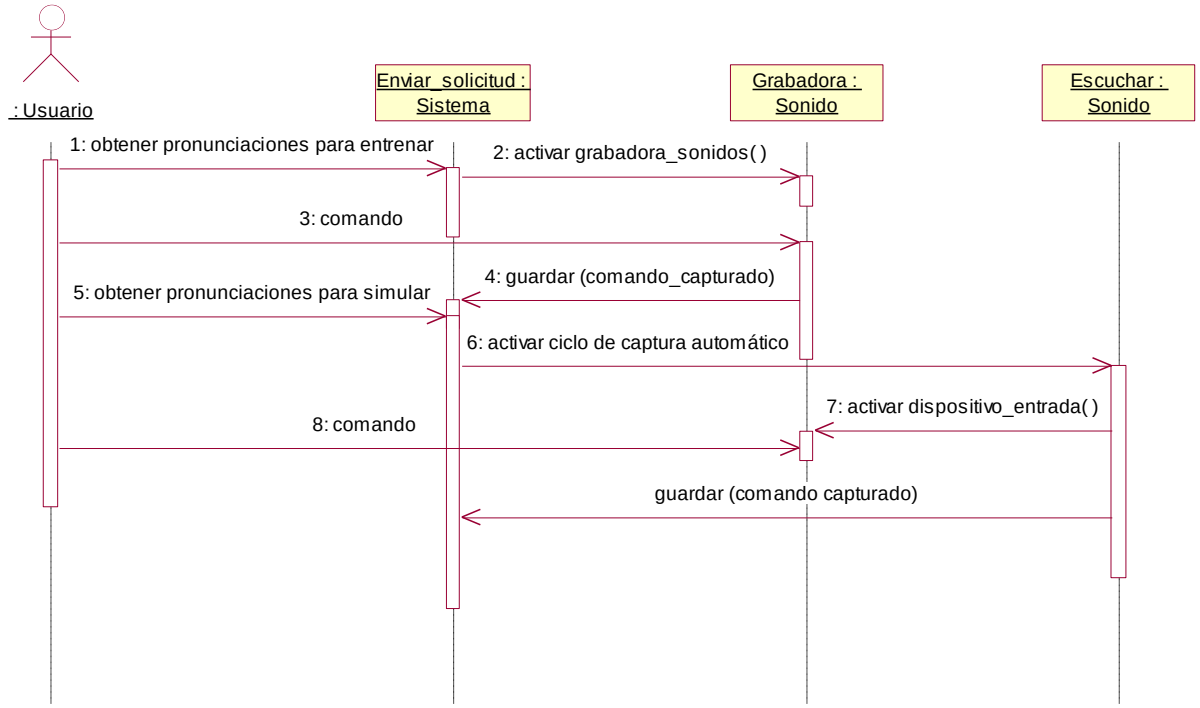

Figura 3. Diagrama de secuencia para el caso de uso captura de señal.

Caso de uso Procesamiento de señal

En este caso de uso, el procesamiento de señal que reciben los comandos se hacen con el objetivo de obtener las matrices de entrenamiento o simulación. Por esta razón la obtención de estos tipos de matrices se hace a partir de N_función (comando_capturado), donde *comando_capturado* corresponde a la pronunciación capturada para entrenar o simular.

N_función es una importante función que se ejecuta en el ambiente Matlab, cuyo argumento *comando_capturado* corresponde a los datos de la señal de audio a procesar, y los valores retornados por esta función son las matrices para entrenar o simular.

El diagrama de secuencia de mensajes para este caso de uso se muestra en la figura 4.

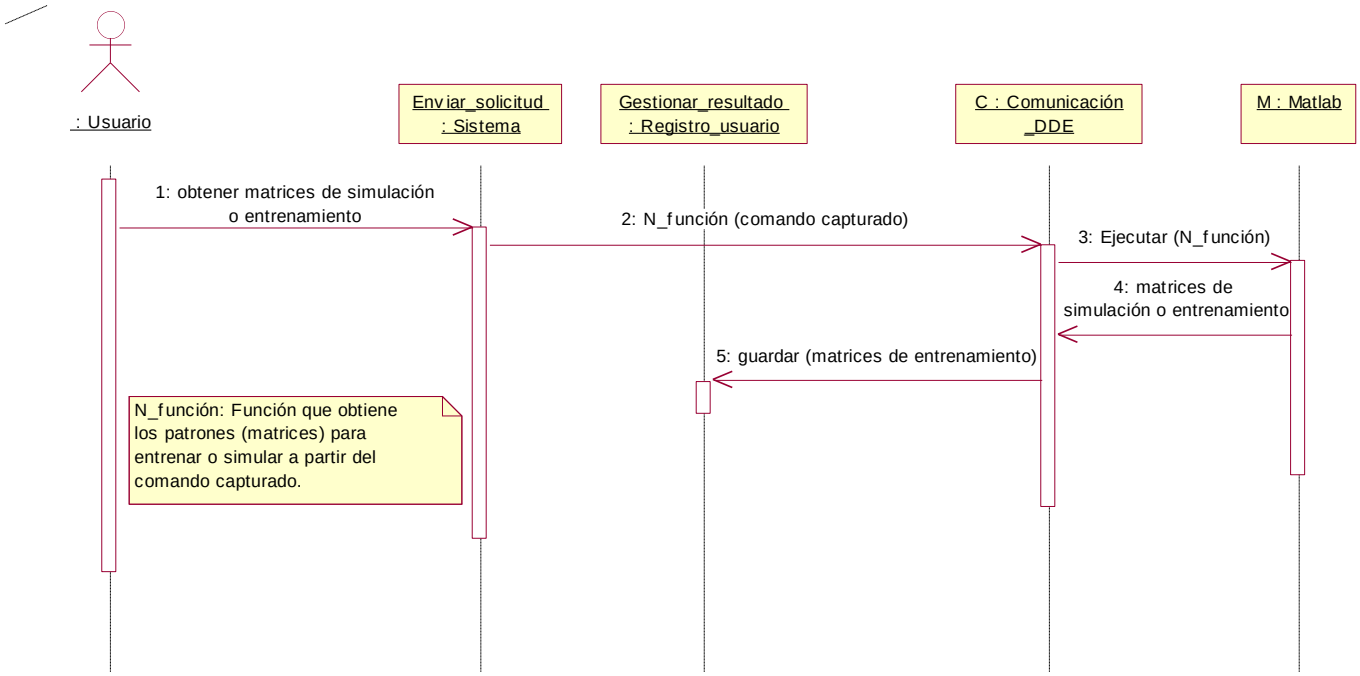

Figura 4. Diagrama de secuencia para el caso de uso procesamiento de señal.

Caso de uso Entrenamiento

En este caso de uso la solicitud de entrenar de Usuario se convierte en *entreno( )*, que es una función realizada en el ambiente matlab encargada del entrenamiento de los sistemas neuronales. Antes de ejecutarse *entreno( )* se necesita ejecutar *obtener (matrices de entrenamiento)* debido a

que, a la función *entreno( ),* le deben ser pasadas las matrices de entrenamiento, que específicamente corresponden a los ejemplos de entrenamiento de la red neuronal.

La ejecución de la función *entreno( )* tiene como resultado la verificación de entrenamiento y las matrices de pesos. Cuando se tiene una verificación de entreno exitoso, se procede a guardar estas matrices mediante *guardar(matriz_pesos),* informando al usuario mediante un mensaje de entrenamiento exitoso. Si se tiene una verificación de entreno no exitoso, las matrices obtenidas no son guardadas y se informa al usuario mediante el uso de un mensaje que debe repetir el la operación de entreno.

El diagrama de secuencia de mensajes se muestra en la figura 5.

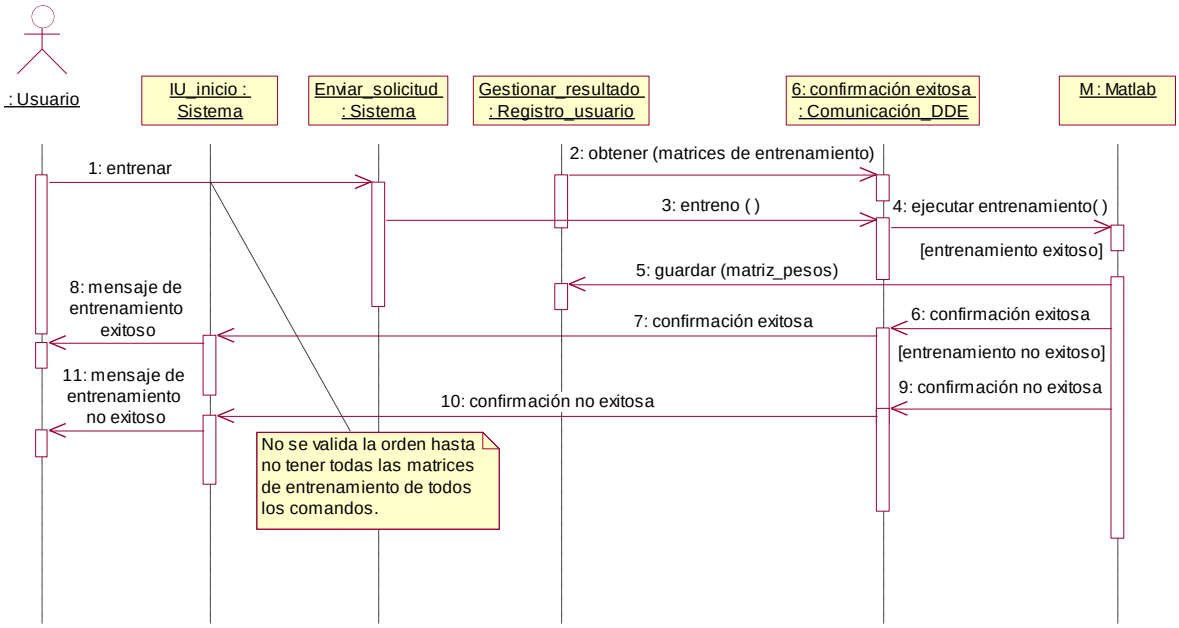

Figura 5. Diagrama de secuencia para el caso de uso Entrenamiento.

Caso de uso Reconocimiento

Usuario envía la orden simular que es convertida en *simulador( )*, que es una función realizada en el ambiente de Matlab encargada de la simulación de los sistemas neuronales. Antes de ejecutar esta función, deben ser ejecutadas las funciones *N_función(comando_capturado)* para obtener los patrones del comando a reconocer, y *obtener(matriz_pesos)* para cargar la red entrenada. De esta forma *simulador( )* hace su trabajo, pues necesita como argumentos la red entrenada y los patrones a reconocer.

Una vez ejecutada la función *simulador( )* en Matlab se obtiene como resultado una indicación directa del comando_reconocido. Por medio de la función *guardar( )* se almacena esta indicación en la clase Datos_usuario para estar disponible a la aplicación asignada.

El diagrama de secuencia de mensajes al efectuar el reconocimiento se muestra en la figura 6.

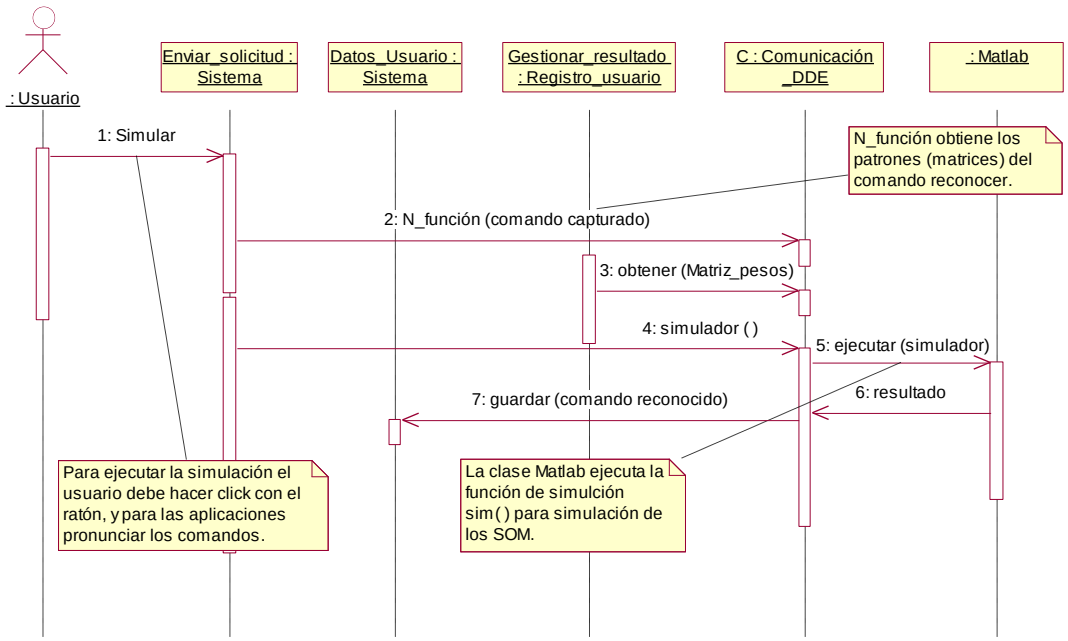

Figura 6. Diagrama de secuencia para el caso de uso Reconocimiento.

Caso de uso Aplicaciones

Cuando Usuario activa la aplicación mediante el uso de *activar_aplicación* la clase Aplicación_SW está atenta a la indicación directa del comando_reconocido mediante la función *obtener (comando_reconocido)* para enviar a la clase Datos_usuario la opción reconocida mediante *obtener (opción). Si opción es dispositivo doméstico* se actúa sobre Software_controlado para mostrar un cuadro de dialogo relacionado con la actividad a realizar, para luego activar o desactivar a Dispositivo_doméstico a través de la clase Puerto. *Si opción es proceso controlado* se actúa sobre Software controlado para realizar el propósito correspondiente, y se espera la indicación directa del comando_reconocido correspondiente al Set_Point.

Con la función *obtener_variable_controlar( )* se determina la variable a controlar de Proceso_controlado (velocidad del motor) para que Datos_control la obtenga junto con el Set_Point y de esta forma ejecutar el algoritmo de control PI. El resultado de este algoritmo, variable_controlada, se envía a Puerto para que Proceso_controlado reciba la señal PWM que controla el funcionamiento del motor.

En la figura 7 se muestra la secuencia de mensajes para este caso de uso.

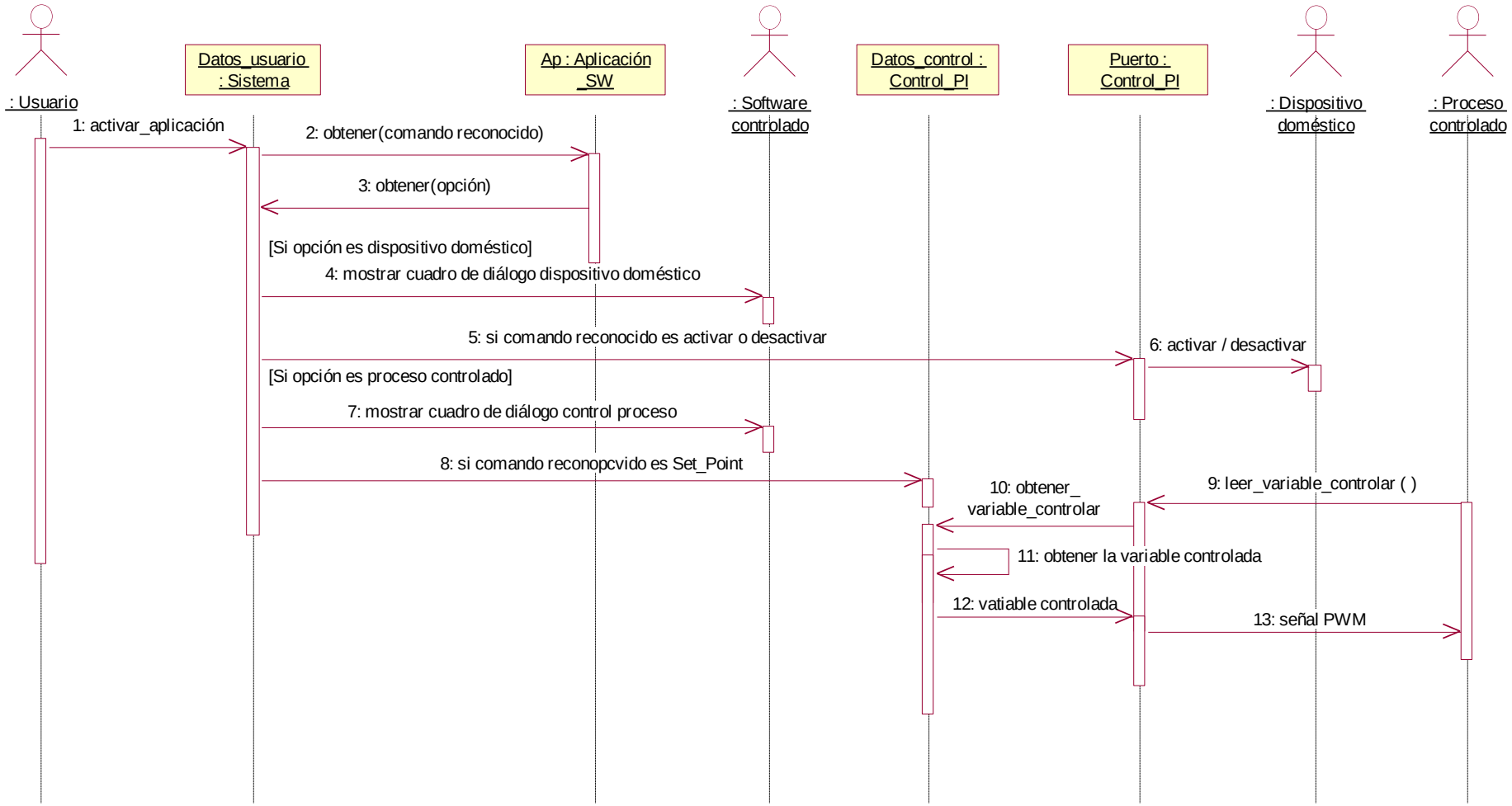

Figura 7. Diagrama de secuencia para el caso de uso Aplicaciones.

2. Diagrama de estados

A continuación se describe el comportamiento dinámico de cada una de las clases mediante el uso de los diagramas de estado.

2.1 Diagrama de Estado para la Clase IU_inicio

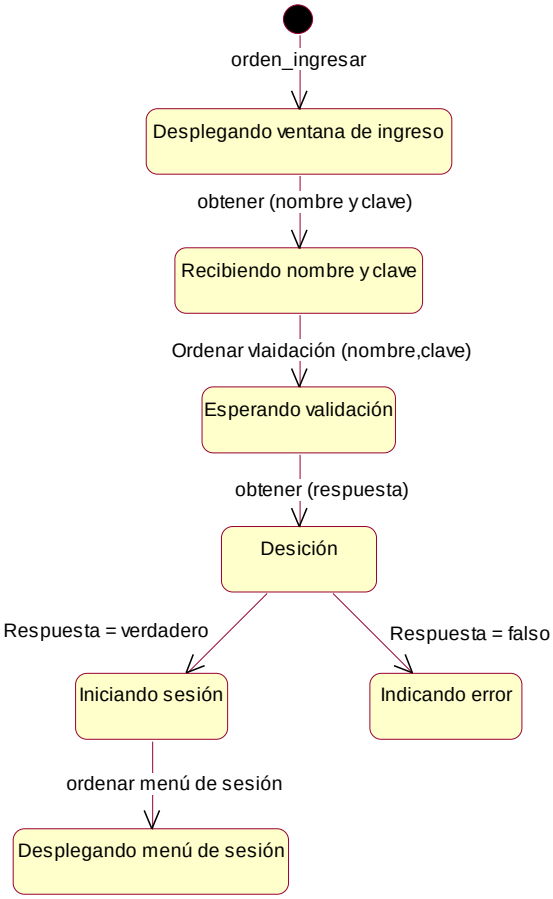

Figura 8(a). Diagrama de estado para la clase IU_inicio.

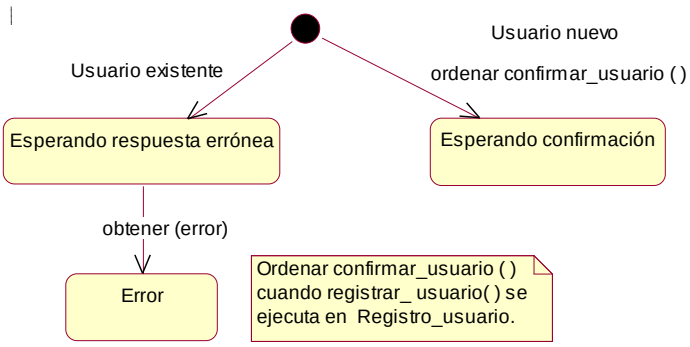

Figura 8(b). Sub-estado usuario

En el sub-estado usuario se evalúa la particularidad de los usuarios existentes o de los nuevos usuarios. Para los usuarios existentes se espera una respuesta errónea en caso de que el nombre introducido sea incorrecto, llevando a un estado de error. Los nuevos usuarios son confirmados y la clase entra a un estado de espera.

El diagrama de estado para la clase IU_inicio se muestra en al figura 8a, y el sub-estado usuario se muestra en la figura 8b.

2.2 Diagrama de Estado para la Clase Validar_ingreso

En el diagrama de estado de esta clase se pueden observar las alternativas de usuario existente o nuevo usuario que da lugar a los sub-estados ingresando usuario existente y registrando nuevo usuario. En el primer caso se envía a la clase Registro_usuario el nombre asociado a la clave para obtener de esta la clave asociada al nombre, y en el segundo se envía nombre y clave a la clase Registro_usuario y se espera el registro de usuario por parte de esta clase.

El diagrama de estado para la clase validar ingreso con los respectivos sub-estados se muestra en la figura 9.

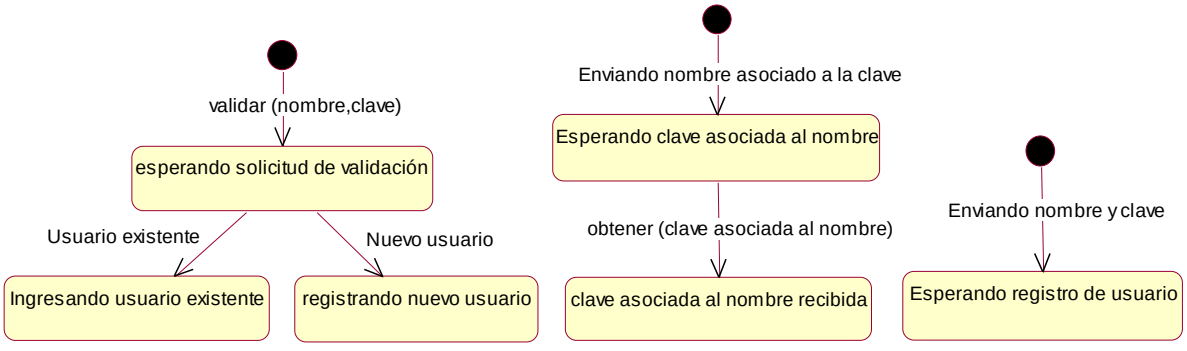

Figura 9. (a) Diagrama de estado para la clase Validar_ingreso. (b) Sub-estado usuario existente. (c) Sub-estado registrar nuevo usuario.

2.3 Diagrama de Estado para la Clase Datos_Usuario

Los estados en esta clase son proporcionados por las operaciones guardar ( ), activar aplicación ( ) obtener ( ) y por las condiciones sobre opción y comando_reconocido .Según estas condiciones esta clase toma estados que soportan las aplicaciones de dispositivo doméstico y control proceso.

El diagrama de estado para esta clase se muestra en la figura 10.

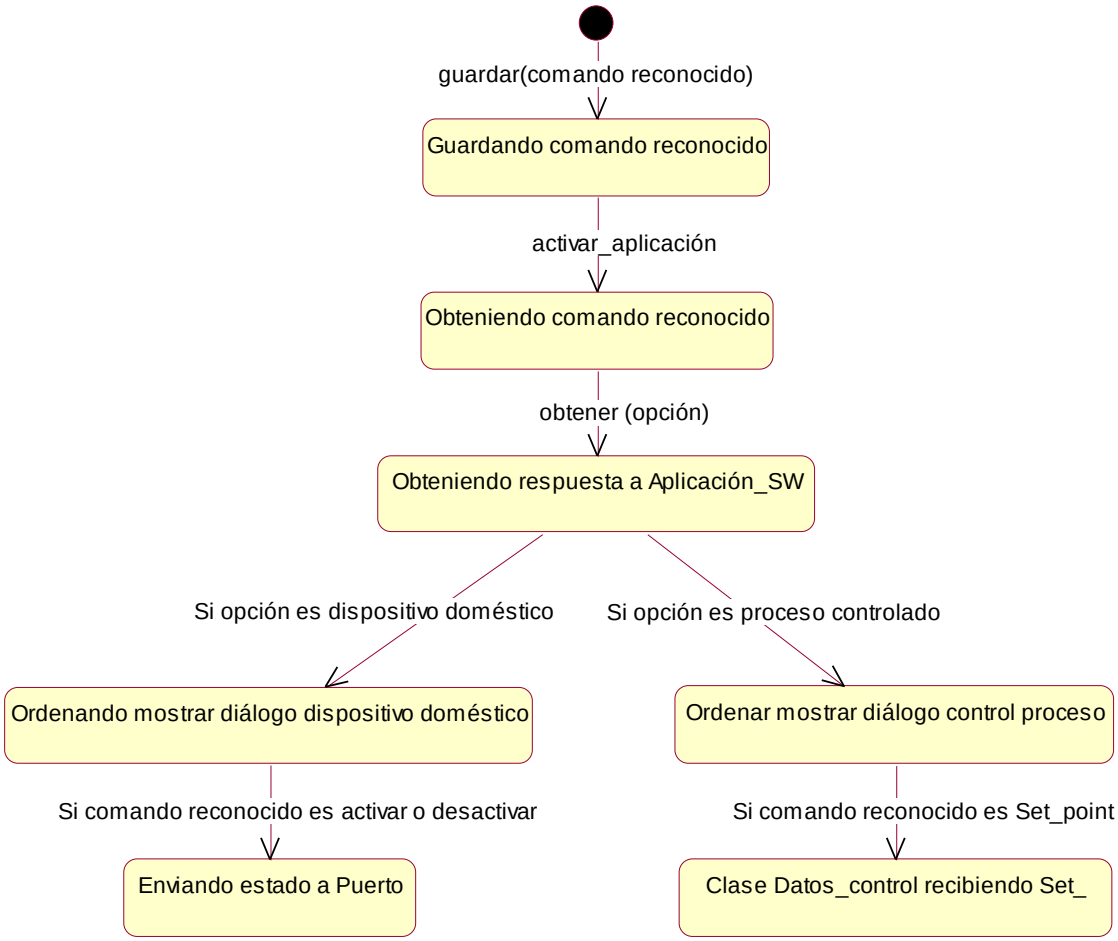

Figura 10. Diagrama de estado para la clase Datos_usuario.

2.4 Diagrama de Estado para la Clase Enviar_solicitud

La operación enviar (solicitud) corresponde a todas las solicitudes de Usuario que tienen como destino la clase Comunicación_DDE para que sean ejecutadas en Matlab. Por esta razón solicitudes como *simular* o *entrenar* son trasladadas a simulador ( ) y entreno ( ) respectivamente. Para el caso de la solicitud *obtener pronunciaciones para entrenar*, esta clase entra directamente al estado *ordenado activación de la grabadora de sonidos*.

En la figura 11 se muestra el diagrama de estado correspondiente a esta clase.

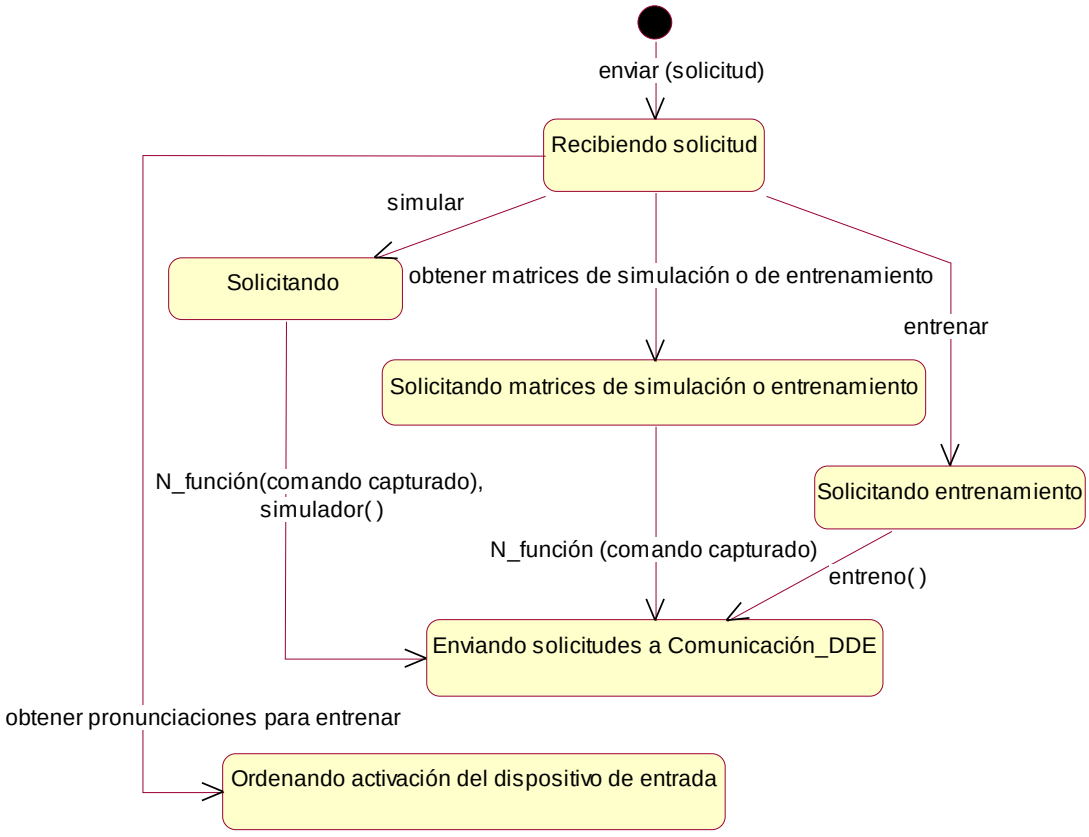

Figura 11. Diagrama de estado para la clase Enviar_solicitud.

2.5 Diagrama de Estado para la Clase Gestionar_resultado

La razón por la cual el primer estado presentado por esta clase es *registrar usuario* se debe a que la primera tarea realizada es asignar un campo de datos al usuario en el momento en que este ingresa por primera vez al sistema.

Una vez realizada la operación confirmar_usuario ( ) se entra al estado *campo de datos asignado* para poder realizar la configuración de los comandos con los que trabajará el sistema, y las operaciones guardar( ) y obtener ( ). La ejecución de estas dos últimas operaciones lleva a esta clase a los estados *matrices guardadas* y *matrices accesadas* respectivamente.

En la figura 12 se observa el diagrama de estados de esta clase.

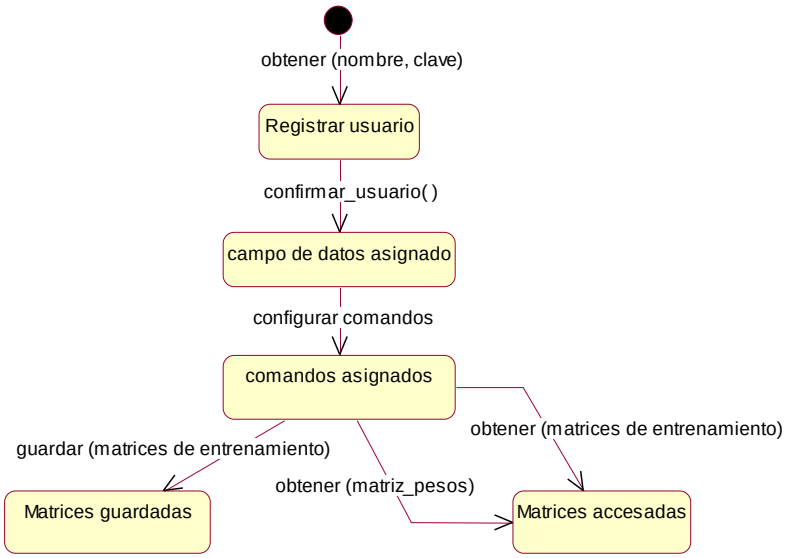

Figura 12. Diagrama de estado para la clase Gestionar resultado

2.6 Diagrama de Estado para la Clase Asocia_nombre

El estado *buscando nombre* es generado por la operación obtener (nombre asociado a la clave) para efectos de buscar en el sistema la existencia del usuario que esta relacionado con la clave introducida.

Las condiciones generadas como resultado de la operación buscar_nombre ( ) generan los estados *enviando error* y enviando *clave asociada al nombre.* La figura 13 muestra estos aspectos.

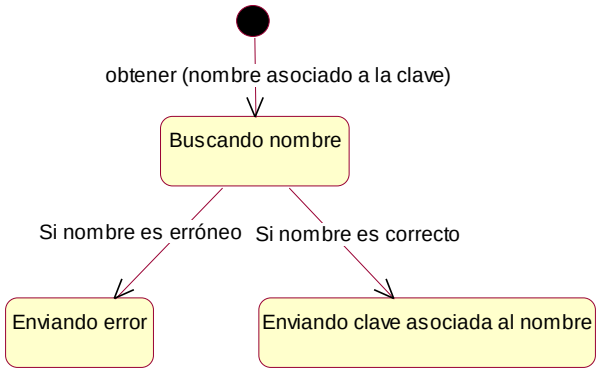

Figura 13. Diagrama de estado para la clase Asocia_nombre.

2.7 Diagrama de Estado para la Clase Grabadora

La operación activar grabadora_sonidos( ) lleva a esta clase al estado grabar para estar atenta al comando pronunciado, el cual genera el sub-estado Comando grabado. La razón por la cual Grabar es un estado principal es por que se debe estar en ese estado cuando el usuario hace sus pronunciaciones. Posteriormente la operación guardar (comando_capturado) se realiza cuando al terminar el estado principal se obtiene la pronunciación para ser guardada.

Estos aspectos se muestran en la figura 14.

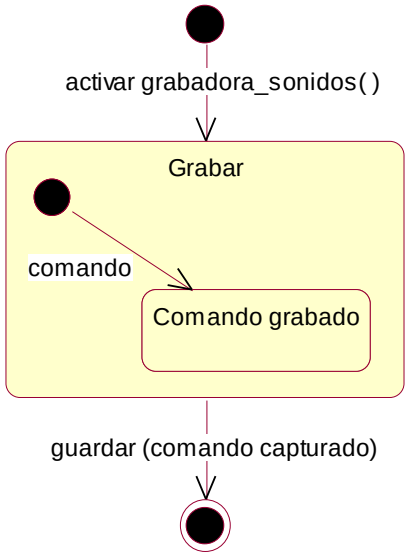

Figura 14. Diagrama de estado para la clase Grabadora.

2.8 Diagrama de Estado para la Clase Escuchar

La operación activar ciclo de captura automático lleva a esta clase al estado principal Escuchar en donde la operación activar dispositivo_entrada activa el sub-estado Grabar, para usar la captura de sonido proporcionada por el *dispositivo de entrada.* Cuando esta clase se encuentra en el estado Escuchar realiza procesos sobre la señal capturada para detectar de todo ese conjunto el comando, siendo esta la razón por la que Escuchar es el estado principal.

Una vez detectado el comando, el estado principal finaliza con la ejecución de la operación guardar( ) para que la clase Enviar_solicitud tenga conocimiento del comando_capturado. En la figura 15 se muestra los estados de esta clase.

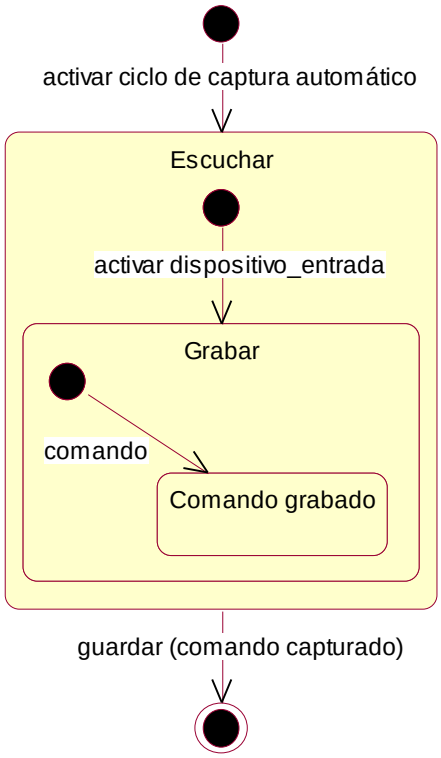

Figura 15. Diagrama de estado para la clase escuchar.

2.9 Diagrama de Estado para la Clase Puerto

La operación activar / desactivar corresponde a un valor particular de *comando_reconocido* proveniente de la clase Datos usuario para llevar a cabo la tarea de activar o desactivar al actor Dispositivo_doméstico, por lo que el estado generado por esta operación es activar / desactivar dispositivo.

leer_variable_controlar( ) es una operación que obtiene la variable a controlar del actor Proceso_controlado, lo que lleva a esta clase al estado *leyendo variable a controlar.* La operación *variable_controlada* corresponde al dato de control que ha de ser llevado al actor Proceso_controlado a través de esta clase, llevándola al estado señal PWM.

En la figura 16 se muestran estos comportamientos.

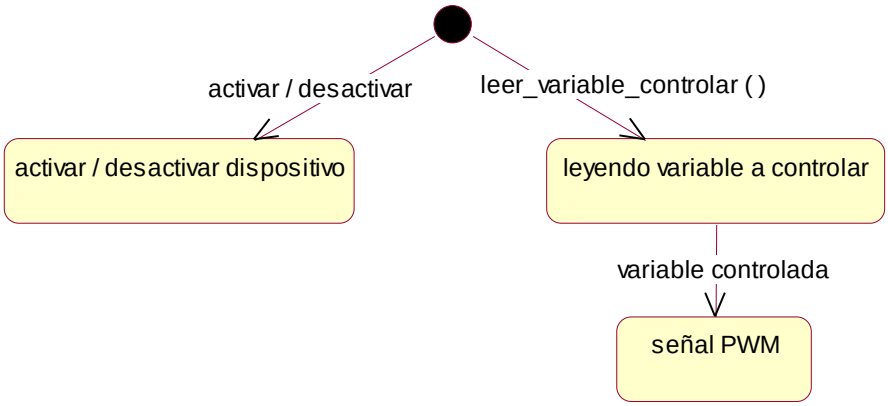

Figura 16. Diagrama de estado para la clase Puerto.

2.10 Diagrama de Estado para la Clase Datos_control

En esta clase la operación *obtener_variable_controlar( )*, lleva al estado de ejecución del algoritmo de control PI, que junto con el Set_point proveniente de Datos_Usuario, obtiene la *variable_controlada( )* del proceso para enviarla a la clase Puerto.

En la figura 17 se muestra este comportamiento.

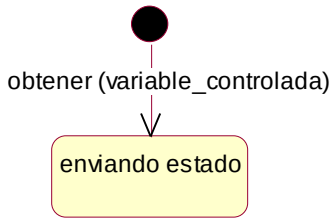

Figura 17. Diagrama de estado para la clase Datos_control.

2.11 Diagrama de Estado para la Clase Comunicación_DDE

En esta clase las operaciones *N_función* ( ), *simulador* ( ) y *entreno* ( ) llevan a un estado de envío de ejecución de estas mismas operaciones. Al finalizar estos estados, se generan transiciones que son el resultado del proceso realizado, y estas transiciones a su vez generan estados que sugieren la realización de operaciones en las clases de destino correspondientes.

Finalmente la operación obtener (matrices de entrenamiento) invoca estados y operaciones destinadas al entrenamiento de los sistemas neuronales.

Todos estos aspectos importantes se pueden observar en la figura 18.

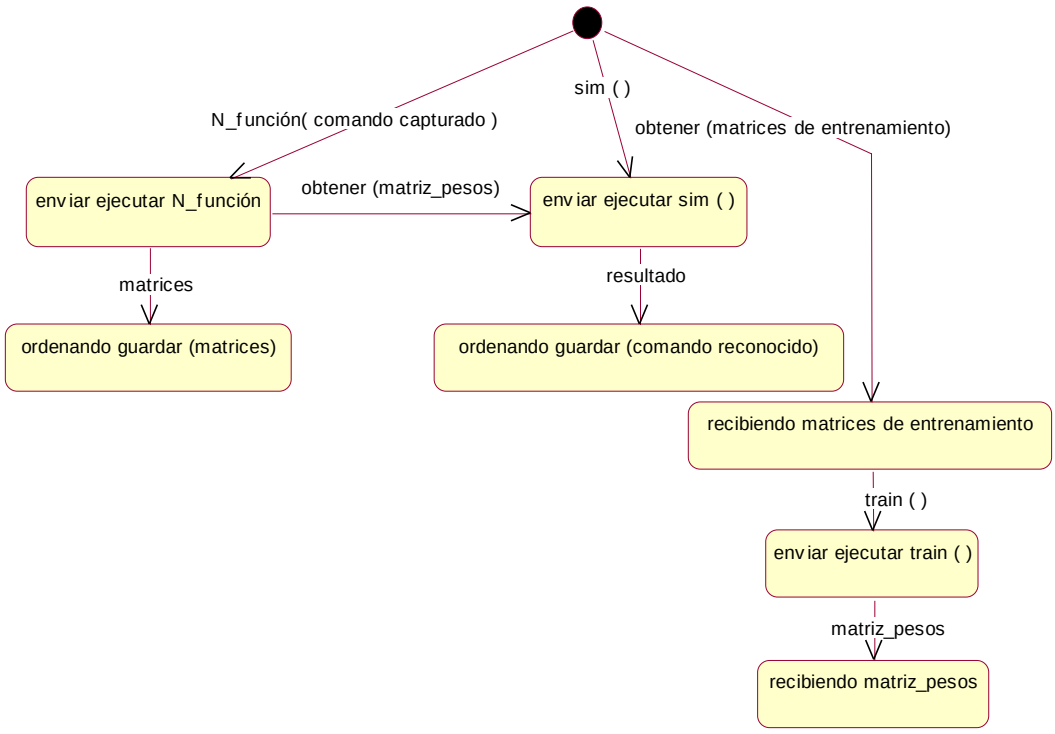

Figura 18. Diagrama de estado para la clase Comunicación_DDE.

2.12 Diagrama de Estado para la Clase Matlab

Como se había comentado antes, la clase matlab es el soporte de los cálculos invocados por las operaciones, *ejecutar (simulador)*, *ejecutar (N_función)* y *ejecutar (entreno).* Cada una de estas operaciones lleva a esta clase a un estado de realizar los cálculos correspondientes.

En la figura 19 se muestra el diagrama de estado para esta clase.

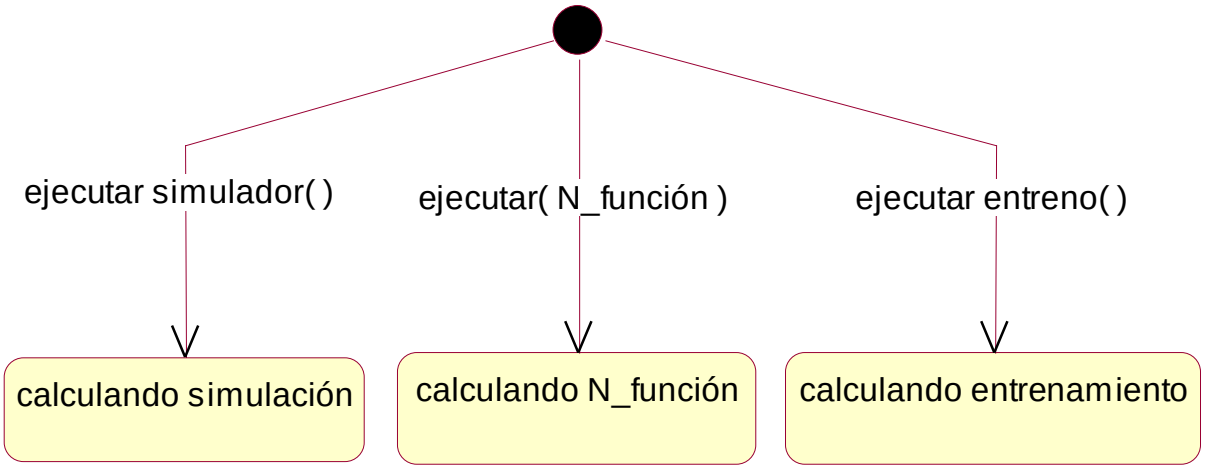

Figura 19. Diagrama de estado para la clase Matlab.

3 Diseño arquitectural

A continuación se relacionan las clases definidas con la arquitectura funcional y física del "Control de Aplicaciones Mediante Comandos Orales Reconocidos por Redes Neuronales".

3.1 Descomposición en Subsistemas

Basados en las actividades realizadas por cada una de las clases definidas en el sistema de control de aplicaciones, se obtiene una arquitectura funcional que da origen al diagrama de paquetes compuesto por el paquete Operaciones de Usuario, el paquete Comunicación, el paquete Cálculos y el paquete Campo de Datos.

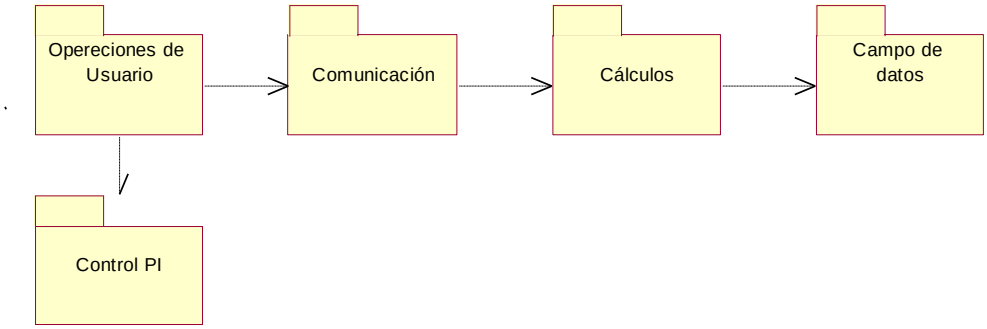

Figura 20. Paquetes del "Control de Aplicaciones Mediante Comandos Orales Reconocidos por Redes Neuronales".

La descripción de cada paquete con las clases que contienen se presenta a continuación.

Paquete Operaciones de Usuario

Paquete que contiene las clases Enviar_solicitud, Grabadora, Escuchar, Puerto, y Datos_control. Es en este paquete donde se interactúa con el usuario final del sistema para realizar operaciones de solicitud y adquisición de datos.

Paquete Control PI

En este paquete están incluidas todas las clases y el algoritmo de control PI que permiten la realización de los cálculos en la obtención la variable controlada para gestionar la velocidad del motor de corriente continua. Los aspectos correspondientes a la ejecución del control del motor están descritos en detalle en el capítulo seis.

Paquete Comunicación

Contiene la clase Comunicación_DDE. Este paquete realiza la conversión de requerimientos del actor Usuario y el envío de datos a las operaciones (funciones) realizadas por Matlab, y además envía los resultados obtenidos por Matlab a las clases que los requieren.

Paquete Cálculos

Este paquete contiene la clase Matlab. Se encarga del soporte de los cálculos correspondientes a las funciones del sistema realizadas en el ambiente Matlab.

Paquete Campo de Datos

Paquete que contiene las clases Datos_usuario y Gestionar_resultado y se encarga del manejo, almacenamiento y acceso de todas las variables (datos) obtenidos en el procesamiento de señal, reconocimiento, entrenamiento y los datos usados en las aplicaciones.

3.2 Definición de la Arquitectura Física

La arquitectura física describe la distribución de los módulos y paquetes en el contexto físico. En la figura 21 se observa la arquitectura del sistema desarrollado, en donde se destaca el módulo Desarrollo C++ conteniendo los paquetes desarrollados con la herramienta Visual C++, y el módulo Ambiente Matlab conteniendo el paquete cálculos.

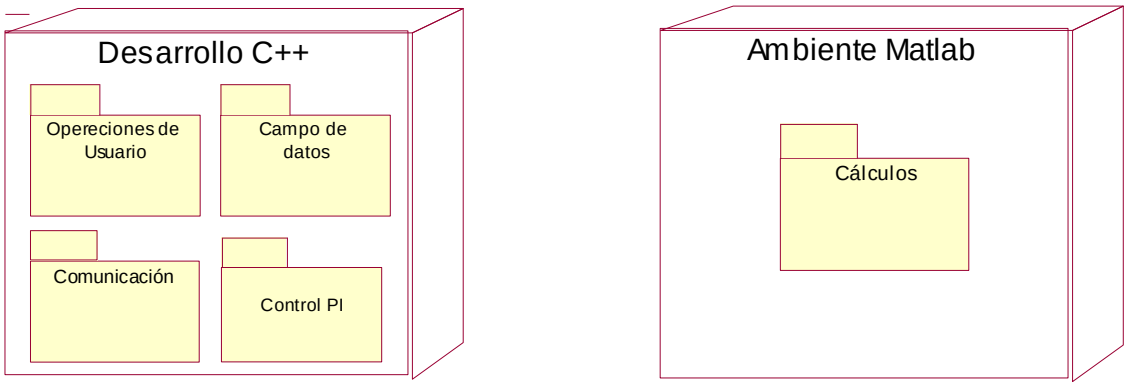

Figura 21. Arquitectura física.