Facultad de Ingeniería

Escuela de Ingeniería Civil Biomédica

Desarrollo de una plataforma de captura de movimiento ambulatorio por medio de sensores inerciales

Por

Sebastián Andrés Sepúlveda San Martín

Trabajo de título para optar al Título de Ingeniero Civil Biomédico

Prof. Guía: Dr. Pablo Reyes

Prof. Co-Guía: Dr. Alejandro Weinstein

Octubre, 2014

Agradecimientos

A mi familia, que a pesar de la distancia; su apoyo se hace presente cada día.

A mis compañeros y amigos, que me acompañaron durante los años de universidad

A mis profesores, en especial, a los Dr. Pablo Reyes y Dr. Alejandro Weinstein, por compartir sus conocimientos, experiencia y fortalecer mi formación.

A todos ellos, muchas gracias por compartir su tiempo conmigo.

Resumen

Palabras claves: Ambulatorio, Biomecánica, Sensores Inerciales.

Resumen: *Los sensores inerciales se han convertido en una alternativa interesante para el estudio del movimiento humano. El avance tecnológico de los Microelectromechanical Systems (MEMS) ha permitido obtener mejores características técnicas y mayor accesibilidad. Usando esta tecnología, los Inertial Measurement Unit (IMU) son dispositivos pequeños que integran sensores inerciales como acelerómetros y giroscopios. Mediante la fusión de sus variables, permiten estimar la postura del IMU en el espacio, respecto a un marco de referencia propio o utilizando el marco terrestre. En la actualidad, el estudio del movimiento por métodos ópticos está confinado a entornos de laboratorio, donde se realizan capturas del movimiento utilizando imagen y vídeo, que junto a marcadores ubicados en el cuerpo permiten medir, dentro del alcance visual del sistema, los movimientos efectuados por el sujeto de estudio. Utilizando sensores inerciales, es posible seguir al sujeto fuera del laboratorio y realizar la captura del movimiento en el hogar del paciente, mientras el deportista entrena en un ambiente específico o en el lugar que el sujeto de estudio trabaja.*

En este trabajo se propone el desarrollo de un prototipo para una plataforma de captura de movimiento ambulatorio por medio de sensores inerciales, estableciendo requerimientos específicos para crear una plataforma realmente ambulatoria, como portabilidad, usabilidad y acceso a controlarla desde prácticamente cualquier dispositivo capaz de conectarse a una red WiFi, como un *smartphone, tablet o un computador.*

La metodología de desarrollo propuesta permite segmentar la problemática de captura del movimiento, acotándolo a unidades modulares de desarrollo. Adicionalmente, se detallan las problemáticas encontradas con las diferentes plataformas de desarrollo utilizadas y el uso de las tecnologías de comunicación.

Los resultados muestran que el diseño propuesto cumple con los objetivos y requisitos propuestos, asegurando modularidad tanto en hardware, software y arquitectura. El dispositivo fue probado para adquirir datos en un estudio de biomecánica en Colombia, logrando con éxito el registro y control de la plataforma con los medios propuestos.

Tabla de Contenidos

1. Introducción **1**
1.1. Objetivo General 2
1.2. Objetivo específico 2

2. Análisis de la problemática **3**
2.1. Estado del arte 5
2.1.1. Respecto a los algoritmos de fusión 5
2.1.2. Respecto a la utilización EMG con IMU 6

3. Desarrollo de la propuesta **6**
3.1. Marco teórico 6
3.2. Diseño de la propuesta 17
3.2.1. Metodología 17
3.2.2. Requisitos 18
3.3. Implementación 18
3.3.1. Selección de los sensores 18
3.3.2. Selección de protocolo de comunicación 20
3.3.3. Selección de la plataforma de desarrollo 21
3.3.4. Hardware del dispositivo 23
3.3.5. Firmware del dispositivo 24
3.3.6. Software del dispositivo 25

4. Resultados **30**

5. Discusión **32**

6. Conclusiones **32**

7. Referencias Bibliográficas **34**

8. Glosario **39**

9. Anexo **40**
9.1. Esquematico de multiplexor I2C PAC9547 40
9.2. Planos de enclosure sensor LSM9DS0 41
9.3. Documentación IMU Graph 43
9.4. Documantación IMU Monitor 86

Desarrollo de una Plataforma de Captura de Movimiento Ambulatorio por medio de Sensores Inerciales

Sebastián Andrés Sepúlveda San Martín
Escuela de Ingeniería Civil Biomédica, Universidad de Valparaíso, Chile

Palabras claves: *Ambulatorio, Biomecánica, Captura del movimiento, Sensores Inerciales.*

1. Introducción

El movimiento es la distorsión del estado natural de un objeto. Es una característica que pueden controlar los seres vivos superiores, y su correcto dominio les puede permitir realizar complejas secuencias de movimientos que resultan en actividades como desplazarse a través de su entorno para sobrevivir; y a diferencia de la gran mayoría de los animales, el humano logró dominar el desplazarse erguido en dos piernas.

Las secuencias de movimientos pueden llegar a ser extremadamente complejas: para un ser humano, impulsos nerviosos originados voluntariamente en el cerebro orquestan la contracción de los músculos que accionan sobre la unidad estructural; el esqueleto, para llevar a cabo el movimiento en forma voluntaria. El cuerpo humano posee en su esqueleto huesos y articulaciones, definiendo las posibilidades de movimiento, o grados de libertad de cada segmento o extremidad del cuerpo.

El movimiento además nos entrega información. Notamos visualmente que el caminar de una persona es anormal. La comparación de condiciones normales y anormales para movimientos "comunes" surge entonces como medición y diagnóstico. Más aún, existen condiciones extraordinarias, como por ejemplo, analizar cómo un deportista de alto rendimiento puede realizar movimientos que un ser humano normal no puede realizar, tanto en sus dimensiones como en los tiempos, extensiones o fuerza del movimiento.

Existe un gran interés -y más aún, una fascinación (Roetenberg, 2006)- por entender, cuantificar y modelar el movimiento. Para esto, capturarlo y registrarlo es el primer paso a seguir al momento de querer analizar el movimiento. El método ampliamente aceptado por la comunidad científica y clínica para la captura del movimiento es a través de MOCAP (*Motion Capture*) por medios ópticos. Este tipo de sistemas es considerado el *Gold Standard* del MOCAP.

La captura de movimiento por medios ópticos consiste en adosar marcadores superficiales a los segmentos del cuerpo de interés para la captura. Al marcar en forma apropiada un segmento en la superficie de la piel, es posible estimar el movimiento que realiza el esqueleto del sujeto de estudio. Estos marcadores son registrados mediante cámaras posicionadas de tal forma que sea posible capturar los marcadores desde al menos 3 posiciones conocidas, generando un espacio de captura delimitado por las capacidades y características de las cámaras para registrar los marcadores en imágenes o videos. Los registros son posteriormente procesados para dejar solamente los marcadores, y por medio de trilateración es posible determinar la posición en el tiempo de un marcador. Posteriormente, se unen los marcadores para determinar a qué segmento del cuerpo pertenece el marcador registrado.

Para asegurar las mejores condiciones de captura del movimiento, es común instalar laboratorios de biomecánica especializados en el tipo de capturas que se desea realizar. La cantidad y tipo de cámaras, el tipo marcadores, el espacio de captura y las condiciones del ambiente (principalmente condiciones lumínicas y nivel de suelo) son controlados para adquirir la captura del movimiento con una precisión menor al milímetro (Field, Stirling, Naghdy, & Pan, 2009). Sin embargo, al establecer un área delimitada de adquisición, como un laboratorio, surge el principal problema de los métodos de adquisición por imágenes y video: Los movimientos están confinados al área de captura y a las condiciones de laboratorio (Mündermann, Corazza, & Andriacchi, 2006).

Analizar el movimiento humano permite diagnosticar enfermedades, seguir la recuperación ante terapias y operaciones (Zheng, Black, & Harris, 2005), controlar el gasto energético ante diferentes actividades (Bouarfa et al., 2014; Dong, Biswas, Montoye, & Pfeiffer, 2013; Lustrek, Cvetkovic, & Kozina, 2012), llegando a clasificar el tipo de actividades que realizan (Pham, Tran, Thang, Cooper, & Kamei, 2011). Estas inquietudes han llevado la captura de movimiento fuera del laboratorio, para medir en el entorno natural y monitorear ambulatoriamente los movimientos. Esto se ve ampliamente aplicado en estudios sujetos al deporte (Haapalainen, Laurinen, Roning, & Kinnunen, 2008; Lee & Shiroma, 2014; Panagiota, Layal, & Stefan, 2012; Wixted et al., 2007) y en la detección de caídas, tanto para prevenirlas como para detectarlas (Aziz, Park, Mori, & Robinovitch, 2014; Bagalà, Klenk, Chiari, Cappello, & Becker, 2012; Bourke et al., 2010; Bourke & Lyons, 2008; Hausdorff, Rios, & Edelberg, 2001; Tsai, 2014), sobre todo en adultos mayores, donde el mayor peligro se presenta en sus hogares y sus actividades cotidianas.

En estos casos, el sistema óptico de MOCAP no se puede utilizar, ya que necesita una referencia fija para realizar las mediciones, donde los elementos naturales del ambiente, como los muebles de la casa, las paredes o incluso los instrumentos que utiliza el deportista pueden generar puntos ciegos en el sistema de referencia. Por tanto, los sistemas ópticos de MOCAP no son apropiados para mediciones ambulatorias del movimiento. La necesidad de contar con sistemas ambulatorios de captura del movimiento ha llevado a utilizar sensores inerciales miniaturizados para lograr este objetivo. Este método logra solucionar la problemática principal, sin embargo, introduce otros problemas característicos de los sensores inerciales, los que deben ser corregidos en su funcionamiento en el tiempo para alcanzar niveles de precisión comparables.

En este trabajo, se propone desarrollar una plataforma de captura de movimiento ambulatoria mediante sensores inerciales, desarrollado con tecnología accesible, *open source* y modular, con el objetivo de que sea posible ajustar la plataforma a las necesidades individuales de la aplicación. Adicionalmente, se propone una solución para controlar la plataforma desde cualquier tipo de dispositivo en forma inalámbrica, pudiendo permitir, incluso, controlarse remotamente a través de internet.

1.1. Objetivo General

Desarrollar un prototipo de la plataforma de captura de movimiento ambulatoria por medio de sensores inerciales.

1.2. Objetivo específico

(1) Determinar la arquitectura apropiada para desarrollar la plataforma
(2) Obtener los datos en tiempo real, registrarlos y evaluar el desempeño del procesamiento.
(3) Implementar una interfaz independiente del sistema operativo o dispositivo para controlar la plataforma
(4) Diseñar el dispositivo en forma modular, permitiendo portabilidad a diferentes arquitecturas y diferentes cantidades de sensores

A continuación, se analiza la problemática, se propone la solución y se detallan las metodologías utilizadas para la implementación del dispositivo. Finalmente, se muestran los resultados obtenidos con el prototipo de la plataforma en pruebas de biomecánica respecto a la utilización en conjunto contra el *Gold Standard*.

Este trabajo se realiza en el contexto del proyecto de tesis del programa de Magister en Ciencias de la Ingeniería, mención en Ingeniería Biomédica de la escuela de Ingeniería Biomédica de la Universidad de Valparaíso. El desarrollo de esta plataforma es parte fundamental del proyecto, que permitirá registrar el movimiento y la actividad de electromuscular en los músculos (EMG) en conjunto con el movimiento, para evaluar y validar mejoras en el procesamiento de posición y postura mediante sensores inerciales.

2. Análisis de la problemática

Los sistemas de captura de movimiento a través de medios ópticos son considerados el *Gold Standard* para las adquisiciones de MOCAP, debido a la gran precisión que se puede lograr con ellos. Sin embargo, presentan una problemática principal: los sistemas ópticos de MOCAP necesitan un sistema de referencia fijo. Por este motivo, la gran mayoría de ellos se instala en entornos controlados y laboratorios especializados.

Esta problemática se vuelve limitante al momento de tener la necesidad de realizar mediciones ambulatorias del movimiento. Por mediciones o capturas ambulatorias del movimiento se entiende un sistema que permita monitorear al sujeto de pruebas mientras el realiza su vida cotidiana, tal como un *Holter* (electrocardiograma portable) permite registrar las señales del corazón durante el día. Respecto a un sistema de captura ambulatorio cualquiera, los sistemas de captura ópticos del movimiento presentan problemáticas difíciles de resolver al momento de adaptarlos a ambulatorios:

(1) Un sistema ambulatorio debe basarse en un sistema de referencia fijo y local.
(2) El almacenamiento de imágenes o video en el tiempo implica grandes cantidades de datos por almacenar.
(3) El espacio delimitado de adquisición impide capturar movimientos de amplio rango, principalmente en aplicaciones deportivas, o con más de una persona.
(4) Existen condiciones difíciles ambientales que no se pueden reproducir o emular en un laboratorio, como por ejemplo ejercicios acuáticos o dependientes del contexto donde se realizan, como el trabajo de un minero en excavaciones profundas.
(5) El consumo energético de un sistema óptico lo hace inviable para operar con baterías durante tiempos prolongados de adquisición.

El monitoreo ambulatorio se ha vuelto una necesidad durante los últimos años. Estudios para estimar los cambios de la marcha en enfermedades del Parkinson, detección y monitoreo de caídas en adultos mayores o el gasto energético requieren un sistema ambulatorio, y además, *wearable*.

Los dispositivos *wearables* (Anliker et al., 2004) son aquellos que se pueden integrar en conjunto con la ropa o como accesorios, dando comodidad al usuario del dispositivo, quien no se da cuenta que está utilizando el dispositivo. Un dispositivo *wearable* es apropiado para mediciones ambulatorias en aplicaciones Biomecánicas (Huang & Hsu, 2005), y en la actualidad se comercializan masivamente como accesorios de *smartphones* y *smartwatches*, permitiendo registrar variables como ritmo cardiaco, pulso, contadores de pasos y calorías quemadas, utilizando sensores miniaturizados y de bajo consumo energético.

Gran parte del progreso en las tecnologías *wearable* han ido de la mano con las tecnologías de miniaturización. En este ámbito, una de las principales tecnologías de miniaturización que ha permitido el desarrollo de sensores cada vez más pequeños, menor consumo energético y menor costo de producción es la tecnología MEMS (*Microelectromechanical Systems*).

La tecnología MEMS ha permitido miniaturizar sensores inerciales, dispositivos que miden las fuerzas o magnitudes que los afectan. He aquí, un salto importante tanto para la biomecánica y la captura del movimiento con fines ambulatorios: los sensores inerciales no necesitan un sistema de referencia fijo para referenciarse en un entorno local, puede utilizar marcos de referencia globales como la gravedad de la tierra y la orientación respecto al norte magnético. Sensores inerciales ya habían sido utilizados en estudios biomecánicos para obtener variables como aceleraciones de segmentos del cuerpo, detección de caídas, gasto energético o postura de diferentes segmentos del cuerpo. Por lo tanto, la captura de movimiento por métodos inerciales resulta beneficiosa para su utilización como mediciones ambulatorias, además, son parte elemental de los nuevos dispositivos s*mart*.

Los métodos inerciales de captura de movimiento utilizan IMU (*Inertial Measurement Unit*) de la misma forma que los marcadores en los métodos ópticos, y se realizan mediciones que permiten determinar la orientación o postura del segmento del cuerpo bajo estudio. Conociendo la ubicación en el cuerpo, cada sensor entrega la información directamente del segmento asociado, por tanto no se requiere reconstrucción de los marcadores.

Los IMU permiten medir directamente magnitudes físicas que luego se puede expresar como postura o posición; mediamente métodos de fusión de sensores. Los sensores se pueden miniaturizar hasta tener dispositivos comerciales de 4mm x 4mm x 1mm; siendo más pequeños que los marcadores ópticos. La cantidad de datos generados se puede procesar digitalmente a altas velocidades, ya que se almacenan como datos (texto) en contra de los métodos ópticos que necesitan conservar la imagen o video.

Diferentes soluciones comerciales utilizan captura de movimiento por IMU, entre ellos, los fabricados por XSens (Xsens, 2014), donde se proveen unidades inalámbricas para realizar mediciones, junto con trajes especiales para adosar los IMUs. Otras aproximaciones utilizan soluciones *wearables*, donde el sensor se ubica dentro del traje.

Los sistemas de captura por IMU solucionan el problema de adquisiciones ambulatorias y entregan más facilidad en el manejo de datos, sin embargo, existen problemas específicos de los sensores inerciales, entre ellos:

(1) Los sensores inerciales deben fusionarse apropiadamente para obtener postura y/o posición. Existen diferentes métodos para lograrlo, pero la gran mayoría tiende a aumentar su error en el tiempo.
(2) Los trajes especiales diseñados por algunos fabricantes restringen el tipo de personas que pueden utilizar el sistema.
(3) Las soluciones de mercado no son modulares: requieren de software especializado que funciona en conjunto con el hardware entregado.
(4) Los algoritmos de fusión y calibración son generalmente propietarios.

La comunidad *open source* ha desarrollado diversos proyectos para utilizar y construir IMU (Barnett, 2014; S. Madgwick, 2013; Varesano, 2011). Librerías para plataformas de desarrollo tan populares como Arduino o Raspberry Pi (RPi) están disponibles para diferentes sensores IMU, siendo alcanzables para todo *hobbista* interesado en medir el movimiento tanto en humanos como en robots. Sin embargo, la mayoría de estos esfuerzos quedan sin validación.

En este trabajo se propone desarrollar un sistema de captura de movimiento por métodos inerciales utilizando sensores y plataformas *open source* de fácil acceso. El sistema debe ser modular; sin impedir la mejora o agregar más sensores junto con establecer un sistema de control y acceso a los datos de forma ambulatoria desde cualquier dispositivo y plataforma. Posteriormente, el dispositivo implementado se comparará contra un *Gold Standard* para determinar las

condiciones iniciales del sistema, y establecer de qué forma sería posible mejorar los resultados de la fusión.

Basado en la propuesta del proyecto de tesis de Magister en Ciencias de la Ingeniería, mención Ingeniería Civil Biomédica, se propone como hipótesis la utilización y validación de un sistema de este tipo en conjunto con la adquisición de electromiografía (EMG).

En este trabajo, se desarrollara la plataforma de adquisición, seleccionando la arquitectura, tanto de hardware como de software, más apropiada a los requerimientos del dispositivo, junto con realizar pruebas de funcionamiento en terreno para corroborar su usabilidad y robustez.

En forma paralela, en la tesis de magister, se realizara la investigación y validación de la hipótesis, con el fin de determinar en qué forma pueden las mediciones EMG mejorar los resultados de posición y postura.

2.1. Estado del arte

El estado del arte describe las investigaciones que se han desarrollado en los diferentes ámbitos de la captura de movimiento, el uso y comparación de métodos ópticos y sensores inerciales y las mejoras aplicables al incluir EMG e IMU.

2.1.1. Respecto a los algoritmos de fusión

Los algoritmos de fusión para sensores inerciales se han desarrollado desde principios del año 1956, durante plena guerra fría. El objetivo principal de la fusión de sensores es obtener la orientación del dispositivo respecto al marco de referencia del sensor o respecto al marco terrestre, y a partir de esto, es posible determinar posiciones y posturas de objetos. El filtro Kalman, desarrollado en aquellos años, fue utilizado en la teoría de control, llegando hasta campos como la electrónica de consumo, el ámbito militar y la salud. En los años 60, el filtro Kalman fue utilizado para navegación en el Proyecto Apollo, estimando trayectorias de las naves espaciales que viajaban a la luna (Grewal & Andrews, 2010). En 1969, el modulo lunar del Apollo 11 utilizo un filtro Kalman, con un computador de 2MHz, 2K de memoria y un sistema de correcciones para los sensores inerciales, logro llevar por primera vez al hombre a la luna.

Desde entonces, el filtro Kalman ha sido el preferido para realizar la fusión de sensores inerciales, y muchos esfuerzos se han realizado para mejorar sus resultados y disminuir el la carga computacional requerida. En el año 2002 se publicó uno de los primeros métodos de fusión utilizando un filtro Kalman que entregaba sus resultados en cuaterniones (Gao, Niu, & Guo, 2002). Diferentes modificaciones al filtro Kalman han permitido mejorar los resultados e integrar más sensores y correcciones a sus errores, incluyendo GPS en un filtro Kalman Scaled Unscented (Khoder, Fassinut-Mombot, & Benjelloun, 2008a, 2008b); alcanzando un error RMS en la estimación de hasta 2,5º , filtros Kalman de doble etapa (Suh, 2010) y la aplicación del filtro Kalman extendido (EKF) (Sabatini, 2006) que ha logrado un error de estimación RMS de hasta 2,06º, que también se ha implementado en dos etapas (Xie, Yang, Yang, & Li, 2012). Aplicaciones para otros filtros se pueden encontrar en redes de sensores para estudiar la locomoción humana (Misgeld, Ruschen, Kim, & Leonhardt, 2013); logrando un error RMS de hasta 5,4º y el seguimiento de articulaciones humanas (To & Mahfouz, 2012) alcanzando un error RMS de hasta 4,09° en extremidades superiores y hasta 4,51º en extremidades inferiores.

Otros algoritmos proponen el uso de filtros DCM (*Direction Cosine Matrix*) (Phuong, Kang, Suh, & Ro, 2009), y mejoras basadas tanto en Kalman como en filtros complementarios, que permiten la integración de los algoritmos de fusión en dispositivos embebidos de bajos recursos, usando un algoritmo de gradientes descendientes (S. O. H. Madgwick, Harrison, & Vaidyanathan, 2011). Este algoritmo ha logrado errores RMS de hasta 1,110° con respecto al uso de un filtro Kalman clásico, que logra un error RMS de hasta 1,334°.

Este último algoritmo se ha hecho muy popular entre las comunidades *open source* (Varesano, 2011) por la facilidad de implementación, bajo costo computacional y precisión comparable a un filtro Kalman. Adicionalmente, se ha trabajado en la inclusión de barómetros a los algoritmos de fusión (Romanovas et al., 2012) para determinar la altura respecto al nivel del mar, como también *footswitches* que en conjunto con algoritmos de *dead-reckoning* permiten estimar el movimiento (Goyal, Ribeiro, Saran, & Kumar, 2011; Yi-fu, Bo, & Jun, 2013). La utilización de GPS (*Global Positioning System*) en conjunto con IMU y otros sensores como *encoders* para la estimación de la pose en navegación de vehículos autónomos (Weinstein & Moore, 2010).

2.1.2. Respecto a la utilización EMG con IMU

Respecto a mediciones simultaneas del movimiento y EMG, muchas aplicaciones combinan el uso de IMU y mediciones de EMG para realizar HCI (*Human Computer Interface*)(Wolf, Assad, Stoica, et al., 2013; Wolf, Assad, Vernacchia, Fromm, & Jethani, 2013; Xiong, Chen, Zhao, Han, & Liu, 2011), reconociendo gestos en la mano para crear interfaces de mouse en computadores. Pocas investigaciones o estudios se han documentado para incluir el EMG como una medición del movimiento en conjunto a IMUs, sin embargo se han manifestado intenciones de la posibilidad que esta inclusión implicaría (Song, Wang, & Guo, 2013a, 2013b).

El EMG por si solo se ha utilizado en análisis de movimiento para extraer movimientos voluntarios de flexión (Ando, Watanabe, & Fujie, 2009). La estimaciones continuas del movimiento de una articulación de las extremidades superiores se han realizado en conjunto a un IMU (Ding, Xiong, Zhao, & Han, 2011; Park, Kawahara, & Uchida, 2008), pero con el objetivo inverso al propuesto por este trabajo. En la gran mayoría de los casos, se utilizan varios electrodos de EMG y redes de clasificación entrenadas para clasificar movimientos (Geng, Yu, You, & Li, 2010), y en específico, se han logrado algoritmos para la extracción de características en el movimiento de la mano (Phinyomark, Hirunviriya, Limsakul, & Phukpattaranont, 2010; Shin, Jeong, Shin, Cho, & Chung, 2013; Xiong, Lin, Zhao, Han, & Liu, 2012). Sobre otras extremidades inferiores se han realizado estudios que mezclan acelerómetros en comparación a la adquisición de EMG de los pies durante el sueño (Terrill et al., 2013).

3. Desarrollo de la propuesta

En el contexto del desarrollo de la propuesta, se propone una metodología de desarrollo de hardware y software para lograr cumplir con el objetivo de diseñar un prototipo para capturar el movimiento en forma ambulatoria utilizando sensores inerciales. Para esto, el marco teórico define alguno de los términos y, principalmente, tecnologías utilizadas para el desarrollo del prototipo. Posteriormente, el diseño de la propuesta muestra en específico la metodología y los requisitos del sistema para alcanzar los objetivos, para finalmente detallar la implementación agrupada temáticamente.

3.1. Marco teórico

(1) Postura: describe la orientación de un objeto respecto a un marco de referencia determinado. En el contexto de este trabajo, se utilizará postura u orientación indistintamente para describir el movimiento angular de un objeto; es decir, todas las posibles rotaciones (sin desplazarse) que pueda realizar un objeto en un punto fijo. La postura, en términos generales, se mide en ángulos respecto a un punto de referencia. Existen denominaciones establecidas en diferentes ámbitos para los movimientos angulares, entre ellos; robótica, aeronáutica, sistemas de coordenadas esféricas e incluso astronomía. Las representaciones de postura más comunes son:

(1) Ángulos de Euler: es la definición común para describir el movimiento angular en planos cartesianos, a través de *yaw*, *pitch* y *roll* (Thrun, Burgard, & Fox, 2005). Los ángulos son medidos desde los ejes cartesianos siguiendo la regla de la mano derecha. La Figura 1 muestra la disposición de los ejes y los nombres otorgados a cada movimiento. Es importante notar que *yaw* puede ser denominado como *heading* dependiendo del contexto en que se utilice. Si bien, indistintamente corresponden a la misma postura, en el punto (10) marco de referencia se describirá las diferencias con que se suelen utilizar.

Existe una problemática en la representación de la postura a través de ángulos tridimensionales, conocido como *Gimbal Lock*. Este fenómeno se produce cuando los ángulos de Euler se paralelizan en el plano cartesiano, quitando un grado de libertad al sistema. En estas condiciones, se pierde la postura y las referencias se deben volver a establecer.

(2) Cuaternion (*Quaternion*): es una representación en cuatro dimensiones de un cuerpo en un espacio tridimensional. Una orientación arbitraria del marco B relativo al marco A se puede obtener a través de una rotación del ángulo θ. Es posible, entonces, representar el cuaternion ${}^{A}_{B}\hat{q}$ como muestra la ecuación 1, basado en la representación de la Figura 2.

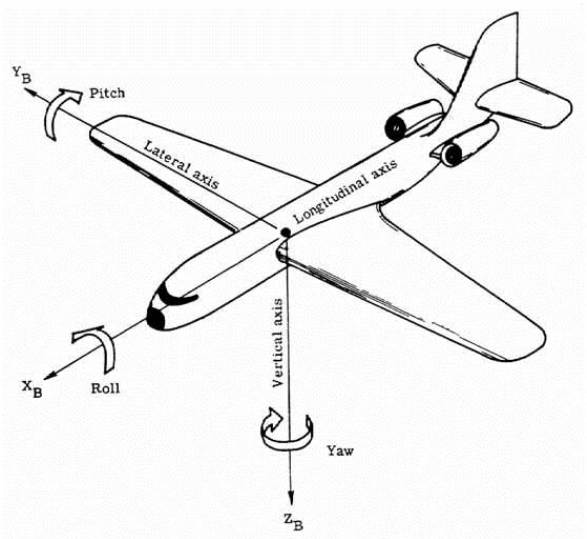


Figura 1.
Representación de los movimientos angulares de yaw, pitch y roll (Mahoney, 2013).

$$ {}^{A}_{B}\hat{q} = [q_1\ q_2\ q_3\ q_4] = [\cos\frac{\theta}{2} - r_x \sin\frac{\theta}{2} - r_y \sin\frac{\theta}{2} - r_z \sin\frac{\theta}{2}] \quad (1) $$

Los cuaterniones como conjunto poseen las operaciones de conjugación, de producto desarrollado mediante la regla de Hamilton (por tanto, los productos de cuaterniones no son conmutativos). Los cuaterniones pueden también convertirse en otros tipos de representaciones, matrices de rotación e incluso, ángulos de Euler, y en este proceso, es posible evitar los errores producidos por el *Gimbal Lock*. Las ecuaciones 2, 3 y 4 muestran la relación entre los ángulos de Euler y los cuaterniones.

$$\phi(roll) = Atan2(2q_3q_4 - 2q_1q_2, 2q_1^2 + 2q_4^2 - 1) \quad (2)$$

$$\theta(pitch) = -\sin^{-1}(2q_2q_4 + 2q_1q_3) \quad (3)$$

$$\psi(yaw) = Atan^2(2q_2q_3 - 2q_1q_4, 2q_1^2 + 2q_1^2 - 1) \quad (4)$$

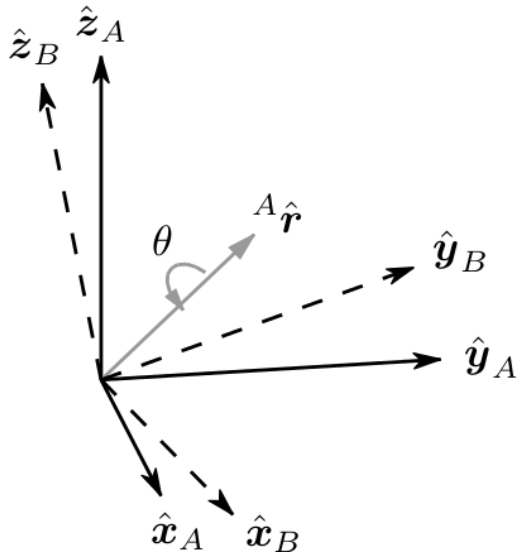


Figura 2.
La orientación del marco B es obtenida por una rotación del marco A en un ángulo θ respecto al eje ${}^Ar$. Se utiliza la notación de superíndices y subíndices adoptada por Craig para definir las referencias, donde el superíndice indica el marco de referencia, mientras que el subíndice describe el marco descrito (Madgwick et al., 2011).

A diferencia de las terminologías utilizadas en robótica, el estado cinemático o postura, donde se utiliza un vector para describir el movimiento en las coordenadas en un plano cartesiano (para movimientos restringidos a entornos planos) más un ángulo de postura, en este trabajo se utilizaran como vectores separados la postura como vector en 3 dimensiones (o 4 en su representación de cuaternion) y como vector de 3 dimensiones para la posición en un momento definido.

(2) Posición: describe la ubicación espacial respecto de un marco de referencia determinado. En el contexto de este trabajo, se utilizará posición, ubicación o localización indistintamente para determinar los movimientos lineales de un objeto (desplazamientos). Las cuantificaciones se realizan sobre un espacio cartesiano, donde se definen los movimientos realizados en cada eje componente.

(3) Acelerómetros: miden las fuerzas de aceleración que los afectan. La forma común de representar la operación de un acelerómetro es a través de un sistema masa-resorte. La Figura 3 muestra un esquema del sistema en un eje. En ausencia de fuerza alguna *-en el espacio exterior-*, la masa debería flotar en el centro del marco: al no existir fuerzas externas, la masa no manifestará peso. Sobre un planeta, estamos sometidos a la fuerza de gravedad de este. Por tanto, la gravedad nos acelera hacia el centro del planeta (y en forma constante). Una situación particular se produce en caída libre, el acelerómetro no manifiesta ser afectado por ninguna fuerza, ya que las fuerzas del marco del acelerómetro van en contra de la fuerza de gravedad (es decir, el marco del acelerómetro sostiene al sistema masa resorte con la misma aceleración que la gravedad pero en sentido contrario). El comprender

correctamente el marco respecto al cual se mide es elemental para determinar con precisión el tipo de fuerza que se está midiendo.

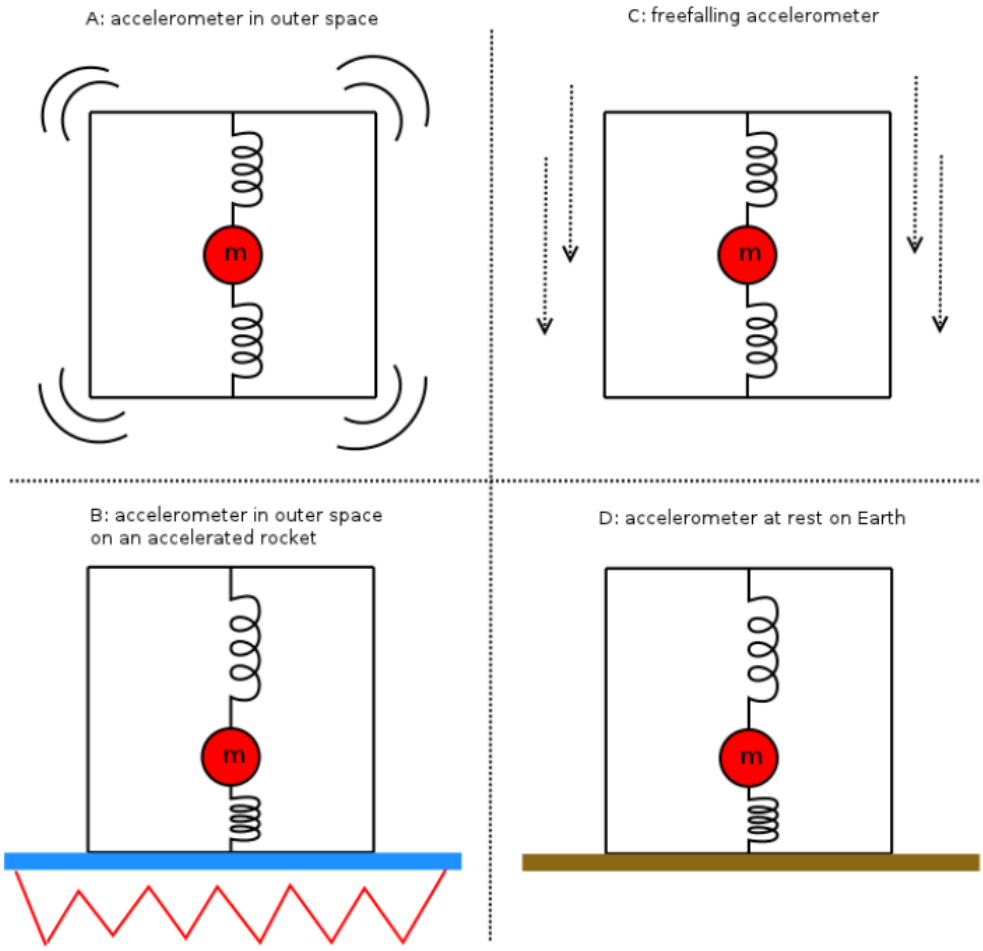


Figura 3.
Efectos de la gravedad y aceleraciones externas en un acelerómetro modelado como sistema masa resorte. En A, se observa el comportamiento de un acelerómetro en el espacio, en B se muestra el comportamiento del acelerómetro dentro de un cohete en el espacio, en C el acelerómetro en caída libre y en D un acelerómetro descansando sobre la tierra (Varesano, 2011).

La ley de Hooke señala que "*un resorte exhibirá una fuerza de restauración proporcional a la cantidad en que se ha expandido dicho resorte*" (Roetenberg, 2006). Gracias a esta ley, el acelerómetro puede medir la magnitud de la fuerza que lo afecta proporcionalmente a las diferencias de longitud que manifiestan sus resortes. La fuerza de gravedad tiene una magnitud de 9,81 m/s^2, equivalente a 1G o 1 fuerza de gravedad del planeta Tierra. Un acelerómetro construido con la tecnología MEMS difiere del modelo masa resorte, sin embargo, realiza el trabajo en forma resistiva o capacitiva. El acelerómetro puede ser utilizado para medir la aceleración de un segmento o estimar la postura a través del *tilt* (inclinación) respecto a la fuerza de la gravedad. Otras aplicaciones incluyen la detección de caídas, golpes y estimación del gasto energético.

La Tabla 1 muestra algunas de las ecuaciones más utilizadas en biomecánica para obtener información a partir de la aceleración. Se observa que, a partir del acelerómetro, se pueden obtener valores de postura y posición. La ecuación de inclinación permite conocer la postura del acelerómetro respecto al eje de la gravedad, asumiendo que el eje sobre el cual se manifiesta la gravedad es fijo. De esta forma, se puede obtener la postura del sensor, siempre y cuando se mantenga fijo un eje paralelo a la fuerza de gravedad. Conociendo la postura

del dispositivo, sería posible estimar las componentes de gravedad para cada eje, permitiendo obtener sólo las aceleraciones líneas (aceleraciones ejercidas sobre el acelerómetro sin contar la fuerza de gravedad). Realizando las integraciones para transformar la aceleración en velocidad, y luego la velocidad en desplazamiento, se puede determinar la posición del acelerómetro. Sin embargo, se presentan restricciones y errores, entre ellos:

(1) Se debe mantener un eje paralelo a la gravedad, limitando los grados de libertad a medir.
(2) Las inclinaciones sólo se pueden medir respecto a la gravedad, y con precisión hasta 90º grados desde el eje de gravedad. En forma posterior, se pierde el paralelismo de un eje con la fuerza de gravedad, siendo necesario volver a establecer el marco de referencia.
(3) No se puede estimar el *yaw* del acelerómetro, ya que estaría girando en torno al eje paralelo a la fuerza de gravedad, en tanto que una rotación no es una fuerza de aceleración en el eje paralelo a la gravedad.
(4) La doble integración en forma numérica del acelerómetro introduce un error de integración que aumentará en el tiempo. Además, es dependiente de las técnicas de integración utilizadas, según el tipo de movimiento, como se determinó en el trabajo de Seminario de Investigación titulado "Monitorización ambulatoria de Coordenadas de posición utilizando Acelerómetros, giroscopios y magnetómetros" (Sepúlveda, 2012).

(4) Giroscopios: miden el movimiento angular que los afecta. En otras palabras, permiten medir la rotación del marco de referencia del giroscopio. Al igual que los acelerómetros, es posible representar los giroscopios por modelos mecánicos simples, como el giroscopio de rueda giratoria, pero difieren mucho de lo utilizado en las tecnologías MEMS.
En un giroscopio MEMS una masa o elemento vibrante (resonante), al ser rotado, se somete al efecto Coriolis que causa una vibración ortogonal a la dirección de la vibración original. Midiendo la vibración producida por el efecto Coriolis, se puede determinar la taza de giro (Roetenberg, 2006), generando la velocidad angular en unidades de grados por segundo. La estructura vibratoria se puede modelar de forma similar al acelerómetro (Figura 4), siguiendo la misma ley de Hooke es posible determinar el movimiento angular. El método de construcción, sin embargo, permite que la masa no esté sujeta a la fuerza de gravedad, por tanto, se mide únicamente la velocidad de los movimientos angulares.
El giroscopio, entonces, resulta el candidato perfecto para determinar la postura de un objeto o cuerpo. Como muestra la , se puede determinar el ángulo en un determinado momento por la integración de los datos. Sin embargo, al igual que con el acelerómetro, la integración numérica introducirá errores que se acumularán en el tiempo de medición. Por otro lado, las mediciones se realizan respecto al marco de referencia del giroscopio, de modo que es necesario establecer un marco de referencia para realizar las mediciones, es decir, determinar cuál de los ejes corresponderá a *yaw*, *pitch* y *roll.*

	Característica	Ecuación	Descripción
Acelerómetro	Normal de aceleración	$RSS = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}$	
	Aceleración normal lineal	$ANL = RSS - 1$	Se obtiene la aceleración lineal, dado que un eje del acelerómetro este siempre paralelo con la fuerza de gravedad
	Pitch / Roll	$\theta = Atan\left(\frac{a_x}{\sqrt{a_x^2 + a_z^2}}\right)$	Donde θ corresponde al ángulo de *tilt*, y considerando que el eje z es paralelo a la fuerza de gravedad, a_x puede ser reemplazado por a_y para obtener el *tilt* en ese eje.
	Velocidad	$v = \frac{1}{t} \times \left(\int \lvert a_x(t) \rvert + \int \lvert a_y(t) \rvert + \int \lvert a_z(t) \rvert\right)$	Si la aceleración es introducida en m/s^2, el resultado será en m/s
	Desplazamiento	$\mathrm{d} = \frac{1}{\mathrm{t}} \times \left(\int \lvert \mathrm{v_x(t)} \rvert + \int \lvert \mathrm{v_y(t)} \rvert + \int \lvert \mathrm{v_z(t)} \rvert\right)$	Si la velocidad es introducida en m/s, el resultado será en metros.
Giroscopio	Ángulos de Euler	$(\psi, \theta, \phi) = \frac{1}{\mathrm{t}} \times \left(\int \lvert \omega_x(\mathrm{t}) \rvert + \int \lvert \omega_y(\mathrm{t}) \rvert + \int \lvert \omega_z(\mathrm{t}) \rvert\right)$	Donde ω corresponde a la velocidad angular medida desde el giroscopio, y (ψ, θ, ϕ) corresponde a *yaw*, *pitch* y *roll*, respectivamente
Magnetómetro	Declinación	$\mathrm{X}_h = m_x \times \cos(\theta) + m_z \times \sin(\theta)$	Donde m representa las mediciones del campo magnético desde el magnetómetro.
	Inclinación	$\mathrm{Y}_h = m_x \times \sin(\phi) \times \sin(\theta) + m_y * \cos(\phi) - m_z * sin(\phi) * \cos(\theta)$	
	Norte magnético	$\psi = Atan^2(Y_h, X_h)$	Donde ψ representa el *heading* respecto al norte magnético.

Tabla 1.
Resumen de cálculos utilizados con acelerómetros, giroscopios y magnetómetros para estimación de postura, posición.

(5) Magnetómetro: Los magnetómetros miden intensidad y dirección de campos magnéticos. Permiten conocer la ubicación del norte magnético, al igual que un compás. Los magnetómetros funcionan en tecnologías MEMS gracias al efecto Hall. Aplicando una corriente eléctrica, magneto-resistores convierten campos magnéticos incidentes en voltajes diferenciales. Conociendo el norte magnético, es posible determinar el *heading* (la postura del dispositivo respecto al norte magnético). Por ser un sensor de campos magnéticos, los magnetómetros están sujetos a una gran cantidad de distorsiones electromagnéticas.
La Tabla 1 resume las ecuaciones que se pueden utilizar para usar el magnetómetro como compás, permitiendo realizar mediciones en el marco de referencia de la Tierra (respecto al norte magnético).

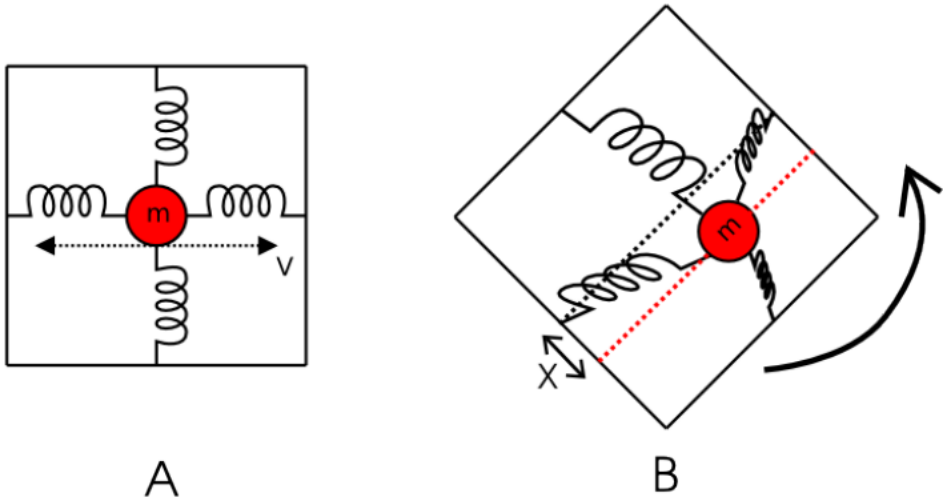


Figura 4.
Modelo de un giroscopio de estructura vibratoria (Varesano, 2011).

(6) (*Microelectromechanical Systems*): los sistemas electromecánicos combinan dispositivos mecánicos miniaturizados y componentes electrónicos para formar dispositivos miniaturizados. La tecnología MEMS permitió a la industria de la fabricación de dispositivos electrónicos incluir en sus componentes vigas, ruedas, diafragmas y resortes. De esta forma, los modelos de acelerómetros, giroscopios y magnetómetros pueden ser miniaturizados y embebidos en silicio. La Figura 5 muestra algunos de los componentes que pueden ser integrados en menos de 1mm.

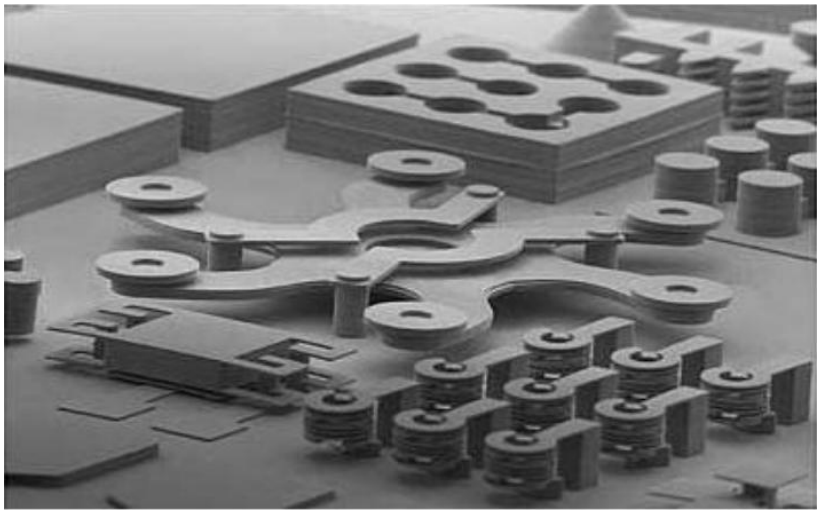


Figura 5.
Magnificación de la superficie de un dispositivo MEMS, donde los elementos mecánicos son visibles, junto con su conexión eléctrica. Los elementos varían en tamaño, desde 50 micrones hasta 5 micrones, tamaño similar al de una glóbulo rojo sanguíneo (Varesano, 2011).

(7) DOF (*Degree of Freedom*): describe los grados de libertad que posee un objeto para realizar movimientos en el espacio. En aplicaciones con sensores inerciales, se acostumbra a entregar 1 DOF a cada dimensión espacial en la que el dispositivo puede medir magnitudes.

(8) IMU (*Inertial Measurement Unit*): se denomina IMU al conjunto de sensores inerciales que permiten estimar la postura de un objeto en el espacio. Para alcanzar gran precisión, se utilizan diferentes sensores para estimar el mismo parámetro y entregar un resultado fusionado de todos los sensores, con menor error de estimación. Se denomina:
   (1) IMU 3DOF: IMU compuesto por un acelerómetro de 3 ejes.
   (2) IMU 6DOF: IMU 3DOF adicionado a un giroscopio de 3 ejes.
   (3) IMU 9DOF: IMU 6DOF adicionado a un magnetómetro de 3 ejes.
   (4) IMU 10DOF: IMU 9DOF adicionado a un barómetro.

(9) Fusión de sensores: las ecuaciones de la Tabla 1 muestran la redundancia que tienen los 3 sensores presentados para medir la postura, respecto a diferentes marcos de referencia. Poseen propiedades específicas, con ventajas y desventajas, pero con la posibilidad de que, respecto al conjunto de mediciones, podríamos obtener una estimación más cercana a la realidad. A este proceso, de combinar convenientemente los resultados de la orientación de los sensores, se le conoce como fusión de sensores, y se puede alcanzar mediante diferentes métodos, diferentes cargas de procesamiento y diferentes precisiones.
   En breve, y para resumir, un IMU 9DOF nos permite:
   (1) Mediante el acelerómetro, medir inclinaciones respecto a la gravedad, siempre que el paralelismo de uno de los ejes con la fuerza de gravedad se mantenga fijo. No se puede obtener la postura de *yaw*.
   (2) Mediante el giroscopio, medir directamente la postura expresada en ángulos de Euler respecto a su propio marco de referencia, con un error de integración que aumentará en el tiempo.
   (3) El magnetómetro permite medir la postura del dispositivo respecto al norte magnético de la tierra, permitiendo medir *yaw* o *heading* respecto con un marco de referencia en la tierra.

   Es posible, entonces, determinar las relaciones que muestra la ecuaciones 5. Donde los subíndices a corresponden a la postura determinada por el acelerómetro, g corresponde a la postura mediante integración del giroscopio (incluido $\Delta\mathrm{e}$ como error de integración) y m como la orientación respecto al norte magnético en función del acelerómetro y los campos magnéticos. La Figura 6 muestra esquemáticamente como se realiza la fusión de los sensores.

$$\begin{aligned} \psi_g + \Delta\mathrm{e} &\cong \psi_{(m,a)} + \mathrm{e}_{(m,a)} \\ \theta_g + \Delta\mathrm{e} &\approx \theta_a + \mathrm{e}_a \\ \phi_g + \Delta\mathrm{e} &\approx \phi_a + \mathrm{e}_a \end{aligned} \tag{5}$$

   Al ponderar apropiadamente cada una de estas posturas, es posible disminuir el error de cada uno de los sensores. El proceso de ponderar apropiadamente los sensores se conoce como filtro complementario, y consiste en asignar diferentes ponderaciones a la postura calculada por cada sensor. Sin embargo, el filtro complementario no permite predicciones ni estimaciones del error introducido por cada sensor. Entonces, es necesario un filtro más

robusto y que permita predecir el error para compensarlo, y el principal candidato es el filtro Kalman.

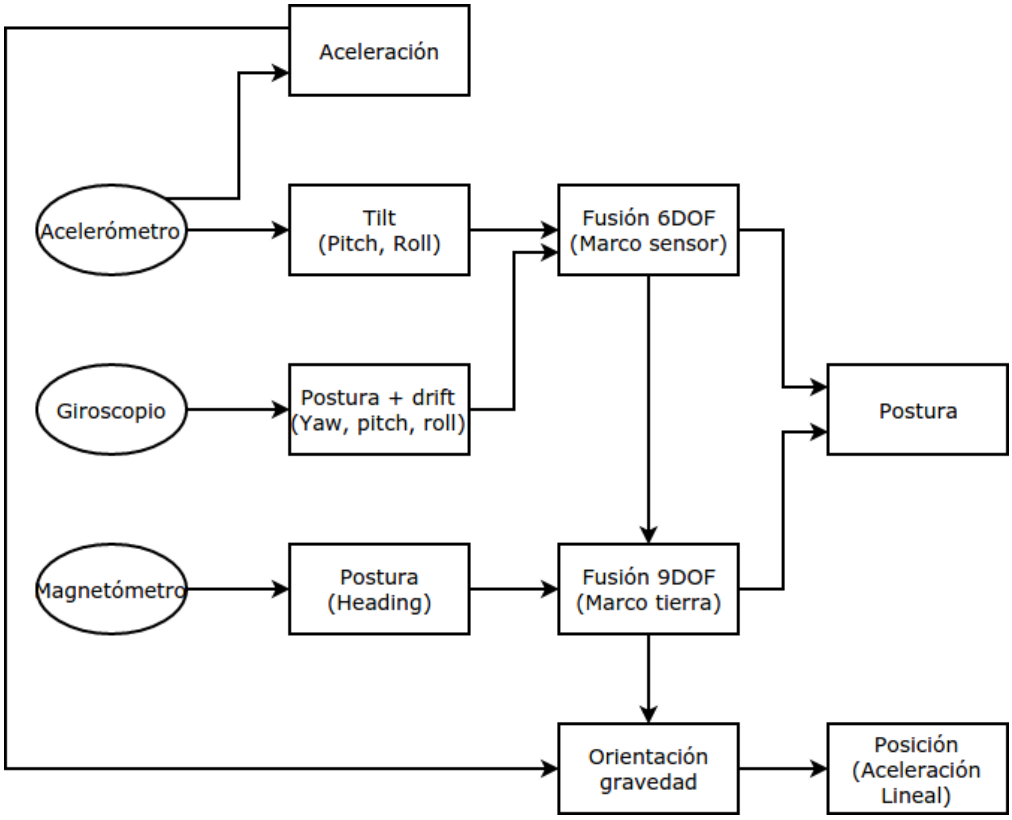


Figura 6.
Diagrama esquemático de proceso de fusión de sensores.

El filtro Kalman es un algoritmo recursivo de datos, que busca estimar los estados en forma óptima, minimizando el error cuadrático medio. El filtro Kalman (Bishop & Welch, 2001) se presenta como modelo lineal, en la forma de la ecuación 6, 7 y 8, donde:

- A, B, y C son matrices.
- k es el índice de tiempo.
- $u(k)$ representa la entrada del sistema
- $x(k)$ es el estado del sistema.
- $y(k)$ es la salida del sistema.
- $w(k)$ es el ruido asociado a la medición.
- $v(k)$ es el ruido asociado al sistema.
- $\hat{x}(k)$ es el estado estimado del sistema.
- $e(k)$ es la diferencia entre los estados, real y estimada.
- $P(n)$ es la matriz de covarianza del error.

El objetivo del filtro es minimizar la ecuación 9 para disminuir el error cuadrático estimando el estado del sistema.

$$x(k+1) = A \times x(k) + B \times u(k) + v(k) \tag{6}$$

$$y(k) = C \times x(k) + \omega(k) \quad (7)$$

$$e(k) = x(k) - \hat{x}(k) \quad (8)$$

$$P(n) = E(e(n) \times e^T(n)) \quad (9)$$

El filtro Kalman presenta una solución general, este tiene que ser adaptado a la problemática a enfrentar. Tal como se revisó en el estado del arte, respecto a los algoritmos de fusión, existen diferentes aplicaciones del filtro Kalman para la fusión de sensores IMU, principalmente, con diferencias en la estimación del error y la representación sobre la cual se realiza la estimación de la postura.

Sin embargo, el filtro Kalman requiere una alta carga computacional. Por este motivo, se han desarrollado algoritmos de fusión que tengan la simplicidad de un filtro complementario y la precisión del filtro Kalman. Sebastian Madgwick (2011) propuso un filtro utilizando gradientes descendientes, logrando cumplir con los requisitos expuestos. En este trabajo, este algoritmo de fusión se utilizará, debido a que se distribuye en forma libre a la comunidad *open source* y es posible implementarlo en micro controladores con bajo poder de cómputo.

La Figura 7 muestra un diagrama en bloques del algoritmo de fusión. El algoritmo incluye etapas de corrección para la compensación de distorsiones magnéticas y los errores de integración (*drift*) para el giroscopio.

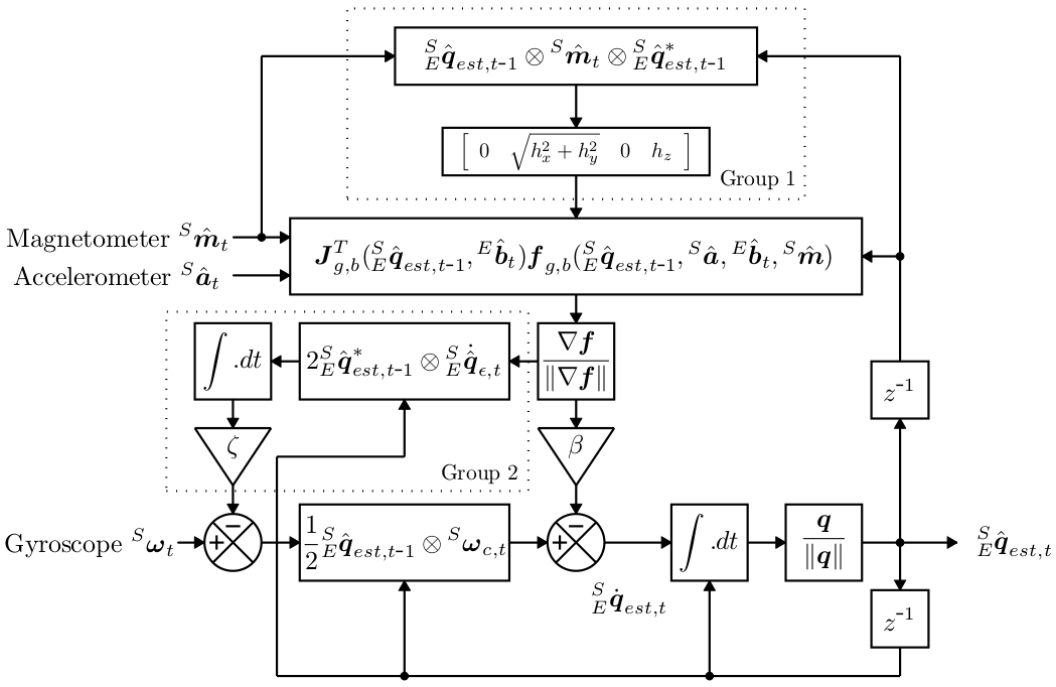


Figura 7.
Diagrama en bloques del algoritmo de gradientes descendientes propuesto por Madgwick para estimar postura utilizando IMU 9DOF. Grupo 1 corresponde a la compensación de distorciones magnéticas, grupo 2 corresponde a la compensación del drift del giroscopio. β corresponde al parámetro definido por el usuario para compensar el error del giroscopio (Madgwick et al., 2011).

El algoritmo comienza tomando la postura por medio del giroscopio, donde cada eje es ubicado en el vector S_ω para su paso a cuaternion describiendo el rango de cambio desde el marco terrestre hacia el marco del dispositivo ($\dot{q}$). La postura del marco de la tierra relativa al sensor en un momento determinado, $q_{w,t}$ puede ser calculada integrando numéricamente la derivada del cuaternion $\dot{q}$.

$$S_\omega = [0\ \omega_x\ \omega_y\ \omega_z] \tag{10}$$

Posteriormente, y para la inicialización del algoritmo, se asume que el acelerómetro sólo será afectado por la fuerza de gravedad, y que el magnetómetro sólo mediara el campo magnético de la tierra. Si se conoce la dirección del campo magnético de la tierra, es posible establecer las mediciones de postura respecto al marco de la tierra. La representación en cuaterniones requiere que se encuentre una única solución al problema donde la estimación de la orientación del sensor, ${}^S_E\hat{q}$, este alineada con la dirección del campo en el marco del tierra. Varios algoritmos de optimización pueden encontrar la solución al problema, siendo por medio de gradientes descendientes el método más simple de implementar y calcular. Se puede realizar una estimación inicial sobre ${}^S_E\hat{q}_0$, por medio del cómputo del error de dirección realizada por la función f y su jacobiano, J. El algoritmo se puede optimizar más considerando que sólo la fuerza de gravedad está presente (en este caso, en el eje Z del acelerómetro), permitiendo definir los vectores $E_{\hat{g}}$ y $S_{\hat{a}}$, para la orientación de la gravedad y los valores de aceleración, respectivamente. El campo magnético de la tierra, se puede representar por sus componentes verticales y horizontales (inclinación y declinación, que son conocidas (mapeadas) según la ubicación en el planeta). Estas mediciones se sustituyen en los vectores $E_{\hat{b}}$ y $S_{\hat{b}}$. Por parte del acelerómetro y el magnetómetro se obtiene la medición de orientación ${}^S_E q_{\nabla,t}$.

La postura del marco de la tierra relativa al sensor ${}^S_E q_{est}$ se obtiene por la fusión de dos orientaciones separadas, ${}^S_E q_{w,t}$ y ${}^S_E q_{\nabla,t}$. La compensación magnética se logra en el grupo 1 de la Figura 7, mediante la definición de $E_{\hat{b}t}$ como muestra la ecuación 12. El grupo 2 de la Figura 7 se encarga de la compensación del giroscopio, mientras que el parámetro β representa el error de medición en el giroscopio. Este parámetro debe ser ajustado al error producido por el giroscopio utilizado.

$$\begin{gathered} E_{\hat{g}} = [0\ \ 0\ \ 0\ \ 1] \\ S_{\hat{g}} = [0\ \ a_x\ \ a_y\ \ a_z] \\ S_{\hat{m}} = [0\ \ m_x\ \ m_y\ \ m_z] \\ E_{\hat{b}} = [0\ \ b_x\ 0\ \ b_y] \end{gathered} \tag{11}$$

$$E_{\hat{b}t} = [0\ \sqrt{h_x^2 + h_y^2}\ \ 0\ \ h_z] \tag{12}$$

Obtenida la postura, es común convertir los resultados de cuaterniones a ángulos de Euler, con el fin de obtener una representación más cercana a nuestra realidad. La postura permite obtener la dirección esperada de la fuerza de gravedad, sin tener que fijar un eje, por tanto, es posible obtener aceleración lineal dinámicamente a través de la fusión de sensores, obtener la postura del objeto o cuerpo, y obtener la orientación esperada del marco terrestre.

El algoritmo descrito se utiliza, modificado, en soluciones comerciales como las de X-Sens, incluyendo mejoras en la determinación inicial de los sensores y la calibración inicial del magnetómetro.

Con el fin de asegurar una correcta estimación inicial del campo magnético, se realiza una medición esférica del campo magnético, permitiendo detectar las distorsiones iniciales del campo magnético y corregirlas en las mediciones siguientes del magnetómetro.

(10) Marco de referencia: durante el desarrollo del trabajo, se ha nombrado en varias oportunidades los marcos de referencia, sin definirlos como tales. Las descripciones de los sensores y posteriormente del algoritmo de fusión permite dar mejores indicios a lo que es un marco de referencia. Para formalizar, la definición, entonces:
   (1) El marco de referencia del sensor, propio o del dispositivo, es el que se utiliza para realizar una medición respecto a un estado anterior del mismo dispositivo (es la característica principal de un sensor inercial). Una medición de este tipo se puede lograr con un sensor IMU 6DOF.
   (2) El marco de referencia en la tierra o terrestre es aquel que utiliza puntos terrestres (norte magnético en componentes verticales y horizontales) para referenciar las mediciones respecto a estos puntos. Una medición de este tipo se puede lograr con un sensor IMU 9DOF.

(11) I2C (*Inter-Integrated Circuit*): es un protocolo de comunicación serial punto multipunto que utiliza 1 bus de datos bidireccional (SDA) y un bus de reloj (SCL) (también conocido como *2-wire*). El reloj puede alcanzar velocidades de 400 KHz en modo *fastmode*. Cada dispositivo tiene un identificador único o dirección I2C. La gran mayoría de los fabricantes de dispositivos ofrecen hasta 2 direcciones I2C seleccionables por hardware.

(12) SPI (*Serial Peripheral Interface Bus*): es un protocolo similar a I2C pero que alcanza mayores velocidades a costa de separar los buses de datos en uno de entrada y otro de salida. Adicionalmente, reemplaza las direcciones I2C por una condición lógica para activar o desactivar la lectura del dispositivo. De esta forma, se alcanzan mayores velocidades de transferencia a costa de mayor cantidad de cables (también conocido como 4-wire).

3.2. Diseño de la propuesta

A continuación se describe la metodología utilizada para resolver y desarrollar el prototipo, luego se especifican los requisitos para posteriormente describir la implementación realizada separada temáticamente.

3.2.1. Metodología

La metodología basada en desarrollo iterativo y creciente (Constantine & Lockwood, 1999; Sommerville, 2005) es ampliamente utilizada en el desarrollo de software y hardware. De esta manera, y mediante iteraciones, se pueden dividir las tareas y funciones e incluirlas en forma incremental hasta alcanzar un proyecto estable. Adicionalmente, este método de desarrollo permite la continuidad de los proyectos, fomentando sus actualizaciones luego de la iteración final. Para este trabajo, la metodología de desarrollo considera:

(1) El problema biomecánico se puede acotar a dos segmentos del cuerpo. De esta forma, es posible desarrollar el hardware y software necesario para un brazo, y una vez verificado y validado el segmento, es posible replicarlo al otro brazo y luego las piernas, finalizando con los segmentos de tronco y cabeza. Esto permite que, la propuesta de desarrollo sea alcanzable y con la modularidad suficiente para alcanzarse en futuros trabajos.
(2) Definida la unidad a desarrollar, se desarrolla el hardware y software en forma iterativa y creciente, hasta alcanzar el seguimiento mediante IMUs.

(3) Para probar la funcionalidad del equipo, ser realizaran pruebas en terreno, controlado el dispositivo en forma remota y realizando mediciones de movimiento estandarizados.
(4) Una vez que el prototipo sea funcional, se realizarán las validaciones utilizando la metodología de Investigación, que permitirá comparar contra el *Gold Standard* el sistema propuesto. Esta etapa esta descrita en el proyecto de tesis de magister y sus resultados son parte de la tesis de magister.
(5) Finalizadas las iteraciones y alcanzando un producto estable; se propone el diseño final que podría replicarse a los otros segmentos.

3.2.2. Requisitos

(1) La plataforma debe ser modular, permitiendo:
    a. Utilizar diferentes cantidades de sensores.
    b. Utilizar diferentes arquitecturas de procesamiento.
    c. Portabilidad de los códigos a otras plataformas o arquitecturas.
    d. Portabilidad para intercambiar el tipo de conexión a la plataforma, en forma alámbrica o inalámbrica.
    e. La plataforma implementada deberá tener la posibilidad de conectarse a una red inalámbrica para monitorear en forma ambulatoria y tiempo real los parámetros.
    f. Para la captura del movimiento, es necesario contar con un sistema capaz de procesar y registrar datos a por lo menos 25Hz por cada sensor, donde se encuentran los máximos contenidos de frecuencia para el movimiento en brazos. Se ha determinado, que para la captura del movimiento humano, es necesario realizar mediciones a más de 50Hz para cumplir con el criterio de Nyquist, llegando, inclusive, a 100Hz, para hacerlo comparable con la mayor cantidad de sistemas (Chen & Bassett, 2005; Lowe & OLaighin, 2014).
(2) El control de la plataforma debe ser independiente de sistemas operativos o dispositivos utilizados, adaptándose apropiadamente al cliente.
(3) La plataforma debe registrar los datos en una memoria interna.
(4) La plataforma debe operar a baterías, minimizando el consumo energético.

3.3. Implementación

Siguiendo la metodología de trabajo se comienza a desarrollar el prototipo de dispositivo. En primera instancia, se seleccionan los sensores IMU a utilizar.

3.3.1. Selección de los sensores

Según los criterios establecidos por Lowe y OLaighin (2014), los componentes de los sensores inerciales utilizados hasta el momento deben cumplir características técnicas mínimas. Adicionalmente, y para cumplir los requisitos impuestos para el desarrollo de este trabajo, se necesita una solución de bajo consumo y en un solo circuito integrado. Debido al nivel de miniaturización de los dispositivos, es importante contar con *breakout boards* para el desarrollo inicial, para no incurrir en pérdidas de tiempo y material al tratar de soldar componentes de dimensiones pequeñas. Bajo estas condiciones, dos sensores IMU 9DOF cumplen los requisitos: MPU-9150 de Invensense y LSM9DS0 de ST Electronics. Las Tablas 2, 3 y 4 muestran los requisitos determinados por Lowe y OLaighin (2014), junto con la comparación de las características técnicas de cada dispositivo.

Ambos sensores cumplen con los requisitos mínimos, y en una primera revisión, ambos son bastante similares. A nivel de *wafle* (oblea sobre la que se montan los componentes integrados), el MPU-9150 está compuesto por un MPU-6050 (Acelerómetro y giroscopio integrados) y un magnetómetro AKM AK8975. La comunicación es únicamente a través de I2C, cada uno de los componentes del silicio tiene una dirección I2C. Adicionalmente, el MPU-9150 posee un DMP (componente integrado propietario de Invensense, *Digital Motion Processor*) que permite realizar la fusión de sensores en el dispositivo, entregando la posibilidad de definir si se desean los resultados en cuaterniones o ángulos de Euler. La solución embebida resulta prometedora, ya que se aliviana la carga de procesamiento del micro controlador. Sin embargo, la fusión que el DMP realiza es sólo para el MPU-6050, por tanto es una fusión 6DOF que no incluye magnetómetro. El magnetómetro posee menor resolución espacial y temporal que sus pares y sus competidores, como se observa en la Tabla 4. Y presenta un punto bajo para el procesamiento, y quizás explica por qué el fabricante no incluyó la fusión 9DOF en el DMP: La frecuencia de muestreo máxima del magnetómetro es mucho menor que la frecuencia de muestreo de los demás sensores. Existen reportes donde el MPU-9150 llega a alcanzar frecuencias de muestreo de hasta 1KS/s, siempre y cuando se desactive el magnetómetro y el DMP (Barnett, 2014). El DMP además aumenta el consumo energético del dispositivo, volviéndolo difícil de controlar en aplicaciones de bajo consumo.

	Requisitos mínimos	MPU-9150	LSM9DS0
Rango	± 8G	± 16G (16 bits)	± 16G (16 bits)
Sensibilidad	3mG/LSB	0,488 mG/LSB	0,732 mG/LSB
Ejes	3	3	3
Ancho de Banda	>100Hz	1KHz	16KHz
Ruido	100 $\mu g/\sqrt{Hz}$	400 $\mu g/\sqrt{Hz}$	N.A.

Tabla 2.
Comparación y características mínimas de características de los acelerómetros utilizados.

	Requisitos mínimos	MPU-9150	LSM9DS0
Rango	>300 °/s	2000 °/s (16 bits)	2000 °/s (16 bits)
Sensibilidad	15 mV/(°/s)	16,4 LSB/(°/s)	14,28 LSB/(°/s)
Ejes	3	3	3
Ancho de Banda	>200KHz	8KHz	8KHz
Ruido	N.A.	0.005 (°/s) /$\sqrt{Hz}$	N.A.
Compensación temperatura	Sí	Si	Si

Tabla 3.
Comparación y características mínimas de características de los giroscopios utilizados.

	MPU-9150	LSM9DS0
Rango	± 12 Gs (13 bits)	± 13 Gs (16 bits)
Sensibilidad	0.3 mGs/LSB	0.48 mGs/LSB
Ejes	3	3
Ancho de Banda	10Hz	100Hz

Tabla 4.
Comparación de características de los magnetómetros utilizados.

Al momento de realizar la fusión de sensores, esto generaría un cuello de botella en el magnetómetro, limitando la frecuencia de muestreo del algoritmo completo.

A diferencia del MPU-9150, el LSM9DS0 a nivel de silicio asocia el magnetómetro con el acelerómetro, y luego adiciona un giroscopio, por el mismo motivo, para comunicarse con todos los sensores se deben utilizar dos direcciones I2C. Una ventaja relativa que ofrece el LSM9DS0 es que puede comunicarse tanto por I2C como por SPI.

Tampoco integra un procesador para realizar la fusión, sin embargo, logra menor consumo de corriente, junto con entregar un mejor magnetómetro. La frecuencia de muestreo del magnetómetro llega hasta los 100S/s, volviéndose un cuello de botella apropiado para la aplicación en que se desea utilizar.

Dadas las condiciones señaladas, se adquirieron 5 IMU LSM9DS0, tanto por la versatilidad de conexiones, bajo consumo energético y mejor frecuencia de muestreo para el más lento de sus dispositivos.

3.3.2. Selección de protocolo de comunicación

El sensor LSM9DS0 puede comunicarse a través de los protocolos seriales I2C o SPI. Si bien el protocolo SPI es más apropiado para la aplicación por permitir velocidades de transferencia superiores al protocolo I2C, se necesitan mayor cantidad de cables para poder utilizarse: dos cables adicionales para realizar la selección del dispositivo, para cada IMU. Esto debido a que, a nivel de *wafle*, el IMU está compuesto por dos integrados, una unidad tiene el acelerómetro y magnetómetro, mientras la otra tiene sólo el giroscopio. Esto significa que, para cada IMU, serán necesarios 6 cables en total: 2 para alimentación, 2 para selección de dispositivo y 2 para comunicación.

El protocolo I2C utiliza menos cables para la comunicación y la selección del dispositivo, a costa de una velocidad menor. Adicionalmente, los desarrolladores de integrados sólo proveen dos direcciones I2C, permitiendo conectar un máximo de dos LSM9DS0 a un mismo canal I2C.

A nivel lógico, existe un problema con las plataformas de desarrollo a utilizar. Los sensores utilizan una lógica de 3.3V; mientras que algunas plataformas de desarrollo tienen comunicación a un nivel lógico mayor, 5V. Si bien, el problema no es limitante, ya que es probable que al interconectar ambos niveles lógicos los niveles altos y bajos se mantengan sobre las zonas de transición, energizar los sensores con voltajes superiores a los sugeridos por el fabricante podría reducir su vida útil. Sería apropiado utilizar un conversor de niveles lógicos tanto para SPI como para I2C, a fin de asegurar una correcta comunicación entre los dispositivos.

En consideración de ambas problemáticas, se utilizó un multiplexor I2C de 8 canales, PCA9547 de NXP Semiconductors, que permitiría conectar hasta 16 IMUs con un multiplexor (es posible conectar un máximo de 8 multiplexores, logrando 128 IMUs).

Este multiplexor ayuda además a compensar las capacitancias de los cables, que son críticas para la correcta comunicación I2C. Adicionalmente, maneja apropiadamente las diferencias de niveles lógicos de los dispositivos.

El cableado debe realizarse con precaución para distancias mayores a 2 metros. Se espera tener más de un metro de cable por cada sensor, por tanto, es importante cumplir con las especificaciones del estándar I2C. Los cables más apropiados para la comunicación son el cable multifilar apantallado AWG 24 y AWG 32. En consideración de las características de movilidad del dispositivo, el cable plano es más flexible, permitiendo mayor comodidad a la hora de realizar movimientos.

3.3.3. Selección de la plataforma de desarrollo

Una vez seleccionado el sensor, se realizó la selección de las plataformas que cumplan de mejor manera los requisitos especificados. Los dispositivos con los que se realizaron pruebas para adquirir, procesar y registrar los datos son:

(1) Arduino Yún: provee un sistema de prototipo rápido junto por medio de un MCU AVR ATmega 32u4 de 8 bits, comunicándose por UART con un chipset Atheros AR9331, que provee capacidad similares a las de un *router* corriendo OpenWrt; una distribución de Linux modificada para *routers*. A través del kernel Linux, es posible montar tarjetas microSD para almacenar información, y el gestor de paquetes de OpenWrt permite instalar utilidades como Python. Esta plataforma, pensada para actuar con el IoT (*Internet of Things*) permite desarrollar proyectos que puedan interactuar directamente en forma inalámbrica con internet o redes locales.

En el firmware de Arduino, se cargó la librería RTIMULib para Arduino (Barnett, 2014), la que permite utilizar indistintamente sensores MPU-9150 o LSM9DS0. Se comprobó la transferencia por medio de un puerto serial virtual para verificar el funcionamiento de la adquisición.

Para el almacenamiento, Arduino Yún posee acceso a una memoria micro SD que es manejada a través del Atheros AR9331. Por tanto los datos deben ser enviados al router antes de ser almacenados.

Con este fin, la comunidad Arduino provee una librería llamada Bridge, que permite realizar el puente a través de UART desde el micro controlador hacia el router. La librería no ofrece el máximo desempeño del dispositivo, además de aumentar el tamaño y el uso de memoria. En búsqueda de una solución de más bajo nivel, se implementó sólo la librería RTIMULib para obtener los datos crudos y enviarlos por UART hacia el router de la plataforma Arduino Yún. De este modo, el micro controlador actúa tan solo como intérprete de datos.

Como el router cuenta con una versión modificada de OpenWrt, es posible instalar los paquetes disponibles en el repositorio de OpenWrt, dentro de los cuales se encuentra Python y varias de sus librerías, entre ellas el *framework* Autobahn.

Autobahn (Tavendo, 2014) es un proyecto *open source* que provee implementaciones de los protocolos WebSocket y WAMP (*Web Application Messaging Protocol*) en Python, JavaScript, Android y C++. Gracias a este *framework*, es posible montar una página web que permita controlar el dispositivo a través de la red y capturar los datos y almacenarlos en forma local o remota. Las posibilidades de contar con el protocolo WebSocket operativo permiten una gran cantidad de aplicaciones basadas en IoT, con actualizaciones en tiempo real.

El *framework* Autobahn, ofrece entonces, las herramientas necesarias para armar un prototipo controlable en forma remota a través de un navegador Web, permitiendo configurar remotamente el dispositivo, controlar la adquisición de los datos y finalmente descargarlos para su análisis.

El firmware de OpenWrt utiliza gran parte de la memoria destinada a programas en el router, por tanto, se vuelve necesario montar la micro SD como parte del sistema de archivos, e instalar las aplicaciones en ella. Se expande entonces, la memoria disponible para el router hacia la tarjeta micro SD, obteniendo una mayor capacidad de almacenamiento. Enlazados al micro SD, se realiza la instalación de los paquetes y servicios:

- *Openssh-sftp-server*, un servidor de sistemas SFTP (*Secure File Transfer Protocol*)
- *Bzip, unzip tar*, aplicaciones para descomprimir archivos.
- *Wget*, aplicación para descargar archivos por línea de comandos.
- *Fdisk, e2fsprogs*, aplicaciones para manejar sistema de ficheros.

- *Python, pyopenssl, python-openssl, python-crypto, python-bzip2, python-sqlite3, python-ncurses*, aplicaciones y librerías de Python necesarias para correr la *framework* Autobahn.
- *Setuptools*, una herramienta de Python que ofrece un repositorio para instalar librerías de Python, entre ellas, Autobahn y sus dependencias: *zope.interface* y *twisted* que permiten la comunicación a bajo nivel entre el UART del micro controlador y el router.
- *PySerial*, librería de Python para acceder al puerto serial (UART).

Una vez compilados e instalados los programas, es necesario desactivar la librería Bridge de Arduino para liberar el puerto serial. Esto se consigue modificando el archivo */etc/inttab*, donde se puede encontrar la línea que registra un intérprete de comandos ASH al puerto serial del router (*/dev/ttyATH0*).
Autobahn ofrece una documentación detallada y ejemplos en su repositorio, entre ellos, el ejemplo ubicado en */examples/twisted/wamp/app/serial2ws* ejemplifica como utilizar el servidor WAMP y WebSockets para comunicar el micro controlador de la plataforma Arduino Yún con el router. El servidor WAMP se encarga de transformar la comunicación UART a un lenguaje entendible por el protocolo TCP/IP, logrando comunicar los datos a través de WebSocket a medida que el micro controlador los envía hacia una página cliente, montada en el mismo servidor. Iniciado el servidor, basta con entrar a la dirección http://arduino.localhost:8080 para acceder a la página web. El detalle de la construcción de la página web se encuentra en la sección 3.3.6 Software del dispositivo.
Tras tener operacional el sistema, se realizan adquisiciones con un sensor, luego con varios sensores y se registran los datos obtenidos en un archivo CSV (*Comma Separated Values*) junto con el tiempo en que se realizó cada adquisición (*time stamp*).
Las pruebas mostraron que la frecuencia de muestro obtenida rondaba los 20Hz con un solo sensor. Sin embargo, al agregar más sensores, el ancho de banda se repartía entre los diferentes sensores, llegando a frecuencias de 4Hz cuando 5 IMUs estaban conectados.
Esta limitante descarta la plataforma Arduino Yún para incluirse en el prototipo.

(2) Nucleo F401RE: con un MCU ARM STM32F401RE de 32 bits, y compatibilidad con dispositivos creados para Arduino (*Shields*), esta plataforma entrega una mayor capacidad de cómputo (con punto flotante por hardware) y la posibilidad de interactuar con más periféricos a costos similares.
El framework Autobah resultó ser una alternativa interesante, y provee la portabilidad para ser implementado en diferentes arquitecturas. Tras implementar el *firmware* para adquirir datos de los sensores, la cantidad de periféricos no incluidos, respecto a la plataforma Arduino Yún, generaría un aumento en el tiempo de desarrollo. A diferencia de la plataforma Arduino Yún, la plataforma Nucleo F401RE no cuenta con almacenamiento directo a una memoria externa, ni un sistema de comunicación inalámbrica.
En vista de las problemáticas presentadas, se decide continuar con la siguiente plataforma, debido a que la cantidad de componentes hardware que habría que implementar supera el tiempo destinado de desarrollo.

(3) Raspberry Pi (Modelo B revisión 2): provee un sistema de Linux embebido sobre un procesador ARM1176JZF-S (ARMv6k), junto a un procesador de gráficos Broadcom VideoCore IV, entregan grandes capacidades de procesamiento y almacenamiento. Dispone de GPIO que permiten establecer comunicación UART, I2C y SPI con otros

dispositivos, además de correr distribuciones como GNU/Linux Debían para ARM, permite conexión a otros dispositivos a través de Ethernet o USB.

La plataforma de desarrollo RPi (Raspberry Pi) provee una mezcla entre la cantidad de periféricos embebidos en el sistema junto con la posibilidad de actuar como un punto de acceso WiFi por medio de sus puertos USB.

Tras la instalación del firmware, como se describe en 3.3.5 Firmware del dispositivo, se portó la aplicación desarrollada para Arduino Yún al RPi, transfiriendo los archivos y modificándolos para volverlos accesibles a Python. Es necesario portar el controlador del multiplexor PCA9547 a Python, tras lo cual se pueden realizar las pruebas de funcionamiento.

Con la configuración implementada, donde es importante señalar la habilitación de las velocidades *fastboot* para el canal I2C, y a través del uso de la librería *multiprocessing* de Python, se lograron frecuencias de muestro de entre 90Hz a 120Hz utilizando 5 IMUs multiplexados, logrando la máxima configuración esperada para el procesamiento de los datos.

Con esto, se formaliza la decisión de utilizar la plataforma RPi para el prototipo, cumpliendo con los requisitos de procesamiento especificados. Para una versión final, la plataforma RPi no permite diseñar soluciones embebidas, debido a que varios de sus componentes son difíciles de encontrar en el mercado, junto con soluciones de soldadura avanzada. Sin embargo, la arquitectura ARM permite portabilidad hacia otras plataformas, manteniendo. Dado que las pruebas demostraron que se necesita un poder computo importante, un MCU ARM de 32 bits es más apropiado para una solución embebida, añadiendo los dispositivos periféricos faltantes como la memoria externa y la comunicación inalámbrica por WiFi.

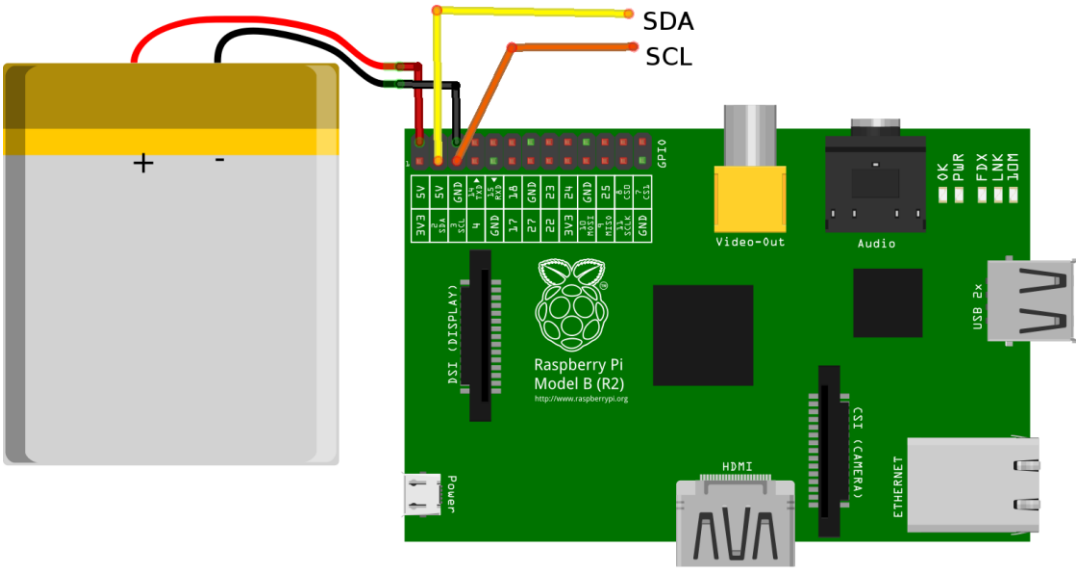


Figura 8.
Diagrama de conexión básica de la plataforma Raspberry Pi.

3.3.4. Hardware del dispositivo

El objetivo de utilizar una plataforma de desarrollo es realizar los prototipos con la menor cantidad de desarrollo de Hardware. En la Figura 8 se observan los componentes de la plataforma, junto con las conexiones básicas para hacer funcionar la plataforma. En uno de los puertos USB se conecta el WiFi, y en los terminales SDA y SCL las conexiones al multiplexor I2C y los sensores IMU fue necesario desarrollar un *breakout board* para el multiplexor PCA9547, debido a que su

encapsulado es de montaje superficial. El esquemático del anexo 9.1 muestra la circuitería necesaria para el multiplexor. El objetivo de esta placa es tener acceso simple a conectar uno o más sensores. Para esto, se habilitaron pines (*headers*) para conectores IDC de cable plano, de 5x2 pines. Esto permite realizar la conexión de los sensores en forma reversible, por tanto el usuario no puede conectar en forma errónea el dispositivo.

A pesar de no tener la misma topología que los *Shield* R3 de Arduino, en el diseño se consideró mantener las dimensiones dentro del área de prototipo de los *Shields*. Esto permitirá portabilidad a las plataformas de desarrollo ya utilizadas, específicamente la plataforma Nucleo F401RE, que se postula como el mejor candidato para una solución final embebida.

Para energizar la plataforma de desarrollo, se utilizó una batería recargable externa de 2800mAh, con la cual se logró hasta 2 horas de adquisición continua, sin implementar ningún tipo de ahorro energético.

Para proteger la electrónica, se utilizó un *enclosure* acrílico para la plataforma RPi, mientras que para cada uno de los sensores se fabricó una caja que permitiera proteger al *breakout board* del sensor LSM9DS0 y aislar sus conexiones del sujeto de pruebas, junto con la posibilidad de sujetar el dispositivo mediante velcros a los diferentes segmentos del cuerpo. La Figura 9 muestra una representación de la caja, mientras que en el anexo 9.3 se incluyen los esquemáticos del dispositivo. El modelo se imprimió en una impresora 3D MCube 2, utilizando como material PLA.

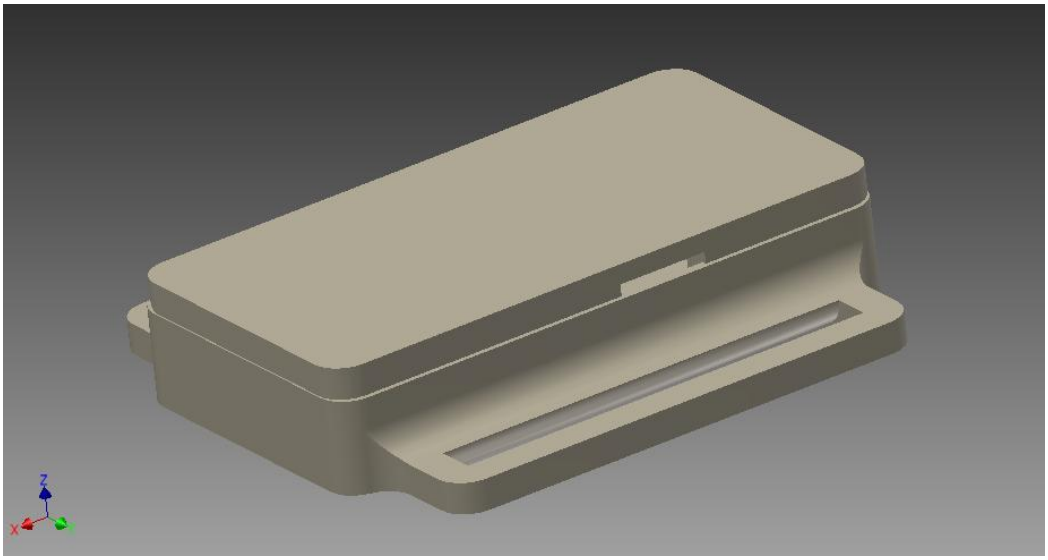


Figura 9.
Representación del diseño del *enclosure* para LSM9DS0.

3.3.5. Firmware del dispositivo

Como firmware de la plataforma de desarrollo RPi, se instaló la distribución Debian GNU/Linux con soporte para punto flotante por Hardware (Linux 3.12.28+ armv6l). Tras la instalación, se habilitó el servidor SSH para acceder en forma remota a través de Ethernet. Siguiendo la documentación para instalar RTIMULib en RPi disponibles en el repositorio de la librería

- Se debe habilitar el puerto I2C agregando los módulos del kernel *i2c-bcm2708* e *i2c-dev* en el archivo de configuración */etc/modules*.

- Instalar el paquete *i2c-tools* a través del gestor de paquetes de Debian.

```
apt-get install i2c-tools
```

- Es recomendable dar privilegios a los usuarios normales para acceder a los canales I2C. Esto se puede lograr creando una regla *udev* con la línea.

```
KERNEL=="i2c-[0-7]",MODE="0666"
```

- Para habilitar la máxima velocidad del canal I2C, se debe agregar la siguiente configuración al archivo */etc/modprobe.d/i2c.conf* para configurar el módulo del kernel en *fastmode*.

```
options i2c_bcm2708 baudrate=400000
```

- Clonar el repositorio git para obtener la última versión de RTIMULib

```
git clone https://github.com/richards-tech/RTIMULib.git
```

- En la carpeta donde fue clonado el repositorio, compilar la librería

```
make
```

- Finalizada la compilación, se puede realizar la instalación para el sistema completo

```
sudo make install
```

- Tras reiniciar el RPi, se puede utilizar la librería desde la línea de comandos. Dentro de la carpeta donde fue clonada la librería, se encuentra una subcarpeta llamada "*Python*", que entrega los enlaces de la librería compilada en C a Cython, para ser utilizada en Python.

- Posteriormente, se configura el RPi como punto de acceso WiFi. Para ello, se debe conectar el dispositivo USB WiFi e instalar y configurar los servicios *hostapd* y *udchpd*.

```
apt-get install hostapd udhcpd
```

- Finalmente, para mantener al dispositivo con el servidor funcionando y ser manejado como un servicio del sistema, se creó un demonio que inicia el servidor y lo reinicia en caso de fallas.

3.3.6. Software del dispositivo

Para probar los algoritmos de fusión y verificar las velocidades de transferencia de información, se desarrolló un software que fuera capaz de graficar en tiempo real datos recibidos a través de un puerto serial y registrarla junto con el tiempo de adquisición (*time stamp*).

En busca de portabilidad y facilidad de desarrollo, se desarrolló en Python una aplicación capaz de cumplir estos requisitos. En primera instancia, se revisaron las diferentes alternativas mediante

librerías o *frameworks* para graficar los datos y desarrollar la interfaz de usuario. El *framework* Qt4 permite diseñar en forma rápida interfaces de usuario que son multiplataforma. Adicionalmente, entrega un API amplia para interactuar con los diferentes elementos de la interfaz. Una vez seleccionado Qt4 como gestor de la interfaz, se buscaron alternativas para graficar en tiempo real. Entre ellas:

- Matplotlib (Hunter, 2007) es la librería más utilizada para hacer gráficos, tanto por su integración en entornos científicos, como su similitud con las funciones de MATLAB. Sin embargo, la librería no es apropiada para graficar en tiempo real.
- PyQwt (Vermeulen & Schiele, 2010) es una librería liviana que permite realizar gráficos en tiempo real. Sin embargo, el proyecto se encuentra sin soporte ni mantenedores.
- Chaco (Enthought, Inc., 2013) es un proyecto que permite crear gráficos de calidad y en tiempo real, sin embargo, no es sencillo integrarlo a una interfaz ni plataforma determinada.
- PyQtGraph (Campagnola, 2014) es una librería que puede ser implementada para procesamiento en tiempo real, crea gráficos de calidad y está desarrollada para compatibilizar con Qt4 y, por sobre todo, con QtDesigner, simplificando la puesta en marcha del gráfico. Está siendo activamente desarrollado, su documentación no está completa, pero posee una gran cantidad de ejemplos. Más aún, funciona en diferentes sistemas operativos y puede ser embebido para crear archivos binarios específicos para cada sistema.

Por las razones expuestas, se utilizó la librería PyQtGraph para realizar los gráficos en tiempo real sobre una interfaz de usuario diseñada en Qt4. Para lograr una optimización del procesamiento de datos, se implementaron y compararon los desempeños de *threads* y *QtThreads*. Python posee cierta limitación para manejar correctamente *threads*, y esto genera que las aplicaciones creadas en Python no utilizan todos los recursos de procesamiento disponibles en forma eficiente. Para solucionarlo, se utilizó la librería *multiprocessing*, que implementa de mejor forma los *threads* a través de procesos de bajo nivel, logrando graficar y registrar sin problemas señales con frecuencias de muestreo de hasta 2 KHz, por medio de un puerto serial. La aplicación es capaz de procesar los datos y realizar la fusión de sensores para obtener postura, junto con calcular compensación de la gravedad para obtener aceleraciones lineales para estimar posición. Además, el usuario puede elegir exportar los datos a un archivo CSV. La implementación espera datos en formato CSV a través del puerto serial y, posteriormente, grafica los datos obtenidos, mientras que la librería PyQtGraph provee acciones con el mouse para auto escalado o realizar la transformada de Fourier en tiempo real, sin necesidad de implementar la funcionalidad. En el anexo 9.3 se encuentra la documentación de la aplicación generada. La Figura 10 muestra la interfaz gráfica y la adquisición de datos.

La aplicación se utilizó para probar otros dispositivos con éxito, e incluso, en otros proyectos, por lo que se distribuye libremente a través del repositorio https://github.com/ssepulveda/RTGraph (Sepúlveda, 2014).

Posteriormente, tras visualizar los gráficos en tiempo real, se encontró la necesidad de visualizar los datos representados en 3D. Nuevamente, reutilizando los procesos de captura ya creados como clases de Python, se buscó utilizar dos herramientas:

- Blender, que puede ejecutar scripts de Python para realizar acciones en forma programática sobre modelos 3D. Sin embargo, los tiempos de respuesta no resultaron óptimos, además de generar problemas de compatibilidad por utilizar Python 3.2 como lenguaje base.
- OpenGL es una de las bases para cualquier aplicación en 3D. Sin embargo, la curva de aprendizaje sobre cómo se debe ejecutar los movimientos, y las acciones sobre las matrices de estado pueden volver la programación confusa. Utilizando herramientas como GLUT,

fue posible crear y unir cubos que representaban los segmentos del cuerpo, moviéndose en tiempo real.

Se desarrolló, entonces, un agregado al programa para graficar en 2D y poder cambiar a una interfaz 3D al mismo tiempo que se mantenían los gráficos 2D en tiempo real. El uso de *multiprocessing* es fundamental en esta tarea, ya que de otra forma las interfaces de usuario y el procesamiento se volvían inestables. En este aspecto, la comunicación de los datos entre diferentes procesos no es trivial, y para poder mantenerlos accesibles desde diferentes procesos se implementaron *queues*, que son también proporcionados por la librería *multiprocessing*.
Los resultados obtenidos permiten observar cómo se comportan las variables en diferentes instancias, y permite modelar el sistema segmento articulación, para visualizar como se manifiesta el movimiento. La Figura 11 muestra la ventana de adquisición con algunos movimientos.

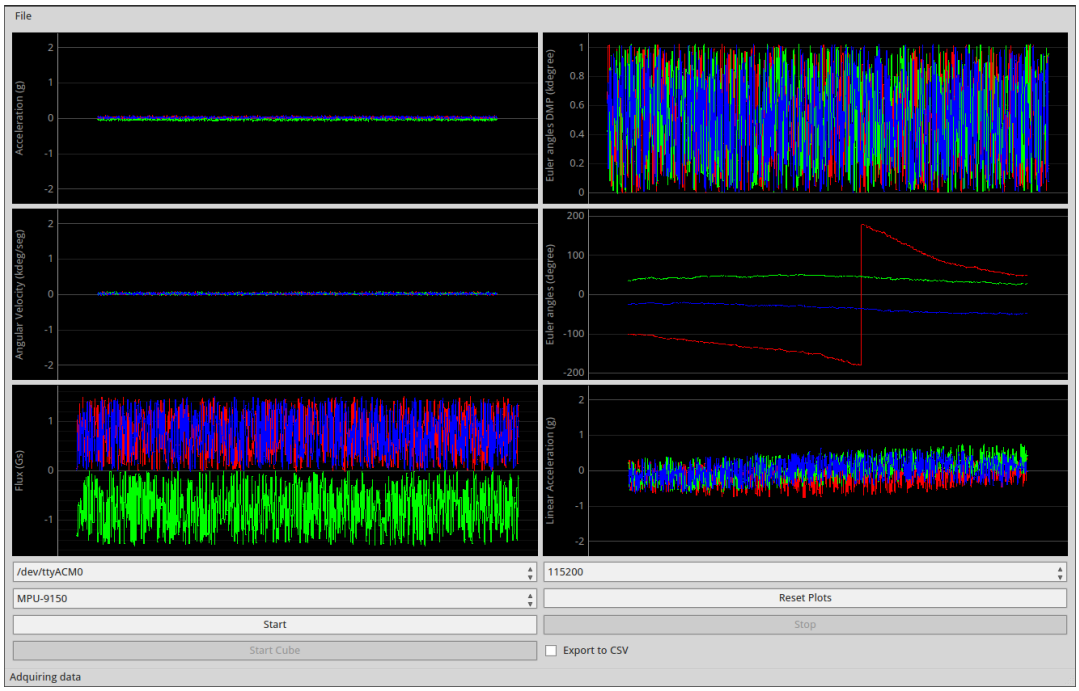


Figura 10.
Interfaz gráfica de RTGraph modificada para graficar aceleración, velocidad angular, flujo magnético, fusión de sensores y aceleración lineal.

Las aplicaciones desarrolladas hasta el momento permitían monitorear los resultados desde el mismo RPi u otro computador. Sin embargo, al momento de implementar una solución accesible a través de red en el RPi, y con el requisito de ser agnóstico del sistema operativo o dispositivo que accede, las aplicaciones propuestas necesitaban de teclado y monitores para operar.
Siguiendo lo implementado en la plataforma de desarrollo Arduino Yún, se porta el *framework* Autobahn para hacer manejar los protocolos WebSocket y WAMP, a través de los cuales se pueda interactuar a bajo nivel con la plataforma RPi y los sensores.

Como las librerías siguen basadas en Python, el proceso de portar es bastante simple. Sólo es necesario reemplazar el sistema WAMP para no atender a llamadas seriales, si no que manejar directamente las llamadas a los sensores. Utilizando los procesos creados con *mutiprocessing* para adquirir y procesar los datos, se crean procesos que permiten manejar los datos y la publicación a través del protocolo WebSocket de la información. El anexo 9.4 presenta la documentación del servidor implementado en Python para Autobahn.

Figura 11.
Modelo brazo y antebrazo creado en OpenGL. Las esferas simulan las articulaciones de hombro y codo.

Resuelto el servidor y la comunicación con los datos, sólo queda entregar una interfaz de usuario amigable. El framework Bootstrap entrega una poderosa herramienta que permite crear páginas web "responsivas" con facilidad. Basado en JavaScript y utilizando las características de HTML5 y CSS3, es posible crear páginas con animaciones y que pueden crear diálogos como las interfaces de usuario común, sin tener que volver a recargar toda la página. De esta forma, la transferencia de datos se disminuye y se mantiene un *look and feel* amigable.

Una de las principales características que ofrece está plataforma es la posibilidad de auto adaptar el tamaño de la página web y la configuración y organización de sus elementos al tamaño de la pantalla. La Figura 12 muestra la visualización de la página web con un tamaño de ventana que permite visualizar todos los elementos, mientras que la Figura 13 muestra cómo se reordenan los elementos de la página a un tamaño más pequeño, junto con crear menús que permiten ahorrar espacio. Este requisito es elemental para lograr una visualización agnóstica tanto al sistema operativo como al dispositivo desde el cual se realiza la visualización.

La página web permite, desde cualquier navegador que soporte JavaScript (prácticamente todos los navegadores, incluidos aquellos de dispositivos móviles como *smartphones* y *tablets*) puede mostrar en tiempo real los datos de los sensores, realizando incluso gráficos. El procesamiento de la información se realiza por el navegador, descargando al servidor (en este caso, el RPi) de tener que actualizar la página web. El protocolo WAMP permite a través de RPC (*Remote Proccesudre Call*) notificar al cliente (el navegador web) que existe nueva información disponible, y este actualiza sólo el campo que ha sido modificado. Al iniciar la carga de la página, todos sus componentes son descargados al navegador, por tanto, toda la tarea de mostrar los resultados queda para el cliente, mientras que el servidor solo notifica los nuevos estados de la información.

Esta aproximación es crucial para soluciones pensadas en IoT, descargando al servidor del procesamiento de la página web y, así mismo, disminuyendo el ancho de banda y el consumo de datos de la página.

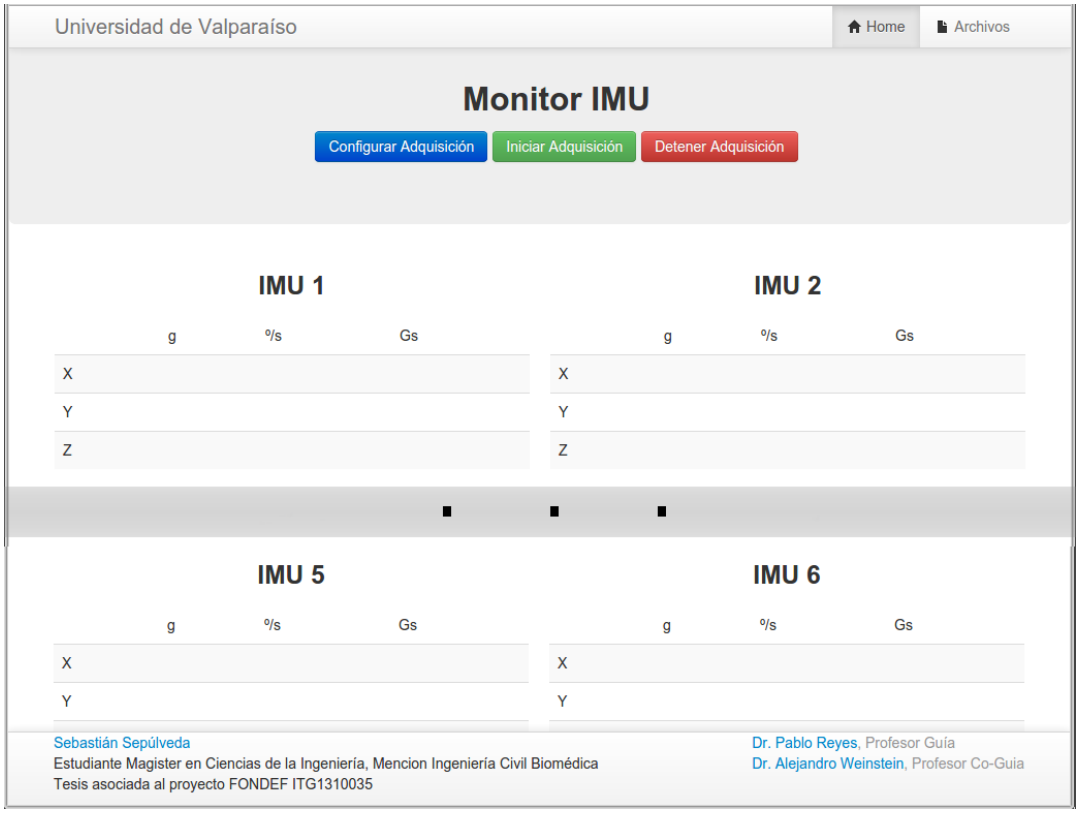


Figura 13.
Página web que entrega el dispositivo al conectarse en forma inalámbrica a la dirección configurada (http://192.168.240.1:8080) dentro de la red WiFi "IMU_MONITOR_UV".

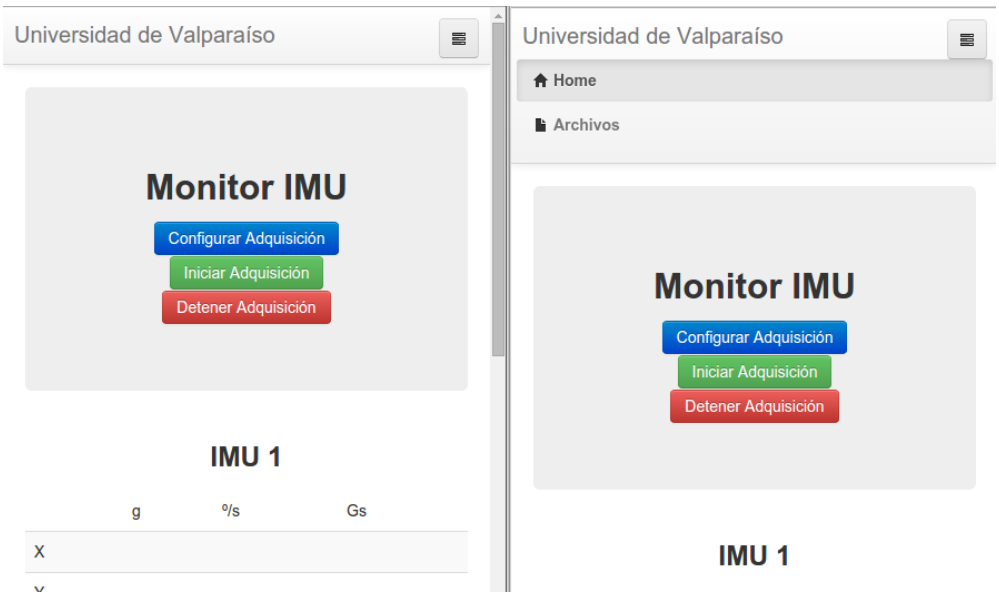


Figura 12.
Tamaño de página reducido para la misma página de la Figura 12. A la izquierda, la página con el menú sin abrir, a la derecha, la página con el menú desplegado.

4. Resultados

Los resultados de este trabajo se pueden separar en la misma secuencia que se utilizó para la implementación de la propuesta.

Respecto a la selección de sensores IMUs, se logró seleccionar el dispositivo más apropiado, tanto para el prototipo como para el desarrollo de una solución final. El sensor LSM9DS0 permite las frecuencias de muestreo más altas con todos sus sensores activados, la mayor resolución para todos sus sensores y el menor consumo, en un mismo encapsulado que el resto de la competencia.

Respecto al protocolo de comunicación, se prefirió sacrificar la máxima velocidad de transferencia de datos posible utilizando I2C en vez de SPI. Sin embargo, se logró disminuir la cantidad de cables necesarios para energizar y comunicarse con el sensor, parte de los requisitos para crear un dispositivo cómodo, manteniendo las características de *wearable*. La decisión del protocolo impacta enormemente en el desarrollo de Hardware, ya que el protocolo SPI necesita disponer de más salidas para activar cada uno de los sensores. En el momento de probar la plataforma con todos los sensores funcionando, no se encontró un cuello de botella en la información, ya que esta fue adquirida para los 5 dispositivos a 120Hz como máximo, para cada uno de los sensores IMU. Esta velocidad corresponde a la más baja entregada por el magnetómetro, por tanto, no se producen cuellos de botella por la utilización del protocolo I2C.

Respecto a la plataforma de desarrollo, la más apropiada para el desarrollo resultó ser la mezcla entre las capacidades embebidas de Arduino Yún y el poder de procesamiento de la arquitectura ARM de Nucleo F401RE. La plataforma Raspberry Pi entrega un control de alto nivel que puede interactuar con componentes de bajo nivel, simplificando el proceso de desarrollo de protocolos y software. Si bien la plataforma Raspberry Pi ya es una solución final, las pruebas realizadas descartan micro procesadores de 8 bits para desempeñar la tarea de manejar múltiples sensores.

Respecto al hardware, gracias a la plataforma seleccionada se mantuvo al mínimo la cantidad de hardware que desarrollar, volviéndose entonces más una tarea de cableado que de montaje de hardware. Adicionalmente, el componente que requería circuitería específica, el multiplexor I2C PCA9547 fue diseñado con el objetivo de ser compatible con el resto de las plataformas y a prueba de fallas en las conexiones de los sensores.

Respecto al firmware, la utilización de un kernel Linux entrega soluciones de alto nivel para el manejo de hilos y eventos. En una solución final, es poco probable contar con las capacidades para instalar un kernel completo de Linux, sin embargo, existen alternativas que permiten lograr un manejo similar de eventos y tareas, y están especialmente diseñadas para arquitecturas ARM, conocidos como sistemas RTOS (*Real Time Operative System*).

Respecto al software, las herramientas desarrolladas implementadas con el objetivo de ser multiplataforma y con una correcta optimización de los recursos permitieron portar con facilidad el software, desde diferentes sistemas operativos hasta diferentes arquitecturas (desde x64 y x86 en el computador hacia ARMv6 en el RPi). La forma modular de la implementación del programa junto con el manejo de las tareas en diferentes *threads* permite la utilización óptima de los recursos del dispositivo. La herramienta para visualizar los datos en tiempo real ha sido utilizada en otros proyectos, por lo tanto la solución es escalable a otras problemáticas. En vista de esto, y como aporte a la comunidad *open source*, la aplicación, la documentación y los requisitos de instalación están disponibles en https://github.com/ssepulveda/RTGraph (Sepúlveda, 2014).

En el contexto del desarrollo del prototipo, la metodología utilizada permitió determinar la mejor alternativa para desarrollar el prototipo, al mismo tiempo que predice la alternativa para el producto final. Los requisitos impuestos de portabilidad permitieron cambiar de forma sencilla entre las diferentes plataformas. La utilización de comunicación inalámbrica para el control y la transferencia de datos, junto con el almacenamiento local de los datos, entregan robustez al sistema, enfatiza la modularidad y provee el acceso con la menor cantidad de recursos adicionales para

controlar el sistema, siendo posible utilizarlos desde un computador, *Tablet* o *Smartphone*. El manejo a través de una página web resulta intuitivo y uniforme en las diferentes plataformas.

El prototipo fue probado en una aplicación real por el profesor Dr. Pablo Reyes en la Universidad de Manuela Beltrán, Bogotá, Colombia. Se adquirieron datos de los sensores inerciales y orientación, mientras se comparaba contra un *Gold-Standard* de MOCAP por medios ópticos. La plataforma fue utilizada para adquirir datos por un voluntario en Colombia. Mientras el profesor Dr. Pablo Reyes realizaba las pruebas, donde pudo enviar los datos por red en formato CSV. Cuando el dispositivo volvió a Chile, comprobamos que los datos aún estaban guardados en el dispositivo. La Figura 14 muestra la utilización del prototipo durante las pruebas. Se observa que, a pesar de utilizar ambos métodos de adquisición, el sistema es lo suficientemente modular y pequeño como para utilizarse en conjunto, es ajustable a cada segmento del cuerpo del sujeto de pruebas y no interfiere mayormente con el movimiento. Las pruebas incluyeron adquirir datos de extremidades superiores, incluyendo los 5 dedos de las manos, demostrando la modularidad que la propuesta del sistema ofrece, y que puede ser mejorada en un dispositivo final.

Finalmente, todos los códigos, esquemáticos y diseños creados se distribuyen en forma *open source* a través de un repositorio https://github.com/ssepulveda/ICB_TT, junto a la documentación para instalar y reproducir los resultados.

Figura 14.
Utilización del prototipo de adquisición de movimiento inercial en la Universidad de Manuela Beltrán, Bogotá, Colombia, supervisado por el profesor Dr. Pablo Reyes. En la imagen se observa el uso conjunto del sistema de captura propuesto y el Gold-Standard por medios ópticos.

5. Discusión

Los sensores IMUs utilizados cumplen con las características técnicas mínimas establecidas en diferentes estudios de biomecánica y soluciones comerciales.
La plataforma utilizada no genera cuellos de botella para la adquisición y procesamiento de información, en tanto que los protocolos de comunicación seleccionados son apropiados para la aplicación.

El hardware implementado es modular y compatible con diferentes plataformas, permitiendo el uso de diferentes arquitecturas con los mismos sensores, respondiendo a los requisitos de modularidad señalados en el diseño de la propuesta. La cantidad de sensores es detectada al momento de iniciar la adquisición, no siendo necesario realizar ninguna configuración especial con el dispositivo. Hasta 6 sensores se pueden adquirir simultáneamente con el prototipo propuesto, con una frecuencia de muestreo máxima de 120Hz, cumpliendo con los requisitos señalados el diseño de la propuesta, respecto a los requerimientos de la plataforma.

El software utilizado es multiplataforma, pudiendo ser ejecutado desde diferentes sistemas operativos con diferentes arquitecturas. Adicionalmente, la solución final implementada como punto de acceso WiFi permite ser agnóstico incluso del dispositivo utilizado, con una interfaz de usuario que se adapta al tamaño de la pantalla del cliente. A diferencia de las soluciones comerciales, que requieren software adicional o dispositivos dedicados, el mismo dispositivo se encarga tanto de procesar los datos como ofrecer una interfaz universal de configuración, control y adquisición de datos.

En vista de los resultados obtenidos, y considerando la exitosa prueba en terreno del prototipo mientras se realizaba comparaciones contra un *Gold Standard*, se concluye que se cumplieron los requisitos impuestos en el diseño de la propuesta, finalizando con una plataforma funcional y operativa.

Adicionalmente, los códigos de fuente y esquemáticos están disponibles en el repositorio https://github.com/ssepulveda/ICB_TT, para todo aquel que desee realizar trabajos o investigaciones con los mismos sensores. Esto promueve el desarrollo de nuevas investigaciones utilizando la misma plataforma. En el contexto del proyecto de tesis de Magister, el desarrollo del prototipo de esta plataforma permitirá adquirir datos de movimiento y electromiografía en forma conjunto, para así investigar como las mediciones de EMG podrían mejorar la fusión de sensores y la estimación de la postura y posición. La metodología de implementación y el énfasis en la modularidad y portabilidad permitirán agregar las características faltantes sin mayores inconvenientes.

6. Conclusiones

El trabajo realizado entrega el prototipo de una plataforma funcional de captura de movimiento, disponible en forma *open source* para que cualquiera la pueda utilizar, modificar y realizar capturas de movimiento.

Las contribuciones de este trabajo se enmarcan desde la prueba de diferentes plataformas de desarrollo en aplicaciones de alta demanda de procesamiento, junto con un conjunto de herramientas que permiten evaluar el desempeño, tanto en tiempo real como a través del análisis de registros el desempeño de las plataformas y los diferentes protocolos de comunicación.

Desde un punto de vista tecnológico, se realiza un aporte importante en la implementación de una plataforma con miras al *Internet of Things* (IoT), permitiendo crear aplicaciones realmente ambulatorias y agnósticas del dispositivo con el cual se controlan. La plataforma ofrece la escalabilidad para conectarse directamente a Internet, o compartir la conexión con otro dispositivo, enviando en tiempo real los datos a un servidor para su almacenamiento y control ambulatorio. En

investigaciones futuras, el manejo de la seguridad y privacidad de los datos es un tema pendiente e importante que se debe asegurar en un dispositivo de utilización clínica.

Esta plataforma será parte esencial del proyecto de tesis del programa de Magister en Ciencias de la Ingeniería, mención Ingeniería Biomédica, ya que permite la captura y registro de los datos en condiciones similares a las comerciales, con un costo mucho menor y personalizado a los requerimientos. Para el caso específico, se agregará un módulo para medir EMG en conjunto con la captura del movimiento.

Durante, la tesis de magister, la plataforma será comparada contra el *Gold-Standard*, lo que pondría a disposición de la comunidad educativa y de *open source* una plataforma realmente valiosa, que podría estar validada respecto a un *Gold-Standard*, entregando un gran valor agregado a la plataforma.

7. Referencias Bibliográficas

Ando, T., Watanabe, M., & Fujie, M. G. (2009). Extraction of voluntary movement for an EMG controlled exoskeltal robot of tremor patients (pp. 120–123). Presented at the 4th International IEEE/EMBS Conference on Neural Engineering, 2009. NER '09. doi:10.1109/NER.2009.5109249

Anliker, U., Beutel, J., Dyer, M., Enzler, R., Lukowicz, P., Thiele, L., & Troster, G. (2004). A systematic approach to the design of distributed wearable systems. *IEEE Transactions on Computers*, *53*(8), 1017–1033. doi:10.1109/TC.2004.36

Aziz, O., Park, E. J., Mori, G., & Robinovitch, S. N. (2014). Distinguishing the causes of falls in humans using an array of wearable tri-axial accelerometers. *Gait & Posture*, *39*(1), 506–512. doi:10.1016/j.gaitpost.2013.08.034

Bagalà, F., Klenk, J., Chiari, L., Cappello, A., & Becker, C. (2012). Accelerometer-based algorithms on real falls database. *Gait & Posture*, *35, Supplement 1*, S28. doi:10.1016/j.gaitpost.2011.09.060

Barnett, R. (2014). RTIMULib V3 – 9-dof IMU fusion with accelerometer and enhanced magnetometer calibration. Retrieved from http://richardstechnotes.wordpress.com/2014/10/13/rtimulib-v3-9-dof-imu-fusion-with-accelerometer-and-enhanced-magnetometer-calibration/ on October, 2014

Bishop, G., & Welch, G. (2001). An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, *8*(27599-23175), 41.

Bouarfa, L., Atallah, L., Kwasnicki, R. M., Pettitt, C., Frost, G., & Yang, G.-Z. (2014). Predicting Free-Living Energy Expenditure Using a Miniaturized Ear-Worn Sensor: An Evaluation Against Doubly Labeled Water. *IEEE Transactions on Biomedical Engineering*, *61*(2), 566–575. doi:10.1109/TBME.2013.2284069

Bourke, A. K., & Lyons, G. M. (2008). A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. *Medical Engineering & Physics*, *30*(1), 84–90. doi:10.1016/j.medengphy.2006.12.001

Bourke, A. K., van de Ven, P., Gamble, M., O'Connor, R., Murphy, K., Bogan, E., … Nelson, J. (2010). Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities. *Journal of Biomechanics*, *43*(15), 3051–3057. doi:10.1016/j.jbiomech.2010.07.005

Campagnola, L. (2014). Scientific Graphics and GUI Library for Python. Retrieved from http://www.pyqtgraph.org/ on October, 2014

Chen, K. Y., & Bassett, D. R. (2005). The technology of accelerometry-based activity monitors: current and future. *Medicine and Science in Sports and Exercise*, *37*(11 Suppl), S490–500.

Constantine, L. L., & Lockwood, L. A. D. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design* (1 edition.). Reading, Mass: Addison-Wesley Professional.

Ding, Q. C., Xiong, A. B., Zhao, X. G., & Han, J. D. (2011). A novel EMG-driven state space model for the estimation of continuous joint movements (pp. 2891–2897). Presented at the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC). doi:10.1109/ICSMC.2011.6084104

Dong, B., Biswas, S., Montoye, A., & Pfeiffer, K. (2013). Comparing metabolic energy expenditure estimation using wearable multi-sensor network and single accelerometer (pp. 2866–2869). Presented at the 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). doi:10.1109/EMBC.2013.6610138

Enthought, Inc. (2013). 2-Dimensional Plotting. Retrieved from http://code.enthought.com/projects/chaco/ on October, 2014

Field, M., Stirling, D., Naghdy, F., & Pan, Z. (2009). Motion capture in robotics review (pp. 1697–1702). Presented at the IEEE International Conference on Control and Automation, 2009. ICCA 2009. doi:10.1109/ICCA.2009.5410185

Gao, Z., Niu, X., & Guo, M. (2002). Quaternion-Based Kalman Filter for Micro-machined Strapdown Attitude Heading Reference System. *Chinese Journal of Aeronautics*, *15*(3), 171–175. doi:10.1016/S1000-9361(11)60149-0

Geng, Y., Yu, L., You, M., & Li, G. (2010). A Pilot Study of EMG Pattern Based Classification of Arm Functional Movements (Vol. 3, pp. 317–320). Presented at the 2010 Second WRI Global Congress on Intelligent Systems (GCIS). doi:10.1109/GCIS.2010.125

Goyal, P., Ribeiro, V. J., Saran, H., & Kumar, A. (2011). Strap-down Pedestrian Dead-Reckoning system (pp. 1–7). Presented at the 2011 International Conference on Indoor Positioning and Indoor Navigation (IPIN). doi:10.1109/IPIN.2011.6071935

Grewal, M. S., & Andrews, A. P. (2010). Applications of Kalman Filtering in Aerospace 1960 to the Present [Historical Perspectives]. *IEEE Control Systems*, *30*(3), 69–78. doi:10.1109/MCS.2010.936465

Haapalainen, E., Laurinen, P., Roning, J., & Kinnunen, H. (2008). Estimation of Exercise Energy Expenditure Using a Wrist-Worn Accelerometer: A Linear Mixed Model Approach with Fixed-Effect Variable Selection (pp. 796–801). Presented at the Seventh International Conference on Machine Learning and Applications, 2008. ICMLA '08. doi:10.1109/ICMLA.2008.131

Hausdorff, J. M., Rios, D. A., & Edelberg, H. K. (2001). Gait variability and fall risk in community-living older adults: A 1-year prospective study. *Archives of Physical Medicine and Rehabilitation*, *82*(8), 1050–1056. doi:10.1053/apmr.2001.24893

Huang, H.-P., & Hsu, L.-P. (2005). Development of a wearable biomedical health-care system (pp. 1760–1765). Presented at the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005). doi:10.1109/IROS.2005.1545390

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, *9*(3), 90–95.

Khoder, W., Fassinut-Mombot, B., & Benjelloun, M. (2008a). Inertial navigation attitude velocity and position algorithms using quaternion Scaled Unscented Kalman filtering (pp. 1754–1759). Presented at the 34th Annual Conference of IEEE Industrial Electronics, 2008. IECON 2008. doi:10.1109/IECON.2008.4758219

Khoder, W., Fassinut-Mombot, B., & Benjelloun, M. (2008b). Quaternion Unscented Kalman Filtering for integrated Inertial Navigation and GPS (pp. 1–8). Presented at the 2008 11th International Conference on Information Fusion.

Lee, I.-M., & Shiroma, E. J. (2014). Using accelerometers to measure physical activity in large-scale epidemiological studies: issues and challenges. *British Journal of Sports Medicine*, *48*(3), 197–201. doi:10.1136/bjsports-2013-093154

Lowe, S. A., & OLaighin, G. (2014). Monitoring human health behaviour in one's living environment: A technological review. *Medical Engineering & Physics*, *36*(2), 147–168. doi:10.1016/j.medengphy.2013.11.010

Lustrek, M., Cvetkovic, B., & Kozina, S. (2012). Energy expenditure estimation with wearable accelerometers (pp. 5–8). Presented at the 2012 IEEE International Symposium on Circuits and Systems (ISCAS). doi:10.1109/ISCAS.2012.6271906

Madgwick, S. (2013). Gait tracking with x-IMU | x-io Technologies. Retrieved from http://www.x-io.co.uk/gait-tracking-with-x-imu/ on October, 2014

Madgwick, S. O. H., Harrison, A. J. L., & Vaidyanathan, R. (2011). Estimation of IMU and MARG orientation using a gradient descent algorithm (pp. 1–7). Presented at the 2011 IEEE International Conference on Rehabilitation Robotics (ICORR). doi:10.1109/ICORR.2011.5975346

Misgeld, B. J. E., Ruschen, D., Kim, S., & Leonhardt, S. (2013). Body sensor network-based strapdown orientation estimation: Application to human locomotion (pp. 1–6). Presented at the 2013 IEEE International Conference on Rehabilitation Robotics (ICORR). doi:10.1109/ICORR.2013.6650480

Mündermann, L., Corazza, S., & Andriacchi, T. P. (2006). The evolution of methods for the capture of human movement leading to markerless motion capture for biomechanical applications. *Journal of NeuroEngineering and Rehabilitation*, *3*(1), 6. doi:10.1186/1743-0003-3-6

Panagiota, A., Layal, S., & Stefan, H. (2012). Assessment of Human Gait Speed and Energy Expenditure Using a Single Triaxial Accelerometer (pp. 184–188). Presented at the 2012 Ninth International Conference on Wearable and Implantable Body Sensor Networks (BSN). doi:10.1109/BSN.2012.7

Park, Y., Kawahara, K., & Uchida, M. (2008). Characteristic extraction of EMG with the apparent movement (pp. 2723–2726). Presented at the SICE Annual Conference, 2008. doi:10.1109/SICE.2008.4655128

Pham, H. V., Tran, K. D., Thang, C., Cooper, E. W., & Kamei, K. (2011). Human Reasoning Awareness Quantified by Self-Organizing Map Using Collaborative Decision Making for Multiple Investment Models (pp. 101–108). Presented at the 2011 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS). doi:10.1109/CISIS.2011.24

Phinyomark, A., Hirunviriya, S., Limsakul, C., & Phukpattaranont, P. (2010). Evaluation of EMG feature extraction for hand movement recognition based on Euclidean distance and standard deviation (pp. 856–860). Presented at the 2010 International Conference on Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON).

Phuong, N. H. Q., Kang, H.-J., Suh, Y.-S., & Ro, Y.-S. (2009). A DCM Based Orientation Estimation Algorithm with an Inertial Measurement Unit and a Magnetic Compass. *Journal of Universal Computer Science*, *15*(4), 859–876.

Roetenberg, D. (2006). *Inertial and Magnetic Sensing of Human Motion*. University of Twente [Host].

Romanovas, M., Goridko, V., Al-Jawad, A., Schwaab, M., Traechtler, M., Klingbeil, L., & Manoli, Y. (2012). A study on indoor pedestrian localization algorithms with foot-mounted sensors (pp. 1–10). Presented at the 2012 International Conference on Indoor Positioning and Indoor Navigation (IPIN). doi:10.1109/IPIN.2012.6418886

Sabatini, A. M. (2006). Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing. *IEEE Transactions on Biomedical Engineering*, *53*(7), 1346–1356. doi:10.1109/TBME.2006.875664

Sepúlveda, S. (2012, December). *Monitorización ambulatoria de coordenadas de posición utilizando acelerómetros, giroscopios y magnetómetros* (Seminario de Investigación). Universidad de Valparaíso, Valparáiso.

Sepúlveda, S. (2014). RTGraph: First release. doi:10.5281/zenodo.12789

Shin, S., Jeong, S., Shin, S., Cho, D., & Chung, S. (2013). Developing a device using accelerometers and EMG for hand movement recognition (pp. 398–402). Presented at the 2013 6th International Conference on Biomedical Engineering and Informatics (BMEI). doi:10.1109/BMEI.2013.6746971

Sommerville, I. (2005). *Ingenieria del software*. Pearson Educacion.

Song, Z., Wang, Z., & Guo, S. (2013a). SEMG-based continuous posture recognition of elbow flexion and extension in sagittal plane (pp. 920–925). Presented at the 2013 IEEE International Conference on Mechatronics and Automation (ICMA). doi:10.1109/ICMA.2013.6618038

Song, Z., Wang, Z., & Guo, S. (2013b). Semg-based posture recognition of elbow flexion and extension (pp. 448–452). Presented at the 2013 ICME International Conference on Complex Medical Engineering (CME). doi:10.1109/ICCME.2013.6548288

Suh, Y.-S. (2010). Orientation Estimation Using a Quaternion-Based Indirect Kalman Filter With Adaptive Estimation of External Acceleration. *IEEE Transactions on Instrumentation and Measurement*, *59*(12), 3296–3305. doi:10.1109/TIM.2010.2047157

Tavendo. (2014). Autobahn. Retrieved from http://autobahn.ws/ on October, 2014

Terrill, P. I., Leong, M., Barton, K., Freakley, C., Downey, C., Vanniekerk, M., … Douglas, J. (2013). Measuring leg movements during sleep using accelerometry: Comparison with EMG and piezo-electric scored events (pp. 6862–6865). Presented at the 2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). doi:10.1109/EMBC.2013.6611134

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. Cambridge, Mass: Mit.

To, G., & Mahfouz, M. R. (2012). Design of wireless inertial trackers for human joint motion analysis (pp. 49–52). Presented at the 2012 IEEE Topical Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS). doi:10.1109/BioWireless.2012.6172737

Tsai, M.-K. (2014). Automatically determining accidental falls in field surveying: A case study of integrating accelerometer determination and image recognition. *Safety Science*, *66*, 19–26. doi:10.1016/j.ssci.2014.01.012

Varesano, F. (2011, April). *Using Arduino for Tangible Human Computer Interaction* (Laurea Magistrale in Metodologie e Sistemi informatici). Universita degli Studi di Torino, Italy. Retrieved from http://www.varesano.net/files/MoS_thesis/thesis.pdf on October, 2014

Vermeulen, G., & Schiele, J. (2010). PyQwt plots data with Numerical Python and PyQt. Retrieved from http://pyqwt.sourceforge.net/ on October, 2014

Weinstein, A. J., & Moore, K. L. (2010). Pose estimation of Ackerman steering vehicles for outdoors autonomous navigation. In *2010 IEEE International Conference on Industrial Technology (ICIT)* (pp. 579–584). doi:10.1109/ICIT.2010.5472738

Wixted, A. J., Thiel, D. V., Hahn, A. G., Gore, C. J., Pyne, D. B., & James, D. A. (2007). Measurement of Energy Expenditure in Elite Athletes Using MEMS-Based Triaxial Accelerometers. *IEEE Sensors Journal*, 7(4), 481–488. doi:10.1109/JSEN.2007.891947

Wolf, M. T., Assad, C., Stoica, A., You, K., Jethani, H., Vernacchia, M. T., … Iwashita, Y. (2013). Decoding static and dynamic arm and hand gestures from the JPL BioSleeve (pp. 1–9). Presented at the 2013 IEEE Aerospace Conference. doi:10.1109/AERO.2013.6497171

Wolf, M. T., Assad, C., Vernacchia, M. T., Fromm, J., & Jethani, H. L. (2013). Gesture-based robot control with variable autonomy from the JPL BioSleeve (pp. 1160–1165). Presented at the 2013 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/ICRA.2013.6630718

Xie, L., Yang, P., Yang, T., & Li, M. (2012). Dual-EKF-Based Real-Time Celestial Navigation for Lunar Rover. *Mathematical Problems in Engineering*, *2012*. doi:10.1155/2012/578719

Xiong, A., Chen, Y., Zhao, X., Han, J., & Liu, G. (2011). A novel HCI based on EMG and IMU (pp. 2653–2657). Presented at the 2011 IEEE International Conference on Robotics and Biomimetics (ROBIO). doi:10.1109/ROBIO.2011.6181705

Xiong, A., Lin, G., Zhao, X., Han, J., & Liu, G. (2012). Feasibility of EMG-based ANN controller for a real-time virtual reality simulation (pp. 2699–2704). Presented at the IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society. doi:10.1109/IECON.2012.6389150

Xsens. (2014). Products. Retrieved from http://www.xsens.com/products/ on October, 2014

Yi-fu, J., Bo, X., & Jun, W. (2013). SINS in-motion alignment and position determination for land-vehicle based on quaternion Kalman filter (pp. 5083–5088). Presented at the Control Conference (CCC), 2013 32nd Chinese.

Zheng, H., Black, N. D., & Harris, D. N. D. (2005). Position-sensing technologies for movement analysis in stroke rehabilitation. *Medical and Biological Engineering and Computing*, *43*(4), 413–420. doi:10.1007/BF02344720

8. Glosario

D

DOF: Grados de Libertad; Define la cantidad de movimientos que puede realizar un objeto.

E

EMG: Acrónimo para Electromiograma, 11

G

Gold Standard: Denominación para un dispostivo o aparato que sirve como referencia o parámetro de comparación.

H

Holter: Holter (dispositivo); registro de señales cardiacas en forma ambulatoria.

I

I2C: Inter-Integrated Circuit; Protocolo de comunicación serial punro-multipunto.
IMU: Unidad de medición Inercial; dispositivo que combina sensores inerciales para medir el movimiento de un objeto, generalmente compuestos por acelerómetros y giroscopios.
IoT: Internet of Things.

M

MEMS: Sistemas Microelectromecánicos; tecnología de miniaturización que permite reducir el tamaño de dispositivos mecánicos.
Motion Capture: (Captura de movimiento) Acrónimo para denominar a un sistema de captura de movimiento.

R

RPi: Acronimo para Raspberry Pi.

S

SPI: Serial Peripheral Interface.

U

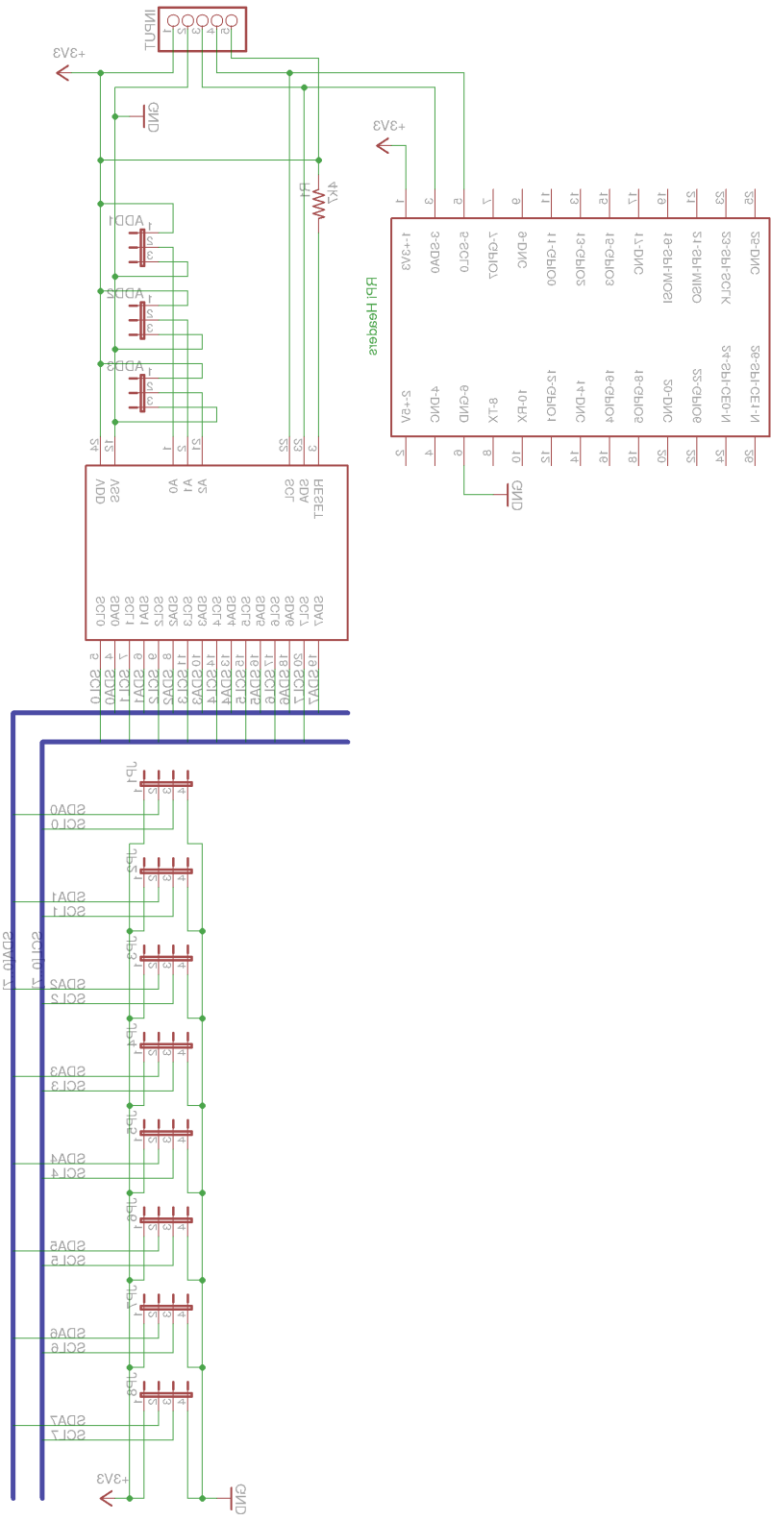
UART: Universal asynchronous receiver/transmitter.

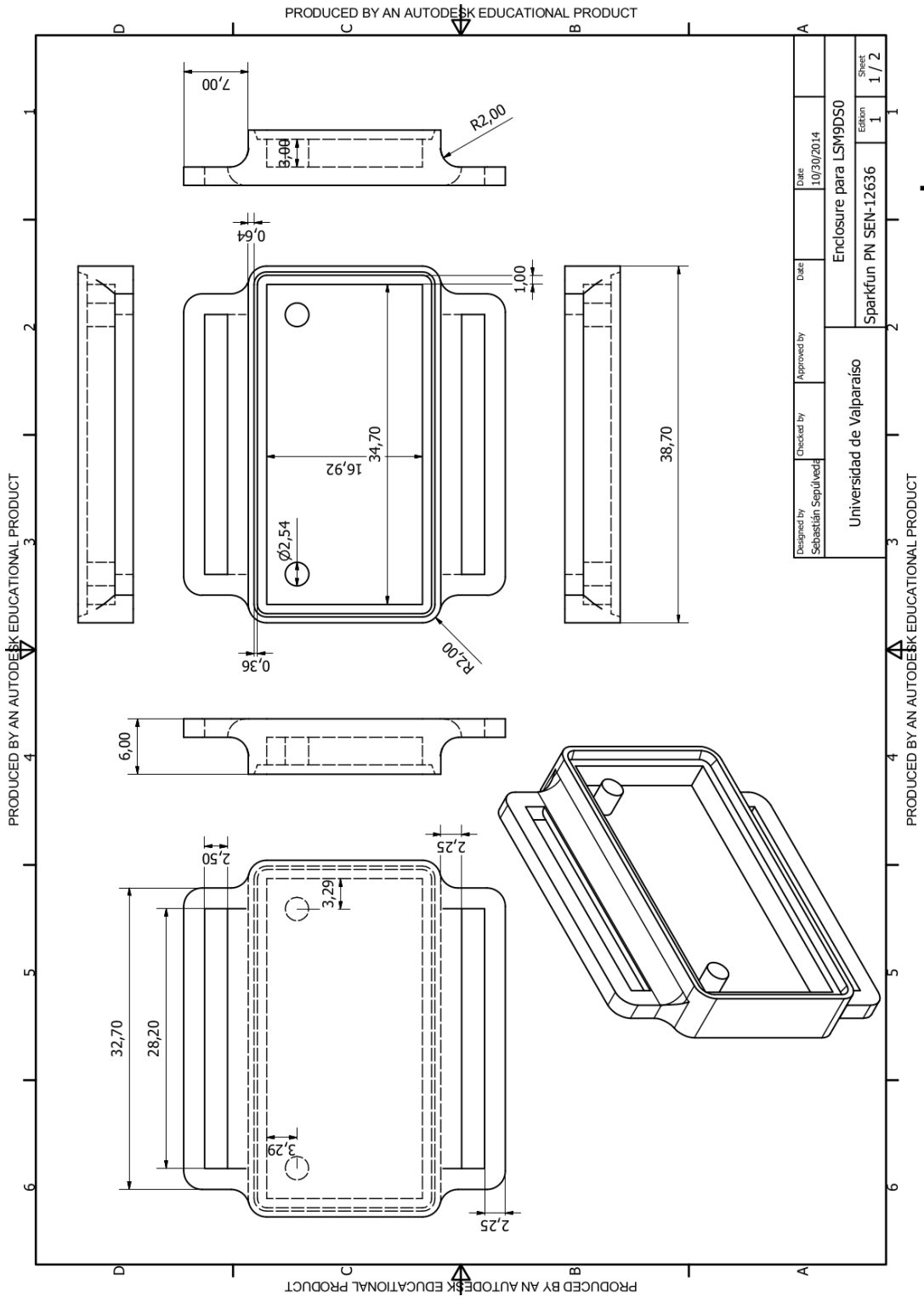
W

wearable: dispositivos electrónicos que se pueden utilizar como ropa o como accesorios de ella.

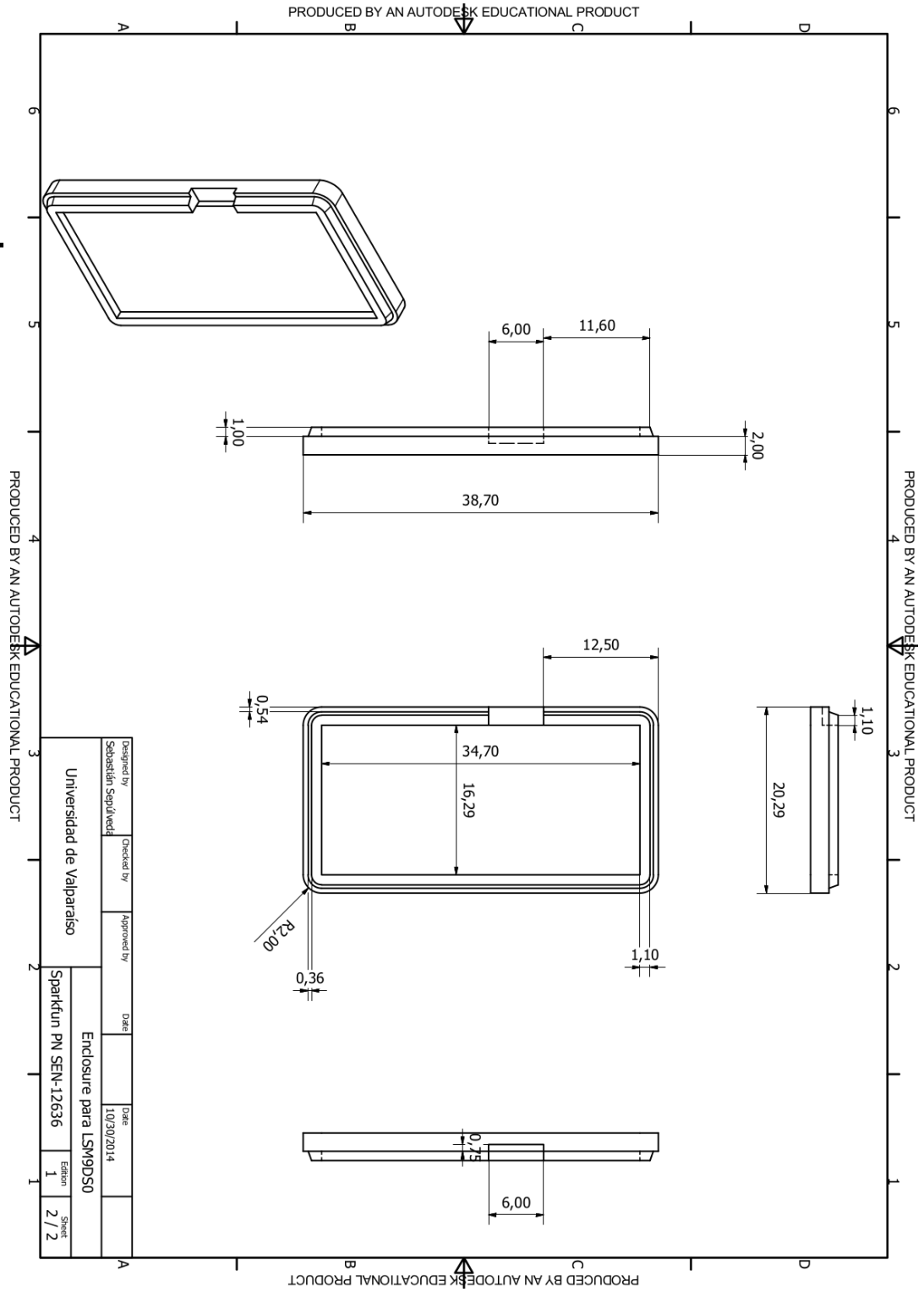
9. Anexo

9.1. Esquemático de multiplexor I2C PAC9547




PRODUCED BY AN AUTODESK EDUCATIONAL PRODUCT
7,00
R2,00
3,00
0,64
1,00
16,92
34,70
38,70
Ø2,54
0,36
R2,00
6,00
2,50
2,25
3,29
32,70
28,20
3,29
2,25
Designed by
Sebastián Sepúlveda
Checked by
Approved by
Date
Date
10/30/2014
Enclosure para LSM9DS0
Sparkfun PN SEN-12636
Edition
1
Sheet
1 / 2
Universidad de Valparaíso

Captura de movimiento ambulatorio



1 Class Documentation

1.1 activityDetection.ActivityDetection Class Reference

Estimates the energy consumption (Kcal) by second.

Public Member Functions

- def __init__
  Constructor for Activity Detection.
- def abs_list
  Take the absolute value of a list.
- def updateActivity
  Update activity detection algorithm with last second data.
- def getEnergyConsumption
  Getter for the last second energy consuption.
- def getEE
  Getter for the EE value.
- def getBMR
  Getter for the BMR value.

Public Attributes

- **BMR**
- **fs**
- **EE**

Static Public Attributes

- int BMR = .023842593
  Default value for BMR.
- int EE = 0
  Predicted consumed energy by activity (EE) in a second.
- int **fs** = .02

1.1.1 Detailed Description

Estimates the energy consumption (Kcal) by second.

Using an accelerometer and aditional data from the user calculates:

- Index of activity (IAA) in a second
- Predicted consumed energy by activity (EE) in a second
- Predicted consumed energy by basal metabolism (BMR) in a second

Author

Dr. Pablo Reyes
Sebastian Sepulveda

Todo Validate algorithms

1.1.2 Constructor & Destructor Documentation

def activityDetection.ActivityDetection.__init__ (*self, kg =* `70, cm =` `170, years =` `20, fs =` `.02`)

Constructor for Activity Detection.

Initializes calculations for the user to determinate the BMR

Parameters

self	The object pointer
kg	Weight of the user, in Kg
cm	Height of the user, in cm
years	Years of the user
fs	Time between samples, in seconds

1.1.3 Member Function Documentation

def activityDetection.ActivityDetection.abs_list (*self, _list*)

Take the absolute value of a list.

In Python, the `abs` function can't return the absolute values from a list. This function is a workaround to this limitation.

Parameters

self	The object pointer
_list	The original list to take the absolutes values

Returns

The absolute values in the original list

def activityDetection.ActivityDetection.getBMR (*self*)

Getter for the BMR value.

Gets the calculated BMR for the user

Parameters

self	The object pointer

Returns

Value of the BMR (per day)

def activityDetection.ActivityDetection.getEE (*self*)

Getter for the EE value.

Gets the last EE result from the last # updateActivity call

Parameters

self	The object pointer

Returns

Last second EE

def activityDetection.ActivityDetection.getEnergyConsumption (*self*)

Getter for the last second energy consuption.

Gets the last energy consumption result from the last updateActivity call

Parameters

self	The object pointer

Returns

Last second energy consumption, in Calories

def activityDetection.ActivityDetection.updateActivity (*self, x, y, z*)

Update activity detection algorithm with last second data.

The function updates the IAA based on the provided accelerations and calculates the energy consuptiom (EE). Updates the data for the algoritm

Parameters

self	The object pointer
x	list with the last second acceleration data in the X axis
y	list with the last second acceleration data in the Y axis
z	list with the last second acceleration data in the Z axis

Todo verify normalization of the IAA

1.1.4 Member Data Documentation

int activityDetection.ActivityDetection.BMR = .023842593 `[static]`

Default value for BMR.

Default BMR calculated at 2060 Kcal (normal person) divided in the seconds of one day (86400 seconds)

int activityDetection.ActivityDetection.EE = 0 `[static]`

Predicted consumed energy by activity (EE) in a second.

Initalized at 0

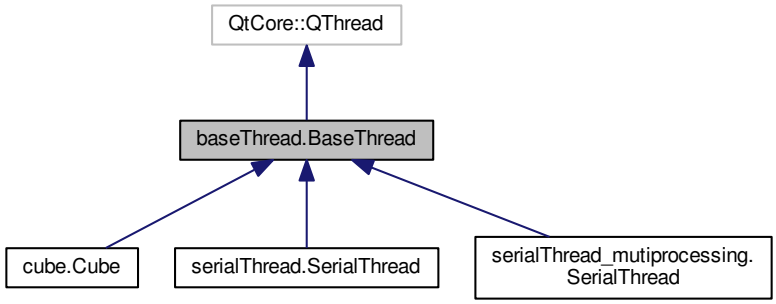
The documentation for this class was generated from the following file:

- Graph/activityDetection.py

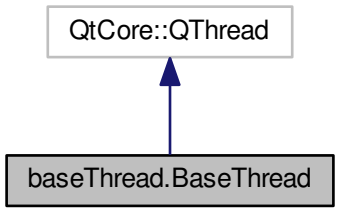
1.2 baseThread.BaseThread Class Reference

Defines basic methods for dealing with Qt Threads and all Threads.

Inheritance diagram for baseThread.BaseThread:



Collaboration diagram for baseThread.BaseThread:



Public Member Functions

- def __init__

  Constructor.
- def __del__

  Deconstructor.

Public Attributes

- exiting

  Notifies all threads to stop.

1.2.1 Detailed Description

Defines basic methods for dealing with Qt Threads and all Threads.

Implement handles for the events of the Qt Threads and UI elements (main thread and UI thread).

1.2.2 Constructor & Destructor Documentation

def baseThread.BaseThread.__init__ (*self*)

Constructor.

Initializes threads

Parameters

self	The objetc pointer

def baseThread.BaseThread.__del__ (*self*)

Deconstructor.

Closes and exits from threads

Parameters

self	The objetc pointer

The documentation for this class was generated from the following file:

- Graph/baseThread.py

1.3 csvExport.CSVExport Class Reference

Export data in CSV format.

Public Member Functions

- def __init__
  Constructor.
- def csvWrite
  Writes a new line of data.

Public Attributes

- **CSV**
- **t0**

1.3.1 Detailed Description

Export data in CSV format.

Export data using date and time as name for the file, avoiding overwriting data in differents adquisitions.

It also registers in the first columnn the elapsed since the first adquisition

Note

The data is stored in a "data" folder (muest be created first)

1.3.2 Constructor & Destructor Documentation

def csvExport.CSVExport.__init__ (*self*)

Constructor.

Gets time to create the filename and instanciate the start of the adquisition

Note

The filename format is Year-Moth-Day_Hours-minutes-seconds.csv

1.3.3 Member Function Documentation

def csvExport.CSVExport.csvWrite (*self, txt*)

Writes a new line of data.

Parameters

self	The object pointer
txt	The data to export

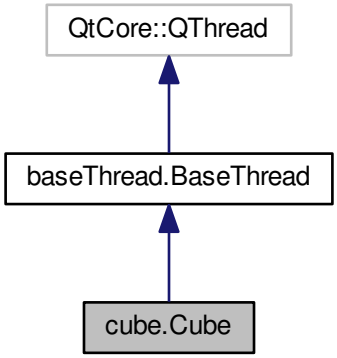
The documentation for this class was generated from the following file:

- Graph/csvExport.py

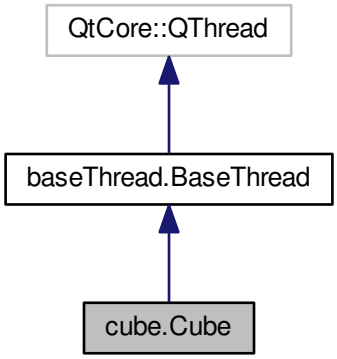
1.4 cube.Cube Class Reference

Displays the orientation as a 3D cube.

Inheritance diagram for cube.Cube:



Collaboration diagram for cube.Cube:



Public Member Functions

- def __init__

  Constructor.
- def updateData

  Gets data from serial thread.
- def run

  Main function.
- def InitGL

  Inits OpenGL.
- def display

  Draws the arms with the rotation data.
- def **newPosition**
- def reshape
- def KeyPressed

  Callback for keypress events.
- def drawForearm

  Draws cube for the foreaarm representation.
- def drawArm

  Draws cube for the arm representation.
- def LoadTextures

  Add textures for the cube.

Public Attributes

- **data**
- **R2**
- **window**

Static Public Attributes

- int **window** = 0
- int **Y** = 0
- int **P** = 0
- int **R** = 0
- list **LAX** = [0.0]
- list **LAY** = [0.0]
- list **LAZ** = [0.0]
- int **X** = 0
- int **Z** = 0

1.4.1 Detailed Description

Displays the orientation as a 3D cube.

1.4.2 Constructor & Destructor Documentation

def cube.Cube.__init__ (*self, data* **)**

Constructor.

Parameters

data	Pass SerialThread object to connect signal

1.4.3 Member Function Documentation

def cube.Cube.display (*self* **)**

Draws the arms with the rotation data.

Parameters

self	The object pointer.

def cube.Cube.drawArm (*self* **)**

Draws cube for the arm representation.

Parameters

self	The object pointer.

def cube.Cube.drawForearm (*self* **)**

Draws cube for the foreaarm representation.

Parameters

self	The object pointer.

def cube.Cube.InitGL (*self, Width, Height*)

Inits OpenGL.

Parameters

self	The object pointer.
Width	size of the Window
Height	size of the Window

def cube.Cube.KeyPressed (*self, args*)

Callback for keypress events.

Parameters

self	The object pointer.
args	Pointer to the args for accesing input data

def cube.Cube.LoadTextures (*self*)

Add textures for the cube.

Applies a png image as a texture for the faces of the cube

Note

experimental

Parameters

self	The object pointer.

def cube.Cube.reshape (*self, Width, Height*)

Parameters

self	The object pointer.
Width	size of the Window
Height	size of the Window

def cube.Cube.run (*self*)

Main function.

Inits and configures all the GLUT parameters to and defines the associated functions to each event

Parameters

self	The object pointer.

def cube.Cube.updateData (*self*)

Gets data from serial thread.

Parameters

self	The object pointer.

The documentation for this class was generated from the following file:

- Graph/cube.py

1.5 fallDetection.FallDetection Class Reference

Detects fall from the accelerometer data.

Public Member Functions

- def __init__
  Constructor for the Fall Detection algorithm.
- def normalVector
  Calculates de normal vector of three vectors.
- def search1LFT
  Finds the first lower value on a list.
- def search2LFT
  Finds the first upper value on a list.
- def updateFall
  Updates the Fall Detection algorithm with the last data.
- def getFallStatus
  Getter for the fall status obtained from the last update.
- def getSemiFallStatus
  Getter for the semi fall status obtained from the last update.

Public Attributes

- **UFT**
- **LFT**
- **fs**
- **POSITION**

Static Public Attributes

- int SIZE = 1024
  Buffer size for SV.
- float UFT = 2.8
  Upper force trigger.
- float LFT = 0.65

Lower force trigger.

- int fs = .02

  Time between samples.

- list SV = [0.0]

  List for SV values.

- list YAW = [.0]

  List for angles monitoring posture.

- list **PITCH** = [.0]
- list **ROLL** = [.0]
- int POSITION = 0

  Indicates for the buffers index.

- DROP = False

  Indicates if a fall was detected.

- **SEMIDROP** = False

1.5.1 Detailed Description

Detects fall from the accelerometer data.

Using an accelerometer calculates:

- Impact normalized force (SV) and impact ranges (UFT, LFT)
- Detectes a range of time during the detected impact (tm, ti)
- Determinates the speed of the fall during the fallind time (V0)
- Detect the posture (change in orientation) respect the initial impact posture. With all these data and determinations is possible to estimate with presition a fall and impact.

Author

Dr. Pablo Reyes
Sebastian Sepulveda

Todo Validate algorithms

1.5.2 Constructor & Destructor Documentation

def fallDetection.FallDetection.__init__ (*self, fs =* .02, *UFT =* 2.8, *LFT =* .65)

Constructor for the Fall Detection algorithm.

Initializes the values of fs, UFT and LFT allowing to change the default values of the algorithms

Parameters

self	The object pointer
fs	Time between samples, in seconds

UFT	UFT trigger, in g (gravities)
LFT	LFT trigger, in g (gravities)

1.5.3 Member Function Documentation

def fallDetection.FallDetection.getFallStatus (*self*)

Getter for the fall status obtained from the last update.

Return the result of the Fall Detection algorithm from the last updateFall call.

Returns

Status of the fall

- **TRUE** : if a fall was detected
- **FALSE** : if a fall wasn't detected

def fallDetection.FallDetection.getSemiFallStatus (*self*)

Getter for the semi fall status obtained from the last update.

Return the result of the Fall Detection algorithm from the last updateFall call before checking the posture

Returns

Status of the semi fall

- **TRUE** : if a semi fall was detected
- **FALSE** : if a semi fall wasn't detected

def fallDetection.FallDetection.normalVector (*self, x, y, z*)

Calculates de normal vector of three vectors.

Uses the math library from Python

Parameters

self	The objetc pointer
x	1st value to normalize
y	2nd value to normalize
z	3rd value to normalize

Returns

Normalized value of x,y and z

def fallDetection.FallDetection.search1LFT (*self, value, vector*)

Finds the first lower value on a list.

Finds the first lower value of LFT on a list

Parameters

self	The objetc pointer
value	The value to find in the list
vector	The list to find on

Returns

The index where the value (or 1st lower) was found

def fallDetection.FallDetection.search2LFT (*self, value, vector*)

Finds the first upper value on a list.

Finds the first upper value of LFT on a list

Parameters

self	The objetc pointer
value	The value to find in the list
vector	The list to find on

Returns

The index where the value (or 1st upper) was found

def fallDetection.FallDetection.updateFall (*self, x, y, z, yaw, pitch, roll*)

Updates the Fall Detection algorithm with the last data.

The function searches for a SV in between the impact range, a fall speed between the time corresponding the impact and the posture when a fall is detected.

Parameters

self	The objetc pointer
x	acceleration in X axis, in g (gravities)
y	acceleration in Y axis, in g (gravities)
z	acceleration in Z axis, in g (gravities)

Note

The algorithm proposed sugest to filter the adquired data with a lowpass 1st order butterworth at 100 Hz. The data adquired from the MPU-9150 (or even ADXL-345) digital accelerometers can (and is) configured to filter the data and outputs filtered raw data. Therefore, this filter is not implemented.
The algorithm has been implemented asumming a sensor fusion running continusly for obtaining yaw, pitch, roll. The class stores the data in a buffer for look back at the posture.

1.5.4 Member Data Documentation

fallDetection.FallDetection.DROP = False `[static]`

Indicates if a fall was detected.

- if **TRUE:** a fall was detected
- if **FALL:** a fall wasn't detected

int fallDetection.FallDetection.fs = .02 [static]

Time between samples.

Time between samples, in seconds

float fallDetection.FallDetection.LFT = 0.65 [static]

Lower force trigger.

Lower force trigger for detecting the start of the impact, in g (gravities)

int fallDetection.FallDetection.SIZE = 1024 [static]

Buffer size for SV.

Determinate the buffer size for storing the SV data

list fallDetection.FallDetection.SV = [0.0] [static]

List for SV values.

Stores the last SV values

float fallDetection.FallDetection.UFT = 2.8 [static]

Upper force trigger.

Upper force trigger for detecting an impact, in g (gravities)

The documentation for this class was generated from the following file:

- Graph/fallDetection.py

1.6 imu.IMU Class Reference

IMU values for scales and data structure in the CSV data received.

Public Member Functions

- def __init__
  Constructor.
- def **getScaleVector**
- def **getRangeAccelerometer**
- def **getRangeGyroscope**
- def **getRangeMagnetometer**
- def **getRangeEuler**

Public Attributes

- **name**
- **ACCEL**
- **GYRO**
- **MAG_T**
- **MAG_GS**

- **EULER**
- **SCALE_VECTOR**
- **ACCEL_RANGE**
- **GYRO_RANGE**
- **MAG_RANGE**
- **EULER_RANGE**

1.6.1 Detailed Description

IMU values for scales and data structure in the CSV data received.

Receives a IMU device name and returns data for scaling each type of data and the correct placement in the CSV format.

Also fixes inverted and swaped axis.

1.6.2 Constructor & Destructor Documentation

def imu.IMU.__init__ (***self, name =*** `'MPU-9150'` **)**

Constructor.

Creates a self.SCALE_VECTOR for scaling the data based on the IMU devices and the arregement on the CSV format.

Parameters

self	The object pointer
name	IMU decive used for sending data

The documentation for this class was generated from the following file:

- Graph/imu.py

1.7 log.Log Class Reference

General logging for debugging.

Public Member Functions

- def __init__

  Constructor.
- def d

  Prints debugging messages.
- def e

  Prints debugging errors.
- def i

  Prints debugging informations.
- def v

  Prints debugging verboses.
- def w

  Prints debugging warnings.

- def setDebug

  Enables/Disables messages from debugs.
- def setError

  Enables/Disables messages from errors.
- def setInfo

  Enables/Disables messages from infos.
- def setVerbose

  Enables/Disables messages from verboses.
- def setWarning

  Enables/Disables messages from warnings.
- def getDebug

  Getter for the status for printing debugs.
- def getError

  Getter for the status for printing errors.
- def getInfo

  Getter for the status for printing infos.
- def getVerbose

  Getter for the status for printing verboses.
- def getWarning

  Getter for the status for printing warnings.

Public Attributes

- **TAG**

Static Public Attributes

- string TAG = ""

  Tag name for the log reference.
- DEBUG = None

  Initial status for the debug methods.
- ERROR = None

  Initial status for the error methods.
- INFO = None

  Initial status for the info methods.
- VERBOSE = None

  Initial status for the verbose methods.
- WARN = None

  Initial status for the warning methods.

1.7.1 Detailed Description

General logging for debugging.

Some simple debugging tools for enabling/disabling prints

1.7.2 Constructor & Destructor Documentation

def log.Log.__init__ (*self, tag, debug =* `True`, *error =* `True`, *info =* `False`, *verbose =* `False`, *warn =* `False`)

Constructor.

Initialices the log and defines the TAG value

Parameters

self	The object pointer
tag	String with the name
debug	define the output of debug methods
error	define the output of error methods
info	define the output of info methods
verbose	define the output of verbose methods
warn	define the output of warning methods

1.7.3 Member Function Documentation

def log.Log.d (*self, txt*)

Prints debugging messages.

Parameters

self	The object pointer
txt	The message to print

def log.Log.e (*self, txt*)

Prints debugging errors.

Parameters

self	The object pointer
txt	The message to print

def log.Log.getDebug (*self*)

Getter for the status for printing debugs.

Parameters

self	The object pointer

Returns

status of the debuggin

def log.Log.getError (*self*)

Getter for the status for printing errors.

Parameters

self	The object pointer

Returns

status of the debuggin

def log.Log.getInfo (*self*)

Getter for the status for printing infos.

Parameters

self	The object pointer

Returns

status of the debuggin

def log.Log.getVerbose (*self*)

Getter for the status for printing verboses.

Parameters

self	The object pointer

Returns

status of the debuggin

def log.Log.getWarning (*self*)

Getter for the status for printing warnings.

Parameters

self	The object pointer

Returns

status of the debuggin

def log.Log.i (*self, txt*)

Prints debugging informations.

Parameters

self	The object pointer
txt	The message to print

def log.Log.setDebug (*self, bool_*)

Enables/Disables messages from debugs.

Parameters

self	The object pointer
*bool_*	The status for the debuggin method • **TRUE** : enabled • **FALSE** : disabled

def log.Log.setError (*self, bool_*)

Enables/Disables messages from errors.

Parameters

self	The object pointer
*bool_*	The status for the debuggin method • **TRUE** : enabled • **FALSE** : disabled

def log.Log.setInfo (*self, bool_*)

Enables/Disables messages from infos.

Parameters

self	The object pointer
*bool_*	The status for the debuggin method • **TRUE** : enabled • **FALSE** : disabled

def log.Log.setVerbose (*self, bool_*)

Enables/Disables messages from verboses.

Parameters

self	The object pointer
*bool_*	The status for the debuggin method • **TRUE** : enabled • **FALSE** : disabled

def log.Log.setWarning (*self, bool_*)

Enables/Disables messages from warnings.

Parameters

self	The object pointer
*bool_*	The status for the debuggin method • **TRUE** : enabled • **FALSE** : disabled

def log.Log.v (*self, txt*)

Prints debugging verboses.

Parameters

self	The object pointer
txt	The message to print

def log.Log.w (*self, txt*)

Prints debugging warnings.

Parameters

self	The object pointer
txt	The message to print

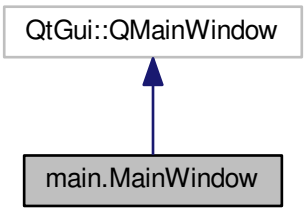
The documentation for this class was generated from the following file:

- Graph/log.py

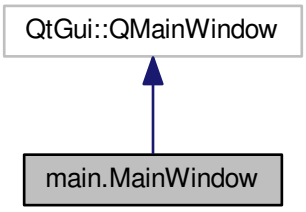
1.8 main.MainWindow Class Reference

Managing and plotting adquired data.

Inheritance diagram for main.MainWindow:



Collaboration diagram for main.MainWindow:



Public Member Functions

- def __init__

  Configures initial settings of the window.
- def start

  Executed when pressing Start Buttton.
- def updateData

  Executed every time new data is adquired.
- def **updateActivity**
- def updatePlot

  Executed every 20 ms for updating plot.
- def stop

  Executed when Stop Button is pressed.
- def configurePlot

  Axis configuration of a PyQtGraph plot.
- def setUILocked

  Set the lock status of the UI elements.

- def getPorts

  Gets avaliable serial ports on the system.
- def resetData

  Clear data buffers for plotting.
- def **startCube**

Public Attributes

- **ui**
- timer

  Reference update plot timer.
- timerActivity

  Qt4 timer for activity.
- data

  Reference for the Threaded serial adquisition.
- cube

  Register the data object with the cube.
- csv

  export data
- activity

  Reference for the activity detection algorithm.
- fall

  Reference for the fall detection algorithm.
- posture

  Reference for.
- AX

  Buffer for Acceleration in X axis.
- AY

  Buffer for Acceleration in Y axis.
- AZ

  Buffer for Acceleration in Z axis.
- GX

  Buffer for Angular Velocity in X axis.
- GY

  Buffer for Angular Velocity in Y axis.
- GZ

  Buffer for Angular Velocity in Z axis.
- MX

  Buffer for Magnetic field in X axis.
- MY

  Buffer for Magnetic field in Y axis.
- MZ

  Buffer for Magnetic field in Z axis.
- Y

  Buffer for Euler angles in X axis (Yaw)
- P

  Buffer for Euler angles in Y axis (Pitch)

- R

  Buffer for Euler angles in Z axis (Roll)
- Y2

  Buffer for Euler angles in X axis (Yaw)
- P2

  Buffer for Euler angles in Y axis (Pitch)
- R2

  Buffer for Euler angles in Z axis (Roll)
- LAX

  Buffer for Linear Acceleration in X axis.
- LAY

  Buffer for Linear Acceleration in Y axis.
- LAZ

  Buffer for Linear Acceleration in Z axis.
- POS

  Indicates the current index for storing data in the buffers.

1.8.1 Detailed Description

Managing and plotting adquired data.

Parameters

QtGui.QMain-Window	Qt4 Main Window reference

1.8.2 Constructor & Destructor Documentation

def main.MainWindow.__init__ (*self*)

Configures initial settings of the window.

Initialize data, plots, imported functions and timers

1.8.3 Member Function Documentation

def main.MainWindow.configurePlot (*self, plot, Range, title, unit*)

Axis configuration of a PyQtGraph plot.

Stops data threads, timers and unlocks UI.

Parameters

self	The object pointer.
plot	The reference to the plot to be configured
Range	The range (+/-) for the Y axis of the plot

title	The title of the plot
unit	The measurement unit for the plot (auto-scaled)

def main.MainWindow.getPorts (*self*)

Gets avaliable serial ports on the system.

Search trough all avaliable serial ports (virtuals included) and lists only those who have a connection avaliable.

Parameters

self	The object pointer.

Return values

portList	list with the serial ports routes

def main.MainWindow.resetData (*self*)

Clear data buffers for plotting.

Fill with zeroes all data buffers used for plotting and position the data.

Size of the buffers can be changed by the class variable WINDOW

Parameters

self	The object pointer.

def main.MainWindow.setUILocked (*self, bool_*)

Set the lock status of the UI elements.

Enables or disables UI elements of the window. Executed during adquisition, blocks UI elements to only enable the Stop button and disable configuration UI elements.

- if `bool_` = **TRUE:** allows configuration of the adquisition
- if `bool_` = **FALSE:** disables configuratin of the adquisition

  Parameters

self	The object pointer.
*bool_*	The status of the UI

def main.MainWindow.start (*self*)

Executed when pressing Start Buttton.

Starts data adquisition and timers, blocking the UI elements

Parameters

self	The object pointer.

def main.MainWindow.stop (*self*)

Executed when Stop Button is pressed.

Stops data threads, timers and unlocks UI.

Parameters

self	The object pointer.

def main.MainWindow.updateData (*self*)

Executed every time new data is adquired.

Executed by serialThread_MPU9150 with QT4 Signal

Parameters

self	The object pointer.

def main.MainWindow.updatePlot (*self*)

Executed every 20 ms for updating plot.

Executed by QtTimer every 20 ms after pressed start button. Stops and reset timer when stop button is pressed.

Parameters

self	The object pointers

1.8.4 Member Data Documentation

main.MainWindow.activity

Reference for the activity detection algorithm.

Activity detection algorithm propossed by Dr. Pablo Reyes

main.MainWindow.AX

Buffer for Acceleration in X axis.

Acceleration in g (gravity)

main.MainWindow.AY

Buffer for Acceleration in Y axis.

Acceleration in g (gravity)

main.MainWindow.AZ

Buffer for Acceleration in Z axis.

Acceleration in g (gravity)

main.MainWindow.data

Reference for the Threaded serial adquisition.

Threaded adquisition for a serial device with interrupts using Qt4 signals

main.MainWindow.fall

Reference for the fall detection algorithm.

Fall detection algorithm propossed by Dr. Pablo Reyes

main.MainWindow.GX

Buffer for Angular Velocity in X axis.

Angular Velocity in degrees/second

main.MainWindow.GY

Buffer for Angular Velocity in Y axis.

Angular Velocity in degrees/second

main.MainWindow.GZ

Buffer for Angular Velocity in Z axis.

Angular Velocity in degrees/second

main.MainWindow.LAX

Buffer for Linear Acceleration in X axis.

Acceleration in g (gravity). The 9DOF sensor fusion algorithm allows to estimate the expected gravity coordinates and dinamically delete it from the measured acceleration, allowing the gravity compensated linear acceleration

main.MainWindow.LAY

Buffer for Linear Acceleration in Y axis.

Acceleration in g (gravity). The 9DOF sensor fusion algorithm allows to estimate the expected gravity coordinates and dinamically delete it from the measured acceleration, allowing the gravity compensated linear acceleration

main.MainWindow.LAZ

Buffer for Linear Acceleration in Z axis.

Acceleration in g (gravity). The 9DOF sensor fusion algorithm allows to estimate the expected gravity coordinates and dinamically delete it from the measured acceleration, allowing the gravity compensated linear acceleration

main.MainWindow.MX

Buffer for Magnetic field in X axis.

Magnetic field flux in Gauss (Gs)

main.MainWindow.MY

Buffer for Magnetic field in Y axis.

Magnetic field flux in Gauss (Gs)

main.MainWindow.MZ

Buffer for Magnetic field in Z axis.

Magnetic field flux in Gauss (Gs)

main.MainWindow.P

Buffer for Euler angles in Y axis (Pitch)

Angles in degrees. Obtained directicly from the DMP of the MPU-9150 internal sensor fusion with 6DOF (Accelerometer + Gyroscope).

main.MainWindow.P2

Buffer for Euler angles in Y axis (Pitch)

Angles in degrees. Obtained from the sensor fusion algorithm (9DOF) developed by Sebastian Madgwick and implemented by Fabio Varesano.

main.MainWindow.POS

Indicates the current index for storing data in the buffers.

Updated and reseted by updateData limits its count to WINDOW variable

main.MainWindow.R

Buffer for Euler angles in Z axis (Roll)

Angles in degrees. Obtained directicly from the DMP of the MPU-9150 internal sensor fusion with 6DOF (Accelerometer + Gyroscope).

main.MainWindow.R2

Buffer for Euler angles in Z axis (Roll)

Angles in degrees. Obtained from the sensor fusion algorithm (9DOF) developed by Sebastian Madgwick and implemented by Fabio Varesano.

main.MainWindow.timer

Reference update plot timer.

Qt4 timer to trigger the function

main.MainWindow.Y

Buffer for Euler angles in X axis (Yaw)

Angles in degrees. Obtained directicly from the DMP of the MPU-9150 internal sensor fusion with 6DOF (Accelerometer + Gyroscope).

main.MainWindow.Y2

Buffer for Euler angles in X axis (Yaw)

Angles in degrees. Obtained from the sensor fusion algorithm (9DOF) developed by Sebastian Madgwick and implemented by Fabio Varesano.

The documentation for this class was generated from the following file:

- Graph/main.py

1.9 posture.Posture Class Reference

Posture estimation based on accelerometer used as a compensated tilt sensor.

Public Member Functions

- def __init__

  Constructor.
- def offset

  Determinates the offset in the accelerometer.
- def compensatedTilt

  Compesated tilt.
- def normalize

  Normalizes de values of the gravity.
- def mean

  Average of a list of values.
- def resetBuffers

  Clean buffers.
- def getTilt

  Getter for the tilt values.

Public Attributes

- **g**
- **angle_frontal**
- **angle_lateral**
- **X**
- **Y**
- **Z**

Static Public Attributes

- int **std** = .05
- list **g** = [0]

1.9.1 Detailed Description

Posture estimation based on accelerometer used as a compensated tilt sensor.

The algoritm calculates the offset due to earth's gravity in the accelerometer axis, provided no other accelerations or forces affects the accelerometer.

The algorithm then determinates the tilt componsated with the estimated position of the gravity, to ensure the measurements are always equals in the coordinates of the accelerometer.

Author

Sebastian Sepulveda

1.9.2 Member Function Documentation

def posture.Posture.compensatedTilt (*self, x, y, z*)

Compesated tilt.

Using the data generated by offset, modifies axis values gatered so the tilt is always measured properly, based on this considerations:

- The earth's gravity is aligned with the z axis (axis pointing in direction of the earth's gravity)
- The Y axis points to the front of the user
- The X axis points to the right of the user

With this considerations, the rest of the modifications based on the displacement of the earth's gravity are reorder to calculate the tilt alway in reference to the earth's gravity

Parameters

self	The object pointer.
x	Acceleration in the x axis, in g
y	Acceleration in the y axis, in g
z	Acceleration in the z axis, in g

def posture.Posture.getTilt (*self, x, y, z*)

Getter for the tilt values.

Parameters

self	The object pointer.
x	Acceleration in the x axis, in g
y	Acceleration in the y axis, in g
z	Acceleration in the z axis, in g

Returns

A list with 2 elements

- **1st** Position show the tilt inclination.
  - **Positive** values means front inclination
  - **Negative** values means backwards inclination
- **2nd** Position show the tilt inclination.
  - **Positive** values means right inclination
  - **Negative** values means left inclination

def posture.Posture.mean (*self, list_*)

Average of a list of values.

Parameters

self	The object pointer.
*list_*	list with values

Returns

Average of the List

def posture.Posture.normalize (*self, value*)

Normalizes de values of the gravity.

Normalizes values in 3 ranges, using the std as desviation from the expected values

Parameters

self	The object pointer.
value	Value to be normalized

Returns

normalized values

- **1** if the gravity points in the direction of the axis
- **0** if the gravity was not detected in the axis
- **1** if the gravity points in the opposite direction of the axis

def posture.Posture.offset (*self, x, y, z*)

Determinates the offset in the accelerometer.

When the accelerometer is static, it's readings should show the effect of the gravity in accelerometer (while it's inside the earth's gravitational force). Therefore, if no other force (normalized magnitud equal to 1 g) the direction of the gravity force could be estimated

An std value is defined for the desviation in the values acceptables

Parameters

self	The object pointer.
x	Acceleration in the x axis, in g
y	Acceleration in the y axis, in g
z	Acceleration in the z axis, in g

def posture.Posture.resetBuffers (*self*)

Clean buffers.

Parameters

self	The object pointer.

The documentation for this class was generated from the following file:

- Graph/posture.py

1.10 sensorFusion.SensorFusion Class Reference

Algorithms and functions for 9DOF fusion.

Public Member Functions

- def __init__

  Constructor.
- def update9DOF

  Update the orientatation with sensors data.
- def LinearAcceleration

  Compensate the accelerometer readings from gravity.
- def Gravity

  Determinates the expected direction of the gravity.
- def getQ

  Getter for the Quaternions.
- def getLinearAcceleration

  Getter for the Linear Acceleration.
- def getGravityDirection

  Getter for the Gravity direction.
- def getYawPitchRoll

  Getter for Orientation as Yaw, Pith, Roll.
- def getEuler

  Getter for Orientation as Euler angles.

Public Attributes

- q

  Last quaternion holder.
- linearAcceleration

  Last Linear accerlation.
- Kp

  Gains of the algorithm.
- Ki

  Gains of the algorithm.
- lastUpdate

  Time between updates.
- **exInt**
- **eyInt**
- **ezInt**

1.10.1 Detailed Description

Algorithms and functions for 9DOF fusion.

Uses data from accelerometers, gyroscopes and magnetometers to do sensor fusion and obtain orientation of the device in a quaternion representation.

Author

Sebastian Sepulveda

1.10.2 Constructor & Destructor Documentation

def sensorFusion.SensorFusion.__init__ (*self*)

Constructor.

Initializes variables for the algorithm

Parameters

self	The object pointer

1.10.3 Member Function Documentation

def sensorFusion.SensorFusion.getEuler (*self, deg =* `False`)

Getter for Orientation as Euler angles.

Return last calculated orientation expresed as euler angles from the last call of updateAHRS

Parameters

self	The object pointer
deg	• **TRUE** : Return angles in degrees • **FALSE** : Return angles in radians

Returns

Vector with Euler angles

Author

Fabio Varesano

def sensorFusion.SensorFusion.getGravityDirection (*self*)

Getter for the Gravity direction.

Return last calculated gravity direction from the last call of updateAHRS

Parameters

self	The object pointer

Returns

Last calculated Gravity Direction

def sensorFusion.SensorFusion.getLinearAcceleration (*self*)

Getter for the Linear Acceleration.

Return last calculated linear acceleration from the last call of updateAHRS

Parameters

self	The object pointer

Returns

Last calculated Linear Acceleration

def sensorFusion.SensorFusion.getQ (*self*)

Getter for the Quaternions.

Return last calculated quaternion from the last call of updateAHRS

Parameters

self	The object pointer

Returns

Last calculated quaternion

def sensorFusion.SensorFusion.getYawPitchRoll (*self, deg* = `False`)

Getter for Orientation as Yaw, Pith, Roll.

Return last calculated orientation expresed as yaw, pitch and roll from the last call of updateAHRS

Parameters

self	The object pointer
deg	• **TRUE** : Return angles in degrees • **FALSE** : Return angles in radians

Returns

Vector with yaw, pitch and roll

Author

Fabio Varesano

def sensorFusion.SensorFusion.Gravity (*self, q*)

Determinates the expected direction of the gravity.

Calculated the expected directions of the gravity in x, y and z frame of reference

Parameters

self	The object pointer

q	Quaternion representing the orientation

Returns

g vector with expected direction of the gravity

def sensorFusion.SensorFusion.LinearAcceleration (*self, q, ax, ay, az*)

Compensate the accelerometer readings from gravity.

Removes the offset generated by the earth's gravity

Parameters

self	The object pointer
q	Quaternion representing the orientation
acc	Readings coming from an accelerometer, in g

Returns

3D vector with the Linear acceleration, in g

Author

Fabio Varesano

def sensorFusion.SensorFusion.update9DOF (*self, gyroscope, accelerometer, magnetometer*)

Update the orientatation with sensors data.

Sensor fusion algorithm ported by Fabio Varesano to Python from the Sebastian Madgwick 9DOF fusion.

Data must be inputed as:

- Accelerations in g (gravities)
- Gyroscope in radians per second (rad/sec)
- Magnetometer in Gauss (Gs)

Parameters

self	The object pointer
gyroscope	Gyroscope data as [X,Y,Z] in rad/sec
accelerometer	Accelerometer data as [X,Y,Z] in g (gravities)
magnetometer	Magnetometer data as [X,Y,Z] in Gs

Author

Fabio Varesano
Sebastian Madgwick

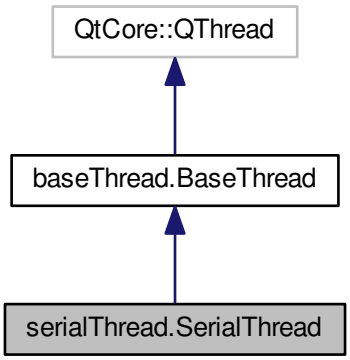
The documentation for this class was generated from the following file:

- Graph/sensorFusion.py

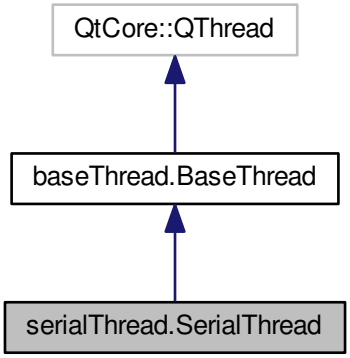
1.11 serialThread.SerialThread Class Reference

Obtain and scale serial data in a thread.

Inheritance diagram for serialThread.SerialThread:



Collaboration diagram for serialThread.SerialThread:



Public Member Functions

- def __init__
  Constructor.
- def openPort
  Open the defined point.

- def run

  The thread process itself.
- def closePort

  Closes the self.serial port and waits for the thread to stop.
- def **getAcceleration**
- def **getLinearAccel**
- def **getGyroscope**
- def **getMagnetometer**
- def **getEulerDMP**
- def **getEuler**

Public Attributes

- **imu**
- **csvScale**
- **csvSize**
- **range_accel**
- **range_gyro**
- **range_mag**
- **range_euler**
- **fusion**
- **LA**
- **E**
- **exiting**

Static Public Attributes

- tuple **ser** = serial.Serial()
- list **A** = [.0]
- list **G** = [.0]
- list **M** = [.0]
- list **E_dmp** = [.3]
- list **E** = [.0]
- list **LA** = [.0]

1.11.1 Detailed Description

Obtain and scale serial data in a thread.

Based on the BaseThread methods, enables a Qt4 thread for adquiring data from a serial port, using a Qt4 signal to interrupt Qt4 main thread process.

Parameters

BaseThread	Basic class for thread handling

1.11.2 Constructor & Destructor Documentation

def serialThread.SerialThread.__init__ (***self, device =*** **'MPU-9150')**

Constructor.

Enables the basic thread functions and logs

Parameters

self	The object pointer

1.11.3 Member Function Documentation

def serialThread.SerialThread.closePort (*self*)

Closes the self.serial port and waits for the thread to stop.

Parameters

self	The objet pointer#

def serialThread.SerialThread.openPort (*self, port =* '/dev/ttyUSB0', *baud =* 115200)

Open the defined point.

Configures the serial port, address and speed, and verifies correct initialization and status

Parameters

self	The object pointer

def serialThread.SerialThread.run (*self*)

The thread process itself.

Defines the "loop" method of the thread

- Obtain self.data from the defined self.serial port
- Separate in CSV format
- Assuming the input from a MPU-9150, scales self.data
- Manage incorrect CSV format (by size)
- Manage incomplete self.data transfer
- Notify main thread (Qt4) with a signal

  Parameters

self	The objetc pointer

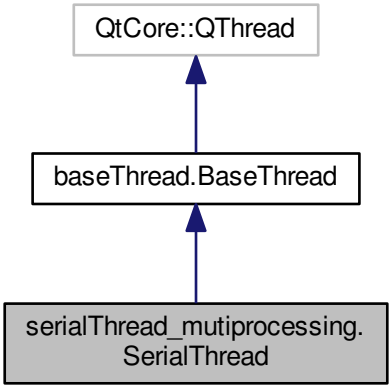
The documentation for this class was generated from the following file:

- Graph/serialThread.py

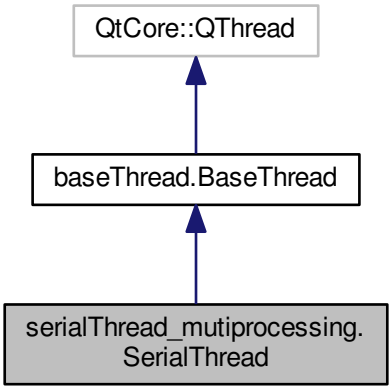
1.12 serialThread_mutiprocessing.SerialThread Class Reference

Obtain and scale serial data in a thread.

Inheritance diagram for serialThread_mutiprocessing.SerialThread:



Collaboration diagram for serialThread_mutiprocessing.SerialThread:



Public Member Functions

- def __init__

  Constructor.
- def openPort

  Open the defined point.
- def run

The thread process itself.

- def closePort

  Closes the self.serial port and waits for the thread to stop.

- def **getAcceleration**
- def **getLinearAccel**
- def **getGyroscope**
- def **getMagnetometer**
- def **getEulerDMP**
- def **getEuler**

Public Attributes

- **imu**
- **csvScale**
- **csvSize**
- **range_accel**
- **range_gyro**
- **range_mag**
- **range_euler**
- **fusion**
- **LA**
- **E**
- **exiting**

Static Public Attributes

- tuple **ser** = serial.Serial()
- list **A** = [.0]
- list **G** = [.0]
- list **M** = [.0]
- list **E_dmp** = [.3]
- list **E** = [.0]
- list **LA** = [.0]

1.12.1 Detailed Description

Obtain and scale serial data in a thread.

Based on the BaseThread methods, enables a Qt4 thread for adquiring data from a serial port, using a Qt4 signal to interrupt Qt4 main thread process.

Parameters

BaseThread	Basic class for thread handling

1.12.2 Constructor & Destructor Documentation

def serialThread_mutiprocessing.SerialThread.__init__ (*self, device =* `'MPU-9150'`)

Constructor.

Enables the basic thread functions and logs

Parameters

self	The object pointer

1.12.3 Member Function Documentation

def serialThread_mutiprocessing.SerialThread.closePort (*self*)

Closes the self.serial port and waits for the thread to stop.

Parameters

self	The objet pointer#

def serialThread_mutiprocessing.SerialThread.openPort (*self, port =* `'/dev/ttyUSB0'` *, baud =* `115200`)

Open the defined point.

Configures the serial port, address and speed, and verifies correct initialization and status

Parameters

self	The object pointer

def serialThread_mutiprocessing.SerialThread.run (*self*)

The thread process itself.

Defines the "loop" method of the thread

- Obtain self.data from the defined self.serial port
- Separate in CSV format
- Assuming the input from a MPU-9150, scales self.data
- Manage incorrect CSV format (by size)
- Manage incomplete self.data transfer
- Notify main thread (Qt4) with a signal

  Parameters

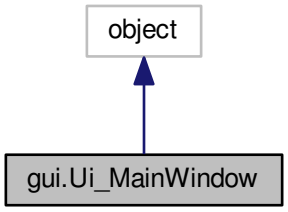
self	The objetc pointer

The documentation for this class was generated from the following file:

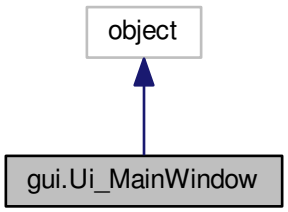
- Graph/serialThread_mutiprocessing.py

1.13 gui.Ui_MainWindow Class Reference

Inheritance diagram for gui.Ui_MainWindow:



Collaboration diagram for gui.Ui_MainWindow:



Public Member Functions

- def **setupUi**
- def **retranslateUi**

Public Attributes

- **centralwidget**
- **gridLayout**
- **cBox_Speed**
- **pButton_Stop**
- **cBox_Port**
- **plt1**
- **plt5**
- **plt4**
- **plt2**

- **pButton_Start**
- **plt3**
- **plt6**
- **pButton_Cube**
- **chBox_export**
- **cBox_IMU**
- **pButton_Reset**
- **statusbar**
- **menubar**
- **menuFile**
- **actionLinear_Acceleration**
- **actionEuler_Rotation**
- **actionLinear_Acceleration_2**
- **actionG_force**
- **actionMeters_seg_2**
- **actionRad_seg**
- **actionDeg_seg**
- **actionYawn_Pitch_Roll**
- **actionEuler_Angles**
- **actionScan_Serial_ports**

The documentation for this class was generated from the following file:

- Graph/gui.py

1 Class Documentation

1.1 csvExport.CSVExport Class Reference

Export data in CSV format.

Public Member Functions

- def __init__
  Constructor.
- def **createFile**
- def **setTitle**
- def **getTitle**
- def csvWrite
  Writes a new line of data.

Public Attributes

- **name**
- **CSV**

1.1.1 Detailed Description

Export data in CSV format.

Export data using date and time as name for the file, avoiding overwriting data in differents adquisitions.

It also registers in the first columnn the elapsed since the first adquisition

Note

The data is stored in a "data" folder (must be created first)

1.1.2 Constructor & Destructor Documentation

def csvExport.CSVExport.__init__ (*self*)

Constructor.

Gets time to create the filename and instanciate the start of the adquisition

Note

The filename format is Year-Moth-Day_Hours-minutes-seconds.csv

1.1.3 Member Function Documentation

def csvExport.CSVExport.csvWrite (*self, txt*)

Writes a new line of data.

Parameters

self	The object pointer
txt	The data to export

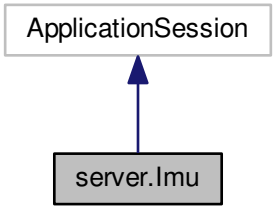
The documentation for this class was generated from the following file:

- Test6/csvExport.py

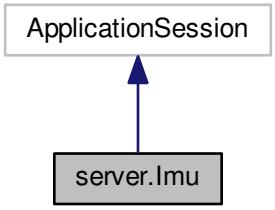
1.2 server.Imu Class Reference

IMU class to access data.

Inheritance diagram for server.Imu:



Collaboration diagram for server.Imu:



Public Member Functions

- def onJoin

  onJoin event (server connected)
- def onLeave

  onLeave event (server disconnected)

- def test

  test function
- def detectImu

  Detects the number of connected IMUs.
- def setFileName

  Sets filename for the CSV export.
- def getFileName

  Gets filename of the current CSV export.
- def startImu

  Starts adquisition.
- def stoptImu

  Stops adquisition.
- def getImuData

  Get data.

Public Attributes

- **app**

1.2.1 Detailed Description

IMU class to access data.

Defines callbacks and register events for WAMP server

Parameters

ApplicationSession	WAMP instance

1.2.2 Member Function Documentation

def server.Imu.detectImu (*self, arg =* `None`)

Detects the number of connected IMUs.

Parameters

self	The object pointer.
arg	Expected data from the WebSocket

Returns

number of IMUs detected

def server.Imu.getFileName (*self, arg =* `None`)

Gets filename of the current CSV export.

Parameters

self	The object pointer.
arg	Expected data from the WebSocket

Returns

file name

def server.Imu.getImuData (*self, arg =* `None`)

Get data.

Gets the current data from the current enabled sensor

Parameters

self	The object pointer.
arg	Expected data from the WebSocket

Returns

sensor data

def server.Imu.onJoin (*self, details*)

onJoin event (server connected)

Starts the MainProcess when server starts, initing the communication thread with the IMU

Parameters

self	The object pointer.
details	WAMP data

def server.Imu.onLeave (*self, details*)

onLeave event (server disconnected)

Stops the MainProcess when server stops.

Parameters

self	The object pointer.
details	WAMP data

def server.Imu.setFileName (*self, arg*)

Sets filename for the CSV export.

Parameters

self	The object pointer.
arg	file name

def server.Imu.startImu (*self, arg =* `None`)

Starts adquisition.

Inits data adquisition from the detected IMUs in another thread

Parameters

self	The object pointer.
arg	Expected data from the WebSocket

def server.Imu.stoptImu (*self, arg =* `None`)

Stops adquisition.

Stops data adquisition and the corresponding thread

Parameters

self	The object pointer.
arg	Expected data from the WebSocket

def server.Imu.test (*self, arg =* `None`)

test function

Checks the WAMP and WebSocket working with a defined Value

Parameters

self	The object pointer.
arg	Expected data from the WebSocket

Returns

Server status

The documentation for this class was generated from the following file:

- Test6/server.py

1.3 log.Log Class Reference

Public Member Functions

- def **__init__**
- def **level**
- def **getLevel**
- def **c**
- def **e**
- def **w**
- def **i**
- def **d**
- def **n**

Public Attributes

- **log**

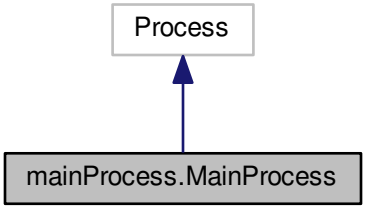
The documentation for this class was generated from the following file:

- Test6/log.py

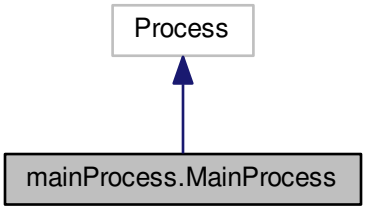
1.4 mainProcess.MainProcess Class Reference

Thread for adquiring data from IMUs.

Inheritance diagram for mainProcess.MainProcess:



Collaboration diagram for mainProcess.MainProcess:



Public Member Functions

- def __init__

  Constructor.
- def detectImu

  Detectes the number of IMUs avaliable in each channel.

- def **setPollInterval**
- def setFileName

  Sets file name for the CSV file.
- def getData

  gets the current data from the current sensor
- def stop

  stops the thread and signals to exit
- def **saveSettings**
- def run

  Thread loop.

Public Attributes

- **exit**
- **csv**
- **i2cMux**
- **settings0**
- **settings1**
- **settings2**
- **settings3**
- **settings4**
- **settings5**
- **imu0**
- **imu1**
- **imu2**
- **imu3**
- **imu4**
- **imu5**
- **settings**
- **imus**
- **detectedIMU**
- **pollInterval**
- **allData**

1.4.1 Detailed Description

Thread for adquiring data from IMUs.

Parameters

Process	the process object

1.4.2 Constructor & Destructor Documentation

def mainProcess.MainProcess.__init__ (*self*)

Constructor.

Inits the process, the CSV file, the PCA9547 Multiplexor, the settings file for each sensor (with calibration)

Parameters

self	The object pointer

1.4.3 Member Function Documentation

def mainProcess.MainProcess.detectImu (*self*)

Detectes the number of IMUs avaliable in each channel.

Parameters

self	The object pointer

Returns

number of detected IMUs

def mainProcess.MainProcess.getData (*self*)

gets the current data from the current sensor

Parameters

self	The object pointer

Returns

all sensor data as a vector

def mainProcess.MainProcess.run (*self*)

Thread loop.

Get data in the polling interval definid time, and stores data form all the avaliable sensor with his identificator and timestamp

Parameters

self	The object pointer

Returns

finalized job

def mainProcess.MainProcess.setFileName (*self, txt*)

Sets file name for the CSV file.

Parameters

self	The object pointer
txt	file name

def mainProcess.MainProcess.stop (*self*)

stops the thread and signals to exit

Parameters

self	The object pointer

The documentation for this class was generated from the following file:

- Test6/mainProcess.py

1.5 pca9547.PCA9547 Class Reference

Class to manage the I2C Multiplexor PCA9547.

Public Member Functions

- def __init__
  Consructor.
- def setChannel
  Sets the current enabled channel.
- def getChannel
  Gets the current enabled channel.
- def resetChannel
  Resets the multiplexor.

Static Public Attributes

- address = None
  Addres of the multiplexor.

1.5.1 Detailed Description

Class to manage the I2C Multiplexor PCA9547.

1.5.2 Constructor & Destructor Documentation

def pca9547.PCA9547.__init__ (*self, address =* `0x70`)

Consructor.

Defines the address for the multiplexor

Parameters

self	The objetc pointer
address	Address to be used

1.5.3 Member Function Documentation

def pca9547.PCA9547.getChannel (*self*)

Gets the current enabled channel.

Parameters

self	The objetc pointer

Returns

the enabled channel number

def pca9547.PCA9547.resetChannel (*self*)

Resets the multiplexor.

When reseted, the PCA9547 restore to the default channel, 0.

Parameters

self	The objetc pointer

def pca9547.PCA9547.setChannel (*self, channel =* 0)

Sets the current enabled channel.

Parameters

self	The objetc pointer
channel	the channel to enable

The documentation for this class was generated from the following file:

- Test6/pca9547.py