InerSens Toolbox

For Use with MATLAB

User's Guide

Version v0.12*(02/04/09)*

Tabla de contenido

Índice de contenido

Introducción

Instalación

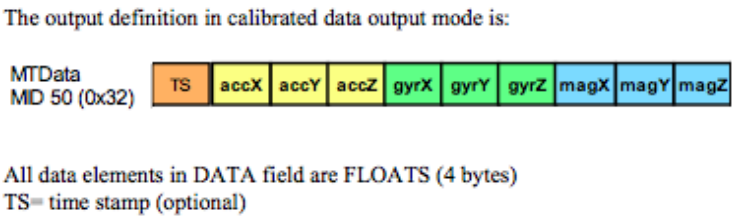
Las herramientas se encuentran en el directorio `silop`. Instálese donde se desee y añádase al path de Matlab. La distribución consta de:

```
-rwx------    1 juan  juan     68 Nov 30 10:22 README
drwx------   36 juan  juan   1224 Nov 30 10:23 html
drwx------   18 juan  juan    612 Nov 30 10:23 silop
```

El archivo README contiene la licencia de la toolbox.. En el directorio `html` están dichas ayudas, que son generadas como se explicará más adelante. En el directorio `silop` residen las funciones de matlab propiamente dichas. Si la instalación es correcta, desde la línea de comandos deberá aparecer una descripción de las funciones disponibles como respuesta a `help silop`.

Primer contacto

La mayor parte de las funciones procesan datos provenientes de los Xsens, u otros IMUS, aunque la toolbox permite trabajar con distintos tipos de sensores. El formato más común para estos datos es el formato de **salida calibrada**, que consiste en 10 medidas en cada periodo de muestreo:



Por tanto la toolbox trabajará con conjuntos de señales similares a estas para cada sensor, (las tres aceleraciones, las tres velocidad de giro , y las tres componentes magnéticas). Dado que cada tipo de sensor es distinto, se debe consultar en cada caso las señales proporcionadas por el mismo.

Las unidades de trabajo son las correspondientes al sistema internacional de medidas: m/s^2 para las aceleraciones, rad/s para las velocidades angulares, y u.a. (unidades arbitrarias sin sentido físico real) para la medida del campo magnético. Todas las funciones de la toolbox esperan trabajar con estas unidades, y producen sus resultados en unidades del sistema internacional.

Desarrollo de aplicaciones rápidas. La estructura de las aplicaciones silop

Esta toolbox proporciona un esquema de desarrollo rápido de aplicaciones. Dichas aplicaciones constan de 6 pasos:

1. Inicio de la aplicación mediante **`initsilop()`**
2. Definición de los sensores a usar mediante **`addimu()`**
3. Conexión con la fuente de datos (Xsens, SF_3D o ficheros de datos) mediante **`connectsilop()`**
4. Selección de los algoritmos a usar mediante **`addalgoritmo()`**
5. Puesta en marcha de la aplicación mediante **`playsilop()`**

6. Finalización del programa pulsando la tecla ESC

La lista completa de algoritmos que se pueden incluir, así como los datos sobre el funcionamiento interno de las aplicaciones silop están en la sección de documentación para el desarrollador.

Para interpretar estas señales se debe tener en cuenta que:

1. Los acelerómetros tienen unos ejes de coordenadas estandar x,y,z

2. La función addimu espera que se indiquen cuales son las relaciones entre esos ejes y los ejes anatómicos (antero-posterior, medio-lateral y vertical). Por defecto si el acelerómetro está en el COG se asocia la x del acelerómetro con el eje anatómico vertical, la y del acelerómetro con el eje anatómico mediolateral y la z del acelerómetro con el eje anatómico antero-posterior. En otros puntos se conservan las señales originales del acelerómetro.

3. Las señales disponibles para los algoritmos se denominarán 'PUNTO.DATO', siendo PUNTO la localización del sensor (que se especifica en la llamada a addimu y puede ser arbitraria, p.e. COG, MUSLO_IZDO, PIE_IZDO, etc..) y DATO la señal a emplear. La lista de señales existentes en cada punto se presenta en pantalla después de la llamada a connectsilop , y será distinta para cada tipo de sensor: pe: (ACC_X,ACC_Y,ACC_Z,G_X,G_Y,G_Z,MG_X,MG_Y,MG_Z).

4. Todas las señales están referidas a los ejes indicados, por lo que para el COG se refieren por defecto a:

    - Coordenada X: El eje que inicialmente coincidiese con la dirección antero-posterior

    - Coordenada Y: El eje que inicialmente coincidiese con la dirección medio-lateral

    - Coordenada Z: El eje que inicialmente coincidiese con la dirección vertical

Estas funciones trabajan con una estructura de datos global denominada **SILOP_CONFIG.** La estructura contiene información necesaria para el funcionamiento del programa. El funcionamiento de dicha estructura es totalmente transparente al usuario.

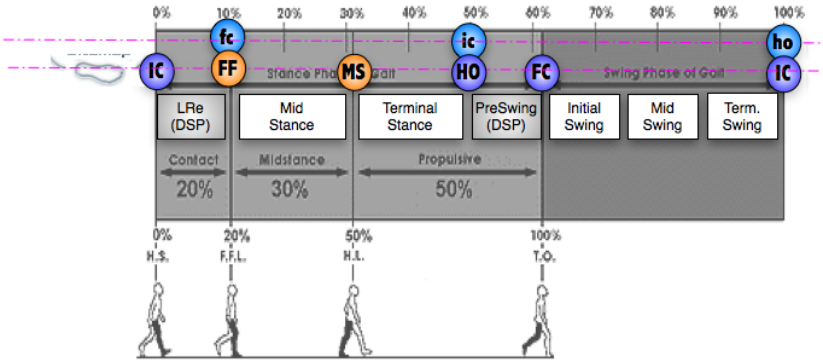
Si algo va mal durante la ejecución del programa, y este se cuelga o se interrumpe mediante la pulsación de CTRL-C el sistema puede quedar en un estado indefinido (datos incorrectos en SILOP_CONFIG). Para detener totalmente el sistema debe usarse en este caso el comando stopsilop()

Durante la ejecución de estas aplicaciones se puede elegir salvar los datos en un fichero con extensión .sl (silop log). Estos ficheros pueden ser recuperados mediante la función loadsilop(), que proporciona información sobre la configuración en la que estaban los sensores durante el experimento, los algoritmos que se ejecutaron, las señales capturadas, y (opcionalmente) los resultados de cada algoritmo. Despues de realizar sobre ellos las modificaciones que se deseen dichos datos pueden ser guardados de nuevo mediante savesilop()

Nomenclatura adoptada en lo referente al análisis del paso

Muchas funciones de esta librería tienen que ver con la locomoción humana. En la literatura especializada referida al análisis del paso coexisten diferentes nomenclaturas. En esta librería adoptaremos principalmente la propuesta en 1989 por el Centro Nacional de Rehabilitación Rancho Los Amigos (RLA) [1], recogidas también por Leonard Elbaum de la Universidad de Florida [2] y por Ellen C. Humphrey de la Northwestern University Medical School [3].

Las nombres de las fases y eventos que caracterizan un ciclo del caminar normal se resumen en la siguiente figura:



Los 5 principales eventos que ocurren en un ciclo del paso, referidos al pie que inicia el apoyo (pie de referencia o ipsilatera), son:

1) IC: Initial Contact (Heel Strike, Heel Contact, Foot Contact): instante del primer contacto del pie con el suelo.
2) FF: Foot Flat: instante en el que toda la planta del pie de referencia se apoya en el suelo (plantar grade).
3) MS: Mid Stance: ocurre cuando el pie opuesto al de referencia (contralateral), al balancearse, adelanta al pie de referencia o apoyo.
4) HO: Heel Off (Foot Off, Heel Rise, Push Off): instante en el que el talón del pie de referencia deja el suelo.
5) FC: Final Contact (Toe Off, Terminal Contact): instante en el que el pie contralateral deja el suelo, normalmente con los dedos del pie.

En la librería SiLoP hay una serie de funciones para detectar estos eventos a partir de señales inerciales, ya sea en linea (online) o fuera de linea (offline). En cada caso, la correspondencia entre la señal y el evento anatómico real se extrae de la literatura especializada (se detalla en cada caso).

Una descusión más detallada de los evento anatómicos del paso se puede encontrar en las referencias anteriores o en [4].

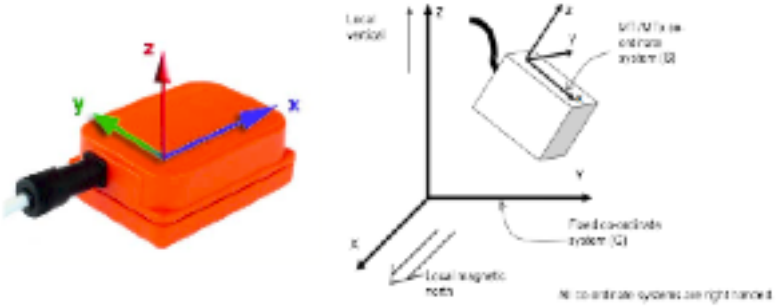
Referencias

1. Perry, D.J., Observational Gait Analysis. 4ª ed. 2004: Rancho Los Amigos National Rehabilitation Center. 72.
2. Elbaum, L. Anatomy, physiology, and biomechanics of walking. 2005 [cited; Class notes]. Available from: http://chua2.fiu.edu/faculty/elbauml/IDH3005-Fall05/Presentation%2010-10v2.doc
3. Humphrey, E.C. Kinesiology. Gait –Part II. 2002 [cited; class notes]. Available from: www.smpp.northwestern.edu/~jim/kinesiology/EllenGaitSlides2002.pdf.
4. Rafael C. González, Juan C. Alvarez, Fases y Eventos del Paso y su Reflejo en las Aceleraciones del COG. Informe Técnico 04-06, SiMuR Laboratory, Departamento de Ingeniería Eléctrica, Universidad de Oviedo, 2006.

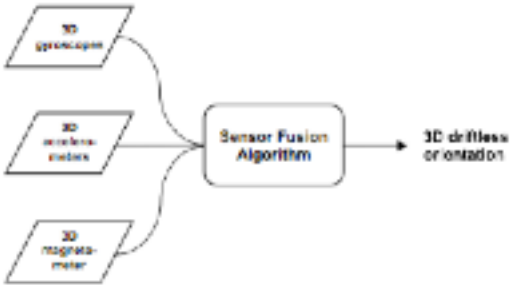
Introducción a los Xsens

Lo que sigue es un resumen de la documentación que acompaña a los Xsens, a la que hay que acudir para más detalles.

Los Xsens son sensores IMU (*Inertial Measurement Unit*) que miden aceleraciones, velocidades de giro e intensidad del campo magnético en tres ejes perpendiculares, o sistema de referencia local asociado a la caja que lo contiene:



Además el IMU incorpora algoritmos de integración sensorial (filtro de kalman) que proporciona la orientación del sensor sin deriva con precisión de 1 grado:



Typical performance characteristics of MTi and MTx orientation output.

Dynamic Range:	all angles in 3D
Angular Resolution:	0.05° RMS [2]
Static Accuracy (roll/pitch):	<0.5°
Static Accuracy (heading)[3]:	<1.0°
Dynamic Accuracy:	2° RMS [4]
Update Rate:	user settable, max 120 Hz [5]

Cuando realizamos una captura de datos, el Xsens devuelve de 1 a 3 archivos (configurable) de datos por sensor, correspondientes a los datos crudos sin calibrar, los datos calibrados y la orientación:

The output definition in calibrated data output mode is:

MTData MID 50 (0x32)

accX	accY	accZ	gyrX	gyrY	gyrZ	magX	magY	magZ	TS

All data elements in DATA field are FLOATS (4 bytes)
TS= time stamp (optional)

La orientación puede venir dada en cualquiera de estos tres formatos: quaternions, ángulos de Euler o matriz de rotación,

The output definition in quaternion output mode is:

MTData MID 50 (0x32)

q0	q1	q2	q3	TS

All data elements in DATA field are FLOATS (4 bytes)
TS= time stamp (optional)

The output definition in Euler-angle output mode is:

MTData MID 50 (0x32)

roll	pitch	yaw	TS

All data elements in DATA field are FLOATS (4 bytes)
TS= time stamp (optional)

The output definition in rotation matrix (DCM) output mode is:

MTData MID 50 (0x32)

a	b	c	d	e	f	g	h	i	TS

All data elements in DATA field are FLOATS (4 bytes)
TS= time stamp (optional)

Los **datos calibrados** vienen en `m/s²`, `rad/s` y `a.u.` (unidades arbitrarias normalizadas al campo magnético terrestre) respectivamente. La lectura de los **datos crudos** exige una lectura del archivo por bytes y la correspondiente conversión a formatos numéricos.

Las características de las señales proporcionadas por cada IMU se resumen en la siguiente tabla:

		rate of turn	acceleration	magnetic field	temperature
Unit		[deg/s]	[m/s²]	[mGauss]	[°C]
Dimensions		3 axes	3 axes	3 axes	-
Full Scale	(units)	+/- 300*	+/- 17	+/- 750	-55…+125
Linearity	(% of FS)	0.1	0.2	0.2	<1
Bias stability	(units 1σ)[11]	5	0.02	0.5	0.5[12]
Scale factor stability	(% 1σ)	-	0.05	0.5	-
Noise density	(units √Hz)	0.1	0.001	0.5 (1σ)	-
Alignment error[13]	(deg)	0.1	0.1	0.1	-
Bandwidth	(Hz)	40	30	10	-

Tabla: Características de las señales proporcionadas por el Xsens

La librería permite tanto trabajar con ficheros previamente capturados como realizar la captura en tiempo real desde los Xsens.

Introducción a los Sparkfun serial 3D accelerometers

El SerAccel v5 es un acelerómetro de tres ejes, con rango de medida de hasta +-6g y conexión serie. El sensor es un MMA7260Q, integrado en un único chip. La velocidad de transferencia de datos llega hasta los 125Hz a 57600bps, en el modo de transferencia ASCII empleado por la toolbox.
La configuración del dispositivo se debe hacer previamente a su uso desde una Terminal seria, empleandose posteriormente los bps ajustados a la hora de hacer la conexión desde Matlab©.
Los datos de aceleración obtenidos vienen dados en g’s. La toolbox los convierte automáticamente a m/s^2 por compatibilidad con el resto de las fuentes de datos.

Referencia

addalgoritmo

Propósito

Añade un algoritmo al sistema de procesamiento de las aplicaciones estandar de la toolbox

Sintaxis

```
addalgoritmo(nombre, valores_retorno, senhales, params);
```

Descripción

`addalgoritmo(nombre, n_valores_retorno, senhales, params, dependencias);`
Añade un algoritmo al sistema de procesamiento de las aplicaciones estandar de la toolbox.
No se pueden incluir algoritmos antes de realizar la conexión mediante `connectsilop()`, ni después de iniciarse el procesamiento con `playsilop()`.

Parámetros de entrada:

`nombre` -> nombre del algoritmo (función de la lista de algoritmos)

`n_valores_retorno` -> Cell array con los nombres de las señales que genera el algoritmo. Aún se permite usar la nomenclatura (obsoleta) de indicar sólo el número de resultados calculados por el algoritmo.

`senhales` -> Cell array con los nombres de las señales a emplear.

`params` -> Array con los parámetros que necesitará el algoritmo.

Parámetros de salida: Ninguno

Las señales disponibles para los algoritmos se denominarán `'PUNTO.DATO'`, siendo `PUNTO` la localización del sensor (especificada por addimu) y DATO la señal a emplear , que debe estar en la lista mostrada en pantalla por connectsilop.

Ejemplos

```
> initsilop();
> addimu('COG',205);
> connectsilop();
> %Añadimos el algoritmo alg_det_mov, que devuelve un dato, necesita la
%aceleracion vertical, tiene dos parámetros k y j
> addalgoritmo('alg_det_mov', {'COG.Mov'}, {'COG.Acc_Z'}, [k, j]);
%Añadimos el algoritmo alg_plot_senhales, que no devuelve nada,
%necesita una lista de señales, y no tiene parámetros
> addalgoritmo('alg_plot_senhales', 0, {'COG.G_X' 'COG.Acc_Z'}, []);
```

addimu

Propósito

Añade un IMU al sistema de procesamiento de las aplicaciones estandar de la toolbox

Sintaxis

```
addimu(posicion,numserie,orientacion);
```

Descripción

`addimu(posicion,numserie,orientacion);` Añade un IMU al sistema de procesamiento de las aplicaciones estandar de la toolbox.
Se debe incluir la lista completa de IMUs a usar antes de realizar la conexión.

Parámetros de entrada:

`posicion` -> Cadena de texto conteniendo la posición en la que está el sensor..

`numserie` -> numero de serie

`orientacion` -> vector que contiene la relación existente entre los ejes del acelerómetro y los ejes anatómicos que se usarán en la aplicación. Este vector debe ser de la forma:

`[eje antero-posterior, eje mediolateral, eje vertical]`.

Por defecto vale [3,-2,1] si el acelerómetro se sitúa en el COG, lo que indica que el eje 3 (Z) del acelerómetro será nuestro eje 1(antero-posterior o X) el eje -2(-Y) del acelerómetro será nuestro eje 2 (vertical o Y) y el eje 1 (X) del acelerómetro será nuestro eje 3 (Vertical o Z).

En otros puntos vale [1,2,3] por defecto, es decir, no se produce reorientación.

Se aceptan valores negativos para indicar que el eje anatómico y el del acelerómetro son opuestos

Muy importante: La orientación definida por este vector debe formar un triedro a derechas. En caso contrario, los resultados pueden ser imprevisibles.

Parámetros de salida: Ninguno

Ejemplos

```
> initsilop();
> addimu('COG',205);
> addimu('PIE_DCHO',252,[3,-2,1]);
```

buscamaximos

Propósito

Detecta todos los máximos de una señal

Sintaxis

```
maximos=buscamaximos(datos)
```

Descripción

Detecta todos los máximos de una función, analizando los cambios de signo de la primera diferencia.

`maximos=buscamaximos(datos);` devuelve un vector que indica la situación de los máximos.
El parámetro de entrada datos debe contener un vector con la señal a analizar.
El parámetro de salida `maximos` es un vector del mismo tamaño que datos, con el valor 1 en la posición de los máximos y cero en el resto.

Ejemplos

buscamaximosth

Propósito

Detecta todos los máximos de una señal que superen un determinado valor umbral

Sintaxis

```
maximos=buscamaximosth(datos,th)
```

Descripción

Detecta todos los máximos de una función que superen un determinado umbral, analizando los cambios de signo de la primera diferencia.

`maximos=buscamaximos(datos);` devuelve un vector que indica la situación de los máximos.
El parámetro de entrada `datos` debe contener un vector con la señal a analizar. `th` contiene el valor umbral por debajo del cual no se buscará un máximo.
El parámetro de salida `maximos` es un vector del mismo tamaño que datos, con el valor 1 en la posición de los máximos y cero en el resto.

Ejemplos

connectsilop

Propósito

Conecta el sistema de procesamiento de las aplicaciones estandar de la toolbox con la fuente de datos escogida

Sintaxis

```
connectsilop(driver, source, freq, updateeach, driver_opt);
```

Descripción

`connectsilop();`
Conecta el sistema con el XbusMaster de acuerdo a la configuración de IMUs previamente escogida, en el puerto 'COM24' a una frecuencia de 100Hz, con actualizaciones cada segundo, y la configuración del XbusMaster por defecto.

`connectsilop(driver);` Permite especificar el dispositivo con el que se va a trabajar. Por ejemplo:

'Xbus' -> XBusMaster
'SF_3D' -> Sparkfun 3D serial accelerometer
'Temporizador' -> Ficheros de datos

`connectsilop(driver, source);` Permite especificar el fichero y/o puerto del que se van a leer los datos. Por defecto COM24.

`connectsilop(driver, source, freq, updateeach);` Permite especificar los detalles del funcionamiento, freq=frecuencia de muestre, updateeach=periodo tras el cual se actualiza la información en pantalla.

`connectsilop(driver, source, freq, updateeach, driver_opt);` Permite especificar opciones especificas al usar el driver. Consulte la documentación del driver para su ajuste.:

Ejemplos

```
> initsilop();
> addimu('COG',204);
> connectsilop();
```

datacrop

Propósito

Eliminar manualmente un rango de datos de un archivo de datos calibrados del Xsens.

Sintaxis

```
datacrop
```

Descripción

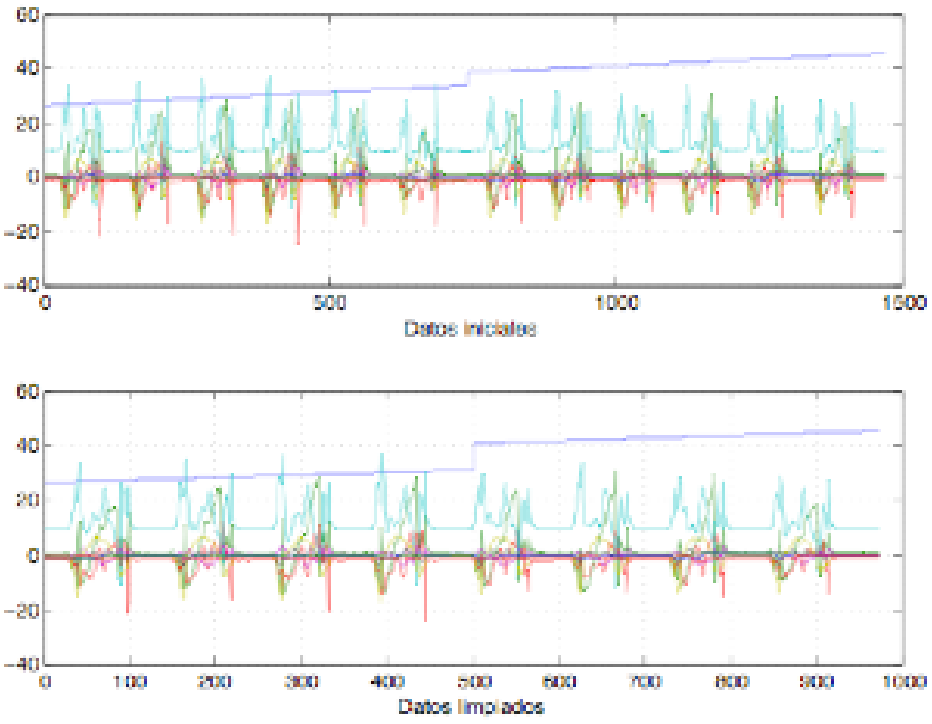
`datacrop` elimina manualmente un rango de datos de un archivo de datos calibrados del Xsens. Primero pregunta por el nombre del archivo de datos. Luego los dibuja para ayudar a definir las muestras inicial y final de datos a eliminar. Al introducir dichas muestras se eliminan los datos comprendidos entre ambas, se dibujan de nuevo los datos recortados, y se escribe en un archivo con el mismo nombre, y acabado en `.clean`

Ejemplos

Del archivo llamado `pie_001.log` eliminamos los datos del 500 al 1000 y guardamos el resto en `pie_001.log.clean`

```
>> datacrop
Nombre del archivo de datos a limpiar: pie_001.log
Archivo leido OK, numero total de muestras:
tam =
       1472  10

Seleccione el intervalo de muestras a eliminar.
Muestra inicial: 500
Muestra final: 1000
```



distancia_arco

Propósito

Calcula la distancia recorrida en un paso basandose en el modelo de movimiento angular a velocidad constante

Sintaxis

```
distancia=distancia_arco(AccVert)
distancia=distancia_arco(AccVert, freq)
distancia=distancia_arco(AccVert, freq, pierna)
```

Descripción

Aplica el modelo que relaciona la distancia recorrida en un paso con la aceleración normal en el instante en de foot-flat.
Esta función es incompatible con la función del mismo nombre disponible en SilopToolbox v0.2 o anterior

`distancia=distancia_arco(AccVert)` realiza los calculos de estimación de distancia AccVert debe contener la aceleración vertical del paso a estudiar mas una decima de segundo antes y después.
`distancia=distancia_arco(AccVert, freq )` permite especificar la frecuenca de muestreo. Por defecto vale 100Hz. El valor se conserva entre llamadas.

`distancia=distancia_arco(AccVert, freq, pierna)` permite especificar como tercer parámetro la longitud de la pierna efectiva del sujeto en estudio. Por defecto vale 1m, el valor se conserva entre distintas llamadas a la función

Referencia: an50 sca3000 accelerometer in velocity, distance and energy

measurement. AN-50. http://www.vti.fi

Ejemplos

distancia_pendulo

Propósito

Calcula la distancia recorrida en un paso mediante el modelo del pendulo invertido desde el COG

Sintaxis

```
distancia = distancia_pendulo(AccVert)
distancia = distancia_pendulo(AccVert,frec)
distancia = distancia_pendulo(AccVert,frec,pierna)
distancia = distancia_pendulo(AccVert,frec,pierna,correccion)
```

Descripción

`distancia = distancia_pendulo(AccVert)` calcula la distancia recorrida en base a la señal de aceleración vertical correspondiente a un paso y la frecuencia de muestreo. Teoricamente los instantes en los que se divida el paso no son relevantes para el resultado, sin embargo por convenio se suele llamar a la función con los datos correspondientes al paso desde un HS hasta el HS siguiente. Por defecto se toma una longitud de pierna de 0.8m

`distancia = distancia_pendulo(AccVert,frec)` permite especificar la frecuencia, que por defecto es de 100Hz. El parámetro se conserva entre llamadas, por lo que sólo es necesario indicarlo la primera vez.

`distancia = distancia_pendulo(AccVert,frec,pierna)` Permite especificar la longitud del radio del péndulo a usar. Anatómicamente esta distancia debe ser medida desde el suelo hasta el COG. Es opciona, por defecto vale 0.8m. El parámetro se conserva entre llamadas, por lo que sólo es necesario indicarlo la primera vez.

`distancia = distancia_pendulo(AccVert,frec,pierna,correccion)` añade un nuevo parámetro que permite desactivar la corrección del drift (corrección =0) Por defecto la corrección está activada. El parámetro se conserva entre llamadas, por lo que sólo es necesario indicarlo la primera vez.

Este método no es equivalente a la descripcon propuesta por Zijlstra, ya que esa propuesta no puede ser aplicada en tiempo real. En la implementación actual de la función cada paso se trata de forma independiente, y se aplica un algoritmo de corrección de la integral de la aceleración en lugar de un filtro paso alto, ya que dicho filtro requiere un alto número de muestras.

Ejemplos

distancia_pendulo_parcial

Propósito

Calcula la distancia recorrida en un paso mediante el modelo del pendulo invertido

mas desplazamiento.

Sintaxis

```
distancia = distancia_pendulo_parcial(AccVert,TO)
distancia = distancia_pendulo_parcial(AccVert,TO,frec)
distancia = distancia_pendulo_parcial(AccVert,TO,frec,hsensor)
distancia = distancia_pendulo_parcial(AccVert,TO,frec,hsensor,pie)
distancia = distancia_pendulo_parcial(AccVert,TO,frec,hsensor,pie,KSP)
```

Descripción

`distancia = distancia_pendulo_parcial(AccVert,TO)` Aplica el modelo del pendulo invertido para calcular el desplazamiento horizontal en función del desplazamiento vertical durante la fase de single stance del paso.
Durante la fase de double stance supone un desplazamiento constante igual al tamaño de pie indicado. Aplica una corrección para eliminar la media de las aceleraciones verticales, lo que es necesario para los casos en los que el drift de la integral es importante.
Los parámetros necearios son: AccVert: La aceleración vertical durante el paso que se está estudiando
`TO: La muestra en la que se produce el evento TO dentro de la secuencia`

`distancia = distancia_pendulo(AccVert,TO,frec)` Permite especificar la frecuencia de muestreo, que por defecto vale 100Hz, y se conserva entre llamadas.
`distancia = distancia_pendulo(AccVert,TO,frec, hsensor)` añade un parámetro extra (hsensor) para especificar la longitud de la pierna (desde el maleolus hasta el trocanter). Por defecto vale 0.8m y se conserva entre llamadas.
`distancia = distancia_pendulo(AccVert,TO,frec,hsensor, pie)` Permite especificar tanto la longitud de la pierna como la del pie (longitud de apoyo desde el calcaño hasta el “cayo”). Por defecto vale 0.15m y se conserva entre llamadas.
`distancia = distancia_pendulo(AccVert,TO,frec,hsensor, pie, KSP)` Permite especificar la altura del sensor en hsensor como la distancia desde el maleolus hasta el sensor (distancia equivalente a la usada por Zijlstra) y aplicar a posteriori un factor de corrección debido a la diferencia de distancia KSP=(distancia de sensor a maleolus)/(distancia de trocanter a maleolus)l. Por defecto KSP vale 1, el valor se conserva entre llamadas.

Modelo aplicado para el cálculo

El método implica la división de cada paso en dos partes, la correspondiente a la fase de single stance y la correspondiente a la fase de double- stance.
Durante la fase de single stance, el COG se desplaza de forma aproximada como un péndulo invertido, por lo que ese es el modelo aplicado. Siguiendo la línea de trabajos anteriores (Zijlstra), la distancia empleada debe ser medida desde el maleolus hasta el trocanter mayor (tobillo a fin de femur).
Dado que en realidad el sensor está situado en una posición más alta, su desplazamiento es mayor que el calculado de acuerdo con dicho modelo. Para corregir eso se aplica una corrección, basada en la diferencia entre la altura a la que está el sensor y la distancia anteriormente comentada. Este factor KSP, se puede calcular como el cociente entre dos distancias: distancia del sensor al maleolus, y distancia del trocanter mayor al maleolus.
Durante la fase de double stance, el desplazamiento del COG se asume que es proporcional al desplazamienot del punto de presión en el pie. Dado que este se produce entre el heel-bone(calcaño) y el primer metatarso (principio del dedo gordo), esa es la distancia que debe ser incluida en el método

Ejemplos

distancia_raizcuarta

Propósito

Calcula la distancia recorrida en un paso mediante el modelo empírico de la raiz cuarta

Sintaxis

```
distancia=distancia_raizcuarta(AccVert)
```

Descripción

Aplica el modelo empírico que relaciona la distancia recorrida en un paso con la raiz cuarta de la amplitud de la aceleración vertical.

`distancia=distancia_raizcuarta(AccVert)`
 realiza los calculos de estimación de distancia AccVert debe contener la aceleración vertical del paso a estudiar. La función no realiza un filtrado previo de la aceleración antes del cálculo de la distancia. Si se quiere imitar fielmente el metoro descrito por Weinberg la señal debe ser previamente filtrada a una frecuencia de 3Hz. (Por ejemplo mediante filtro0(AccVert,26,0.06) si estamos trabajando a 100Hz)

Los datos de distancia proporcionados no están calibrados. La calibración debe ser realizada por el usuario de la función

Referencia: H. Weinberg, “Using the adxl202 in pedometer and personal navigation applications,” 2002

Ejemplos

ejes_anatomicos

Propósito

Realinear los datos tomados por el sensor xSens de forma que las aceleraciones se correspondan con los ejes anatómicos y no con los ejes del dispositivo.

Sintaxis

```
acc_c=ejes_anatomicos(acc,acc_parcial)

acc_c=ejes_anatomicos(acc,acc_parcial,R)

[acc_c,R]=ejes_anatomicos(datos1,datos2,R)
```

Descripción

Tomando como base una señal, en la que los instantes iniciales la única aceleración es la de la gravedad realinea los ejes de referencia para que las aceleraciones se correspondan con los ejes antero-posterior, medio-lateral y vertical.

```
acc_c=ejes_anatomicos(acc,acc_parcial)
```

Toma los datos de acc que es una matriz que debe contener todos los datos de las tres aceleraciones del sensor y de acc_parcial conteniendo los mismos datos pero limitados al intervalo a estudiar

La función asume que los datos se tomaron de acuerdo a la orientación estandar del sensor xSens, y realiza la orientación de forma que se devuelven en acc_c las aceleraciones corregidas.

```
acc_c=ejes_anatomicos(acc,acc_parcial,R)
```

Toma un parámetro añadido que indica la orientación del sensor. Esta orientación puede venir indicada de 3 posibles formas:

1. R puede ser una matriz de rotacion que se aplicará directamente para transformar los datos
2. R puede ser un vector de 3 elementos, indicando que aceleraciones (1,2,3,-1,-2,-3) se corresponden de forma aproximada con las aceleraciones antero-posterior, medio-lateral y vertical.

```
[acc_c,R]=ejes_anatomicos(datos1,datos2,R)
```

Además de la señal de aceleraciones corregida devuelve un segundo parámetro (R) que contiene la matriz de rotación aplicada para la corrección

Ejemplos

entrena_ident_act

Propósito

Ajusta los parámetros de una red neuronal para que realice la identificación del movimiento realizado por un individuo, en base a un conjunto de datos patrón previamente capturados. La frecuencia de muestreo debe ser OBLIGATORIAMENTE de 50Hz.

Sintaxis

```
[redneuronal,parametros]=
entrena_ident_act(datos_bajarescaleras,datos_bajarrampa,datos_andar,
                                        datos_subirrampa,datos_subirescaleras)
```

Descripción

Entrena la red neuronal para un individuo mediante el método descrito en el estudio *Accelerometry Based Clasiffication of Walking Patterns Using Time Frecuency Analysis* de Ning Wang y Eliathamby Ambikairajah.

Los parámetros de entrada son los siguientes:

- `datos_bajarescaleras`: variable tipo cell array dentro de la cuál cada uno de los elementos es una matriz con los datos de cuando el individuo se encontraba bajando escaleras.
- `datos_bajarrampa`: variable tipo cell dentro de la cuál cada uno de los elementos es una matriz con los datos de cuando el individuo se encontraba bajando rampa.
- `datos_andar`:variable tipo cell dentro de la cuál cada uno de los elementos es una matriz con los datos de cuando el individuo se encontraba andando.
- `datos_subirrampa`:variable tipo cell dentro de la cuál cada uno de los elementos es una matriz con los datos de cuando el individuo se encontraba subiendo rampa.
- `datos_subirescaleras`:variable tipo cell dentro de la cuál cada uno de los elementos es una matriz con los datos de cuando el individuo se encontraba subiendo escaleras.

Las matrices que se encuentran formando parte de las variables cell tienen que tener al menos 256 filas (cada fila es una muestra) y 4 columnas (opcionalmente 10). Estas columnas tienen que tener el formato descrito en la página 5, sección Primer Contacto.

Los parámetros de salida son: redneuronal y parametros, valores que deben ser usados por `ident_act` para la posterior identificación de una actividad

Ejemplos

```
Se crean los cell array
> datos_bajarescaleras=cell(1,3);
> datos_subirescaleras=cell(1,3);
...
Se van cargando los cell array
> datos_bajarescaleras{1,1}=load('./entrenamiento/bajarescaleras1.log');
> datos_bajarescaleras{1,2}=load('./entrenamiento/bajarescaleras2.log');
...
Se entrena la red
> [redneuronal,parametros]=entrena_ident_act (datos_bajarescaleras, datos_bajarrampa, datos_andar,
                              datos_subirrampa, datos_subirescaleras)
```

evaluasalto

Propósito

Calcula los parámetros más relevantes de un salto en base a los eventos detectados previamente

Sintaxis

```
duracion=evaluasalto(tiempos)
[duracion,altura,energía]=evaluasalto(tiempos)
[duracion,altura,energía]=evaluasalto(tiempos, frecuencia)
[duracion,altura,energía]=evaluasalto(tiempos, frecuencia, peso)
```

Descripción

`duracion=evaluasalto(tiempos)` calcula un vector con la duracion en segundos de cada uno de los saltos identificados en la secuencia `tiempos`.Este parámetro debe haber sido obtenido previamente mediante una llamada a eventossalto(...)
`[duracion,altura,energía]=evaluasalto(tiempos)` calcula también la altura alcanzada en cada salto, (en metros)y la energía aplicada para llegar a dicha altura (en Julios).
`[duracion,altura,energía]=evaluasalto(tiempos, frecuencia)` permite indicar la frecuencia de muestreo a la que se tomaron los datos, que por defecto es de 100Hz
`[duracion,altura,energía]=evaluasalto(tiempos, frecuencia, peso)` permite indicar el peso conjunto del atleta mas las pesas que este lleve. Por defecto este parámetro son 75Kg.

Ejemplos

Detectamos los eventos de un grupo de saltos y calculamos la duración y altura asociadas.

```
>> tiempos=eventossalto(datos,50);
>> [duracion,altura,energia]=evaluasalto(tiempos,50)
duracion =
   0.5800
   0.6200
altura =
   0.4125
   0.4714
energia =
  303.5046
  346.8108
```

evaluasentadilla

Propósito

Calcula los parámetros más relevantes de un conjunto de sentadillas salto en base a los eventos detectados previamente

Sintaxis

```
evaluasentadillas(tiempos)
evaluasentadillas(tiempos,frecuencia)
evaluasentadillas(tiempos,frecuencia,peso)
desplazamiento=evaluasentadillas(tiempos,frecuencia,peso)
[desplazamiento,velmax,velmed,fmax,fmed,potmax,potmed]=
                                        evaluasentadillas(tiempos,frecuencia,peso)
```

Descripción

`evaluasentadillas(tiempos)` calcula los vectores de velocidad, posición, fuerza y potencia instantáneos correspondientes a un conjunto de sentadillas y los representa gráficamente. El parámetro tiempos tiene que ser una matriz de tiempos tal y como se calcula por `eventossentadillas`

`evaluasentadillas(tiempos,frecuencia)` permite especificar la frecuencia de muestreo, mediante el parámetro `frecuencia. El valor por defecto del parámetro es de 100Hz.`

`evaluasentadillas(tiempos,frecuencia,peso)` permite especificar también la carga desplazada por el atleta. El valor por defecto es de 75Kg.

`desplazamiento=evaluasentadillas(tiempos,frecuencia,peso)` calcula un vector `desplazamiento` que contiene los valores de desplazamiento de cada una de las sentadillas. Cuando `evaluasentadillas` proporciona parámetros de salida detiene la representación gráfica.

`[desplazamiento,velmax,velmed,fmax,fmed,potmax,potmed]=`
`evaluasentadillas(tiempos,frecuencia,peso)` calcula los siguientes parámetros adicionales:

- `velmax` - matriz nx2 con las velocidades máximas de cada sentadilla (excéntrica y concéntrica)
- `velmed` - matriz nx2 con las velocidades medias de cada sentadilla (excéntrica y concéntrica)
- `fmax` - matriz nx2 con las fuerzas máximas de cada sentadilla (excéntrica y concéntrica)
- `fmed` - matriz nx2 con las fuerzas máximas de cada sentadilla (excéntrica y concéntrica)
- `potmax` - matriz nx2 con las potencias máximas de cada sentadilla (excéntrica y concéntrica)
- `potmed` - matriz nx2 con las potencias medias de cada sentadilla (excéntrica y concéntrica)

Ejemplos

Detectamos los eventos de un grupo de saltos y representamos gráficamente los resultados.

```
>> tiempos=eventossentadillas(datos,50);
>> evaluasentadillas(tiempos,50)
>>
```

eventpierectoff

Propósito

Detección de los principales eventos con un giroscopo situado sobre el metatarso del pie.

Sintaxis

```
tiempos=eventpierectoff(giroY,frec)
tiempos=eventpierectoff(giroY,frec,vthres)
```

Descripción

La siguiente detección de eventos se basa en la descripción dada en [1]. Se parte de un giróscopo situado entre el metatarso y las falanges de los dedos del pie, para medir el giro del pie en el plano sagital:

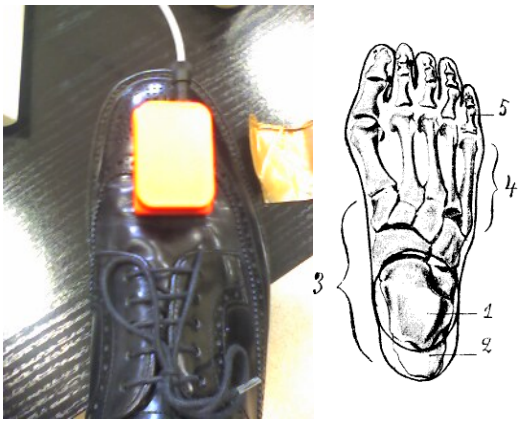
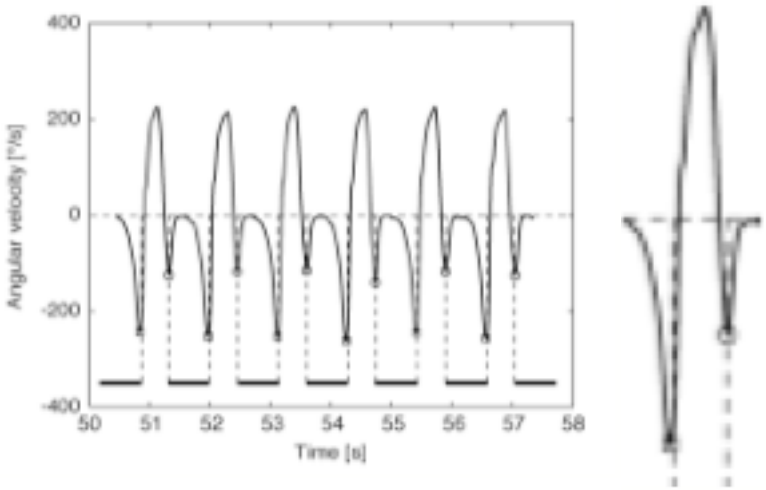


Figura: Foot (anatomic): 1-Ankle 2-Heel bone 3-Instep 4-Metatarsus 5-Toe

La señal del giro sagital presenta entonces el siguiente aspecto:



donde se pueden reconocer sucesivamente cinco eventos característicos relacionados con el pie en cuestión:

- Heel Off (HO) o levantamiento del talón: la señal valía cero y empieza bruscamente a caer a valores negativos
- Final Contact (FC) o levantamiento del pie: el pico negativo mas grande de la señal
- Mid-swing (MW) o pie en el aire: el pico positivo más grande (y único)
- Initial Contact (IC) o primer contacto con el suelo: el pico negativo siguiente al MW, inferior en valor absoluto al del FC
- Foot Flat (FF) o contacto planar del pie con el suelo: la señal vuelve a un valor aproximadamente cero.

Esta función busca los FC e IC a partir de la señal del giro filtrada a 6 Hz, como los máximos de la señal negativa (a partir de un cierto threshold). El evento FF se busca desde cada IC, cuando el giro está cerca de cero (a partir de un cierto threshold). Análogamente, el evento HO se busca hacia atrás desde cada FC cuando el giro está cerca de cero (a partir de un cierto threshold). El MW es el único máximo positivo, si se quiere utilizando también un threshold en su búsqueda. Los valores por defecto de estos 4 thresholds son [60 20 20 160] deg/s, respectivamente.

`tiempos=eventpierectoff(giroY,frec)` busca los eventos a partir de la señal del giróscopo `giroY` leído en deg/s y muestreada con frecuencia `frec` herzios, y devuelve una matriz `tiempos` con 5 columnas por cada muestra de la señal [HO FC MW IC FF] puestas a uno si se detectó el evento correspondiente en esa muestra ó con un cero en caso contrario. Se utilizan los valores por defecto de los cuatro thresholds anteriores, a saber [60 20 20 160] deg/s, respectivamente.

`tiempos=eventpierectoff(giroY,frec,vthres)` busca los eventos como antes, pero seleccionando los valores de los cuatro thresholds implicados `vthres`, a saber [th_ICyFC th_FF th_HO th_MW] en deg/s.

Ejemplos

Detectamos los eventos de la señal `giroY` y devolvemos 5 vectores con los instantes en los que se detectaron los 5 eventos de estudio, respectivamente:

```
>> tiempos=eventpierectoff(giroY*(180/pi),frecuencia,[90 20 20 160]);
>> giroYf=tiempos(:,6)*(pi/180);
>> ho=find(tiempos(:,1)==1);
>> fc=find(tiempos(:,2)==1);
>> mw=find(tiempos(:,3)==1);
>> ic=find(tiempos(:,4)==1);
>> ff=find(tiempos(:,5)==1);
```

Referencias

1. Sabatini, A. M., C. Martelloni, et al. (2005). "Assessment of walking features from foot inertial sensing." IEEE Transactions on Biomedical Engineering 52(3): 486-494.

eventosCOGrecto

Propósito

Detección de los principales eventos con un acelerómetro situado en el COG

Sintaxis

```
tiempos=eventosCOGrecto(accHor,accVert,frec)
```

Descripción

La siguiente detección de eventos se principalmente en la descripción dada por Auvinet.

Los eventos reconocidos son:

1. El evento identificado como "foot flat" (FF) por Auvinet, correspondiente al instante de máxima carga de peso.
2. El instante de Heel Strike, (HS)
3. El instante de Toe Off
4. El instnate de mid stance, MS;
5. El instante que probablemente se corresponde con el push-off. Incorrectamente identificado por Auvinet como el midstance (Auvinet-2002).

Esta función busca los FC e IC a partir de la señal de aceleraciones original, procediendo para ello a realizar los filtrados apropiados. Debido a estos procedimientos de filtrado la función no es válida para ser aplicad online.

Los parámetros de entrada son:

accHor: un vector conteniendo la aceleración horizontal de todo el intervalo a estudiar

accVert: un vector conteniendo la aceleración verticla para el mismo intervalo

frec: un entero que contiene la frecuencia de muestreo. Es opcional y por defecto vale 100Hz.

La función devuelve una matriz de tiempos, en el que cada columna se corresponde con la siguiente información:

1. El vector de aceleraciones verticales original
2. El vector de aceleraciones horizontales original
3. Un vector con el valor 1 en los instantes correspondientes a FF
4. Un vector con el valor 1 en los instantes correspondientes a HS
5. Un vector con el valor 1 en los instantes correspondientes a TO
6. Un vector con el valor 1 en los instantes correspondientes a MS
7. Un vector con el valor 1 en los instantes correspondientes a ¿HO?

Patrones de aceleraciones en el paso

Patrón de la aceleración vertical

La siguiente figura resume cómo es el patrón de la aceleración vertical según la literatura referenciada y con la nomenclatura antes expuesta. El patrón tiene 6 puntos significativos, con tres picos positivos (1,3,5), uno negativo (6), y dos mínimos positivos (2,4):

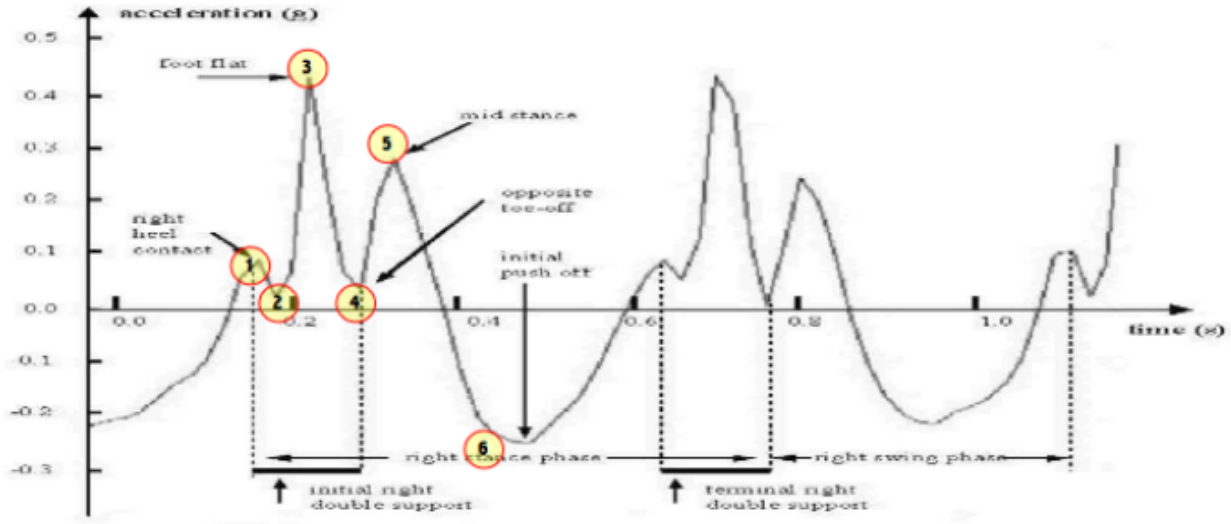


Fig. 1: patron de la aceleración vertical del COG.

Las relaciones más notables con los eventos del paso serían:

El máximo (3) corresponde a un evento "anatómico" (FF, en medio de la fase ***Contact***).

El máximo (5) corresponde a un evento "anatómico" (MS, en medio del ***Midstance***). Según algunos autores. Otros lo identifican con el punto final de la fase, en el mínimo (6)

Los tres picos positivos (1,3,5) se corresponden, sucesivamente, a un evento de "marca real" (HC) y dos "anatómicos" (FF, MS). El primero de ellos (1) no coincide con los resultados experimentales que se comentarán posteriormente.

El pico negativo (6) se corresponde al evento "marca" HO. Tampoco coincide con los resultados experimentales.

Los 3 eventos "marca" (IC, HO, TC) se corresponden con el primer máximo, el mínimo negativo y el mínimo entre los dos últimos máximos (1,6,4) respectivamente.

Del mínimo al máximo (6-1-2-3-4) es la fase de propulsión.

El punto (2) no tiene significado conocido.

Patrón de la aceleración anteroposterior

La siguiente figura muestra la aceleración antero-posterior del COG. Consta de dos máximos (7,1) y dos mínimos (3,5) consecutivos que se concentran en un 30% del total del paso. Los puntos 8 y 4 son los correspondientes valles entre los máximos (8 entre 7 y 1) y los mínimos (el 4 entre 3 y 5). El punto 2 no se ha utilizado, para facilitar posteriores comparaciones con la aceleración vertical. Del punto mínimo 5 al máximo 7 hay un aumento lineal de la aceleración que pasa por el punto 6 ó HO, pero que no se detecta en la señal .

[Brandes06, Zijlstra04]

5 EVENTOS:
IC ó FC: Initial Contact (Feet Contact)
HC ó HS: Heel Contact (Heel Strike)
FF: Feet Flat
MS: Mid Stance [EN CUESTIÓN...]
HO ó HR ó PO: Heel Off (Heel Rise, Push Off)
TC: Terminal Contact
TO: Toe Off

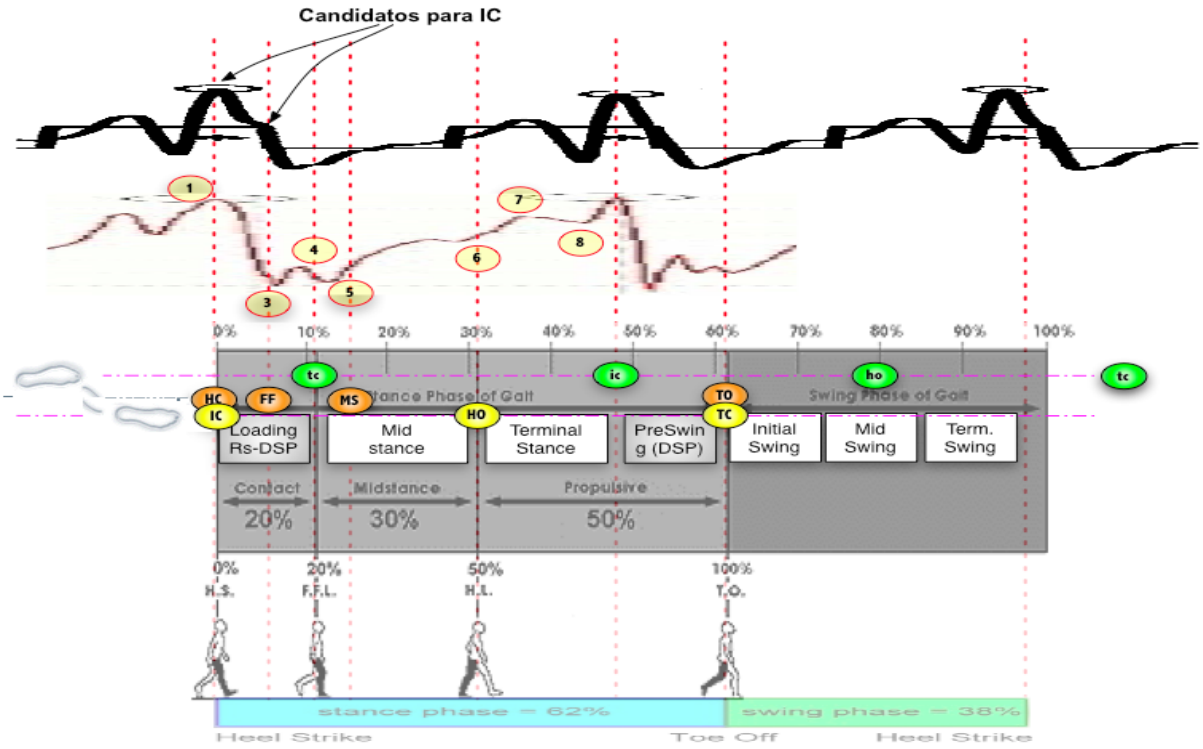


Fig. 2: Patrón de la aceleración antero-posterior del centro de masas.

Las relaciones de este patrón de curva con los eventos del paso son:

No está claro si el evento IC corresponde al punto (1) de aceleración máxima ó a un punto intermedio antes del mínimo (3).

El mínimo principal (3) corresponde al evento FF.

De (1) a (3) el COG decelera, y el resto del ciclo acelera. Una explicación razonable es que durante la fase de contacto se frena por efecto del contacto mismo, mientras que el resto del tiempo se acelera por el impulso que se imprime para avanzar. La relación en cuanto a duración entre aceleración/frenado es de un 4/1. Si fuera, por ejemplo, de un 5/1 implicaría un modo de caminar más económico: más avance con menos esfuerzo (hipótesis sin verificar).

Si escogemos como evento IC el del punto intermedio entre 1 y 3, la interpretación sería que el frenado comienza algo antes del IC, y que la aceleración acaba en el HO.

Ejemplos

eventospiraguas

Propósito

Detecta los principales eventos de cada palada, en una secuencia de datos capturada previamente. Por ahora sólo se detecta el instante de máximo empuje.

Sintaxis

```
tiempos=eventospiraguas(AccHor)
tiempos=eventospiraguas(AccHor,frecuencia)
```

Descripción

`tiempos=eventospiraguas(AccHor, frecuencia)` calcula los instantes en los que cada palada produce la máxima aceleración de avance. El vector `AccHor` debe contener la aceleración de avance de la piragua. Este vector puede tener el tamaño que se quiera, pero debe corresponderse únicamente a datos de avance de la piragua.
El parámetro `frecuencia` es opcional y por defecto vale 100Hz.
La función devuelve un vector de tiempos, que contiene en la primera columna la aceleración, y en la segunda columna un 1 en los instantes en los que se ha detectado el evento y un 0 en el resto

Ejemplos

Detectamos los eventos y luego representamos la señal de aceleración junto a los eventos detectados.

```
>> tiempos=eventospiraguas(datos,100);
>> plot(tiempos)
```

eventossalto

Propósito

Detecta los principales eventos de cada salto, en una secuencia de datos capturada previamente. Se detectan punto de inicio y final de salto, así como puntos auxiliares

Sintaxis

```
tiempos=eventossalto(AccVert)
tiempos=eventossalto(AccVert,frecuencia)
```

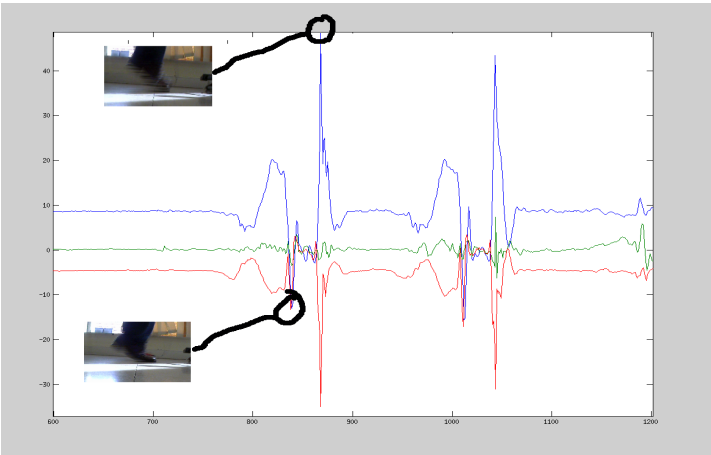
Descripción

`tiempos=eventossalto(AccVert)` calcula los instantes relevantes de luna secuencia de saltos.. El vector `AccVert` debe contener la aceleración vertical del COG, correspondiente a uno o más saltos. La señal puede incuir periodos en los que se haya permanecido estático, pero no datos correspondientes a otros movimientos. Si la señal tiene algún salto incompleto los eventos detectados pueden ir incorrectamente emparejados.

`tiempos=eventossalto(AccVert,frecuencia)` Permite especificar mediante el parámetro `frecuencia` la frecuencia de muestreo, es opcional y por defecto vale 100Hz.

La función devuelve una matriz de tiempos, en el que cada columna se corresponde con la siguiente información:

1. 1 vector de aceleraciones verticales original
2. Un vector con el valor 1 en los instantes correspondientes al inicio del salto (indicado en la gráfica)
3. Un vector con el valor 1 en los instantes correspondientes al inicio del contacto (paso por g anterior al fin del salto)
4. Un vector con el valor 1 en los instantes correspondientes al fin del salto (apoyo intenso, indicado en la gráfica))
5. Un vector con el valor 1 en los instantes correspondientes al inicio de la preparación del contacto. Mínimo de la aceleración justo anterior al contacto



Ejemplos

eventossentadillas

Propósito

Detecta los principales eventos de cada sentadilla, en una secuencia de datos capturada previamente. Se detectan punto de inicio y final de sentadilla, así como el punto central de cada una

Sintaxis

```
tiempos=eventossentadillas(AccVert)
tiempos=eventossentadillas(AccVert,frecuencia)
```

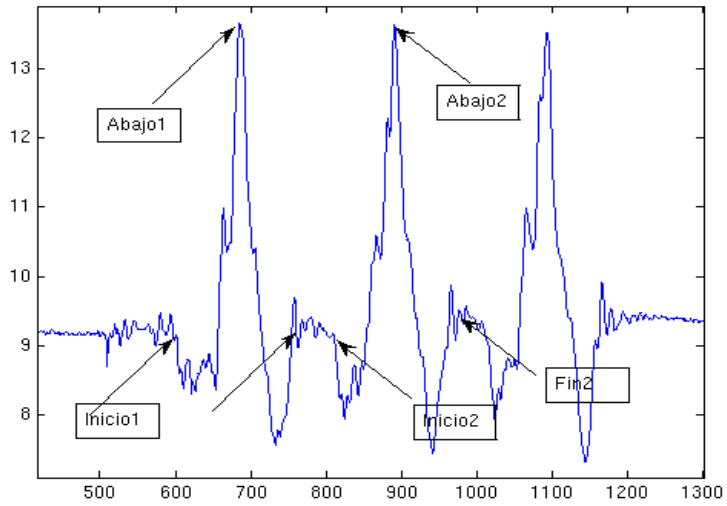
Descripción

`tiempos=eventossentadillas(AccVert)` calcula los instantes relevantes de luna secuencia de sentadillas. El vector `AccVert` debe contener la aceleración vertical del COG, correspondiente a una o más sentadillas. La señal puede incuir periodos en los que se haya permanecido estático, pero no datos correspondientes a otros movimientos. Si la señal tiene algúna sentadilla incompleta los eventos detectados pueden ir incorrectamente emparejados.

`tiempos=eventossentadillas(AccVert,frecuencia)` Permite especificar mediante el parámetro `frecuencia` la frecuencia de muestreo, es opcional y por defecto vale 100Hz.

La función devuelve una matriz de tiempos, en el que cada columna se corresponde con la siguiente información:

1. vector de aceleraciones verticales original
2. Un vector con el valor 1 en los instantes correspondientes al inicio de la sentadilla (indicado en la gráfica)
3. Un vector con el valor 1 en los instantes correspondientes al centro de la sentadilla (marcados como abajo en la gráfica)
4. Un vector con el valor 1 en los instantes correspondientes al fin de la sentadilla (ver gráfica)



Ejemplos

eventos_RT

Propósito

Detecta eventos de Contacto Inicial y Contacto Final del paso en Tiempo Real mediante las señales de aceleración medidas en el COG. La frecuencia de muestreo debe ser 100Hz.

Sintaxis

```
[retardo_hs,retardo_to]=eventos_RT(AccAntPost,AccVert)

[retardo_hs,retardo_to]=eventos_RT(AccAntPost,AccVert,reset)
```

Descripción

`[retardo_hs,retardo_to]= eventosRT(AccAntPost,AccVert)` Devuelve 0 en cada una de las variables de salida, si no se ha detectado ningún evento de Contacto Inicial o Final (respectivamente), y devuelve el retardo en muestras del punto de estimación de Contacto Inicial/Final en otro caso.
Los parámetros de entrada `AccAntPost` y `AccVert` deben ser la última muestra de aceleración antero-posterior y la última muestra de aceleración vertical respectivamente, de un acelerómetro situado en el COG.
Llamadas consecutivas a esta función cada vez que un dato está disponible permiten la detección del evento de Contacto Inicial y el Contacto Final en Tiempo Real.

`eventosRT(AccAntPost,AccVert,reset)` permite reinicar los valores internos usados por lal función, para que comience a procear los datos de nuevo desde cero. Esto permite pasar a un nuevo experimento. El valor del parámetro `reset` puede ser cualquiera.

Ejemplos

```
% Leer un fichero de datos:
cog=load('datacog.log');
% Detectar eventos procesando datos 1 a 1:
for i=1:size(cog),
[reths, retto]=eventos_RT(cog(i,4),cog(i,2));
if (reths>0)  hs(i-reths)=1; end
if (retto>0)  to(i-retto)=1; end
end

% Dibujo de los resultados:
subplot(211), plot(cog(:,2:2:4));
subplot(212), bar(hs);
```

frecuenciapaladas

Propósito

Calcula la frecuencia de paladas de un remero

Sintaxis

```
frecuencias=frecuenciapaladas(tiempos)
frecuencias=frecuenciapaladas(tiempos,freq)
```

Descripción

`frecuencias=frecuenciapaladas(tiempos,freq)` calcula un vector con la frecuencia instantanea de cada una de las paladas realizadas.
El parámetro `tiempos` indicado debe ser un vector con 1 en cada instante en el que se detecto una palada y ceros en el resto de datos. El parámetro `freq` es opcional, y sirve para indicar la frecuencia de muestreo. Por defecto es de 100Hz.

Ejemplos

Detectamos los eventos de un tramo de datos, y calculamos la frecuencia de paladas asociada. Ambas funciones trabajan con la frecuencia por defecto de 100Hz.

```
>> tiempos=eventospiraguas(datos);
>> frecuencia=frecuenciapaladas(tiempos(:,2))
frecuencia =

   60.0000
   63.8298
   61.8557
   69.7674
   66.6667
   70.5882
   63.8298
     ...
```

filtro0

Propósito

Realiza un filtrado paso bajo FIR de fase cero.

Sintaxis

```
Y=filtro0(datos,orden,corte)
```

Descripción

`Y=filtro0(datos,orden,corte);` realiza un filtrado de tipo FIR sobre la señal dada.
El parámetro de entrada `datos` debe contener un vector con la señal a analizar.
`orden` es el orden del filtro a aplicar
`corte` es la frecuencia de corte normalizada. La frecuencia de corte debe estar entre 0 y 1, con 1

correspondiendo a la mitad de la frecuencia de muestreo

La salida (`Y`) se corresponde con la señal filtrada

Ejemplos

```
filtramos a 2.5Hz una señal muestreada a 100Hz. fcorte=0.05*100/2

> filtrado=filtro0(datos,60,0.05);
```

ident_act

Propósito

Identifica la actividad que se está realizando en base a una red neuronal previamente entrenada.

Las actividades identificadas son caminar y subir/bajar escaleras/rampas. La frecuencia de muestreo debe ser OBLIGATORIAMENTE de 50Hz.

Sintaxis

```
identificacion=ident_act(redneuronal,parametros,datos)
```

Descripción

Identifica la actividad que se está realizando en base a una red neuronal previamente entrenada.

Las actividades identificadas son caminar y subir/bajar escaleras/rampas. Para ello emplea el método descrito en el estudio de Ning Wang y Eliathamby Ambikairajah *Accelerometry Based Classification of Walking Patterns Using Time-Frecuency Analysis*.

Los parámetros de entrada a esta función son:

- `redneuronal`, parámetro devuelto por `entrena_ident_act`
- `parametros`, parámetro devuelto por `entrena_ident_act`
- `datos`: Matriz con los datos correspondientes a la señal en la que se quiere hacer la identificación. Tiene que tener 128 filas (una por periodo de muestreo) y 10 columnas, donde cada una de ellas presenta el formato indicado en la página 5, sección Primer Contacto. Si tiene más de 128 filas se ignorarán todas salvo las primeras 128.

El único parámetro de salida es `identificacion`, que es un vector de ¿128? elementos, indicando la actividad realizada en cada instante, mediante el siguiente código:

Código	1	2	3	4	5
Significado	bajando escaleras	bajando una rampa	andando	subiendo una rampa	subiendo escaleras.

Ejemplos

initsilop

Propósito

Inicializa el sistema de procesamiento de las aplicaciones estandar de la toolbox

Sintaxis

```
initsilop();
```

Descripción

`initsilop()` Inicializa el sistema de procesamiento de las aplicaciones estandar de la toolbox.

Debe ser el primer comando usado en dichas aplicaciones.
Después de la llamada a la función la aplicación SILOP queda lista para trabajar con 0 sensores y 0 algoritmos. Se deben añadir sensores y algoritmos a partir de este punto.

Ejemplos

```
> initsilop();
```

integrasig

Propósito

Integración de una señal hasta un cierto instante final, a partir del cual se supone que la señal integrada vale cero, y suponiendo que la condición inicial es cero

Sintaxis

```
v = integrasig(a)
v = integrasig(a,Eventof)
v = integrasig(a,Eventof,R)
v = integrasig(a,Eventof,R,Freq)
```

Descripción

`v = integrasig(a)` integra la señal `a` completa según la expresión:

$$v(i) = \sum_{j=0}^{i} a(j)$$

El vector resultante tiene el mismo tamaño que el original.

`v = integrasig(a, Eventof)` integra la señal `a` hasta el instante final `Eventof`, a partir del cual se supone que la señal integrada vale cero, y por tanto el resto del vector de la señal integrada se completa con ceros.

`v = integrasig(Sig,Eventof,R)` integra la señal `a` hasta el instante final `Eventof`, y para forzar a que la señal integrada valga cero en el instante final de la integración, se aplica un coeficiente k(i) de valores 1 en el instante inicial, y 0 en el instante final de la integración, según:

$$v(i) = k(i) \sum_{j=0}^{i} a(j)$$

El coeficiente de reseteo responde a la expresión matemática de una curva definida por un coeficiente `R` que tiene como valores extremos [0,1] entre los instantes inicial y final de la integración:

$$k(i) = \left(\frac{n_T - i}{n_T - 1} \right)^R$$

por ejemplo, si R=1, k(i) toma valores entre k=1 cuando `i=1` y k=0 cuando `i=`n_T (ver la recta en negro en la figura siguiente). Si 0<R<1 la recta se convierte en una curva (la linea en rojo para R=0.1) cuyo efecto es un reseteo más acusado cuanto más nos acercamos al instante final de integración. Si R=0 no se hace reseteo ninguno.

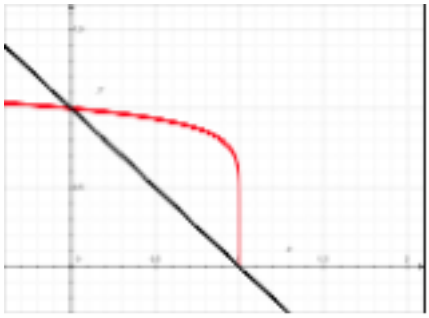


Figura 1. Modos de resetear la integral. El coeficiente k(i) varía entre 1 y 0. El coeficiente R hace que dicha variación deje de ser lineal si R es distinto de 1. En la gráfica se dibujan R=1 (recta en negro) y R=0.1(curva en rojo); el efecto de elevar a 0.1 es el de empezar el reseteo más bien al final del tramo que al principio.

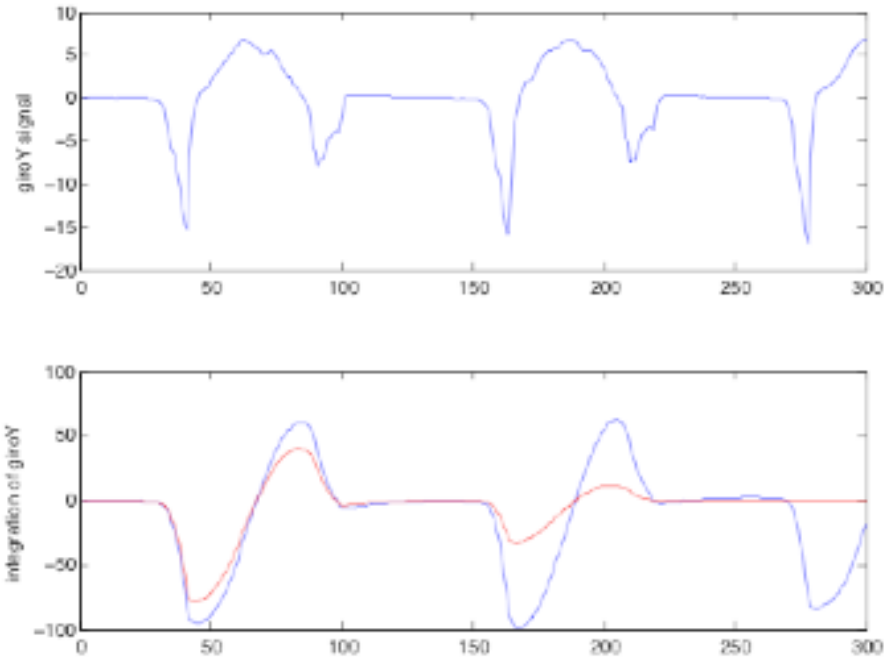
`v = integrasig(a,Eventof,R,f)` hace la misma integración anterior, siendo f la frecuencia de muestreo (en Hz) de la señal que se integra:

$$v(i) = \frac{1}{f} k(i) \sum_{j=0}^{i} a(j)$$

Ejemplos

Integramos los primeros 300 elementos de `giroY`, primero sin reseteo, y luego con reseteo lineal hasta el instante 250 de integración, para generar la figura siguiente:

```
>> subplot(211)
>> plot(giroY(1:300,1))
>> subplot(212)
>> plot(integrasig(giroY(1:300)))
>> hold
>> plot(integrasig(giroY(1:300),250,1),'r')
```



limpia_estatico

Propósito

Detectar los instantes de tiempo en los que se permanece estatico

al principio de los experimentos.

Sintaxis

```
estatico=limpia_estatico(aceleraciones,freq)

[estatico,aceleraciones]=limpia_estatico(aceleraciones,freq)
```

Descripción

`limpia_estatico` detectar los instantes de tiempo en los que se permanece estatico al principio de los experimentos, analizando a partir de que instante las variaciones de la señal empiezan a ser significativas.

Los parámetros de entrada deben ser un vector de aceleraciones capturado por los xsens (con la aceleración vertical en la última componenente) y la frecuencia de muestreo.

`estatico=limpia_estatico(aceleraciones,freq)` devuelve un vector de con ceros y unos, que indican los instantes de tiempo en los que ha habido o no aceleraciones.

`[estatico,aceleraciones]=limpia_estatico(aceleraciones,freq)` devuelve un segundo vector (aceleraciones) que contiene las aceleraciones corregido, de forma que en los instantes en los que se está estático valgan EXACTAMENTE ax=ay=0, y az=9.81, siendo ax la aceleración anteroposterior, ay la medio-lateral y az la vertical.

Ejemplos

loadsilop

Propósito

Carga los datos de un fichero de almacenamiento `.sl`

Sintaxis

```
[Config,captura] = loadsilop(fichero)
```

Descripción

`[Config,captura] = loadsilop(fichero)` Carga los datos de un fichero de almacenamiento .sl. Los datos quedan disponibles en las variables `captura` y `CONFIG`

Parámetros de entrada: El nombre del fichero `.sl` que se tiene que cargar.

Parámetros de salida:

- `Config`: Estructura de configuración de los sensores y algoritmos de la aplicación
- `captura`: Matriz con los datos capturados por los sensores y los resultados de los algoritmos(si están disponibles en el fichero)

Ejemplos

orientacioncompas

Propósito

Calcula la orientación en base a los datos de un compás/brújula situado en el COG

Sintaxis

```
[angulo,fiable] = orientacioncompas(campox,campoy,campoz)
[angulo,fiable] = orientacioncompas(campox,campoy,campoz,angulo0)
```

Descripción

`angulo = orientacioncompas(campox,campoy,campoz)` calcula el ángulo girado en cada instante de tiempo para el que se proporcionen los valores del vecor magnético. campox, campoy y campoz deben ser vectores conteniendo las componentes del campo magnético en cada dirección (antero-posterior, medio-lateral y vertical).

`angulo = orientacioncompas(campox,campoy,campoz,angulo0)` El valor de angulo0 permite indicar que ángulo absoluto se corresponde con el cero en el marco de referencia escogido. Por defecto ambos ángulos son iguales. El valor de angulo0 proporcionado se conserva entre llamadas, por lo que no es necesario especificarlo en cada llamada sucesiva.

Ejemplos

Obtenemos la dirección del campo inicial

```
>> angulo=orientacioncompas(1,1,1,0);
angulo=
     0.7854
```

P

Asignamos ese como el valor inicial y ralizamos los cálculos de cualquier otro ángulo en funcion de él.

```
>>orientacioncompas(1,1,1,angulo)
angulo=
     0
```

Cualquier otro ángulo se obtiene en base al indicado de referencia

```
>>orientacioncompas(1,1,1)
ans=
     0
>>orientacioncompas([1,2],[1,3],[1,4])
ans=
     0   0.1974
```

orientaciongiroscopo

Propósito

Calcula la orientación en base a los datos de un giróscopo situado en el COG

Sintaxis

```
angulo = orientaciongiroscopo(velgiro)
angulo = orientaciongiroscopo(velgiro,angulo0)
angulo = orientaciongiroscopo(velgiro,angulo0,freq)
```

Descripción

`angulo = orientaciongiroscopo(velgiro)` calcula el ángulo girado en cada instante de tiempo proporcionado por el vector velgiro. Velgiro puede tener el número de muestras que se desee. Los valores del ángulo inicial y de la frecuencia de muestreo se conservan de anteriores llamadas a la función. En el caso de que esta sea la primera llamada, se inicializan a sus valores por defecto (angulo=0, y frecuencia=100Hz).

`angulo = orientaciongiroscopo(velgiro,angulo0)` permite especificar un ángulo inicial a partir del cual realizar la integral.

`angulo = orientaciongiroscopo(velgiro,angulo0,freq)` permite especificar la frecuencia de muestreo. Esta es la forma preferida para llamar a la función por primera vez.

Ejemplos

Inicializamos el ángulo a 90º (pi/2) y una frecuencia de muestreo de 50Hz

```
>> angulo=orientaciongiroscopo(0,pi/2,50);
angulo=
     1.5708
```

Proporcionamos un vector con 3 velocidades y vemos los ángulos resultantes

```
>>orientaciongiroscopo([1,1,2])
ans=
     1.5908   1.6108  1.6508
```

orientacionkalman

Propósito

Calcula la orientación en base a los datos de un giróscopo y una brújula, situados ambos en el COG

Sintaxis

```
angulo=orientacionkalman(velgiro, campox,campoy,campoz)
angulo=orientacionkalman(velgiro, campox,campoy,campoz, angulo0)
angulo=orientacionkalman(velgiro, campox,campoy,campoz, angulo0,freq)
angulo=orientacionkalman(velgiro, campox,campoy,campoz, angulo0,freq,reset)
```

Descripción

`angulo = orientacionkalman(velgiro, campox,campoy,campoz)` calcula el ángulo girado en cada instante de tiempo proporcionado por los vectores velgiro y campox/y/z. Estos vectores pueden tener el número de muestras que se desee, pero deben ser todos del mismo tamaño. Los valores del ángulo inicial y de la frecuencia de muestreo se conservan de anteriores llamadas a la función. En el caso de que esta sea la primera llamada, se inicializan a sus valores por defecto (angulo=0, y frecuencia=100Hz).
Es importante que la función orientacioncompas NO debe ser llamada usando su parámetro de ángulo por defecto mientras se usa esta función, ya que ambas llamadas pueden interferir.
`angulo=orientacionkalman(velgiro, campox,campoy,campoz, angulo0)` permite especificar un ángulo inicial correspondiente al primer periodo de muestreo.
`angulo=orientacionkalman(velgiro, campox,campoy,campoz, angulo0, freq)` permite indicar la frecuencia de muestreo
`angulo=orientacionkalman(velgiro, campox,campoy,campoz, angulo0,freq,reset)` permite reiniciar de forma manueal todos los parámetros del filtro de Kalman a sus valores por defecto.

Modelo empleado:
El modelo de sistema empleado es:

$$\begin{bmatrix}\theta_{t+1}\\ \omega_{t+1}\\ b_{t+1}\end{bmatrix}=\begin{bmatrix}1 & T & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\theta_t\\ \omega_t\\ b_t\end{bmatrix}+\begin{bmatrix}v_1\\ v_2\\ v_3\end{bmatrix}$$

$$\begin{bmatrix}\theta_c\\ \omega_g\end{bmatrix}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 1\end{bmatrix}\begin{bmatrix}\theta_t\\ \omega_t\\ b_t\end{bmatrix}+\begin{bmatrix}n_1\\ n_2\end{bmatrix}$$

Los ruidos seleccionados para el proceso son:

1. v1=0, ya que la velocidad real determina totalmente la posición real
2. v2=0.1 variaciones que puede sufrir la velocidad.
3. v3=1e-6 variaciones que puede sufrir el bias. Esto supone asumirlo casi constante, ya que la mayor parte del bias vendrá del sensor y su colocación. Las fuentes de variación posibles son debidas a deslizamientos del sensor y/o cambios de orientación posteriores a la colocación.

Los ruidos elegidos para los sensores son:

- n1=0.1 ruido en el sensor de medida de hasta 1º/seg.
- n2={10,1e10} Cuando no se tiene motivos para suponer que la brújula está siendo perturbada su ruido se iguala a 10º. Cuando se considera que la brújula está siendo perturbada se fija su ruido a 1e10, lo que equivale a desactivarla y usar durante ese periodo únicamente el giróscopo.

Experimentalmente se ha determinado que los mejores resultados se obtienen cuando se desactiva la brújula si se cumple una de las dos siguientes condiciones:

- La componente vertical del campo magnético es excesivamente alta (mayor que 1 en valor absoluto) (en las unidades empleadas por los Xsens)
- La diferencia entre la orientación determinada por el filtro de Kalman y la determinada por el compas es superior a 5º.

La condición que se ha comprobado que funciona bien para reiniciar la brújula es:

- La diferencia entre el ángulo marcado por el filtro de Kalman y por la brújula baja de 2º

Ejemplos

Inicializamos el ángulo a 90º (pi/2) y una frecuencia de muestreo de 50Hz

```
>> angulo=orientacionkalman(0, 1,2,1, pi/2,50);
angulo=
     1.5708
```

playsilop

Propósito

Realiza el procesamiento de acuerdo a los IMUS y algoritmos indicados. Debe ser llamado después de todos los comandos de configuración

Sintaxis

```
playsilop(salvar,fichero);
```

Descripción

`playsilop();` Inicia el procesamiento de datos de acuerdo con la configuración escogida, y permanece en el hasta que se terminen los datos o **hasta que se pulse la tecla ESC.** No se debe terminar mediante la tecla Ctrl-C

`playsilop(salvar,fichero);` Permite guardar los datos en un archivo `.sl` (silop log). Si `salvar` toma el valor 1 se guardan los datos de la captura, si `salvar` toma el valor 2 se guardan también los resultados de los algoritmos.
El archivo por defectgo es `datos.sl`, aunque el nombre se puede modificar mediante el parámetro `fichero`. El fichero `.sl` realmente es un archivo de tipo zip, que contiene dentro 2 o tres ficheros, que son:

- config.mat: Fichero de Matlab con la estructura CONFIG en la que aparecen los sensores usados, y las columans correspondientes a cada dato.
- datos.log: Fichero de texto con los datos correspondientes a las señales capturadas. No se incluyen marcas de tiempos
- datos_alg.log: Fichero de texto con los datos correspondientes a los resultados de los algoritmos.

Ejemplos

```
> initsilop();
> addimu(.....);
> connectsilop();
> addalgoritmo(.....);
> playsilop();
```

savesilop

Propósito

Guarda los datos de un fichero de almacenamiento .sl.

Sintaxis

```
savesilop(fichero,Config,captura)
```

Descripción

`savesilop(fichero,Config,captura)` Guarda los datos en un fichero de almacenamiento .sl. Los datos se toman de las variables `captura` y `CONFIG`.

Parámetros de entrada: El nombre del fichero que se tiene que cargar, la estructura de configuración de los datos y los datos capturados.

Parámetros de salida: Ninguno

Ejemplos

`Rotación de los datos de aceleración de acuerdo a una matriz que se especifique.`

```
[config,captura]=loadsilop('exterior1.sl');
orden=[-3,-2,-1];
Rot=zeros(3,3);
for k=1:3
   Rot(k,abs(orden(k)))=sign(orden(k));
end;
captura(:,config.SENHALES.COG.Acc_X:config.SENHALES.COG.Acc_Z)=
                    captura(:,config.SENHALES.COG.Acc_X:config.SENHALES.COG.Acc_Z)*Rot';

savesilop('exterior1modificado.sl',config,captura)%
```

silopdemo

Propósito

Demostración de las capacidades de la toolbox

Sintaxis

```
echodemo silopdemo
```

Descripción

Muestra la forma de usar la toolbox para el desarrollo rápido de aplicaciones mediante un ejemplo de uso. Se puede ejecutar mediante el comando indicado, desde el menú inicio de Matlab o desde la lista de demos de Matlab (comando `demo`)

Ejemplos

stopsilop

Propósito

Detiene el sistema de procesamiento de las aplicaciones estandar de la toolbox.

Sintaxis

```
stopsilop(modo);
```

Descripción

`stopsilop()` Detiene el sistema de procesamiento de las aplicaciones estandar de la toolbox. Se puede reiniciar de nuevo mediante `playsilop()`.
`stopsilop(1)` Detiene el sistema de procesamiento de las aplicaciones estandar de la toolbox y destruye todas las conexiones asociadas a la misma (temporizadores, puertos, etc..). En este caso no se puede reiniciar de nuevo mediante `playsilop()`, y se debe reconfigurar de nuevo toda la aplicación.

Debe ser usado de forma explicita únicamente cuando el sistema no se detenga correctamente de forma automática. Este comando detiene los temporizadores y/o la lectura de datos desde el Xbus.
En el caso de que una aplicación se detenga de forma imprevista (p.e. mediante C-c) se debe usar `stopsilop()` de forma manual para detener los sistemas que hayan quedado funcionando.
Si se guardan ficheros de datos, esta es también la función que crea el `.sl` definitivo, por lo que los ficheros no estarán disponibles hasta que `stopsilop` se haya ejecutado.

Ejemplos

```
> ...
> playsilop();
error: ....
> stopsilop();
```

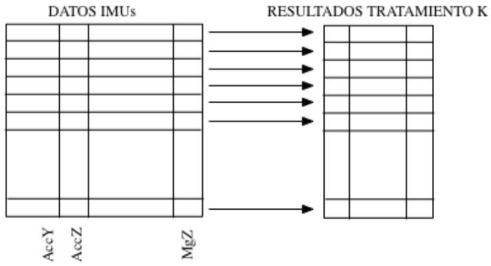
Referencia para el desarrollador

Se incluye a continuación información para el desarrollo de aplicaciones rápidas silop, así como documentación sobre aquellas funciones que sólo son de utilidad para el desarrollador de aplicaciones.

Funcionamiento del Núcleo

El núcleo del SiLoP se basa en una ventana lógica de datos (matriz) que contiene las últimas N muestras de los IMUs (el dato más nuevo se podrá encontrar en la última fila de la matriz). Cada columna de esta ventana será una variable medida por los IMUs. El tiempo total de datos disponible se corresponde con el parámetro `updateeach` de `connectsilop()`

Cada algoritmo de tratamiento dispone de una matriz para almacenar sus resultados. El número de columnas es configurable, permitiendo la adaptación a las necesidades concretas del algoritmo. Cada fila de esta ventana de tratamientos se asocia biunívocamente a una fila de la ventana de muestras. De esta manera, la fila f de la submatriz de tratamiento deberá contener el resultado de aplicar el tratamiento especificado a la muestra f de la ventana de datos.

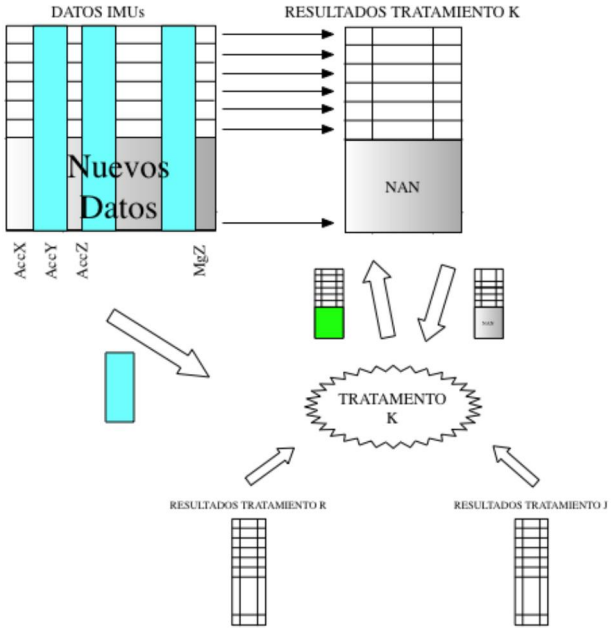


En el momento en que se reciba una nueva muestra de los IMUs:

La ventana de datos se desplaza para dejar espacio a los nuevos datos, los cuales serán insertados por el final.

Todas las ventanas de resultados asociadas a los distintos tratamientos se desplazan en la misma proporción rellenándose con valores NaN las filas correspondientes a las nuevas muestras. De esta manera, un valor NaN en una determinada posición de la ventana de tratamientos ha de interpretarse como un dato aún no calculado.

Se procede a invocar secuencialmente a todos los algoritmos de tratamiento. Cada algoritmo recibirá como parámetro (entre otros) el estado actual de la matriz de resultados correspondiente, que ha de procesar y devolver. Recibirá además una submatriz construida a partir de determinadas columnas de la matriz de datos muestreados, ventanas de resultados de otros algoritmos, y parámetros específicos de configuración, todo ello configurable a partir de la función `addalgoritmo`.



alg_bar_senhales

Propósito

Representa los valores de las señales seleccionadas, durante los últimos segundos usando un gráfico de barras. Muy útil cuando los valores sólo se muestran en instantes concretos, rodeados de NaN.

Sintaxis

```
addalgoritmo('alg_bar_senhales', 0, {'senales'}, []);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función representa gráficamente la dependencia que se desee.
No devuelve ningún valor. El procesamiento se realiza para cada dato disponible.

alg_det_event

Propósito

Detecta la existencia de los eventos HS y TO del paso

Sintaxis

```
addalgoritmo('alg_det_event',{'COG.HS','COG.TO'},{'COG.Acc_Z','COG.Acc_X'},[]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la localización de los eventos Initial Contact y Final Contact.
Devuelve un valor, indicando no hay evento (0) o hay evento (1) para cada instante de tiempo. El procesamiento se realiza para cada muestra disponible
Como datos de entrada usa únicamente la aceleración vertical y antero-posterior del COG, no tiene parámetros de configuración.

alg_det_mov

Propósito

Detectar lal existencia del movimiento

Sintaxis

```
addalgoritmo('alg_det_mov', {'COG.Mov'}, {'COG.Acc_Z'}, []);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `adddalgoritmo`.

Esta función determina si se está realizando algún movimiento, o si se se está estático.
Devuelve un valor, indicando estático (0) o en movimiento (1) para cada instante de tiempo. El procesamiento se realiza en tramos de 50 muestras.
Como datos de entrada usa únicamente la aceleración vertical del COG, no tiene parámetros de configuración.
Idealmente las funciones de estimación de movimiento deberían depender todas de esta, y actuar sólo cuando exista dicho movimiento.

alg_ejes_anatomicos

Propósito

Reorientar las señales de uno o varios sensores de acuerdo a los ejes anatómicos.

Sintaxis

```
addalgoritmo('alg_ejes_anatomicos', 0, {lista de señales}, [{lista de sensores}]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función reorienta las señales de los sensores indicados en la lista de sensores de acuerdo a los ejes anatómicos. Es un wrapper de la funcion `ejesanatomicos` en la que está la documentación completa.
Los primeros 200 datos medidos se usan para calcular la matriz de calibración. A partir de ese instante simplemente se aplica la matriz de rotación calculada en cada uno de los sensores. La matriz de rotacion aplicada queda disponible en `SILOP_CONFIG.SENHALES.Nombre_del_sensor.Rotacion`.

Como datos de entrada usa las tres señales de aceleración correspondientes a cada uno de los sensores que se quiera calibrar.
Sus parámetros de configuración son la lista de sensores que se quiere calibrar, especificada mediante el nombre de los puntos en los que están situados.

alg_est_2d

Propósito

Estima la posición 2d

Sintaxis

```
addalgoritmo('alg_est_2d', {'P.X','P.Y'}, {'P.Dist','P.Orient'}, []);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la posición al final de cada paso.
Devuelve dos valores, indicando las coordenadas (x,y) al final de cada paso, y NaN en los instantes en los que no se ha terminado un paso. El procesamiento se realiza para cada paso disponible.
Como datos de entrada usa las señales de distancia y orientación de cada paso, que generalmente habrán sido calculadas mediante `alg_est_dist` y `alg_est_orient`.

alg_est_dist_arco

Propósito

Estima la longitud de los pasos mediante el modelo desarrollado por VTI

Sintaxis

```
addalgoritmo('alg_est_dist_arco', {'COG.Dist'}, {'COG.Acc_Z','COG.HS'}, [freq,pierna]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la longitud de cada paso. Es un wrapper de la función `distancia_arco`. Devuelve un valor, indicando la distancia del paso, y NaN en los isntantes en los que no se ha terminado un paso. El procesamiento se realiza para cada paso disponible

Como datos de entrada usa únicamente la aceleración vertical del COG y la localización de los eventos Heel-Strike (Initial Contact).

Sus parámetros de configuración son la frecuencia de muestreo y la longitud de la pierna efectiva (ambos opcionales).

alg_est_dist_pendulo

Propósito

Estima la longitud de los pasos mediante el modelo del pendulo

Sintaxis

```
addalgoritmo('alg_est_dist_pendulo', {'COG.Dist'}, {'COG.Acc_Z','COG.HS'}, [freq,pierna]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la longitud de cada paso. Es un wrapper de la función `distancia_pendulo`. Devuelve un valor, indicando la distancia del paso, y NaN en los isntantes en los que no se ha terminado un paso. El procesamiento se realiza para cada paso disponible

Como datos de entrada usa la aceleración vertical del COG y la localización de los eventos Heel-Strike (Initial Contact) .

Sus parámetros de entrada son la frecuencia de muestreo y la longitud de la pierna (ambos opcionales).

Depende de la detección de eventos realizada por `alg_det_event`.

alg_est_dist_r4

Propósito

Estima la longitud de los pasos mediante la raiz cuarta

Sintaxis

```
addalgoritmo('alg_est_dist_r4', {'COG.Dist'}, {'COG.Acc_Z','COG.HS'}, []);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la longitud de cada paso mediante el modelo de la raiz cuarta. Es un wrapper de la función `distancia_raizcuarta`.
Devuelve un valor, indicando la distancia del paso, y NaN en los isntantes en los que no se ha terminado un paso. El procesamiento se realiza para cada paso disponible

Como datos de entrada usa la aceleración vertical del COG y los eventos del paso, estimados normalmente mediante una llamada a `alg_det_event`.

No tiene parámetros de configuración.

alg_est_dist_simur

Propósito

Estima la longitud de los pasos mediante el modelo del péndulo parcial

Sintaxis

```
addalgoritmo('alg_est_dist_simur', {'COG.Dist'}, {'COG.Acc_Z','COG.HS'}, [freq,hsensor,pie]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la longitud de cada paso mediante el modelo del péndulo parcial. Es un wrapper de la función `distancia_penduloparcial`.

Devuelve un valor, indicando la distancia del paso, y NaN en los isntantes en los que no se ha terminado un paso. El procesamiento se realiza para cada paso disponible

Como datos de entrada usa únicamente la aceleración vertical del COG y la localización del evento Heel Strike, normalmente calculado por `alg_det_event`.

Sus parámetros de entrada son la frecuencia de muestreo, la altura del sensor y el tamaño del pie (todos opcionales).

alg_est_orient_compas

Propósito

Estima la orientación mediante la información de la brújula. Esta función es un wrapper de `orientacioncompas()`

Sintaxis

```
addalgoritmo('alg_est_orient_compas',{'COG.Orient'},{'COG.MG_X','COG.MG_Y','COG.MG_Z'},[freq]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la orientación mediante la información de la brújula. Consulte la función `orientacioncompas()` para obtener más información.
Devuelve un valor, indicando el ángulo con la orientación en cada instante. El procesamiento se realiza para cada muestra disponible
Como datos de entrada usa las tres componentes del campo magnético.
Como parámetro se le puede pasar (opcionalmente) la frecuencia de muestreo.

alg_est_orient_gyro

Propósito

Estima la orientación mediante la información de un giróscopo

Sintaxis

```
addalgoritmo('alg_est_orient_gyro', {'COG.Orient'}, {'COG.G_Z'}, [freq]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la orientación mediante un giróscopo. Es un wrapper de la función `orientaciongiroscopo()`
Devuelve un valor, indicando el ángulo con la orientación en cada instante. El procesamiento se realiza para cada muestra disponible
Como datos de entrada usa únicamente la velocidad de giro vertical del COG.
El único parámetro, opcional, es la frecuencia.

alg_est_orient_kalman

Propósito

Estima la orientación mediante la información del giróscopo y la brújula. Esta función es un wrapper de `orientacionkalman()`

Sintaxis

```
addalgoritmo('alg_est_orient_kalman',{'COG.Orient'},{'COG.G_X','COG.MG_X','COG.MG_Y','COG.MG_Z'},
[freq]);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función determina la orientación mediante la información de la brújula y el giróscopo. Consulte la función `orientacionkalman()` para obtener más información.
Devuelve un valor, indicando el ángulo con la orientación en cada instante. El procesamiento se realiza para cada muestra disponible
Como datos de entrada usa las tres componentes del campo magnético y los datos de un giróscopo.
Como parámetro se le puede pasar (opcionalmente) la frecuencia de muestreo.

alg_plot_pos2d

Propósito

Representa la posición 2d estimada durante los últimos segundos.

Sintaxis

```
addalgoritmo('alg_plot_pos2d', 1, {'Pos.X','Pos.Y'}, []);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función representa la posición 2d durante todo el recorrido.
El resultado devuelto es para consumo propio (no redibujar datos antiguos)
El procesamiento se realiza para cada dato disponible
Como datos de entrada usa las estimaciones de posición, normalmente realizadas por `alg_est_2d`.

alg_plot_senhales

Propósito

Representa los valores de las señales seleccionadas, durante los últimos segundos.

Sintaxis

```
addalgoritmo('alg_plot_senhales', 0, {lista de señales}, []);
```

Descripción

Como todas las funciones alg_*, esta no está pensada para ser llamada directamente, sino por medio de `addalgoritmo`.

Esta función representa gráficamente las señales que se desee.
No devuelve ningún valor. El procesamiento se realiza para cada dato disponible.
Como datos de entrada usa únicamente las señales que se le especifiquen, no tiene parámetros de configuración.

energiawavelet

Propósito

realizar la descomposición wavelet de una señal

Sintaxis

```
[energiatotal,desvstdwavelet,valorRMS]=energiawavelet(aceleracion)
```

Descripción

Realiza la descomposición de una señal en sus correspondientes wavelets. Los parámetros son:
`aceleracion`:señal a procesar

`energiatotal`:Vector que tiene las energías de cada una de las componentes wavelet obtenidas de la descomposición de la señal aceleracion.

`desvstdwavelet`:Vector que tiene las desviaciones estándar de cada una de las componentes wavelet.

`valorRMS`:Vector que contiene los valores de la energía contenida en cada componente de la descomposición en wavelets

Ejemplos

getkey

Propósito

rget a single keypress

Sintaxis

```
ch=getkey()
```

Descripción

`CH = GETKEY()` waits (0.1 decimas de segundo como mucho después de adaptarse a la toolbox) for a keypress and returns the ASCII code. Accepts all ascii characters, including `backspace(8)`, `space(32)`, `enter(13)`, etc, that can be typed on the keyboard. Non-ascii keys (ctrl, alt, ..)

return a NaN. `CH` is a double.

`CH = GETKEY('non-ascii')` uses non-documented matlab 6.5 features to
return a string describing the key pressed. In this way keys like ctrl, alt, tab, etc. can also distinguished. `CH` is a string.

This function is kind of a workaround for `getch` in C. It uses a modal, but non-visible window, which does show up in the taskbar.

C-language keywords: `KBHIT`, `KEYPRESS`, `GETKEY`, `GETCH`

Ejemplos

```
fprintf('\nPress any key: ') ;

ch = getkey ;

fprintf('%c\n',ch) ;


fprintf('\nPress the Ctrl-key: ') ;

if strcmp(getkey('non-ascii'),'control'),

  fprintf('OK\n') ;

else

  fprintf(' ... wrong key ...\n') ;

end
```

localmaxima

Propósito

Determina los puntos que son máximos locales de una función en un entorno

Sintaxis

```
maximos=localmaxima(datos,N)
```

Descripción

Detecta todos los puntos de una función que son mayores que los N puntos anteriores y posteriores.

`maximos=localmaxima(datos);` devuelve un vector con la lista de posiciones en la que se encontraron los máximos.
El parámetro de entrada `datos` debe contener un vector con la señal a analizar. `N` contiene el número de muestras a analizar hacia delante y hacia atrás de la señal.
El parámetro de salida `maximos` es un vector con la lista de posiciones en la que se encontraron los máximos, o un vector vacio si no se encontraron dichos máximos

Ejemplos

ReqObjectAlignment

Propósito

Envía el mensaje ReqObjectAlignment al objeto XBusMaster. El proceso se queda bloqueado hasta recibir la información

Sintaxis

```
XBusMaster=ReqConfiguration(XBusMaster)
```

Descripción

Envía el mensaje RequestConfiguration al objeto XBusMaster. El proceso se queda bloqueado hasta recibir la información

Input parameters:

`XbusMaster`**:** Objeto con la información del dispositivo.

Output parameters:

`XbusMaster`**:** Es el mismo objeto de entrada que puede haber sido modificado durante la llamada. La información estará disponible en XbusMaster.Conf.Dev(k).Orientacion

Referencia:

“MTi and MTx User Manual and Technical Documentation.” Document MT0100P, Revision I, Enero 2007. Xsens Technologies B.V.

“XM-B Technical Documentation.” Document XM0101P, Revision C, Febrero 2007. Xsens Technologies B.V.

“MTi and MTx Low-Level Communication Documentation.” Document MT0101P, Revision F, Septiembre 2006. Xsens Technologies B.V.

“Users Manual Xbus Master B.” Document XM0100P, Revision D, Febrero 2007. Xsens Technologies B.V.

SetObjectAlignement

Propósito

Envía el mensaje SetObjectAlignment a los dispositivos conectados al objeto XBusMaster. El proceso se queda bloqueado hasta recibir la información.

Sintaxis

```
XbusMaster=SetPeriod(XBusMaster, k, matriz)
```

Descripción

Envía el mensaje SetPeriod al objeto XBusMaster. El proceso se queda bloqueado hasta recibir la información

Input parameters:

`XbusMaster`: Objeto con la información del dispositivo.

`K`: Número de dispositivo al que aplicar el cambio, de acuerdo con la numeración que el Xsens asigna automáticamente (de 1 a número de sensores).

`matriz`: Matriz de rotación que se aplicará para reorientar los ejes del dispositivo

Output parameters:

`XbusMaster`: Es el mismo objeto de entrada que puede haber sido modificado durante la llamada.

Referencia:

“MTi and MTx User Manual and Technical Documentation.” Document MT0100P, Revision I, Enero 2007. Xsens Technologies B.V.

“XM-B Technical Documentation.” Document XM0101P, Revision C, Febrero 2007. Xsens Technologies B.V.

“MTi and MTx Low-Level Communication Documentation.” Document MT0101P, Revision F, Septiembre 2006. Xsens Technologies B.V.

“Users Manual Xbus Master B.” Document XM0100P, Revision D, Febrero 2007. Xsens Technologies B.V.

Apéndice 1: Instrucciones para incorporar nuevas funciones a InerSensTB

Formato de los archivos

Los scripts o funciones de la librería deberán tener el formato que se describe a continuación para facilitar su uso y el proceso de generación automática de las ayudas. La siguiente plantilla se puede emplear para este fin:

```
% NOMBREENMAYUSCULAS Short one line description
%
% NOMBREENMAYUSCULAS Exhaustive and long description of functionality of m-file.
%
% Syntax:  [ParaOut] = nameofmfile(ParaIn)
%
% Input parameters:
%    ParaIn    - Vector or matrix containing the initial individuals
%                   may be any number of individuals
%                if PopInit is empty, ...
%    ParaIn2   - (optional) Scalar or vector containing ...
%                ParaIn2(1): InitRand (name of option)
%                            short description of option
%                            0: ...
%                            >0: ...
%                            standard: 0.25
%
% Output parameters:
%    ParaOut   - description
%    ParaOut2  - (optional) description
%
% Examples:
%
% % Short description of example, followed by Matlab code line
% >> matlab code of example
%
% See also: prprintf, findfiles, straddtime

% Author:   Hartmut Pohlheim
% History:  12.11.2000  file created
%                       full description at the top
%           19.11.2000  suggestions for in-code comments added


function [ParaOut] = nameofmfile(ParaIn)

% Check input parameters



% End of function
```

La primera línea es una descripción concisa de lo que hace la función, que es la que aparece en la ayuda en linea de matlab.

El siguiente bloque de texto, separado de la línea anterior por una línea en blanco, es una descripción más detallada de la función, entre 3 o 4 a 50 líneas para las funciones más complejas. Esta descripción es la que aparece al hacer un help de la función. Todo el bloque, incluídas las líneas en blanco, deben ir precedidas por el signo %.

El tercer bloque describe la sintáxis de la función, con todos los posibles parámetros:

```
[FirstParaOut, SecondParaOut] = nameofmfile(FirstParaIn, SecondParaIn, ThirdParaIn)
```

Es muy importante una detallada descripción de los parámetros de entrada y de salida:

- si el parámetro es opcional, escribir (optional) delante de su descripcion

- el tipo de dato esperado: escalar, real, complejo, vector, o cualquier otra combinación.
- si de un parámetro solo son aceptables un conjunto de valores aceptables

A modo de ejemplo:

```
% Input parameters:
%    PopInit   - Vector or matrix containing the initial individuals
%                   may be any number of individuals
%                if PopInit is empty, a uniform at random initialization
%                   of individuals takes place
%    Nind      - Scalar containing the number of individuals to create at all
%    VLUB      - (optional) Matrix containing the boundaries of the variables
%    InitOpt   - (optional) Scalar or vector containing the parameters for initialization
%                InitOpt(1): InitRand
%                            level of randomization of (inoculated) individuals
%                            0: no randomization, no similar individuals are produced
%                           >0: randomize individuals using the following equation
%                               (randn/4 * InitRand * domain of variable)
%                            standard: 0.25
%                InitOpt(2): InitNindKeep
%                            keep preinitialized individuals in population (unchanged)
%                            0: keep none of them
%                            > 0 (max 1): scalar (percentage of population size) how
%                               many individuals from PopInit to copy to Chrom
%                            standard: 0.2 (keep not more than 20% of individuals
%                                      in final population)
%                InitOpt(3): InitNindUniform
%                            create some individuals uniform at random in defined
%                            domain of variables (boundaries VLUB) - uses the
%                            standard init functions initrp, initip, initbp and initpp
%                            0: create none
%                            > 0 (max 1): scalar (percentage of population size) how
%                            standard: 0 (create no individuals uniform at random)
%                               many individuals to create uniform at random
%
% Output parameters:
%    Chrom     - Matrix containing the individuals of the current
%                population. Each row corresponds to one individual's
%                representation.
%    VLUB      - (optional) Matrix containing the (new) boundaries of the
%                variables
```

A continuación viene la sección de ejemplos, con llamadas a la función, empezando por el caso más sencillo e incluyendo algún caso de los más complejos.

La línea “See also:” es muy útil al proporcionar acceso directo a otras funciones relacionadas. Por último, y en un segundo bloque de comentario, viene una nota con el autor y la historia de modificaciones a la función.

Durante la función los comentarios han de hacerse con la extensión suficiente para que se pueda leer el código directamente.

Chequeo de parámetros

La primera operación que se debe realizar ya en la función propiamente dicha es el chequeo de parámetros, comprobando:

- Si todos los parámetros necesarios están presentes (si no, error)
- Los parámetros opcionales o no presentes han de ser puestos a su valor por defecto.
- Si los parámetros están dentro del rango de valores admisibles, y realizar su reajuste si es posible, enviando solo un warning.

A modo de ejemplo:

```
% Set standard initialization parameter
```

```
   InitOptStandard = [0.2, 0.25, 0];
   % InitRand = 0.3 (use normal random inoculation with 30% of domain of variables),
   % InitNindKeep = 0.2 (keep 20% maximal)
   % InitNindUniform = 0 (create no random individuals)

% Check input parameters
   % At least the first 2 input parameters are necessary
   if nargin < 2,  error('Not enough input parameters (at least 2 parameters -
individuals and Nind)'); end
   if isnan(PopInit), PopInit = []; end
   % The 3. input parameter is optional, default is []
   if nargin < 3,  VLUB = []; end

   % the 4. input parameter is optional and can contain multiple options
   if nargin < 4, InitOpt = []; end
   if isnan(InitOpt), InitOpt = []; end
   % When too many options are contained in InitOpt, issue a warning and
   % shorten the parameter vector
   if length(InitOpt) > length(InitOptStandard),
      InitOpt = InitOpt(1:length(InitOptStandard));
      warning(' Too many parameters in InitOpt! InitOpt was shortened.');
   end
```

La documentación generada con estas directrices podrá utilizarse además para la documentación de Referencia que se irá incorporando en este mismo documento, donde ya se pueden añadir gráficos, tablas u otras ayudas o explicaciones más visuales.

Generación automática de los archivos de ayuda

En el directorio `html` residen archivos de ayuda, que son generados de manera automática. Basta correr el script `generaAyudas` desde una terminal de cualquier sistema unix con perl instalado.

Funciones ocultas

Siguiendo los consejos de Mathworks, se dejará dentro del directorio silop/silop/private las funciones de uso exclusivo por otras funciones o por otros desarrolladores y que se supone que ningún usuario querrá llamar. Las funciones que vayan a ser usadas exclusivamente por otra, y que no sean de utilidad tampoco para otros desarrolladores deben ir anidadas dentro del fichero de la función que las llame.

Funciones no usadas

Es recomendable dejar dentro del directorio silop/silop/archive aquellas funciones que hallan sido de utilidad durante algún tiempo, aunque dichas funciones hayan dejado de ser parte necesaria de la toolbox, y no puedas ser usadas sin reincluirlas correctamente en silop/silop/private o silop/silop.

Apéndice 2: Descripción de la orientación en 3D

Quaternions

Cada IMU de los Xsens proporciona 9 medidas: 3 aceleraciones, 3 velocidades angulares y 3 campos magnéticos.

Angulos de Euler

Cada IMU de los Xsens proporciona 9 medidas: 3 aceleraciones, 3 velocidades angulares y 3 campos magnéticos.

Matriz de Rotación

Cada IMU de los Xsens proporciona 9 medidas: 3 aceleraciones, 3 velocidades angulares y 3 campos magnéticos.

Apéndice 3: Instrucciones para la creación de un nuevo driver de dispositivo.

Cada driver estará contenido en un fichero denominado driver_NOMBRE.m, que incorporara la función driver_NOMBRE.

```
function [retorno,senhales]=driver_Nombre(operacion,parametros)
```

Esta función debe estar preparada para recibir dos *parámetros de entrada*:

- Una cadena de caracteres: `operación`, que permite la selección del bloque de código a utilizar (llamando a la función que corresponde en cada caso). Los posibles valores de esta cadena de texto son:: *'create', 'connect', 'configura', 'gotoconfig', 'gotomeasuremtne', 'destruye'*.
- Un campo `parametros`, que incluirá:
  - En el bloque 'create' `parametros` será un vector con los parámetros que defina el driver como necesarios.
  - En el bloque 'configura', `parametros` será un vector con el dispositivo a usar, y la estructura de señales a manejar.
  - En el resto de bloques, `parametros` será directamente un vector con el dispositivo a usar

Los parámetros devueltos son:

a. `retorno`: El manejador del driver que debe ser usado para las siguientes llamadas
b. `senhales`: La estructura de señales, que salvo que se indique lo contrario sera `[]`

Según el valor del parámetro `operación`, el driver debe realizar las siguientes operaciones

'`create`' :

Crear una estructura de datos, en la que almacenar los parámetros de funcionamiento que se le indican.Estos serán `{source, freq, updateeach, ns, driver_opt}`

- El puerto de comunicaciones a traves del cual se accede a los datos parametros{1};
- La frecuencia de adquisición a la que se deben obtener los datos parametros{2}
- La frecuencia a la que se deben actualizar los datos (e indirectamente el tamaño de los buffers) parametros{3}
- El numero de sensores conectados al dispositivo (si admite varios)
- Opciones especificas del driver driver_opt=parametros{5};

Cualquier otro parametro auxiliar que necesite puede también guardarse en esta estructura (p.e. Un objeto puerto serie para la comunicación posterior).

La función debe comprobar que los datos proporcionados tengan sentido, y no creara el objeto en caso contrario.

Debe devolver la estructura así creada, o arrojar un error si no se ha podido crear correctamente.

'`connect`'.

Esta función debe configurar de forma correcta el puerto de comunicaciones que se usará para comunicarse con el dispositivo, y comprobar que puede abrir dicho puerto.

Aún no iniciará las comunicaciones.

La función devolverá la estructura modoficada en caso de que todo vaya correctamente o lanzará un error en caso contrario.

'gotoconfig'

Esta función debe dejar al dispositivo listo para enviarle datos de configuración. Dependiendo del funcionamiento del mismo, puede ser una función vacia. La función no puede asumir que el dispositivo se encuentre en un estado determinado.
La función devolverá la estructura que recibe o lanzará un error en caso de no poder hacer su tarea

'configura'

Esta función configurará al dispositivo de acuerdo a los datos existentes en la estructura del driver, y que habían sido cargados previamente mediante la función create. Para ello realizará las comunicaciones que se correspondan con el dispositivo.
El segundo parámetro recibido por esta función es una estrutura que contiene:

1. Un campo ***NUMEROSENHALES*** , que indica las señales existentes. De mano será 0.
2. Un campo para cada sensor del dispositivo, con el nombre arbitrario que el usuario le haya asignado en addimu. Este campo será una estructura, que contendrá a su vez los siguientes apartados
    1. Un campo **Serie**, con el número de serie y/o etiqueta que permita identificar cada sensor en los sistemas que tengan varios.
    2. Un campo **R** con un vector que indica la orientación del sensor, de acuerdo a las especificacioens de addimu. Si es posible se configurará el dispositivo para que aplique esta rotación por hardware. En caso contrario se realizará por software durante la captura de datos.

La función debe detectar las señales que genera el dispositvo, calcular en que columna de la matriz de adquisición de datos quedará dicha señal e incluir en la estructura de señales, dentro del sensor correspondiente, un campo con el nombre de cada señal, conteniendo el número de dicha columna.
Ej: senhales.sensor_en_el_COG.AceleracionVertical=37; Si fuese la columna 37. Asimismo presentará un mensaje en pantalla indicando la señal añadida.
Finalmente se modificará el campo ***NUMEROSENHALES,*** para indicar cual es la última señal añadida.
En caso de que todo vaya bien se devolverá la estructura modificada. En caso contrario se arrojará un error.

'gotomeasurement'

Esta función pasa a modo medida un dispositivo que esté correctamente configurado. Asimismo debe configurar una función de callback para que se llame cada vez que los datos deban estar disponibles de acuerdo a la configuración especificada con create.
Si todo va bien la función devolverá la estructura del objeto, en caso contrario lanzará un error.
La función de callback, que debe estar incluida dentro del propio driver debe:

- Leer los datos del dispositivo
- Aplicarles la rotación apropiada (si no se ha reallizado por hardware)
- Rellenar la variable global SILOP_DATA_BUFFER de forma que en cada columna se encuentre los datos previamente indicados.
- Es conveniente que muestra por línea de comandos el número de datos leído, a modo de indicador de avance.

'destruye'

Esta función debe destruir toda la infraestructura de comunicaciones de la manera más limpia posible. Para ello, si es posible, intentará detener en primer lugar el hardware, a continuación cerrar las comunicaciones con el, y finalmente borrar los datos de la estructura del driver. Devolverá un array vacio, representando a la estructura destruida.

Ejemplo de driver:

```
function [retorno,senhales]=driver_SF_3D(operacion,parametros)
    senhales=[];
    switch operacion
        case 'create'
            retorno=creasf3d(parametros);
        case 'connect'
            retorno=connectsf3d(parametros);
        case 'configura'
            [retorno,senhales]=configurasf3d(parametros);
        case 'gotoconfig'
            retorno=sf3dgotoconfig(parametros);
        case 'gotomeasurement'
            retorno=sf3dgotomeasurement(parametros);
        case 'destruye'
            retorno=[];
            destruyesf3d(parametros);
        otherwise
            disp('error, el driver no soporta la operaciÃ³n indicada');
            retorno=[];
    end
end

function sf3d=creasf3d(parametros)
    source=parametros{1};
    freq=parametros{2};
    updateeach=parametros{3};
    ns=parametros{4};
    driver_opt=parametros{5};
    if (length(driver_opt)<1)
        bps=9600;
    else
         bps=driver_opt(1);
    end
    if (length(driver_opt)<2)
        modo=0;
    else
        modo=driver_opt(2);
    end
    % Calculamos el numero de muestras almacenadas en el buffer
    sf3d.freq=freq;
    sf3d.buffer=updateeach*freq;
    if (ns>1)
        error('Los sparkfun 3D solo tienen un sensor, demasiados IMUS')
    end
    try
        sf3d.puerto=serial(source);
    catch ME
        disp ('Imposible conectarse al puerto serie');
        rethrow (ME);
    end
    sf3d.modo=modo;
    switch (sf3d.modo)
        case 0,
            sf3d.Data=3;
            sf3d.DataLength=29;%bytes de cada mensaje
        otherwise,
            error('Modo invalido');
    end;
    sf3d.bps=bps;
end

function sf3d=connectsf3d(parametros)
    sf3d=parametros;
    % Configurar el objeto serie
    sf3d.puerto.BaudRate=sf3d.bps;
    sf3d.puerto.DataBits=8;
    sf3d.puerto.FlowControl='none';
```

```
    sf3d.puerto.Parity='none';
    sf3d.puerto.StopBits=1;
    sf3d.puerto.ReadAsyncMode = 'continuous';
    sf3d.puerto.ByteOrder = 'littleEndian';
    sf3d.puerto.BytesAvailableFcnCount = sf3d.DataLength*sf3d.buffer;
    sf3d.puerto.BytesAvailableFcnMode = 'byte';
    sf3d.puerto.InputBufferSize = 2*sf3d.DataLength*sf3d.buffer;
    sf3d.puerto.OutputBufferSize = 512;
    sf3d.puerto.Tag = 'SparkFun_3D';
    sf3d.puerto.Timeout = 10;
    fopen(sf3d.puerto);
end

function [sf3d,senhales]=configurasf3d(parametros)
    sf3d=parametros{1};
    senhales=parametros{2};
    sf3d=sf3dsetperiod(sf3d);
    numero=2;
    disp('Los sparkfun 3d no soportan reorientacion por hardware');
    disp('se ignora la orientacion especificada mediante addimu');
    posiciones=fieldnames(senhales);
    senhales.(posiciones{numero}).Acc_Z = 3;
    disp(['Anadida senhal ',posiciones{sensor},'.Acc_Z']);
    senhales.(posiciones{numero}).Acc_Y = 2;
    disp(['Anadida senhal ',posiciones{sensor},'.Acc_Y']);
    senhales.(posiciones{numero}).Acc_X = 1;
    disp(['Anadida senhal ',posiciones{sensor},'.Acc_X']);
    senhales.NUMEROSENHALES=3;
end

function sf3d=destruyesf3d(sf3d)
    try
        fclose(sf3d.puerto);
    catch ME
    end
    delete(sf3d.puerto);
    clear sf3d
    sf3d=[];
end

function sf3d=sf3dgotoconfig(sf3d)
    %Limpiamos todo lo que puede quedar en el buffer de medidas anteriores
    sf3d.puerto.Timeout=1;
    while (sf3d.puerto.BytesAvailable>0)
        % Vaciar el puerto
        disp(['>>> Se descartaran ' int2str(sf3d.puerto.BytesAvailable) ' datos']);
        fread(sf3d.puerto, sf3d.puerto.BytesAvailable,'uint8');
    end
    % Envia el mensaje de paso a config. Codigo hexadecimal 13
    % Cuerpo del mensaje
    msg=[hex2dec('13')]; %#ok<NBRAK>
    % Se envia por el puerto serie
    fwrite(sf3d.puerto,msg,'uint8','async');
    pause(1);
    %Ya deberiamos estar en modo config.
    %comprobamos la respuesta.
    [ack,cnt,msg]=fread(sf3d.puerto, sf3d.puerto.BytesAvailable, 'uint8');
    if (~isempty(msg))
        error('no se ha recibido la respuesta al comando gotoconfig');
    else
        if (sum(ack(end-11:end))~=859)
            error('Error de checksum durante gotoconfig');
        end
    end
end

%Funcion para el paso a modo medida
function sf3d=sf3dgotomeasurement(sf3d)
    global SILOP_DATA_BUFFER;
    SILOP_DATA_BUFFER=[];
    % Cuerpo del mensaje (excepto el byte de checksum)
    msg='x';
    % Se envia por el puerto serie
    if (sf3d.puerto.BytesAvailable>0)
        disp(['>>> Se descartaran ' int2str(sf3d.puerto.BytesAvailable) ' datos']);
```

```
        fread(sf3d.puerto,sf3d.puerto.BytesAvailable,'uint8');
    end
    % El valor del TimeOut se fija a 2 segundos.
    sf3d.puerto.Timeout=2;
    fwrite(sf3d.puerto,msg,'uint8','async');
    leersf3dDatahandle=@leersf3dData;
    sf3d.puerto.BytesAvailableFcn={leersf3dDatahandle, sf3d};
end

%Funcion para laeer los datos del puerto serie. Llamada por una callback
function leersf3dData(obj,event,sf3d)  %#ok<INUSL>
    global SILOP_DATA_BUFFER
    [data,cnt,msg]=fread(obj,[sf3d.DataLength sf3d.buffer],'uint8');
    if (~isempty(msg))
        disp(msg);
        error('error en la lectura de datos');
    end
    % Procesar los datos de 1 mensaje
    if (any(data(2,:)-'X'))
        disp('>>>> ERROR durante la captura de datos');
    end
    if (any(data(11,:)-'Y'))
        disp('>>>> ERROR durante la captura de datos');
    end
    if (any(data(20,:)-'Z'))
        disp('>>>> ERROR durante la captura de datos');
    end
    % procesar la informacion
    for k=1:sf3d.buffer
        ax=9.8*str2double(char(data(  4:  9,k)));
        ay=9.8*str2double(char(data(13:18,k)));
        az=9.8*str2double(char(data(22:27,k)));
        SILOP_DATA_BUFFER=[SILOP_DATA_BUFFER; ax ay az]; %#ok<AGROW>
    end
    disp('leido un segundo de datos');
end

function sf3d=sf3dsetperiod(sf3d)
    freq=sf3d.freq;
    if (sf3d.puerto.BytesAvailable>0)
        disp(['>>> Se descartaran ' int2str(sf3d.puerto.BytesAvailable) ' datos']);
        fread(sf3d.puerto, sf3d.puerto.BytesAvailable,'uint8');
    end
    % Cuerpo del mensaje
    msg='3';
    fwrite(sf3d.puerto,msg,'uint8','async');
    % Se espera a recibir la contestacion
    [ack,cnt,msg]=fread(sf3d.puerto,78,'uint8');
    if (~isempty(msg))
        disp(msg);
        error('no se ha recibido respuesta al comando setperiod');
    end
    if (sf3d.puerto.BytesAvailable>0)
        fread(sf3d.puerto, sf3d.puerto.BytesAvailable,'uint8');
    end
    ack=ack(end-10:end-1);%Me quedo con los 10 penultimos,    if ack(10)=='H'
        %solo dos cifras
        frecuencia=str2double(char(ack(8:9)));
    else
        %tres cifras
        frecuencia=str2double(char(ack(8:10)));
    end
    while (frecuencia~=freq)
        if (frecuencia>freq)
            msg='-';
        else
            msg='+';
        end
        fwrite(sf3d.puerto,msg,'uint8','async');
        % Se espera a recibir la contestacion
        [ack,cnt,msg]=fread(sf3d.puerto,10,'uint8');
        if (~isempty(msg))
            disp(msg);
            error('no se ha recibido respuesta al cambiar la frecuencia de muestreo');
        end
```

```
        if (sf3d.puerto.BytesAvailable>0)
            fread(sf3d.puerto, sf3d.puerto.BytesAvailable,'uint8');
        end
        if ack(8)=='H'
            %solo dos cifras
            frecuencia=str2double(char(ack(6:7)));
        else
            %tres cifras
            frecuencia=str2double(char(ack(6:8)));
        end
    end
    %Salimos del modo de frecuencia
    msg='x';
    fwrite(sf3d.puerto,msg,'uint8','async');
    [ack,cnt,msg]=fread(sf3d.puerto, 196,'uint8');
    if (~isempty(msg))
        disp(msg);
        error('no se ha recibido respuesta al salir del ajuste de frecuencia');
    end
    if (sf3d.puerto.BytesAvailable>0)
        fread(sf3d.puerto, sf3d.puerto.BytesAvailable,'uint8');
    end
end
```

Drivers estandar

Los siguientes drivers están incluidos de forma estandar en la toolbox:

Driver	Especificaciones
SF_3D	Driver para el manejo de los sparkfun serial 3D accelerometers Limitaciones:No realizan la reorientación de acuerdo a addimu driver_opt: `bps` bits por segundo de la comunicación serie
Xbus	Driver para el manejo de los sensores de Xsens mediante un Xbus Master driver_opt: [`bps,modo`] bits por segundo de la comunicación serie y modo modo=0. Datos calibrados modo=1. Datos calibrados y orientaciones. modo=2. Datos calibrados y orientados (euler) Limitaciones. Independientemente del modo sólo se leen aceleraciones, velocidades de giro y vectores magnéticos.
MTiG	Driver para el manejo de los sensores de MtiG de Xsens driver_opt: [`bps,modo`] bits por segundo de la comunicación serie y modo modo=0. Datos calibrados modo=1. Datos calibrados y orientaciones. modo=2. Datos calibrados y orientados (euler) modo=3. Datos calibrados y matriz de orientacion modo=4: Posición(LLA) + Velocidad + Status (del GPS) modo=5: Datos calibrados+ Posición(LLA) + Velocidad + Status (del GPS)

	modo=6: Datos RAW Inertial+GPS
Temporizador	Driver para el manejo de ficheros .sl de la toolbox, .log de Xsens, y .tana previamente capturados