GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMAS DE PREVENCIÓN DE CAÍDAS PARA ESTRUCTURAS BÍPEDAS Y AUTÓNOMAS

Autor: Alejandro Díaz de Argandoña Araujo

Director: Juan Peña Juarez

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Sistemas de Prevención de Caídas para Estructuras Bípedas y Autónomas

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/2020 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Alejandro Díaz de Argandoña Araujo Fecha: 11 / 7 / 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Juan Peña Juarez Fecha: 11 / 7 / 2020

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMAS DE PREVENCIÓN DE CAÍDAS PARA ESTRUCTURAS BÍPEDAS Y AUTÓNOMAS

Autor: Alejandro Díaz de Argandoña Araujo

Director: Juan Peña Juarez

Madrid

«*Why do we fall, sir? So that we can learn to pick ourselves up.* »
Alfred Pennyworth

A los que me enseñáis a levantarme

SISTEMAS DE PREVENCIÓN DE CAÍDAS PARA ESTRUCTURAS BÍPEDAS Y AUTÓNOMAS

Autor: Díaz de Argandoña Araujo, Alejandro.
Director: Peña Juarez, Juan.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

Este proyecto analiza las caídas de los robots bípedos y las personas, y plantea posibles soluciones para la prevención y detección de caídas. Así pues, este trabajo se divide en el análisis de caídas, el desarrollo de un algoritmo y una solución adecuada para cada caso, y la verificación del funcionamiento de ambas con una simulación.

Palabras clave: Prevención de caídas, detección de caídas, posicionamiento, estructuras bípedas, MPU 6050.

1. Introducción

Las caídas suponen un problema muy grande tanto para personas como para robots bípedos.

En las personas supone un gran riesgo para la salud y un enorme gasto de dinero en sanidad. Según la OMS, cada año se producen 646.000 muertes humanas por caídas, lo que convierte a las caídas en la segunda causa mundial de muerte por lesiones no intencionales, solamente por detrás de los traumatismos causados por el tránsito.

Para los robots bípedos, las caídas durante el desarrollo del algoritmo para aprender a caminar, aunque son necesarias para registrar datos que generen respuestas en diversos escenarios, causan demora y esfuerzo humano en el proceso al tener que levantarlos sucesivamente de manera manual.

2. Definición del proyecto

Tras analizar las caídas de robots bípedos y personas, planteo un algoritmo que permita detectar las caídas para permitir una rápida respuesta. Además, diseño un sistema específico para cada caso que permite prevenir las caídas en robots bípedos y detectarlas en personas. Finalmente, realizo una simulación donde obtengo resultados acerca de la fiabilidad del algoritmo.

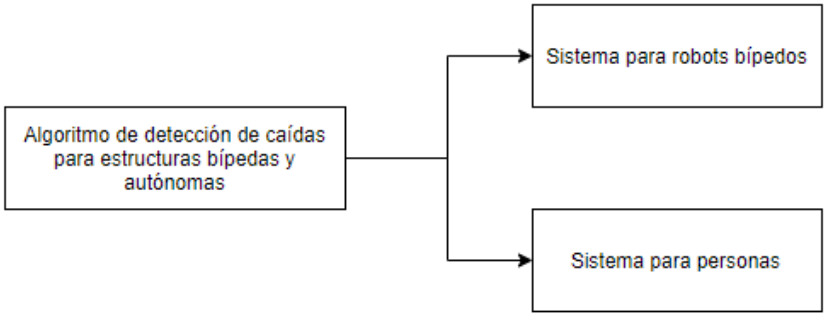


Ilustración 1. Esquema de Trabajo

3. Descripción del Algoritmo y los Sistemas

El algoritmo recopila los datos de un sensor MPU 6050 en Arduino y los envía a una Raspberry PI 3 que analiza en el programa desarrollado en Java los datos recibidos para determinar si se ha producido una caída. En caso de detectar una caída se abre una GUI para confirmar o desmentir la caída y, en caso de confirmarse, se guardan los datos de dicha caída en una base de datos MySQL, lo que permite la posterior visualización y el análisis de los datos.

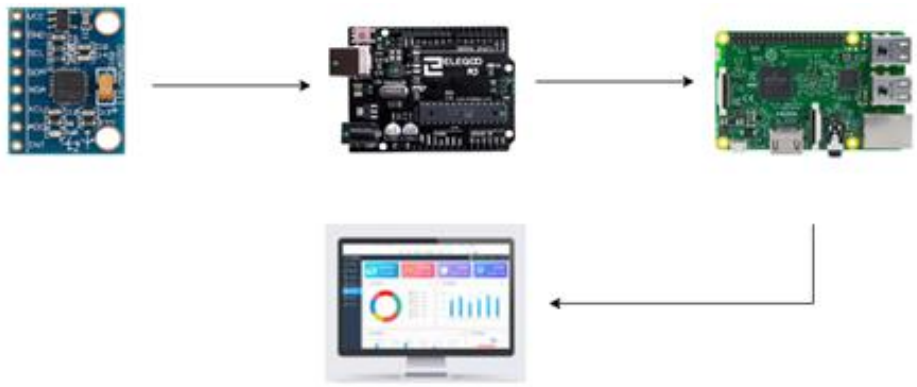


Ilustración 2. Metodología

El sistema diseñado para la prevención de caídas en el entrenamiento de los robots bípedos consiste en un robot de ayuda que procesa el algoritmo diseñado. Este robot ayudante sigue en todo momento al robot entrenado y permite el inicio de las caídas para que el robot entrenado registre los datos, pero evita que se complete la caída devolviendo al robot entrenado a la posición de entrenamiento.

El sistema propuesto para la detección de caídas en personas es una chaleco o banda similar a los utilizados por deportistas para registrar el rendimiento de sus entrenamientos. Al incorporar el sensor y el algoritmo diseñado, se puede crear una aplicación con conectividad Bluetooth que permita la interacción con el usuario cuando se detecta una caída.

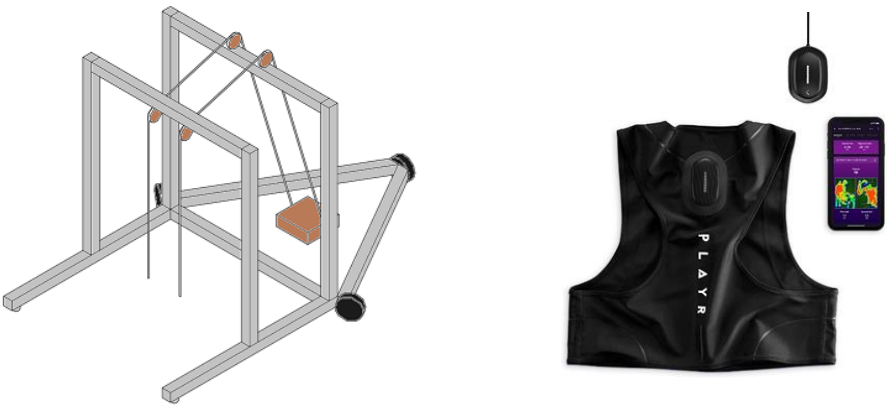


Ilustración 3. Sistemas propuestos

4. Resultados

Ilustración 4. Resultados

Los resultados obtenidos del algoritmo son bastante positivos. Según el análisis estadístico realizado en la simulación del uso del algoritmo en ambos sistemas, se puede afirmar que funciona en un 90% de los casos cuando este es aplicado al sistema para robots y en un 80% aplicado al sistema para personas a un 5% del nivel de significación bajo las condiciones en las que se ha desarrollado y probado.

Estos resultados eran de esperar, pues el algoritmo diseñado es mucho más complejo en el caso de detectar caídas para personas que para detectar el inicio de caídas en robots. Esta complejidad se ve compensada por otro lado con el sistema para robots propuesto, que es más sofisticado que el sistema para personas.

5. Conclusiones

Se han logrado los objetivos iniciales de manera muy satisfactoria y se ha demostrado que es posible implementar un algoritmo con el MPU 6050 que mejore tanto la seguridad de las personas como el entrenamiento de los robots bípedos.

Queda pendiente como ampliación del trabajo la construcción física de los sistemas propuestos ya que debido a la situación excepcional del covid-19 los laboratorios quedaron cerrados y no se pudo finalizar la construcción de ambos.

6. Referencias

[1] Kempe, V. (2011). *Inertial MEMS: Principles and Practice.* Cambridge University Press

[2] Pilla, S. D. (2009). *Slip, Trip, and Fall Prevention.* CRC Press

[3] Koks, D. (2006). Explorations in Mathematical Physics: The Concepts Behind an Elegant Language. En D. Koks, *Explorations in Mathematical Physics: The Concepts Behind an Elegant Language* (págs. 181-183). Springer.

[4] Rui Wang, J. L. (2019). *Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions.* San Francisco: Cornell University.

FALL PREVENTION SYSTEMS FOR BIPEDAL AUTONOMOUS STRUCTURES

Author: Díaz de Argandoña Araujo, Alejandro.
Supervisor: Peña Juarez, Juan.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

In this project I analyze the falls of bipedal robots and humans to introduce solutions to prevent and detect these falls. Thus, this project its split into the analysis of falls, the development of an algorithm and a suitable solution for each case, and the verification of the proper operation of both with a simulation.

Keywords: Fall prevention, fall detection, positioning, bipedal structures, MPU 6050

1. Introduction

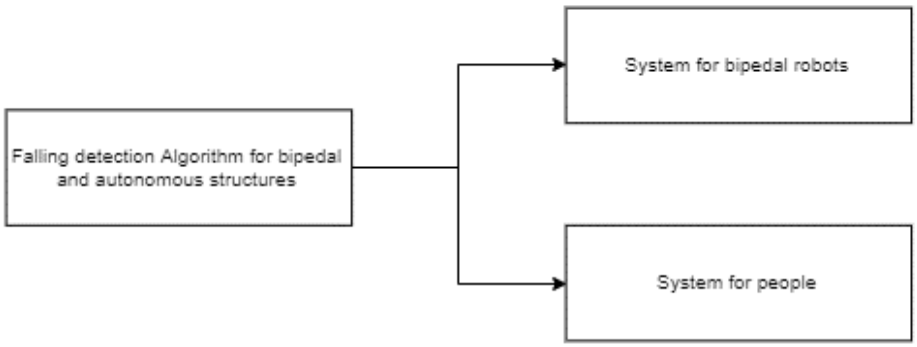
Falls represent a huge problem for both people and bipedal robots.

For people, it is a great health risk and a huge waste of money on health care. According to the WHO, there are 646,000 human deaths from falls each year, making falls the second largest cause of death from unintentional injury worldwide.

For bipedal robots, falls in the process of learning the walking algorithm, although necessary to record data that generate responses in various scenarios, cause delays and wastes human efforts in the process by having to lift the robots successively manually.

2. Project definition

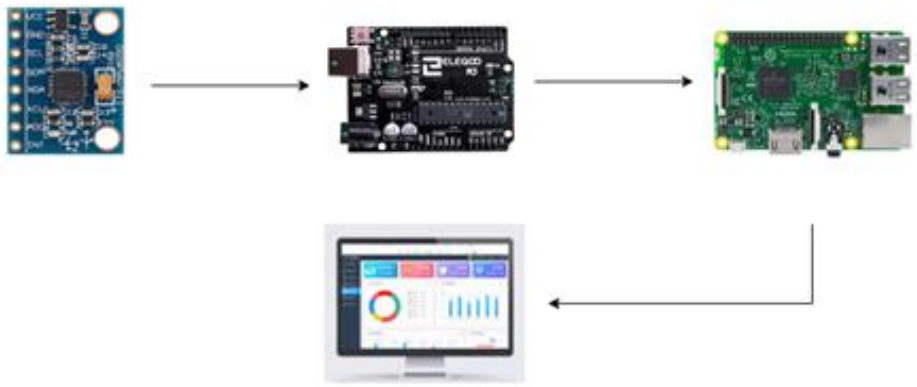
After analyzing the falls of bipedal robots and people, I propose an algorithm to detect the falls to allow a rapid response. In addition, I design a specific system for each case that allows to prevent the falls in bipedal robots and to detect them in people. Finally, I perform a simulation where I obtain results about the reliability of the algorithm.



Ilustration 1. Working plan

3. Algorithm and Systems description

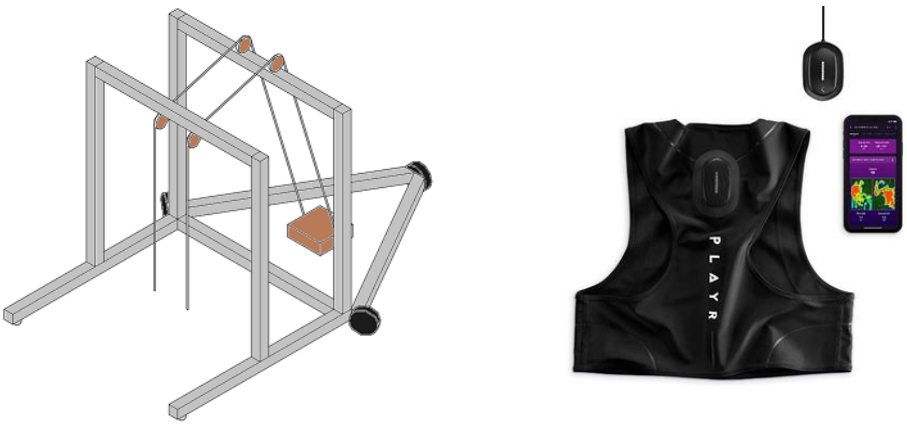
The algorithm collects the data from the MPU 6050 sensor in Arduino and sends it to a Raspberry PI 3 which analyzes the received data in a Java-developed program to determine if a fall has occurred. If a fall is detected, a GUI is opened to confirm or deny the drop, and if confirmed, the data from the drop is stored in a MySQL database, allowing the visualization and data analysis afterwards.



Ilustration 2. Methodology

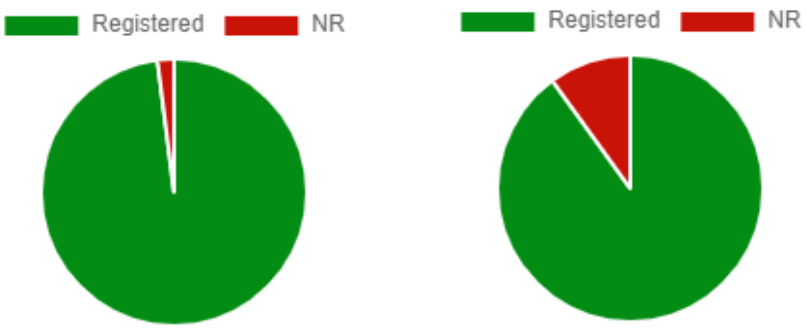
The system designed for fall prevention in the training of bipedal robots consists of a helper robot that processes the designed algorithm and acts like a nanny for the trained robot. This helper robot always follows the trained robot and allows the beginning of the fall to record data by the trained robot but avoids completing the fall by returning the trained robot to its training position.

The system proposed for the detection of falls in people is a vest or band like those used by athletes to record their training and performance stats. By including the sensor and the algorithm in this design, we can create an application with Bluetooth connectivity that can not only detect the falls but also allow the interaction with the user.



Ilustration 3. Suggested systems

4. Results



Ilustration 4. Results

The results obtained from the algorithm are quite positive. According to the statistical analysis carried out in the simulation of the use of the algorithm in both systems, it can be stated that it has worked in 90% of the cases when it is applied to the system for robots and in 80% of the cases when it is applied to the system for people under the conditions in which it has been designed and tested.

This result we expected since the designed algorithm is much more complex in the case of detecting falls for people than to detect the start of falls in robots. This complexity is compensated by the proposed system for robots that is more sophisticated than the system for people.

5. Conclusions

The initial goals have been achieved very satisfactorily and it has been demonstrated that it is possible to implement an algorithm with the MPU 6050 that improves both the safety of people and the training of bipedal robots.

The physical construction of the proposed systems is still pending as an extension of the work, since due to the exceptional situation of covid-19 the laboratories were closed and the construction of both could not be completed.

6. References

[1] Kempe, V. (2011). *Inertial MEMS: Principles and Practice.* Cambridge University Press

[2] Pilla, S. D. (2009). *Slip, Trip, and Fall Prevention.* CRC Press

[3] Koks, D. (2006). Explorations in Mathematical Physics: The Concepts Behind an Elegant Language. En D. Koks, *Explorations in Mathematical Physics: The Concepts Behind an Elegant Language* (págs. 181-183). Springer.

[4] Rui Wang, J. L. (2019). *Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions.* San Francisco: Cornell University.

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

Desde la creación de los primeros robots y máquinas programables, las estructuras bípedas han sido de gran interés ya que estas pueden desempeñar actividades guiadas por humanos que sustituyen el trabajo y esfuerzo físico que supone para una persona el desplazarse de un lugar a otro. La naturaleza ha dotado a las personas la habilidad de moverse a través de dos piernas, una capacidad única que nos ha permitido evolucionar como especie.

Recrear esta obra de ingeniería en robots supone ser capaces de inventar máquinas con la capacidad de desplazarse en numerosos tipos de terrenos sin importar las características particulares de cada superficie ni la cantidad de obstáculos que pueda haber a lo largo de un recorrido. Esto supone una gran ventaja respecto a los robots que se desplazan con ruedas, que, a pesar de ser más estables, no son capaces de superar ciertos obstáculos ni adaptarse a distintos tipos de superficies.

Es por eso por lo que se han intentado numerosas técnicas de control para el balance de un robot bípedo, sin embargo, para el desarrollo del software que debe implementarse en robots se ha de pasar por un largo proceso de prueba y error. Mediante Robot Learning, el cruce entre machine learning (aprendizaje automático de máquinas) y robótica, un robot bípedo es capaz de perfeccionar los algoritmos que hay detrás para poder caminar correctamente, no obstante, esto conlleva dos problemas principales.

En primer lugar, como típicamente la manera en la que un robot bípedo se entrena es dejándolo caer y levantándolo, los robots pueden sufrir daños durante el aprendizaje. Las caídas durante el aprendizaje son por tanto inevitables, los robots forzosamente han de caer para aprender a corregir los fallos que provocan las caídas. Además, aunque cuantos más grados de libertad tenga el robot, más capacidad tendrá para realizar movimientos complejos, el aprendizaje del algoritmo tendrá una mayor complejidad (Knight, 2004).

Aparte del tiempo invertido y los daños colaterales durante el entrenamiento, normalmente es una persona la encargada de tener que volver a poner en pie al robot para seguir con el aprendizaje, lo cual lleva tiempo y esfuerzo. Este problema, que fue planteado por el profesor Kim Joohyung de la Universidad de Illinois en Urbana-Champaign, sugiere que, si fuéramos capaces de desarrollar un mecanismo para aligerar el tiempo de aprendizaje y automatizar el sistema, podríamos entrenar a los robots bípedos mucho más rápido y ahorrar en esfuerzo humano.

Por otro lado, según la Organización Mundial de la Salud, las caídas son un importante problema mundial de salud pública. Se calcula que cada año se producen 646.000 muertes humanas por caídas, lo que convierte a las caídas en la segunda causa mundial de muerte por lesiones no intencionales, solamente por detrás de los traumatismos causados por el tránsito. Cada año se producen 37,3 millones de caídas que, a pesar no ser mortales, requieren de asistencia médica y supone un elevado costo económico y la pérdida de más de 17 millones de años de vida ajustados por discapacidad (Organización Mundial de la Salud, 2018).

Las caídas son especialmente sensibles en los mayores de 65 años cuyas defensas son más débiles. Conforme las personas crecemos y llegamos a una avanzada edad, los reflejos se vuelven más lentos y los sentidos naturales se vuelven menos precisos. Esto provoca que los adultos más mayores no puedan moverse con la seguridad con la que lo hacían antes. Por ello, si fuéramos capaces de desarrollar un sistema cómodo mediante el cual una persona pudiera salir a caminar o hacer ejercicio con mayor seguridad, no solamente supondría un gran ahorro en costes médicos, sino que también podría salvar muchas vidas.

Queda pues demostrado el alto interés que supone encontrar un sistema para detectar y prevenir caídas en estructuras bípedas. Por un lado, para ser capaces de entrenar robots de manera más eficiente, y por otra para detectar caídas en personas que suelen traducirse en causas de fallecimiento o problemas de salud terribles.

Capítulo 2. ESTADO DE LA CUESTIÓN

Actualmente existe una concienciación sobre la gravedad que tienen las caídas en las personas y las consecuencias que estas suponen. Por ello, se han creado sistemas para detectar caídas nocturnas en personas de avanzada edad colocando sensores en habitaciones que detectan cuando un usuario se levanta para deambular (Serenocare, 2017). También existe protección individual que limita la fuerza del impacto contra el usuario para trabajadores que desarrollan su actividad laboral en las alturas (Vertice Vertical, 2015). Sin embargo, no existe actualmente un sistema cómodo y fiable para para la detección de caídas en situaciones cuotidianas más allá de *smartbands* o *smartwatches* como el Apple Watch, que al contar con un sensor único localizado en la muñeca hace que la detección de caídas no sea tan fiable.

Para las principales personas de riesgo, es decir, personas mayores que caminan por la calle de manera habitual, sería conveniente tener un sistema detección de caídas fiable para poder tener una respuesta rápida y minimizar los efectos de esta. Prueba del interés que supone encontrar un producto con estas características es la obtención de fondos que obtuvo un grupo de estudiantes de la universidad de Illinois al presentar en el 2019 Health Make-A-Thon la idea de incorporar sensores en unas zapatillas para resolver este problema. No obstante, el prototipo aún no se ha implantado y se siguen buscando soluciones.

Un enfoque más apropiado para este problema se puede encontrar en un artículo publicado recientemente en (Peter Ma, 2020). Este cómo las caídas se han convertido en la causa más frecuente de muerte para personas mayores de 65 años y explica cómo las nuevas tecnologías podrían ayudar. Las gafas Bose Frames diseñadas para proporcionar audio al usuario, incorpora tecnología programable como acelerómetros, giroscopios y conectividad Bluetooth que puede ser reprogramada con un algoritmo de detección de caídas.

Estas gafas pueden ser una gran opción para los más mayores ya que pasa desapercibida ya que según el artículo, un estudio realizado en Holanda revela que más del 90% de las personas llevan gafas tras cumplir los 50, y casi todo el mundo pasados los 75. Además, se propone el uso de una aplicación Bluetooth que pueda conectar el móvil con las gafas cuando detecten una caída.

En cuanto a los robots bípedos, a través de Inteligenica Artificial y Machine Learning los algoritmos de movimiento son capaces de mejorar las técnicas de control del balance. Para llevar a cabo este aprendizaje se recomienda dividir el proceso en fases y aplicar técnicas como POET (Paired Open-Ended Trailblazer) usado por Uber para generar entornos de aprendizaje complejos y diversos (Rui Wang, 2019). Aun así, si un robot en aprendizaje se cae, tiene que ir una persona a levantarlo y continuar con el proceso, lo cual demora el desarrollo de los algoritmos.

También, dados los recientes avances tecnológicos en IoT y 5G, muchas compañías tecnológicas están viendo las posibilidades que ofrece automatizar tareas desarrolladas por humanos a través de robots bípedos. Por ello, varias empresas han desarrollado robots humanoides como el Asimo de Honda, el Qrio de Sony o el Robotis OP2 de Robotis. Pero el problema principal sigue siendo el desarrollo de algoritmos que permitan la correcta estabilidad del humanoide ante diversas situaciones. Por ello, convendría poder aplicar también una mejora que pudiera acelerar el proceso de aprendizaje de los robots bípedos.

A pesar de que existe mucho interés en resolver estos problemas, no se ha dado aun con la tecla definitiva y en este trabajo se intentará crear un algoritmo válido para ambos problemas y diseñar un sistema específico para cada caso.

Capítulo 3. DEFINICIÓN DEL TRABAJO

3.1 OBJETIVOS

El objetivo será pues enfrentarnos a estos problemas con un sistema personalizado de prevención de caídas para minimizar el impacto de estas. Para ello, se planteará un algoritmo que permita prevenir las caídas o detectarlas, según el caso, para permitir una rápida respuesta. Además, se implementará dicho algoritmo en el sistema personalizado para robots bípedos y humanos y se comparará el resultado de ambos sistemas.

Los objetivos del proyecto se pueden resumir en los siguientes puntos:

- Desarrollar un algoritmo mediante sensores colocados estratégicamente para detectar las caídas de una estructura bípeda de manera más fiable que los sistemas actuales
- Proponer un prototipo de sistema para robots que, implementando el algoritmo, permita entrenar a los robots bípedos de manera más eficiente, reduciendo la necesidad del trabajo humano.
- Proponer un prototipo de sistema para humanos que, implementando el algoritmo, permita una respuesta más rápida ante caídas que pongan el peligro de las personas.

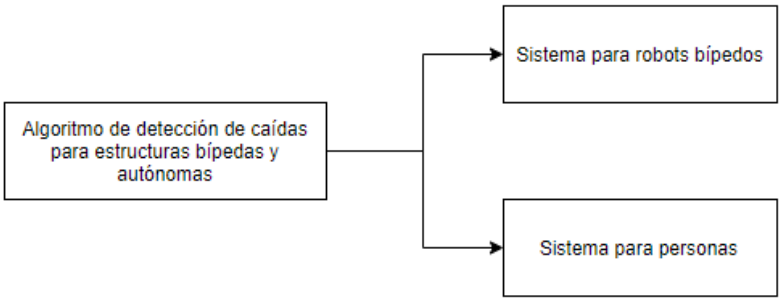


Figura 1. Esquema de Trabajo

3.2 PLANIFICACIÓN

En el siguiente cronograma se explica cómo se va a resolver el problema, así como las técnicas y procedimientos se usarán.

Semana 1	Sistema para robots	Se propondrá un sistema para robots bípedos mediante herramientas de simulación y diseños CAD
Semana 2	Sistema para humanos	Se propondrá un sistema para humanos mediante herramientas de simulación y diseños CAD
Semana 3	Adquisición de datos	Se utilizará una placa Arduino Uno para la recolección de datos a través de sensores de movimiento
Semana 4 **Semana 5** **Semana 6** **Semana 7** **Semana 8**	Desarrollo SW	Se desarrollará el algoritmo capaz de detectar caídas en estructuras bípedas con Java usando una Raspberry Pi 3 para poder implementarlo en ambos sistemas y realizar simulaciones
Semana 9	Simulación	Se simulará la ejecución del software en ambos sistemas y se obtendrán resultados
Semana 10	Análisis comparativo	Se compararán los resultados obtenidos de las implementaciones del software y se elaborará una interfaz de presentación

Tabla 1. Planificación

Teniendo en cuenta que el proyecto ha de contar con unas 335 horas, he dividido el cronograma del proyecto en 10 semanas, de tal manera que trabajando unas 5 o 6 horas por día se puedan llegar a cumplir los objetivos establecidos. El cronograma es orientativo y puede estar sujeto a modificaciones si el director del proyecto lo cree conveniente.

Tal y como se refleja en la planificación, la parte principal del proyecto se centra en la parte del desarrollo del software, ya que no existen limitaciones de trabajo en ese aspecto, al contrario que en la parte hardware, que estará sujeta a las disponibilidades del momento dado el estado de emergencia por el covid-19.

3.3 METODOLOGÍA

Desarrollar un algoritmo implementable en dos sistemas personalizados para cada caso que permita minimizar los efectos de las caídas utilizando como recursos hardware los siguientes componentes

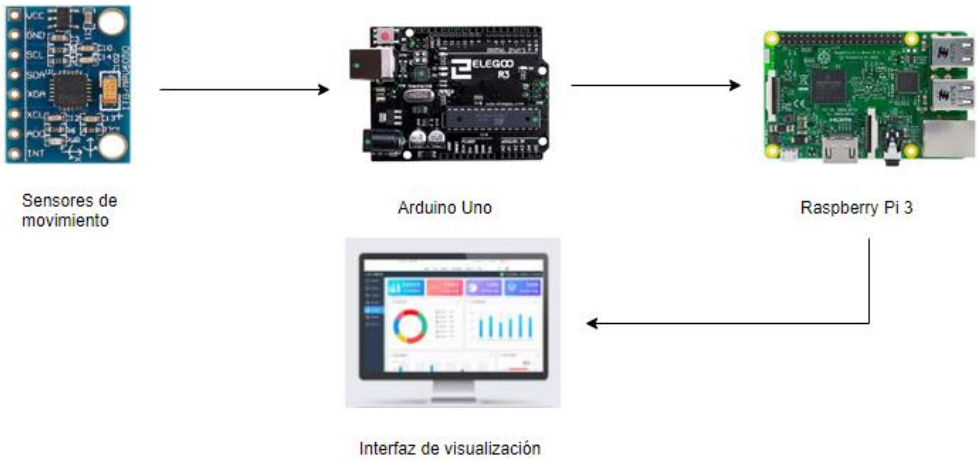


Figura 2. Metodología

3.4 ESTIMACIÓN ECONÓMICA

Teniendo en cuenta los recursos empleados se puede estimar el coste económico del proyecto

Recurso	***Precio (€)***
3 sensores MPU 6050	11,99
Cables DuPont	6,99
Arduino UNO R3	9,99
Raspberry PI 3	37,44
Voltímetro A830L	3,79
Portátil Lenovo G50	599,99
Robot Humanoide Robotis OP2	10.260,00
Ingeniero de Telecomunicaciones (300h)	6.000
Total	**16.930,19 €**

Tabla 2. Estimación Económica del Proyecto

Capítulo 4. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Se desarrollará un algoritmo aplicable en ambos casos que recoge información mediante sensores y detecta cuando una estructura bípeda se cae. De esta manera se pueden reforzar las técnicas empleadas actualmente y predecir o detectar la caída de un sistema bípedo de manera más fiable.

Dado que actualmente estamos en un estado de confinamiento por la pandemia global producida por el covid-19, los laboratorios de la universidad no están accesibles. Por ello, para minimizar el impacto del virus, el desarrollo del proyecto se centrará en la parte software y en simulaciones realizadas con las herramientas disponibles.

En lo que respecta a la parte hardware, se utilizarán una placa de Arduino Uno para recoger información a través de sensores como acelerómetros o giroscopios, y una Raspberry Pi 3 que implementa un sistema operativo de Linux a través de una tarjeta microSD para procesar los datos y generar los outputs deseados mediante el algoritmo. Además, una vez se haya recogido los inputs y generado los outputs, estos se almacenarán en una base de datos para representar gráficas de visualización y elaborar así un análisis comparativo.

Todos estos medios y recursos no deberían tener ningún problema de disponibilidad ya que en tiendas online como Amazon se pueden encargar sin la necesidad de realizar ningún desplazamiento desde casa. En caso de que se requiera soldar alguno de los componentes utilizados se acudirá a la sede de la empresa de telecomunicaciones Grupo Netel de Alicante, donde disponen de estaciones de soldadura.

El método de trabajo será utilizar la placa Arduino para recibir la información de los sensores a utilizar, mientras que la Raspberry ejecutará el algoritmo correspondiente. Al separar en módulos las distintas funcionalidades se permite el uso de un PC entre la Arduino y la Raspberry para depurar los procesos.

De esta manera, al finalizar el trabajo podremos tener una versión inicial portable del sistema implantable a las soluciones propuestas para robots bípedos y humanos. Una vez terminado el algoritmo y verificado su correcto funcionamiento, se propondrá un prototipo de sistema para cada caso y se realizarán las simulaciones correspondientes.

4.1 MPU 6050

El MPU-6050 es una unidad de medición inercial (IMU) de seis grados de libertad (6DOF) fabricado por TDK InvenSense que cuenta con un acelerómetro y un giroscopio de 3 ejes cada uno en un mismo circuito integrado de silicio. Este modelo es uno de los primeros dispositivos de detección de movimiento diseñado para cumplir con requisitos de bajo consumo, bajo coste y alto rendimiento.

El MPU-6050 incorpora la tecnología MotionFusion de InvenSense y un firmware de calibración en tiempo de ejecución que permite una calibración óptima de los sensores y garantiza el correcto funcionamiento de los algoritmos de combinación de los sensores. Gracias al DMP (Digital Motion Processor) incorporado en, el dispositivo puede procesar algoritmos complejos combinando los 6 ejes (InvenSense Inc., 2013).

La comunicación de este modelo admite tanto conexiones por SPI (Serial Peripheral Interface) como por bus I2C (Inter-Integrated Circuit). La transmisión de medidas a la Arduino Uno será pues sencilla ya que únicamente se requieren dos cables para el bus I2C, uno para la señal de reloj (SCL) y otro para el envío de datos (SDA).

La alimentación que necesita el MPU 6050 debe estar entre 2.375 y 3.46 voltios con una tolerancia del 5% según la hoja de especificaciones. Al conectar la unidad a la Arduino se obtendrá la tensión necesaria ya que en este tipo de módulos se incorpora un regulador de voltaje que permite la alimentación directa de 5 voltios proporcionada por placas como la Arduino.

Las medidas que realiza son de 16 bits ya que dispone de conversores analógicos digitales (ADC) de ese tamaño. Por tanto, las valores raw proporcionados tendrán un rango de medición entre -32768 y +32767, es decir de $-2^{n-1} <-> 2^{n-1}-1$, siendo n el número de bits. El rango del acelerómetro puede ser ajustado a ±2g, ±4g, ±8g y ±16g, mientras que el del giroscopio puede estarlo a ±250, ±500, ±1000 y ±2000°/sec.

A continuación, podemos ver las características del sensor y el diagrama de bloques proporcionado por la hoja de especificaciones del fabricante (InvenSense Inc., 2013).

Part #	Gyro Full Scale Range	Gyro Sensitivity	Gyro Rate Noise	Accel Full Scale Range
UNITS:	(°/sec)	(LSB/°/sec)	dps/√Hz	(g)
 MPU-6050	±250	131	0.005	±2
	±500	65.5	0.005	±4
	±1000	32.8	0.005	±8
	±2000	16.4	0.005	±16

Figura 3. Especificaciones MPU 6050 [6]

Accel Sensitivity	Digital Output	Logic Supply Voltage	Operating Voltage Supply	Package Size
LSB/g		(V)	(V +/-5%)	(mm)
16384 8192 4096 2048	I²C	1.8V±5% or VDD	2.375V–3.46V	4x4x0.9

Figura 4. Especificaciones MPU 6050 [6]

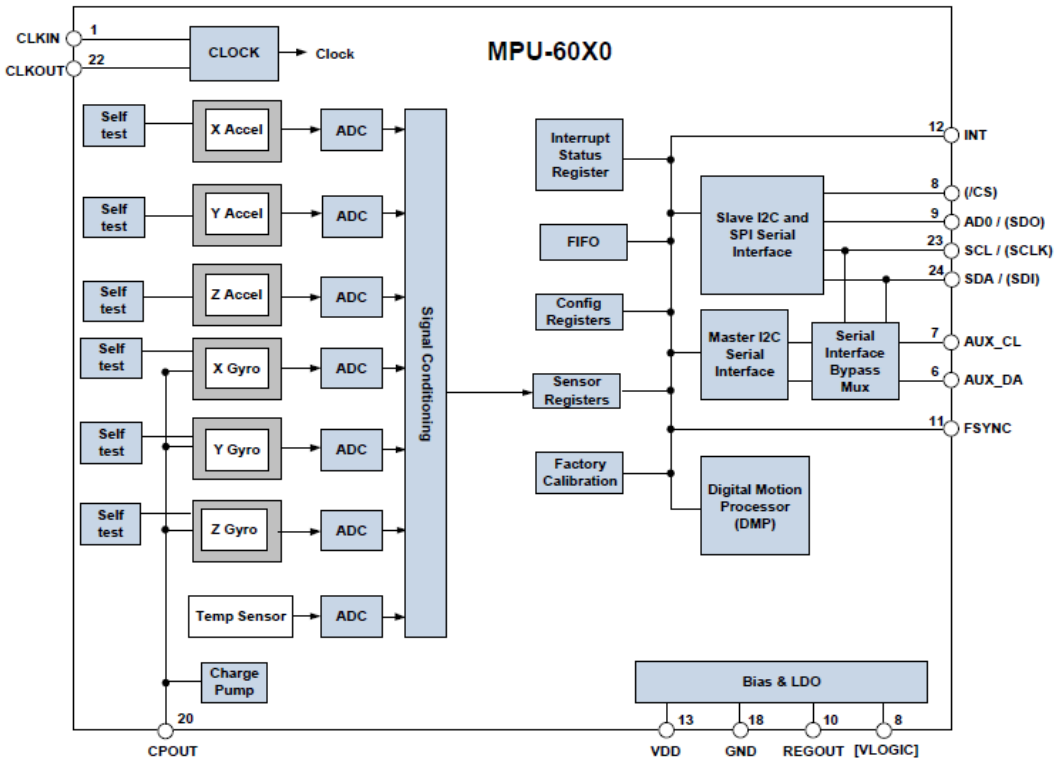


Figura 5. Diagrama de Bloques MPU 6050 [6]

El MPU-6050 contiene los siguientes bloques y funciones:

- Sensor giroscopio MEMS (microelectromechanical systems) de 3 ejes con ADC de 16 bits y acondicionamiento de señal
- Sensor acelerómetro MEMS de 3 ejes con ADC de 16 bits y acondicionamiento de señal
- Motor DMP (Digital Motion Processor)
- Interfaces primarias de comunicaciones serie I2C y SPI (solo MPU-6000)
- Interfaz serial I2C auxiliar para magnetómetros de terceros y otros sensores.
- Reloj
- Registros de datos del sensor
- FIFO
- Interrupciones
- Sensor de temperatura de salida digital
- Autodiagnóstico de giroscopio y acelerómetro
- Regulador lineal de tensión LDO (low-dropout)
- Multiplicador de tensión

4.2 ESQUEMA DE MONTAJE

Para conectar los sensores del MPU-6050 a la Arduino Uno, primero soldamos los pines al módulo y los conectamos con la Arduino a través de cables Dupont macho-macho con los pines correspondientes tal y como muestran las siguientes figuras.

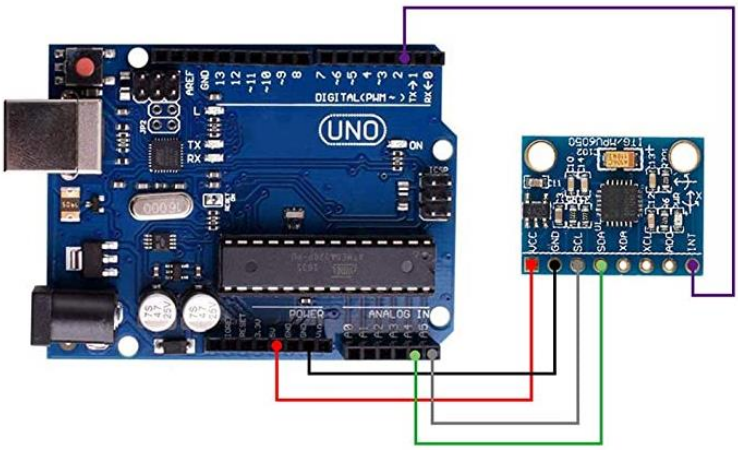


Figura 6. Esquema de Montaje

MPU 6050	*Arduino*
VCC	VCC
GND	GND
SDA	A4
SCL	A5
INT	A2

Tabla 3. Conexiones de los Pines

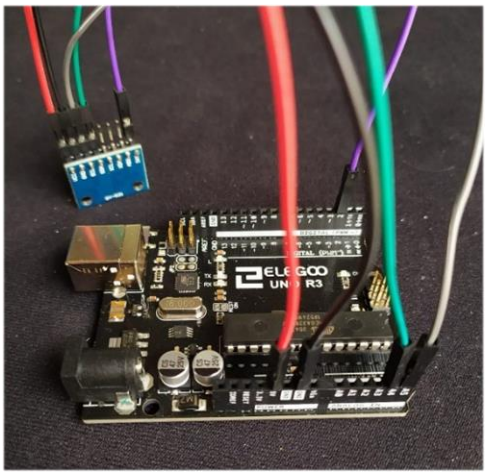


Figura 7. Conexiones en Arduino

Figura 8. Conexiones en el MPU 6050

Conectando la Arduino al ordenador a través de un cable USB B-USB A somos capaces de proporcionar la tensión necesaria para alimentar la placa y el sensor. Podemos realizar una comprobación de esto midiendo los pines de tensión de 5V y tierra con un voltímetro.

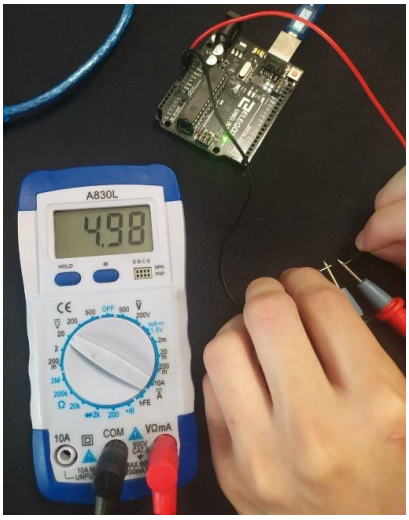


Figura 9. Medición con Voltímetro

4.3 ACELERÓMETRO

A continuación, estudiamos los datos que registra el sensor de movimiento para decidir qué datos queremos usar y en que formato en función de cómo los vayamos a usar en el algoritmo.

El acelerómetro juega un papel fundamental en este proyecto, ya que junto al giroscopio seremos capaces de determinar la posición, velocidad y orientación de una estructura bípeda Las unidades de medición inercial (IMUs) son dispositivos electrónicos que mide e la velocidad, orientación y fuerzas gravitacionales utilizando precisamente acelerómetros y giroscopios.

En los últimos años, estos dispositivos han crecido en popularidad por su reducido tamaño, bajo coste y alta durabilidad. Esto es debido a la mejora de sistemas electromecánicos (MEMS), que se refiere a la tecnología electromecánica de dispositivos microscópicos. Estos son los utilizados por ejemplo en *smartphones* y *tablets* para detectar la rotación de pantalla.

El acelerómetro mide la aceleración a la que está sometido un cuerpo, que equivale a la variación de velocidad de un cuerpo respecto del tiempo tal y como expresa matemáticamente la siguiente ecuación:

$$\vec{a} = \frac{d\vec{v}}{dt} = \frac{d^2\vec{x}}{d^2t}$$

Siendo $\vec{a}$, $\vec{v}$ y x los vectores de aceleración (m/s2), velocidad (m/s) y posición (m) de un cuerpo en el espacio.

Las leyes fundamentales de la dinámica explican el comportamiento que tiene un cuerpo cuando interactúa con fuerzas externas y la aceleración que estas fuerzas provocan.

Según la primera ley de Newton todo cuerpo permanece en estado de reposo o de movimiento rectilíneo uniforme a menos que una fuerza externa actúe sobre él.

$$\sum \vec{F} = 0 \leftrightarrow \frac{d\vec{v}}{dt} = 0$$

Siendo $\sum \vec{F}$ el sumatorio de vectores de fuerza que interactúan sobre un cuerpo, dando como resultado el vector de fuerza neta. Por tanto, es necesaria una fuerza neta no nula para poder registrar una aceleración.

Además, la segunda ley de Newton establece que la aceleración a la que está sometido un cuerpo obedece a la siguiente relación.

$$\vec{F} = m \times \vec{a}$$

Por tanto, la aceleración que un cuerpo adquiere en su trayectoria es directamente proporcional a la fuerza neta no nula que se aplica sobre este e inversamente proporcional a la masa del cuerpo.

Los acelerómetros fabricados en los IMUs tienen en cuenta estas leyes para poder determinar la aceleración registrada por un cuerpo a partir de unos muelles que registran las fuerzas a las que está sometido.

Los acelerómetros incorporan un cuerpo sólido que encierra una masa sostenida por diversos muelles. Estos muelles están adheridos al cuerpo sólida que está fijo, por tanto, cuando el acelerómetro se mueve, la masa del interior sufre una variación de movimiento cuya fuerza es sufrida por los muelles y queda registrada por los sistemas MEMS (Kempe, 2011).

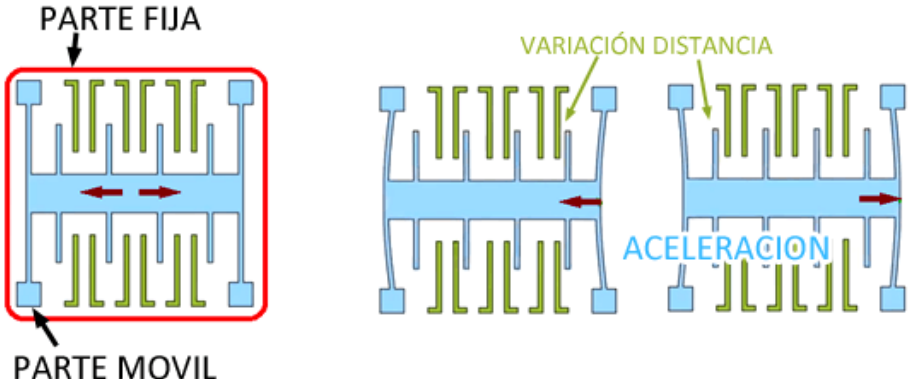


Figura 10. Mecanismo del Acelerómetro [11]

Así, a través de la ley de Hooke y la segunda ley de Newton somos capaces de determinar la aceleración a la que está sometido un cuerpo en los 3 ejes X, Y y Z.

$$\vec{F} = k \times x = m \times \vec{a}$$

Habrá que tener en cuenta la aceleración de la gravedad terrestre a la que está sometido un cuerpo, tal y como hemos comprobado en el ejemplo de la lectura anterior.

Es posible utilizar las aceleraciones obtenidas en los 3 ejes para calcular la velocidad y la posición de un cuerpo, sin embargo, esto no es una buena idea ya que las integrales respecto del tiempo acumularían errores de medición y de ruido. Es por tanto una opción más aconsejable y precisa utilizar también un giroscopio para obtener la velocidad angular del cuerpo y complementar así los cálculos de posicionamiento de un cuerpo.

4.4 GIROSCOPIO

Un giroscopio es un dispositivo mecánico capaz de medir la orientación en el espacio de un cuerpo midiendo el ángulo de rotación girado por un determinado mecanismo. Las medidas tomadas por los giroscopios son relativas tomadas respecto de una referencia arbitraria, a diferencia de las medidas absolutas tomadas por los acelerómetros donde existe una referencia absoluta.

En los dispositivos que cuentan con tecnología MEMS como el MPU-6050 utilizan giroscopios de estructura vibrante (Kempe, 2011) que aprovechan la fuerza del efecto Coriolis para registrar la rotación en los ejes de giro Z, Y y X de forma independiente, también conocidos como guiñada, cabeceo y alabeo (*yaw*, *pitch* y *roll*).

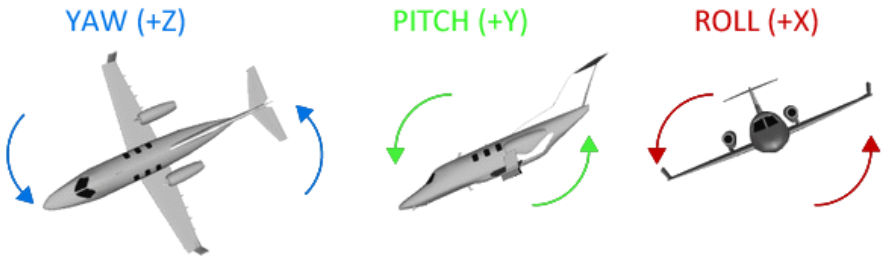


Figura 11. Yaw, Pitch y Roll [11]

Mediante una estructura fija acoplada a unos brazos vibrantes se puede determinar la orientación de un cuerpo en base a la deformación producida en estos brazos por la fuerza de Coriolis, cuyo valor es el siguiente.

$$\vec{F_c} = -2m(\vec{w} \times \vec{v})$$

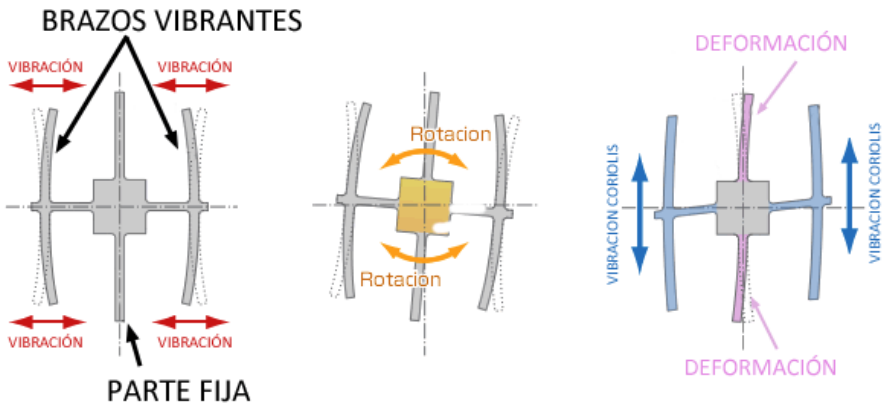


Figura 12. Mecanismo del Giroscopio [11]

De esta manera se puede registrar la velocidad angular, que es la variación del ángulo recorrido respecto del tiempo.

$$\omega = \frac{d\theta}{dt}$$

Para obtener por tanto la posición del cuerpo, habría que realizar la integral de la velocidad angular respecto del tiempo, pero de nuevo las mediciones integrales provoca la acumulación de errores y ruido a medio o largo plazo, lo que genera deriva (drift).

Capítulo 5. OBTENCIÓN DE DATOS

Antes de empezar a desarrollar el algoritmo se han de tomar los datos necesarios para poder representar y monitorizar la posición actual de una estructura bípeda. Para obtener estos datos, se utilizarán las unidades de medición inercial MPU-6050 estudiadas anteriormente, un sensor ampliamente utilizado por su precio tan asequible que ronda en torno al euro y su alta calidad. Además, este incorpora la electrónica necesaria para conectarla de forma sencilla a la placa de desarrollo Arduino Uno.

Al ser ampliamente utilizado, existe una gran cantidad de información disponible acerca del MPU-6059, así como librerías ya escritas para facilitar la obtención de datos. Algunos de los ejemplos más destacados de internet son los del ingeniero Luis Llanos de Zaragoza o los del blog americano How To Mechatronics, donde se realizan experimentos con este dispositivo en diversas plataformas, incluida la Arduino.

5.1 ACELERACIONES Y VELOCIDADES ANGULARES

Para leer los valores del MPU-6050 utilizaremos las librerías I2Cdev y MPU6050 desarrolladas por el ingeniero Jeff Rowberg, que proporciona códigos de ejemplo con este dispositivo que optimiza las comunicaciones I2C (Dejan, 2019). Varias fuentes explican cómo obtener las distintas lecturas de datos a partir de estas librerías, realizando ejemplos con modificaciones del código de Rowberg. Tomaremos estos ejemplos como punto de partida para obtener las medidas necesarias para el desarrollo del algoritmo de detección de caídas.

Realizando una primera lectura del sensor comprobaremos que el sensor y la lectura de datos funciona correctamente si nos encaja con los valores esperados de la realidad. Para ello, emplearemos el siguiente código en la Arduino.

```
#define VCC2  4 // define pin 4 or any other digial pin here as VCC2

#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

const int mpuAddress = 0x68;  // Puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress);

int ax, ay, az;
int gx, gy, gz;


// Factores de conversion
const float accScale = 2.0 * 9.81 / 32768.0;
const float gyroScale = 250.0 / 32768.0;

void printTab()
{
   Serial.print(F("\t"));
}


//Mostrar medidas RAW
void printRAW()
{
   Serial.println(F("Valores RAW:"));
   Serial.print(F("a[x y z](raw) g[x y z](raw):t --> "));
   Serial.print(ax); printTab();
   Serial.print(ay); printTab();
   Serial.print(az); printTab();
   Serial.print(gx); printTab();
   Serial.print(gy); printTab();
   Serial.println(gz);
}

// Mostrar medidas en Sistema Internacional
void printSI()
{
   Serial.println(F("Valores SI:"));
   Serial.print(F("a[x y z](m/s2) g[x y z](deg/s):t --> "));
   Serial.print("a");
   Serial.print(ax * accScale);
   Serial.print(" ");
   Serial.print(ay * accScale);
   Serial.print(" ");
   Serial.print(az * accScale);
   Serial.print(" ");
   Serial.print(gx * gyroScale);
   Serial.print(" ");
```

```
    Serial.print(gy * gyroScale);
    Serial.print(" ");
    Serial.print(gz * gyroScale);
    Serial.print(" ");
    Serial.print("z");

}

void setup()
{
    pinMode(VCC2,OUTPUT);//define a digital pin as output
    digitalWrite(VCC2, HIGH);// set the above pin as HIGH so it acts as 5V
    Serial.begin(115200);
    Wire.begin();
    mpu.initialize();
    Serial.println(mpu.testConnection() ? F("IMU iniciado correctamente") :
F("Error al iniciar IMU"));
}

void loop()
{
    // Leer las aceleraciones y velocidades angulares
    mpu.getAcceleration(&ax, &ay, &az);
    mpu.getRotation(&gx, &gy, &gz);

    //printRAW();
    printSI();

    delay(100);
}
```

Es importante que al leer el monitor serie utilicemos la misma tasa de baudios, es decir, el número de señales transmitidas y recibidas por segundo, para que podamos ver con claridad las lecturas de los datos transmitidos por el sensor, en este caso el valor utilizado es de 115200 baudios. Teniendo esto último en cuenta, abrimos el monitor serie y observamos los siguientes resultados.

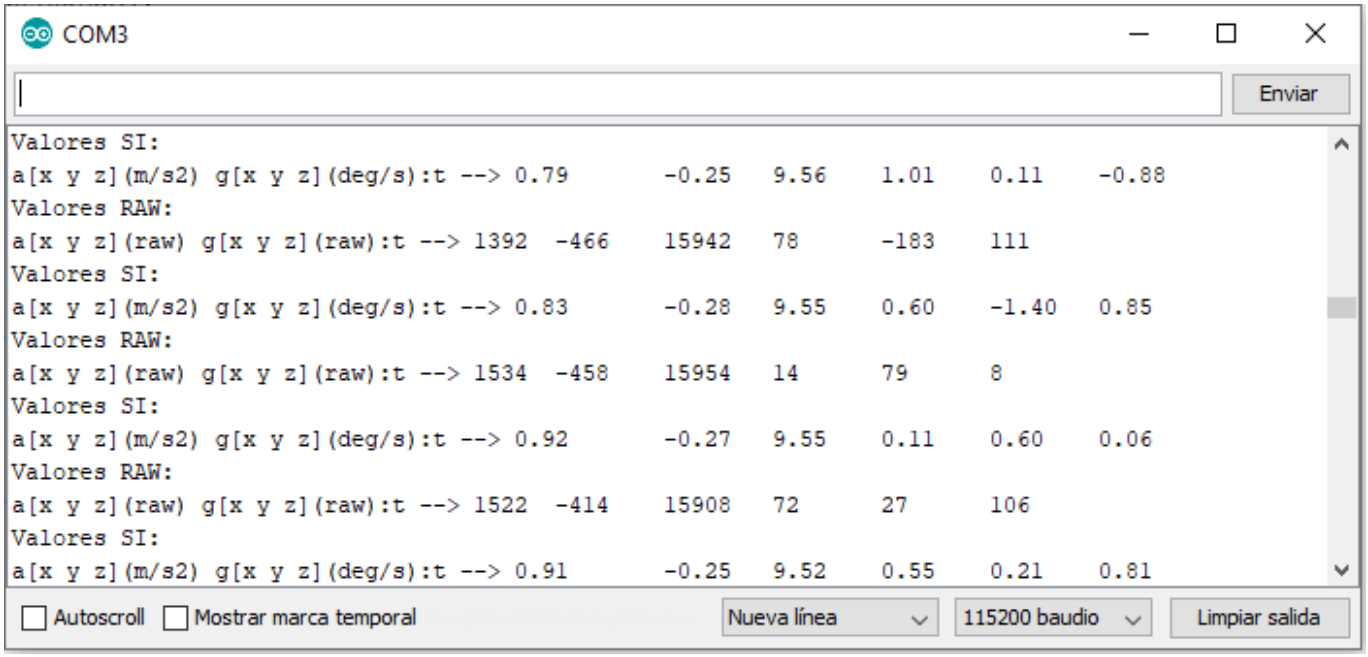


Figura 13. Aceleraciones y Velocidades Angulares

Como esperábamos, si colocamos el sensor de manera plana sobre su eje X-Y en una superficie horizontal y con Z apuntando hacia arriba, podemos observar como las aceleraciones obtenidas, ajustadas en el SI, son aproximadamente 0 para los ejes x e y, ya que no se registra ninguna aceleración en esos ejes, y una aceleración aproximada de 9,8 en z, que corresponde a la lectura de la gravedad. Esto se debe a que los muelles del sistema microelectromecánico del acelerómetro son sensibles también a la gravedad.

Además, como queríamos comprobar, al permanecer en posición de reposo casi absoluto, el sensor registra valores muy reducidos de giro en los 3 ejes. Por lo tanto, queda demostrada que la lectura de valores en sistema internacional del sensor es correcta a través de las librerías utilizadas.

En este caso, estamos obteniendo únicamente los valores de aceleración (m/s^2) y de velocidad angular (θ/s), pero si queremos valores de posicionamiento, los cuales serían muy útiles para desarrollar un algoritmo de detección de caídas, debemos combinar la información de los registros de aceleración y giro.

5.2 POSICIONAMIENTO

Una opción sería obtener la orientación a partir de la información del acelerómetro. Podríamos calcular la inclinación del MPU-6050 proyectando la medición de la gravedad y empleando las relaciones trigonométricas de un plano inclinado. Sin embargo, las medidas de un acelerómetro no son adecuadas para calcular las medidas de velocidad, y menos aún las de posicionamiento, ya que como comentamos, se acumulan errores de medición y existe ruido (Llamas, 2018).

$$\theta = atan\frac{A_X}{A_Z}$$

Otra opción sería obtener la orientación a partir de la información del giroscopio. Podríamos obtener el ángulo de posición del sensor integrando la velocidad de giro respecto del tiempo, pero las mediciones integrales provocan la acumulación de errores y ruido a medio y largo plazo, lo que genera lo que errores de deriva.

$$\theta_{gyro} = \omega_{gyro} \cdot \Delta t$$

Es decir, mientras que los acelerómetros no tienen deriva a medio o largo plazo, pero no son fiables a corto plazo debido a los movimientos del sensor, con los giroscopios ocurre lo contrario, son fiables a corto plazo, pero no a medio o largo plazo donde también presentan deriva.

Por tanto, para obtener una lectura fiable de la posición del cuerpo, será necesario combinar las ventajas de ambos con la información fiable del acelerómetro y del giroscopio, ya que son complementarias.

Para ello, podemos combinar la información en el IMU utilizando un filtro para la señal como el filtro de Kalman, pero al suponer un cálculo extra para la Arduino se suele utilizar un filtro más sencillo como el filtro complementario. Sin embargo, esto tampoco será necesario.

Precisamente, el DMP del MPU-6050 combina los datos del acelerómetro y del giroscopio para corregir los efectos de deriva acumulados en ambos sensores, por lo que podemos obtener valores más precisos de la orientación del cuerpo sin utilizar un filtro complementario ni realizar cálculos en la Arduino. Es aquí donde entra en juego el pin INT del MPU-6050, que se conecta al pin de interrupciones de la Arduino.

Existe más de una manera de expresar la orientación relativa de dos sistemas, por ejemplo, con ángulos de Euler, matrices de rotación o cuaterniones. Uno de los sistemas más utilizados son los ángulos de navegación, llamados en matemáticas ángulos de Tait-Bryan, que son una variación de los ángulos de Euler. Estos ángulos nos permiten describir el posicionamiento de un cuerpo a través de las mediciones en los ejes de giro *yaw*, *pitch* y *roll* ya vistos de manera sencilla e intuitiva.

La gran desventaja que sufren estos ángulos, como todos los sistemas de Euler, es el del bloqueo cardán (*gimbal lock*). Este fenómeno consiste en la pérdida de un grado de libertad cuando dos ejes se colocan en paralelo, bloqueando el sistema en una rotación (Koks, 2006). Este se daría por ejemplo en el caso en que el eje Y gire 90 grados, en ese caso el eje X y Z coincidirían y el sistema quedaría degenerado a 2DOF.

Aunque normalmente esta desventaja es tan grande que en la mayoría proyectos se prefiere utilizar cuaterniones, para desarrollar el algoritmo deseado en este proyecto no es necesario complicarse, ya que las caídas no sobrepasan los 90 grados en pitch y roll, y cuando se aproximan, es uno de los indicativos que nos sirven para indicar que la caída ha ocurrido, y entonces el objetivo del algoritmo estaría ya cumplido.

Los cuaterniones son una extensión de los números reales similar a la de los números complejos. Estos utilizan 3 unidades imaginarias i, j y k donde se cumple la siguiente relación

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$$

Aunque las librerías que utilizamos trabajan con cuaterniones para obtener los datos del sensor, en el algoritmo utilizaremos los valores traducidos a *yaw*, *pitch* y *roll* para poder desarrollar el algoritmo sin valores complejos sino con vectores de 9 números reales (Ax, Ay, Az, Gx, Gy, Gz, Yaw, Pitch, Roll).

Antes de obtener los valores de posicionamiento, es necesario calibrar el DMP para que las medidas sean precisas. Para ello, hemos de obtener los valores de offset que ajusta nuestro MPU 6050 concreto. Utilizando uno de los códigos proporcionados por la librería de Rowberg, el cual está incluido en el anexo, obtenemos los siguientes *offsets*.

```
Sensor readings with offsets:   7       0       16370   -1      1       -1
Your offsets:   -3764   206     1680    -32     88      -1

Data is printed as: acelX acelY acelZ giroX giroY giroZ
Check that your sensor readings are close to 0 0 16384 0 0 0
If calibration was succesful write down your offsets so you can set them in your projects
```

Figura 14. Valores de Calibración del MPU 6050

Una vez obtenidos los *offsets* de calibración, podemos incluirlos en el código que programaremos para obtener todos los valores que deseamos obtener. Cabe destacar que el DMP tarda unos 15 segundos aproximadamente en calibrarse, por lo que cuando ejecutemos el algoritmo deberemos asegurarnos de esperar el tiempo necesario para obtener valores fiables.

Por tanto, los inputs que vamos a obtener del sensor para desarrollar el algoritmo son: la aceleración de los 3 ejes, la velocidad angular de los 3 ejes de rotación, y el posicionamiento del sensor con los valores de *yaw*, *pitch* y *roll*. El programa utilizado para este objetivo en la Arduino puede encontrarse en el anexo y la información recibida por el algoritmo queda resumida en la siguiente imagen.

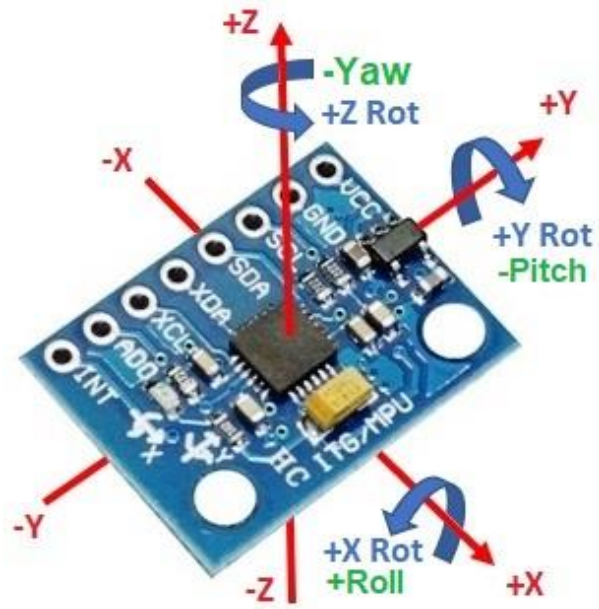


Figura 15. Direcciones y Sentidos de los Datos en el MPU 6050

En resumen, con el sensor MPU 6050 obtendremos la siguiente información de una estructura bípeda tomando el tronco como punto de referencia.

- Los valores de aceleración lineal en los ejes X, Y, Z
- Los valores de velocidad angular en los ejes de giro X, Y, Z
- Los valores de yaw, pitch y roll.

Recordemos que los valores de aceleración lineal y velocidad angular están escalados a ±2g, es decir ±19,62 m/s^2 y a ±250 º/s, por lo que esos serán los valores máximos que se podrán registrar.

Por otro lado, los valores de posicionamiento en yaw, pitch y roll pueden registrar los valores de ±180º en yaw y de ±90º en pitch y roll. Esto se debe a que el sensor, al utilizar los ángulos de Tait-Bryan, mide los ángulos respecto de la horizontal.

5.3 TRANSFERIR LOS DATOS AL PC

Para poder leer fácilmente los valores en el algoritmo, enviamos por el puerto serie los 9 valores con un protocolo muy sencillo. Enviamos un carácter 'a' como inicio de trama, los 9 valores seguidos de un carácter de espacio, y finalmente un carácter 'z' como final de trama. De esta manera facilitamos la lectura en Java de los vectores de valores.

$$a\ Ax\ Ay\ Ax\ Gx\ Gy\ Gz\ Yaw\ Pitch\ Roll\ z$$

Para trasferir los datos al PC utilizamos un USB-TTL que conecta la Arduino con el PC. Necesitaremos por tanto activar el PIN 4 de la Arduino para que estando en alta tensión pueda servir de alimentación para el USB-TTL. Al igual que antes, comprobamos la tensión midiendo aproximadamente 5V entre el PIN 4 y el PIN GND con el voltímetro.

A continuación, podemos comprobar en un lector de puerto serie como Putty que se están leyendo los datos correctamente.

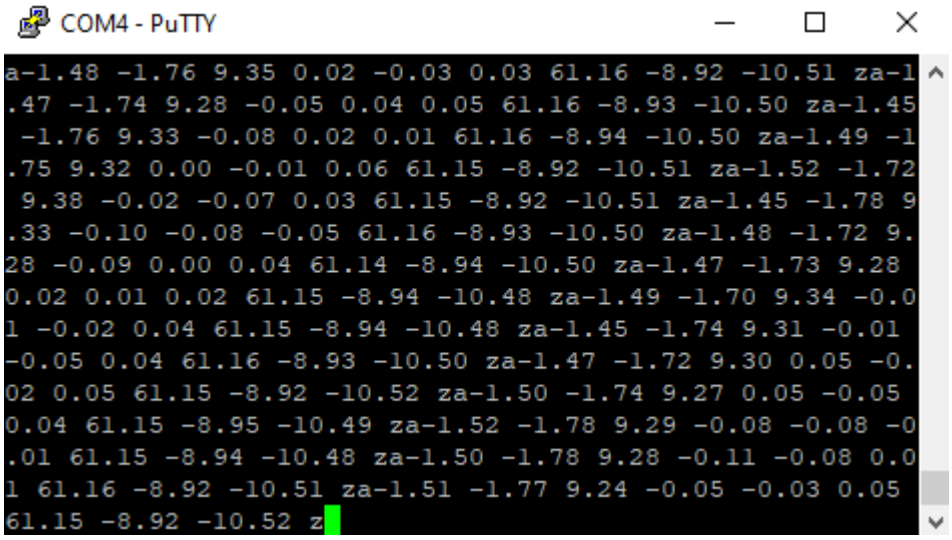


Figura 16. Valores leídos en Putty

Ahora que sabemos que los datos son recibidos por el PC, podremos instalar una máquina virtual como VirtualBox VM y una imagen de Ubuntu 20.0.4 para empezar a desarrollar el algoritmo dentro de un entorno Linux similar al de la RaspBerry Pi. De esta manera, en el futuro podremos transferir el archivo jar del algoritmo al entorno portable.

5.4 LECTURA DE DATOS EN JAVA

Para desarrollar el algoritmo utilizaré Java como lenguaje de programación, ya que existen varias librerías para leer los datos de los puertos serie del PC. En concreto, utilizaré la librería *jSerialComm*, por ser una versión más sencilla y renovada de la *RxTx* y de la ya deprecada *Java Communications API* (Fazecast Inc, 2020). Además, esta librería es muy sencilla de utilizar, pues solo hace falta descargar el archivo jar correspondiente y añadirlo al entorno de desarrollo, en este caso Eclipse.

5.5 MATRIZ DE DATOS

Para poder contar con los inputs necesarios, almacenaremos en una lista los últimos 15 vectores de valores recibidos por el puerto serie. De esta manera, seremos capaces de generar una matriz de 15 x 9 con toda la información de los últimos 1.5 segundos detectada por el sensor, ya que cada 0,1 segundos se envían nuevos datos. Esto nos permitirá averiguar los patrones seguidos por las caídas producidas en robots bípedos y en personas.

La matriz de datos tiene la siguiente forma.

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-

Tabla 4. Ejemplo de la Matriz de Datos

Esta matriz será actualizada continuamente mediante eventos Java gracias a la librería *jSerialComm*. Para poder ejecutar el algoritmo de análisis de manera continua sin que se produzca un corte en la información proporcionada, será necesario ejecutar el algoritmo a la mitad de tiempo aproximadamente del tiempo que registramos información de una caída, es decir, cada 0.7 segundos. De esta manera se obtiene una continuidad en el análisis y somos capaces de analizar efectivamente un tiempo completo de 1.5 segundos, tiempo más que suficiente para detectar una caída.

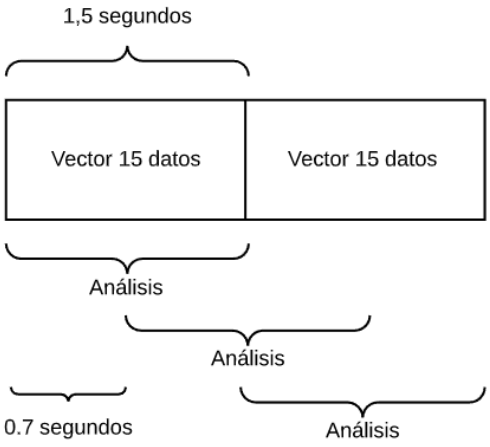


Figura 17. Análisis de Datos

Capítulo 6. ANÁLISIS DE CAÍDAS EN ROBOTS BÍPEDOS

En el caso de la detección de caídas en robots, únicamente nos fijaremos en los valores de posicionamiento, ya que, al querer prevenir la caída, es importante que detectemos una caída al inicio de esta y en esos instantes la aceleración no es muy significativa.

Nos centraremos en el caso concreto de utilizar un Robotis OP2, en el que se ha medido que, para registrar una serie de movimientos concretos, un ángulo en el tronco de más de 30 grados respecto al eje vertical de posición nos indica un estado de inestabilidad en el que el robot está iniciando su caída.

Tomamos el valor de 30º porque es un punto de no retorno en el grupo de movimientos estudiados en el movimiento del modelo Robotis OP2, ya que antes de la llegada del covid-19 era el modelo con el que se estaba trabajando físicamente. El Robotis OP2 para hacer las pruebas fue prestado por la universidad de Illinois en Urbana-Champaign.

Asumimos pues que este ángulo es suficiente para permitir una reacción a tiempo por parte del sistema utilizado para prevenir el impacto de la caída. Dado que este ángulo variara de un modelo a otro, es importante saber con qué modelo se está trabajando y calcular el ángulo del punto de no retorno en las caídas pada cada caso.

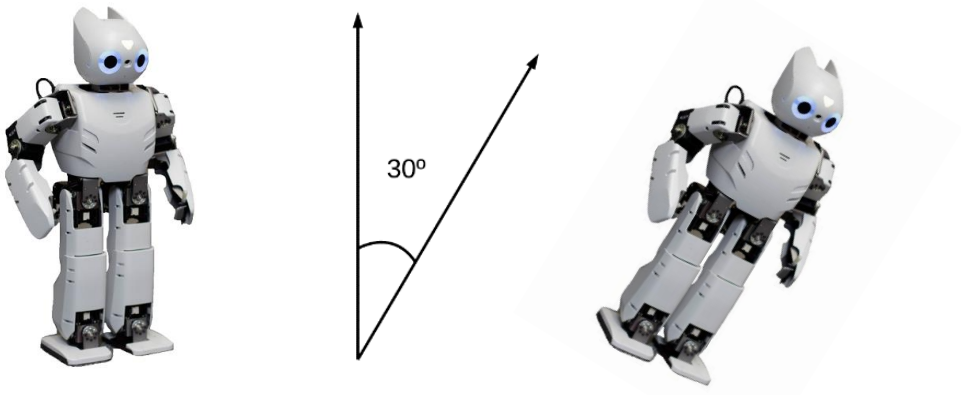


Figura 18. Ángulo de no Retorno en la Caída del Robotis OP2

Capítulo 7. ANÁLISIS DE CAÍDAS EN PERSONAS

En el caso de detectar una caída en las personas, nos fijaremos en los patrones que comúnmente se dan cuando una persona se cae. Esta parte es más compleja ya que existen diversos tipos de caídas.

En los sistemas de detección de caídas ya utilizados en *smartbands*, se registran caídas en función de los distintos movimientos realizados por las personas. En el caso del Apple Watch los movimientos que registran una caída son caer, tropezar y resbalar (*fall*, *trip*, y *slip*). Estos movimientos quedan registrados en base a los movimientos realizados por la muñeca, que es donde se encuentran todos los sensores de las *smartbands*. Actualmente existen estudios como (Pilla, 2009) sobre cómo tratar de evitar la exposición a estas caídas en lugares públicas mediante la implementación de medidas de seguridad.

Figura 19. Fall, Trip, Slip [16]

Sin embargo, la detección de caídas a través de movimientos detectados por sensores en la muñeca conlleva la posibilidad de detectar falsos positivos, ya que es común realizar movimientos bruscos y pronunciados por ejemplo en actividades deportivas o de ocio. Por eso, si contamos con un sistema que reciba y analice la trayectoria no de la muñeca sino del tronco humano, la detección sería más fiable, puesto que el posicionamiento del cuerpo detecta con mayor precisión la trayectoria del cuerpo más que el del brazo o la muñeca. Y es ahí, precisamente donde ubicaremos en las personas el sensor MPU-6050, en el tronco, al igual que para los robots bípedos.

Para detectar la caída de una persona nos basaremos en que, en cualquier caída, se repiten una serie de patrones inevitables. Si somos capaces de identificar los estados por los que pasa una persona cuando se cae, podremos definir el algoritmo con la seguridad de detectar la caída de una persona.

Para identificar los patrones, hemos de buscar qué tienen en común todas las caídas. Para ello, analizaremos con nuestro sensor los 3 tipos de caídas definidas por Apple.
Realizaremos un análisis de cada tipo de caída situando el MPU-6050 en el pecho de una persona con una altura estándar de 1,70 metros de altura. Alargaremos los cables dupont con más conexiones para permitir registrar la caída entera sin que el sensor se desconecte de la Arduino.

En la figura siguiente podemos ver la posición del sensor respecto del cuerpo desde una perspectiva frontal y lateral.

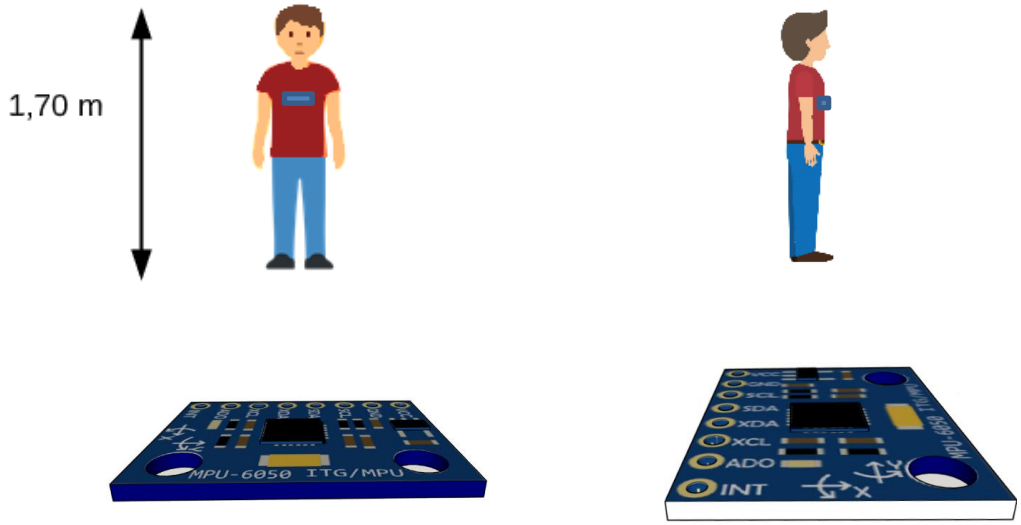


Figura 20. Perspectiva frontal y lateral del MPU 6050 [3]

Para analizar los distintos tipos de caídas, plantearemos varios modelos que simplifiquen la realidad para ver si estos pueden ser utilizados a la hora de analizar el patrón de las caídas que nos sirva para desarrollar el algoritmo.

7.1 TRIP

Este tipo de caídas son normalmente provocadas por un obstáculo inesperado en el camino de la persona, tales como un bordillo, un escalón o cualquier objeto sólido que está fijo y en nuestra trayectoria. En nuestro ejemplo lo representaremos con una piedra.

A continuación, analizamos el comportamiento esperado de una caída por tropiezo, simplificada en 5 momentos representativos, y representamos los vectores de velocidad sobre el centro de masa y el posicionamiento. Al representar la caída de manera totalmente frontal, podemos utilizar únicamente 2 dimensiones en este ejemplo.

Conviene recordar que las aceleraciones registradas por el sensor no tienen por qué coincidir exactamente con las aceleraciones sufridas por la persona, ya que estamos concentrando el registro en un punto situado en el tronco. Sin embargo, como el centro de masa de la persona se halla cercano a este, podemos suponer que los valores obtenidos son una buena aproximación.

Como reglas, recordemos que, cuando el centro de gravedad cae en la vertical fuera de la base del cuerpo, este tenderá a caer, y será el momento en el que habrá un punto de no retorno en la caída. Además, dado que estamos ante un campo gravitatorio uniforme, podemos asumir que el centro de masa equivale al centro de gravedad.

7.1.1 ANÁLISIS TEÓRICO

Esta caída se caracteriza por tener vectores de velocidad equivalentes a los de un tiro parabólico más los de un movimiento circular uniformemente acelerado sobre el centro de masa, que asumimos se encuentra en un punto cercano al lugar del sensor.

La caída empieza con el tropiezo con un obstáculo mientras la persona anda hacia adelante. En ese instante, el cuerpo comienza a rotar manteniendo un centro de giro en el tronco inferior, por lo que el tronco superior se desplaza con más velocidad que el inferior.

Por otro lado, se produce un movimiento parabólico en el que el tronco superior avanza con velocidad constante hacia adelante y con una velocidad cada vez mayor hacia abajo debido a la aceleración de la gravedad.

En este tipo de caídas, es frecuente que la persona que se está cayendo utilice las manos para minimizar el impacto al final de la caída. En el momento en el que sacamos las manos hacia adelante, que suele ser a mitad del giro, el cuerpo tiende a reducir la velocidad del giro, ya que, por la 3ª ley de Newton, se ejerce una fuerza en el cuerpo contraria al sentido de la extensión de las manos de la misma magnitud.

En el instante final, nos fijamos en que el cuerpo impacta con una velocidad que será decelerada por el golpe contra el suelo, que puede dejar inconsciente a la persona que se cae.

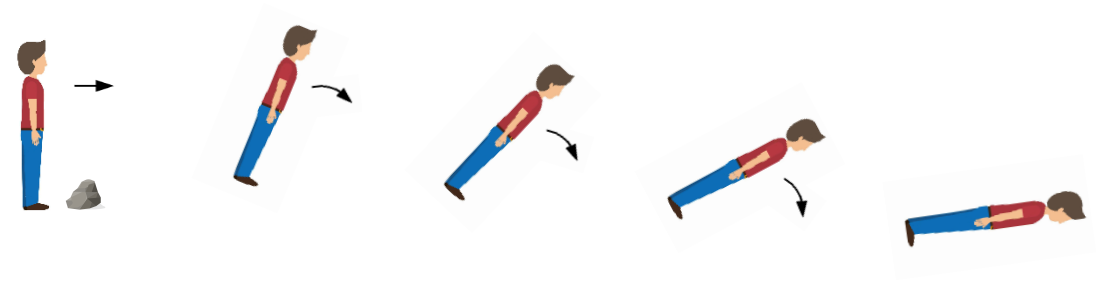


Figura 21. Posicionamiento de la persona en Trip

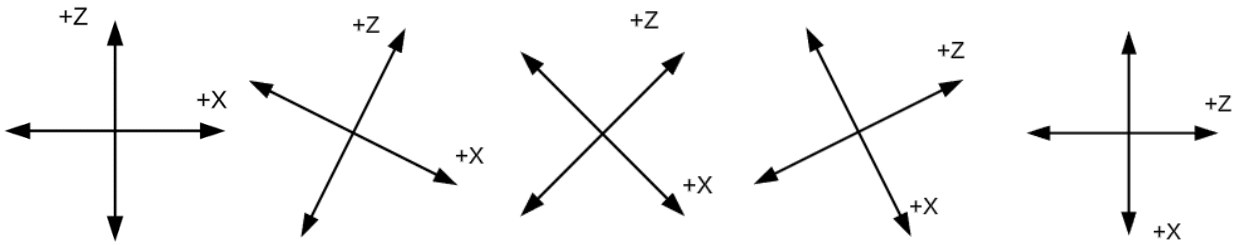


Figura 22. Posicionamiento del MPU 6050 en Trip

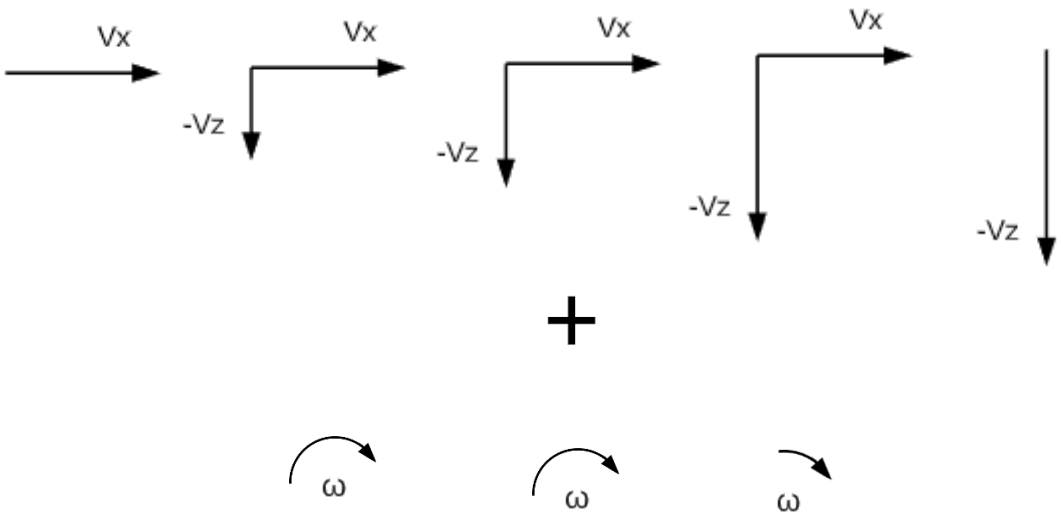


Figura 23. Vectores de Velocidad en Trip

Los valores aproximados que esperamos de una caída frontal ideal de tipo tropezar son los siguientes:

Ax	*Ay*	*Az*	*Gx*	*Gy*	*Gz*	*Yaw*	*Pitch*	*Roll*
0	0	9.8	0	+	0	X	0	0
--	0	--	0	++	0	X	--	0
--	0	--	0	+	0	X	--	0
--	0	--	0	-	0	X	--	0
-9.8	0	0	0	--	0	X	-90	0

Tabla 5. Valores estimados en Trip

- Ax: Parte de un valor de 0 y disminuye progresivamente hasta -9.8, es decir, el valor de la aceleración de la gravedad.
- Ay: Permanece a 0 puesto que no hay movimiento a lo largo del eje y
- Az: Parte de +9.8 y disminuye progresivamente hasta 0
- Gx: Permanece a 0 puesto que no hay giros en el eje de giro x
- Gy: Aumenta al principio y disminuye al final debido al impacto
- Gz: Permanece a 0 puesto que no hay giros en el eje de giro z
- Yaw: Es indiferente, permanece constante
- Pitch: Disminuye progresivamente desde 0 en la posición vertical del cuerpo hasta -90 en la posición horizontal.
- Roll: Permanece a 0 puesto que no hay movimiento de giro en x

7.1.2 ANÁLISIS EXPERIMENTAL

A continuación, tomamos 3 muestras representativas al azar de un grupo de caídas de este tipo y analizamos el experimento:

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
0,19	0,14	10,14	0,4	4,3	0,81	102,09	-3,43	2,34
2,88	-1,27	7,1	-0,29	7,23	8,97	102,05	-4,1	2,56
0,99	-1,74	7,56	-4,24	4,97	7,83	101,67	-4,96	2,73
-0,56	-0,09	5,26	6,97	2,72	0,18	87,88	-14,76	1,73
-8,52	2,42	3,42	7,25	20,01	6,32	87,95	-15,13	2,68
10,78	-4,5	3,2	2,48	11,18	2,27	88,24	-15,76	3,78
-15,87	-3,47	19,62	1,79	-16,1	-3,62	88,54	-16,54	4,62
-19,62	19,62	5,71	21,24	18,31	9,99	91,63	-59,57	6,74
-10,91	10,66	-0,67	-16,44	15,73	4,03	87,92	-62,91	9,9
-1,52	3,07	-2,98	-1,11	-2,78	-2,54	84,88	-62,02	10,39
-5,94	1,28	1,97	1,04	0,56	-3,94	81,07	-60,74	11,89
-10,07	3,36	0,88	2,75	-1,57	-2,46	74,41	-71,55	12,23
-8,75	1,56	2,41	1,24	-2,26	-2,89	74,68	-71,76	12,01
-8,75	2,61	-0,01	0,52	0,35	0,61	74,89	-72,03	10,78
-9,2	2,56	0,32	0,51	-0,77	-2,02	75	-72,32	10,59

Tabla 6. Matriz de Datos del primer ejemplo de Trip

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
-1,09	-0,78	9,52	0,13	1,8	0,98	56,38	-6,71	-3,84
-0,7	-0,58	9	-0,08	1,17	2,33	56,28	-6,89	-3,83
-1,26	-0,28	9,16	1,54	2,93	2,7	56,18	-8,04	-3,85
-2,33	-0,79	9,34	0,76	5,12	1,18	56,09	-10,18	-3,91
-0,67	-1	7,91	2,62	5,2	0,99	51,54	-22,02	0,88
-1,36	0,45	6,65	3,93	10,68	1,43	51,54	-22,5	1,15
-3,82	2,32	3,08	0,92	11,41	5,85	51,36	-23,17	1,55
10,47	-1,19	5,28	-9,4	6,53	10,46	50,99	-24,35	2,29
-19,62	15,1	-15,52	6,94	17,56	-2,72	46,78	-48,17	3,24
-19,62	13,92	-5,05	23,55	-87,87	14,02	48,01	-51,57	4,15
-0,6	-9,31	3,83	-7,83	21,96	-13,98	47,04	-54,12	2,96
-19,47	1,55	0,46	5,16	23,62	5,11	46,02	-59,15	5,6
-5,44	2,62	1,56	6,02	5,32	-6,39	68,82	-72,66	9,05
-11,92	3,73	-1,9	-1,49	4,3	3	71,21	-74,42	8,48
-7,49	3,5	-2,73	-2,5	-5,58	2,21	73,78	-75,92	8,08

Tabla 7. Matriz de Datos del segundo ejemplo de Trip

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
-1,59	0,16	9,57	0,31	1,03	0,18	-35,74	-7,76	3,55
-2,47	0,78	9,5	0,56	2,06	0,48	-35,82	-7,75	3,47
-1,76	0,76	8,94	0,24	2,96	0,77	-35,93	-7,79	3,44
-2,39	0,73	8,4	1	5,72	0,85	-38,6	-15,48	5,06
-3,04	0,72	7,74	1,33	10,26	0,88	-38,84	-16,13	5,23
-4,61	0,95	7,05	1,69	8,47	-1,37	-39,16	-16,87	5,42
-4,85	1,49	4,64	6,57	14,35	0,92	-39,5	-17,67	5,62
-3,91	2,82	-0,07	5,85	14,01	-2,48	-50,23	-49,56	7,88
-14,54	2,71	4,23	2,65	5,28	0,83	-50,84	-58,18	8,11
-7,98	3,17	0,43	2,97	10,88	8,24	-55,25	-59,42	8,4
-19,62	13,07	6,81	-12	-9,33	-2,37	-61,66	-60,36	8,72
-19,62	4,28	-19,62	-6,58	10,82	12,91	-71,45	-78,11	9,8
-9,46	0,9	0,19	-8,71	0,6	1,55	-85,72	-75,57	12,48
-10,47	-2,08	2,58	-2,71	-6,83	-4,58	-91,4	-70,21	13,34
-8,44	4,46	-4,56	-3,88	4,18	2,17	-92,19	-68,57	13,83

Tabla 8. Matriz de Datos del tercer ejemplo de Trip

7.1.3 OBSERVACIONES

- Ax: Como esperábamos, parte de valores cercanos a 0 y termina en valores cercanos a -9.8. Las variaciones iniciales pueden producirse debido a la postura corporal natural de la persona. Observamos como en el tramo final se registran valores muy negativos antes de estabilizarse al valor de la gravedad, esto se debe al impacto en el suelo, que es muy variable entre cada caída.
- Ay: Observamos valores distintos de 0 aunque reducidos puesto que la caída se ha realizado intencionadamente hacia delante.
- Az: Como esperábamos, parte de valores cercanos a 9.8 y termina en valores cercanos a 0. Al igual que en Ax, se observan los datos del impacto antes del impacto.
- Gx: Igual que Ay.
- Gy: Como esperábamos, observamos valores positivos al principio y algunos negativos al final cuando el cuerpo deja de rotar.
- Gz: Igual que Ay.
- Yaw: Existen ligeras variaciones debido a los movimientos de cuerpo durante la caída y posterior al impacto. A veces incrementa o disminuye levemente dependiendo del sentido hacia el cual esté girando el cuerpo.
- Pitch: Disminuye considerablemente como esperábamos.
- Roll: Igual que Ay.

7.1.4 PATRÓN

Identificando los datos más significativos comunes en todas las caídas podemos definir un patrón para una caída frontal del tipo tropezar.

- Partimos desde una posición de pie en la que la mayoría de la aceleración de la gravedad es soportada por el eje Az, mientras que los ejes Ax y Ay son considerablemente inferiores. Además, el eje de Yaw tiene total libertad, mientras que Pitch y Roll no superan los 45º en situaciones normales.
- Al iniciar la caída se producen giros positivos en el eje de giro y
- Se supera un ángulo de 45º respecto del eje vertical del cuerpo, en este caso en Pitch.
- Durante la caída se produce un intercambio de aceleraciones entre los ejes Ax y Az del valor de 9.8, aunque es precisamente en torno a los 45º cuando podemos confirmar que Az se reduce notablemente,
- En todas las caídas se produce un impacto, en los cuales registramos aceleraciones cercanas a 19.62, valor límite de nuestro sensor. No obstante, el impacto es el momento más impredecible, ya que el más mínimo giro determina que eje absorbe la deceleración del impacto.
- Al final de la caída, podemos saber que la persona permanece en posición casi horizontal dado el inclinado ángulo de Pitch y que la gravedad es repartida entre los ejes x e y.

Con esta información, podemos definir un patrón de estados con los elementos que podemos predecir en una caída frontal del tipo tropezar.

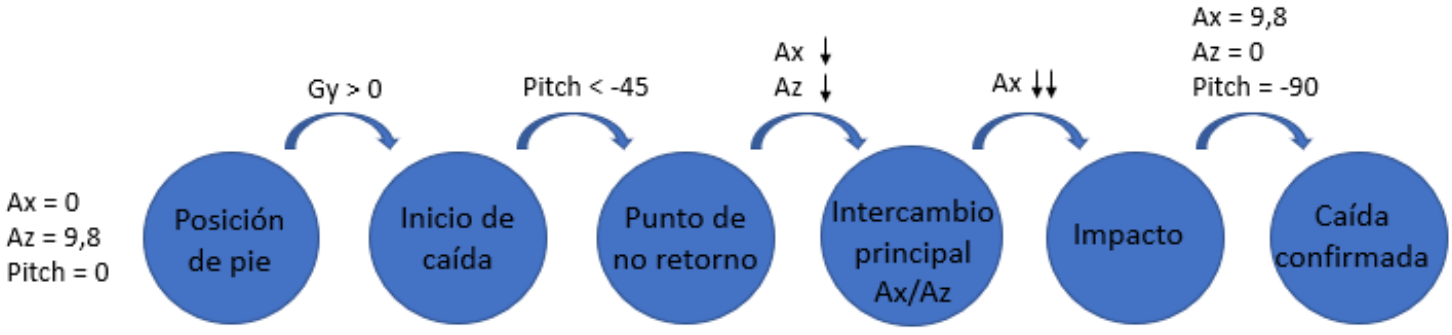


Figura 24. Patrón de Trip

7.2 SLIP

En este caso, analizaremos la caída del tipo resbalar hacia atrás. Para ello repetiremos el mismo planteamiento que en el caso anterior, primero realizaremos el análisis teórico del comportamiento físico esperado de la caída y posteriormente verificaremos la veracidad de la teoría con un modelo experimental.

Este tipo de caídas suele ser provocado por pisar sobre un objeto o suelo resbaladizo, lo cual hace que perdamos el equilibrio y realicemos un giro brusco antes de caer al suelo. En la cultura de los videojuegos y animaciones, este tipo de objeto suele representarse con una piel de plátano, tal y como reflejamos en este ejemplo.

Esta caída se caracteriza porque a diferencia de las caídas del tipo tropezar, el centro de gravedad permanece en el mismo sitio en los instantes iniciales de la caída. Esto se debe a que en primer lugar se realiza el movimiento de rotación del cuerpo, en este caso hacia atrás seguido de la caída hacia abajo provocada por la gravedad. Es decir, el movimiento tiene una fase de rotación donde la persona pierde la estabilidad, y otra fase de caída vertical.

Cabe destacar que, en la realidad, estos modelos nunca se producirán en condiciones ideales, sino que, aunque una caída se pueda aproximar más a un tipo u otro, siempre será una combinación de distintos tipos de caídas.

7.2.1 ANÁLISIS TEÓRICO

En primer lugar, al pisar sobre algo resbaladizo el cuerpo tiende a girar con un centro de rotación situado aproximadamente en la mitad del cuerpo, es por eso por lo que en las dos primeras imágenes observamos al cuerpo rotar con violencia y perdiendo el equilibrio.

A continuación, las personas tendemos por naturaleza a reaccionar frente a este evento y para compensar este desequilibrio balanceamos los brazos hacia atrás para estabilizar un poco nuestra caída y frenar parcialmente el movimiento de giro. De nuevo es la misma idea de la 3ª ley de Newton que en el caso 1 cuando sacábamos los brazos hacia adelante.

Sin embargo, en esos momentos ya se ha producido el punto de no retorno en la caída y el cuerpo comienza un movimiento de caída libre hacia abajo y manteniendo parte de la inercia del giro inicial hasta que finalmente impacta contra el suelo.

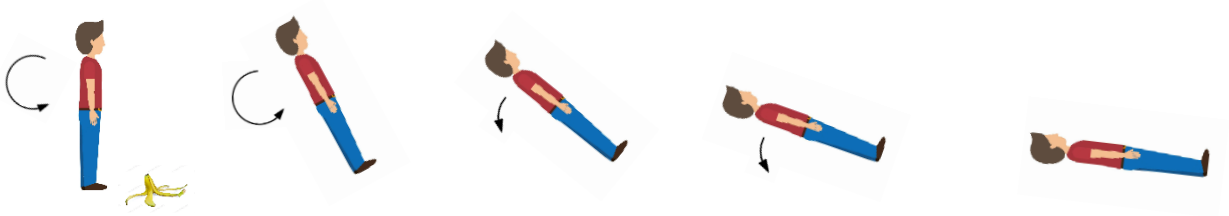


Figura 25. Posicionamiento de la persona en Slip

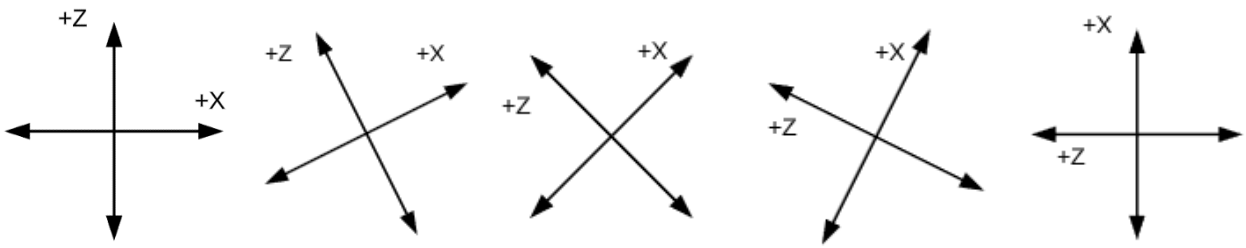


Figura 26. Posicionamiento del sensor en Slip

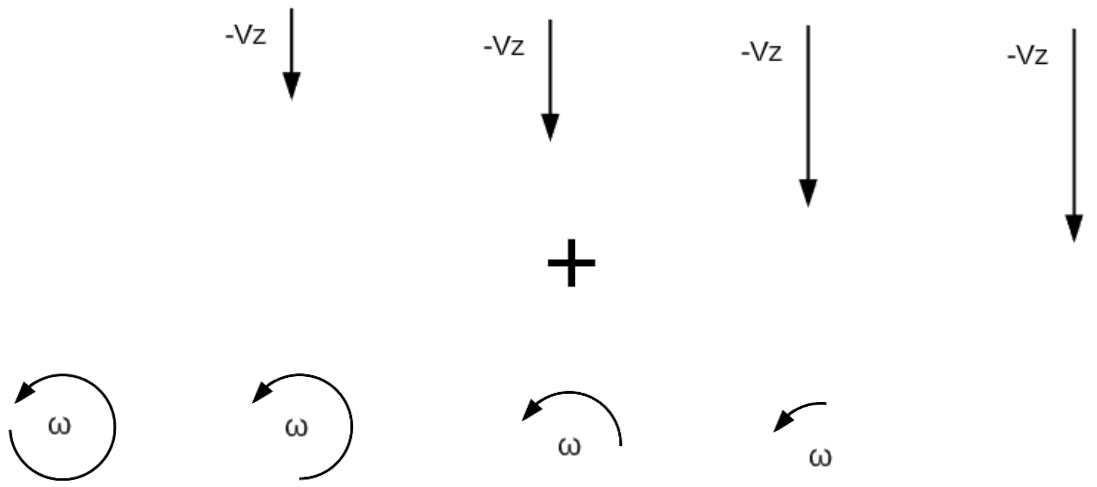


Figura 27. Vectores de velocidad en Slip

Los valores esperados de una caída hacia atrás ideal de tipo resbalar son los siguientes

Ax	***Ay***	***Az***	***Gx***	***Gy***	***Gz***	***Yaw***	***Pitch***	***Roll***
0	0	9.8	0	--	0	X	0	0
++	0	--	0	--	0	X	++	0
++	0	--	0	-	0	X	++	0
++	0	--	0	+	0	X	++	0
9.8	0	0	0	+	0	X	90	0

Tabla 9. Valores estimados en Slip

- Ax: Parte de un valor de 0 y aumenta progresivamente hasta +9.8, es decir, el valor de la aceleración de la gravedad.
- Ay: Permanece a 0 puesto que no hay movimiento a lo largo del eje y
- Az: Parte de +9.8 y disminuye progresivamente hasta 0
- Gx: Permanece a 0 puesto que no hay giros en el eje de giro x
- Gy: Disminuye al principio bruscamente y aumenta al final
- Gz: Permanece a 0 puesto que no hay giros en el eje de giro z
- Yaw: Es indiferente, permanece constante
- Pitch: Aumenta progresivamente desde 0 en la posición vertical del cuerpo hasta +90 en la posición horizontal.
- Roll: Permanece a 0 puesto que no hay movimiento de giro en x

Como vemos, la única diferencia esperada respecto al caso 1 es que la aceleración en Ax esta vez progresará hacia valores positivos, los giros en Gy deberán ser negativos y mucho más pronunciados al inicio, y la orientación de Pitch irá aumentando en vez de disminuyendo.

7.2.2 ANÁLISIS EXPERIMENTAL

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
1	-0,65	11,26	0,56	2,01	0,53	-73,63	7,2	-0,39
0,67	-1,14	9,7	-0,85	-3,82	2,04	-67,11	7,76	1,72
0,57	-2,02	5,83	-4,03	-1,93	-2,89	-67,32	12,18	1,52
2,65	0,51	10,02	-6,61	-4,75	-1,59	-67,3	12,53	1,41
1,79	-0,03	8,57	-2,04	-9,22	-5,71	-67,22	13,92	1,29
4,82	0,41	7,63	2,63	-7,57	-3,32	-60,19	40,74	-1,53
4,59	0,53	3,1	-0,5	-35,95	4,73	-60,4	41,82	-1,19
6,13	2,29	6,18	2,1	-30,96	10,27	-61,05	42,05	-1,41
19,62	4,93	-17,51	26,1	20,48	12,54	-61,44	42,79	-1,88
15,89	-16,07	3,38	6,14	9,53	-5,75	-92,96	71,84	-7,03
5,42	1,59	-14,82	-3,94	-3,18	-7,94	-92,63	71,35	-6,76
5,5	-2,22	-8,23	-10,2	-8,57	0,35	-91,86	71,89	-6,02
10,03	-3,76	-1,72	-0,83	-3,86	-3,43	-91,48	73,91	-5,61
13,06	1,7	1,84	2,38	1,69	-2,27	-97,11	80,59	-3
10,76	1,14	-0,11	1,14	-3,62	2,49	-96,53	81,17	-2,67

Tabla 10. Matriz de Datos del primer ejemplo de Slip

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
-1,46	-0,98	9,61	-0,23	0,14	1,08	22,01	-9,05	-9,59
-1,82	-0,47	9,44	0,53	-0,27	2,55	22,06	-9	-9,6
-0,56	0,08	9,46	2,38	-4,26	1,11	19,87	-8,21	-7,9
-0,84	-2,47	9,18	-0,54	-14,7	-2,97	19,67	-7,66	-7,47
1,87	-0,18	0,78	2,17	-9,15	-18,23	19,28	-7,13	-7
7,52	0,17	11,12	2,09	-16,88	-6,87	18,72	-6,3	-6,53
6,03	-0,52	5,09	-0,78	-17,51	8,46	43,71	37,57	-5,35
9,49	3,68	2,62	4,17	-4,25	5,97	43,11	39,37	-6,24
4,9	2,48	8,67	4,2	-42,46	3,46	42,43	40,83	-7,28
19,62	13,69	-9,32	-21,84	-30,7	-12,89	42,13	42,19	-8,18
8,07	8,61	-6,78	-12,18	-10,07	6,67	86,8	82,19	5,79
13,62	-0,33	-19,62	-2,17	2,2	4,68	89,83	83,45	5,55
14,02	2,78	-1,34	5,74	7,46	1,15	94,06	84,44	5,49
13,98	3,75	3,36	2,84	2,81	0,35	96,72	84,21	5,75
9,03	0,28	1,75	-1,05	3,01	-2,5	94,41	86,35	3,61

Tabla 11. Matriz de Datos del segundo ejemplo de Slip

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
-0,63	-1,02	10,9	0,14	1,14	0,63	-15,3	-4,86	-6,47
-0,92	-1,1	9,35	0,22	0,4	0,89	-15,38	-4,95	-6,44
-0,31	-0,84	9,76	0,02	-1,29	-0,18	-15,49	-4,95	-6,44
0,16	-0,67	12,24	3,13	-9,31	-2,2	-15,53	-4,88	-6,42
0,23	0,88	-0,42	3,14	-16,83	-9,51	-9,56	10,08	-7,27
0,04	-1,88	5,79	-1,43	-16,76	-5,73	-8,7	11,77	-6,67
-3,58	-5,94	9,15	-1,14	-22,23	5,1	-8,08	13,53	-6,12
-0,21	-0,16	-2,18	2,34	-14,55	-6,64	-7,64	15,29	-5,8
-9,47	-19,62	19,62	20,46	-30,73	27,6	-26,98	44,82	-1,99
5,53	10,02	10,08	-7,08	1,32	-29,09	-32,25	47,34	-3,61
12,04	1,07	-5,99	4,3	0,49	1,69	-33,31	50,25	-3,68
7,25	1,26	-1,62	-0,74	9,69	-1,08	-32,05	56,63	-2,64
12,86	2,71	3,92	0	1,37	-1,58	-58,67	82,95	6,7
13,88	3,45	2,29	-2,91	-2,73	-3,94	-55	82,66	6,88
11,7	1,6	-0,36	0,69	-2,46	-0,24	-53,01	82,74	6,68

Tabla 12. Matriz de Datos del tercer ejemplo de Slip

7.2.3 OBSERVACIONES

- Ax: Como esperábamos, parte de valores cercanos a 0 y termina en valores cercanos a 9.8. Las variaciones iniciales pueden producirse debido a la postura corporal natural de la persona. Observamos como en el tramo final se registran valores muy positivos antes de estabilizarse al valor de la gravedad, esto se debe al impacto en el suelo, que es muy variable entre cada caída.
- Ay: Observamos valores distintos de 0 aunque reducidos puesto que la caída se ha realizado intencionadamente hacia atrás. En el momento del impacto el eje y ha absorbido parte de la deceleración.
- Az: Como esperábamos, parte de valores cercanos a 9.8 y termina en valores cercanos a 0. Al igual que en Ax, se observan los datos del impacto antes del impacto.
- Gx: Igual que Ay.
- Gy: Como esperábamos, observamos valores muy negativos al principio y algunos negativos al final cuando el cuerpo deja de rotar.
- Gz: Igual que Ay.
- Yaw: Existen ligeras variaciones debido a los movimientos de cuerpo durante la caída y posterior al impacto. A veces incrementa o disminuye levemente dependiendo del sentido hacia el cual esté girando el cuerpo.
- Pitch: Aumenta considerablemente como esperábamos en un rango aproximado de 0 a 90 grados.
- Roll: Igual que Ay.

7.2.4 PATRÓN

Si nos fijamos bien, los casos 1 y 2 son muy similares. La única diferencia es que mientras que en el caso 1 existe velocidad en el eje horizontal y vertical debido a la trayectoria parabólica, en el caso 2 existe un movimiento de rotación del cuerpo sobre sí mismo y posteriormente una caída libre en dirección vertical hacia abajo.

Por tanto, para representar el caso 2 podemos utilizar un patrón muy similar al del primer caso.

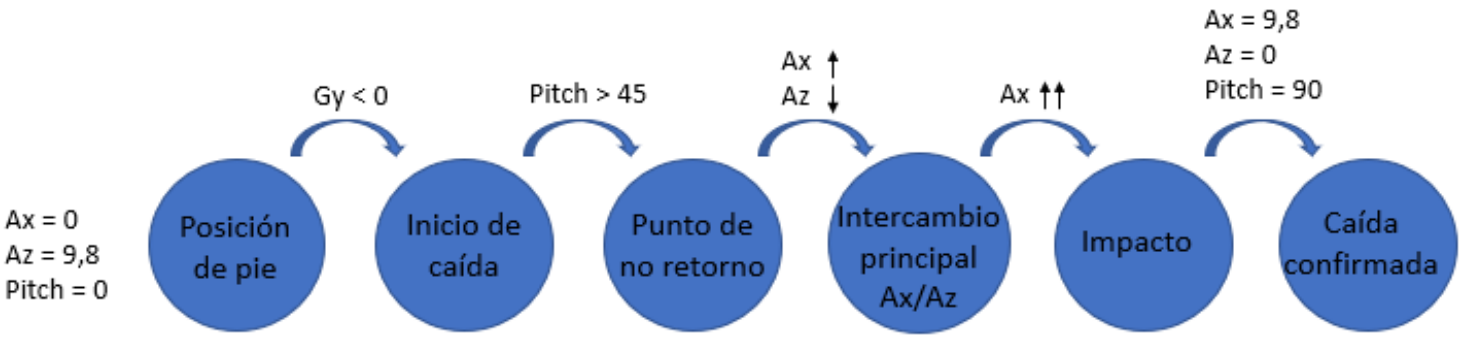


Figura 28. Patrón de Slip

7.3 FALL

También cabe la posibilidad de que primero se produzca la caída hacia abajo y posteriormente el cuerpo se gire hasta quedar en el suelo. Este es el caso de Fall, producido en situaciones en las que, en vez de tropezar o resbalar con algo, es el propio cuerpo el que responde de manera anormal, por ejemplo, al sufrir un calambre en las piernas o al padecer una lesión de rodilla o de cadera, muy frecuentes en edades avanzadas.

En estos casos el movimiento producido es inverso al anterior, es decir, en vez de producirse un giro seguido de una caída libre, primero se produce una caída brusca del cuerpo en dirección vertical hacia abajo flexionando las rodillas y finalmente se produciría el giro del tronco y la caída correspondiente.

7.3.1 ANÁLISIS TEÓRICO

Los valores esperados son muy parecidos a los anteriores, pero en este caso durante las posiciones 1 y 2 se produce un movimiento de caída libre seguido por un movimiento de rotación en las posiciones 3, 4 y 5. El impacto en este tipo de caída es menos intenso que en las caídas tropezar y resbalar, esto se debe a que el impacto se produce en dos tiempos. Primero, cuando el cuerpo se desploma, cae hasta que las rodillas, los muslos o la cadera, en función de la dirección de la caída, impactan en el suelo frenando la segunda parte de la caída. A continuación, la parte superior del cuerpo termina de caer rotando hasta impactar con el suelo.

El cuerpo realiza menos recorrido que en los casos anteriores, ya que no se trata de un movimiento parabólico como en el caso 1, ni de un movimiento de rotación seguido por una caída libre con todo el cuerpo como en el caso 2. Este caso es el inverso al caso 2, se produce una caída libre seguido de un movimiento de giro únicamente del tronco superior, es por eso por lo que este tipo de caídas, aunque suelen ser las más habituales en personas mayores por fallos en la musculatura, son los menos fuertes, pero igualmente muy peligrosos.

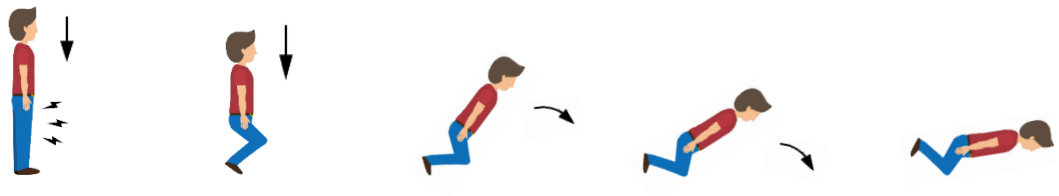


Figura 29. Posicionamiento de la persona en Fall

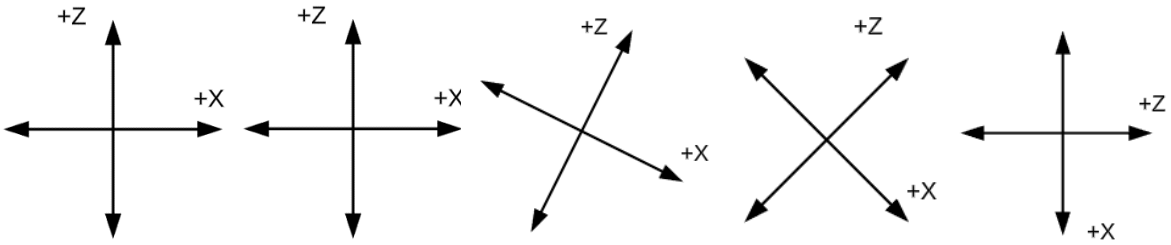


Figura 30. Posicionamiento del sensor en Fall

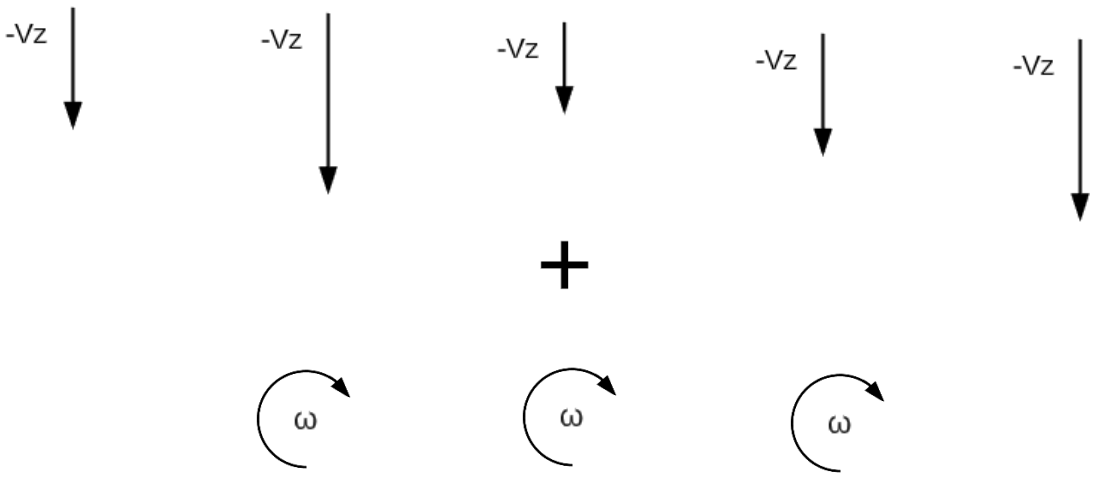


Figura 31. Vectores de velocidad en Fall

Los valores esperados de una caída hacia adelante ideal del tipo caer son los siguientes

Ax	*Ay*	*Az*	*Gx*	*Gy*	*Gz*	*Yaw*	*Pitch*	*Roll*
0	0	9.8	0	0	0	X	0	0
--	0	--	0	0	0	X	0	0
--	0	--	0	+	0	X	--	0
--	0	--	0	+	0	X	--	0
-9.8	0	0	0	-	0	X	-90	0

Tabla 13. Valores esperados en Fall

- Ax: Parte de un valor de 0 y disminuye progresivamente hasta -9.8, es decir, el valor de la aceleración de la gravedad.
- Ay: Permanece a 0 puesto que no hay movimiento a lo largo del eje y
- Az: Parte de +9.8 y disminuye progresivamente hasta 0
- Gx: Permanece a 0 puesto que no hay giros en el eje de giro x
- Gy: Permanece a 0 al principio durante la caída libre y aumenta cuando pasamos a la segunda fase de la caída. Al final encontramos algún valor negativo cuando se deja de girar.
- Gz: Permanece a 0 puesto que no hay giros en el eje de giro z
- Yaw: Es indiferente, permanece constante
- Pitch: Al principio permanece en valores cercanos a 0 en la primera fase de caída libre y posteriormente disminuye progresivamente desde 0 en la posición vertical del cuerpo hasta -90 en la posición horizontal.
- Roll: Permanece a 0 puesto que no hay movimiento de giro en x

Como vemos, los cambios más significativos están en Gy, donde los primeros valores han de ser cercanos a 0 porque nos encontramos en un movimiento de caída libre en la primera fase donde no hay giros. Como consecuencia pitch permanece también a 0 en la primera fase y disminuye progresivamente a en la segunda fase.

7.3.2 ANÁLISIS EXPERIMENTAL

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
1,78	0,2	9,14	-0,42	-2,62	0,54	-3,34	6,91	3,25
1,76	-0,76	5,01	1,97	4,1	1,08	-3,71	6,82	0,48
2,67	0,29	1,75	-0,71	8,13	-2,37	-3,95	6,46	0,62
0,24	0,84	6,86	-1,54	12,35	-4,07	-4,2	6,06	0,88
0,17	1,53	17,5	0,78	13,22	-0,9	-4,42	5,84	1,14
2,13	-0,36	12,47	4,45	20,9	2,98	-0,82	-8,46	4,74
-3,67	1,68	4,37	1,79	24,28	-0,62	-1,05	-10,79	5,4
-7,17	2	0,29	1,37	23,84	-4,1	-1,07	-13,76	5,55
-9,4	-1,01	0,11	-2,61	15,13	-3,36	-0,89	-17,25	6,89
-17,9	0,17	-0,39	6,79	9,9	-4,95	48,65	-85,71	1,16
-11,33	0,43	-1,24	-2,43	0,62	-4,01	62,79	-86,99	0,57
-10,34	-3,85	-3,56	0,46	-3,02	2,01	72,35	-87,85	0,03
-9,25	-0,12	0,41	-1,89	0,02	0,34	80,1	-88,67	-0,06
-9,8	2,24	-0,43	-1,14	2,3	-2,87	87,96	-86,45	-3,53
-9,74	-1,52	-0,37	-1,07	-3,17	-5,99	91,78	-86,45	-3,54

Tabla 14. Matriz de Datos del primer ejemplo de Fall

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
1,22	-0,11	8,83	1,19	-0,73	0,12	-0,85	5,71	-1,54
1,07	-0,16	9,37	-0,06	0,33	-0,05	-0,86	5,76	-1,55
1,31	-0,3	9,13	1,51	0,04	0,07	-0,88	5,89	-1,57
1,44	0,91	-0,74	-2,27	-2,26	-3,64	-0,53	2,11	1,81
1,24	0,35	2,45	-3,06	12,68	-1,85	-0,04	2,53	1,61
-0,5	-2,37	19,62	-4,14	12,94	4,3	0,23	2,79	1,39
4,76	-0,06	8,9	5,78	22,69	7,9	0,19	2,7	1,3
-3,09	1,33	-3,81	5,06	32,72	-2,03	-1,61	-40,41	4,49
-15,66	0,05	-5,9	8,08	38,74	-3,12	-0,89	-44,88	4,85
-11,33	-1,36	-1,87	-5,33	2,94	0,97	0,16	-49,89	5,17
-13,72	1,77	0,79	1,69	3,91	-1,22	1,64	-54,16	5,19
-9,9	1,04	-0,57	-1,87	-1,02	1,2	3,81	-85,44	4,53
-9,94	1,47	-1,06	-0,79	0,87	1,61	5,33	-85,28	4,7
-9,46	0,87	-0,46	0,43	1,39	3,02	13,2	-85,03	4,93
-9,81	1,73	-0,62	-1,48	-0,13	-0,37	15,69	-84,72	5,2

Tabla 15. Matriz de Datos del segundo ejemplo de Fall

Ax	Ay	Az	Gx	Gy	Gz	Yaw	Pitch	Roll
1,22	-0,11	8,83	1,19	-0,73	0,12	-0,85	5,71	-1,54
1,07	-0,16	9,37	-0,06	0,33	-0,05	-0,86	5,76	-1,55
1,31	-0,3	9,13	1,51	0,04	0,07	-0,88	5,89	-1,57
1,44	0,91	-0,74	-2,27	-2,26	-3,64	-0,53	2,11	1,81
1,24	0,35	2,45	-3,06	12,68	-1,85	-0,04	2,53	1,61
-0,5	-2,37	19,62	-4,14	12,94	4,3	0,23	2,79	1,39
4,76	-0,06	8,9	5,78	22,69	7,9	0,19	2,7	1,3
-3,09	1,33	-3,81	5,06	32,72	-2,03	-1,61	-40,41	4,49
-15,66	0,05	-5,9	8,08	38,74	-3,12	-0,89	-44,88	4,85
-11,33	-1,36	-1,87	-5,33	2,94	0,97	0,16	-49,89	5,17
-13,72	1,77	0,79	1,69	3,91	-1,22	1,64	-54,16	5,19
-9,9	1,04	-0,57	-1,87	-1,02	1,2	3,81	-85,44	4,53
-9,94	1,47	-1,06	-0,79	0,87	1,61	5,33	-85,28	4,7
-9,46	0,87	-0,46	0,43	1,39	3,02	13,2	-85,03	4,93
-9,81	1,73	-0,62	-1,48	-0,13	-0,37	15,69	-84,72	5,2

Tabla 16. Matriz de Datos del tercer ejemplo de Slip

7.3.3 OBSERVACIONES

- Ax: Como esperábamos, parte de valores cercanos a 0 y termina en valores cercanos a -9.8. Observamos como existen grandes variaciones a mitad y al final de la caída correspondiendo a la transición de la primera a la segunda fase y al impacto final.
- Ay: Observamos valores cercanos 0 puesto que la caída se ha realizado intencionadamente hacia adelante.
- Az: Como esperábamos, parte de valores cercanos a 9.8 y termina en valores cercanos a 0. Al igual que en Ax, se observan grandes picos en la transición de la caída vertical al movimiento de giro y en el impacto final.
- Gx: Igual que Ay.
- Gy: Como esperábamos, valores muy reducidos al principio donde apenas hay movimiento de giro en la fase de caída vertical, mientras que al inicio de la segunda fase observamos valores muy positivos ya que se produce una aceleración angular en el l inicio de la fase de giro.
- Gz: Igual que Ay.
- Yaw: Existen ligeras variaciones en cada fase y en ocasiones podemos observar una transición mayor en el cambio de fases en la caída.
- Pitch: Aumenta considerablemente como esperábamos en un rango aproximado de 0 a 90 grados.
- Roll: Igual que Ay.

7.3.4 PATRÓN

Al tratarse de una caída en dos tiempos, podemos diferenciar la transición entre ambas fases. Inicialmente el cuerpo se encuentra erguido, observamos que la aceleración en Ax es muy pequeña y que en Az es cercana a la gravedad.

A continuación, se produce la primera fase de caída libre hacia abajo, donde la aceleración en Az se reduce a valores cercanos a 0 y al frenar aumenta bruscamente. Es entonces cuando segunda fase de rotación hasta caer al suelo entra en juego.

En la segunda fase observamos como al principio la aceleración en Ax es muy negativo y el giro en y es muy positivo, lo que provoca una reducción drástica del pitch, indicando que el cuerpo ha comenzado a girar y caer hasta el suelo.

Finalmente, tanto los valores de aceleración en Ax y Az como el pitch se estabilizan tras el impacto final.

Si nos fijamos, este caso también es muy similar al caso 1 pero añadiendo la fase inicial de la fase de caída libre. El patrón que observamos en este tipo de caídas es el siguiente.

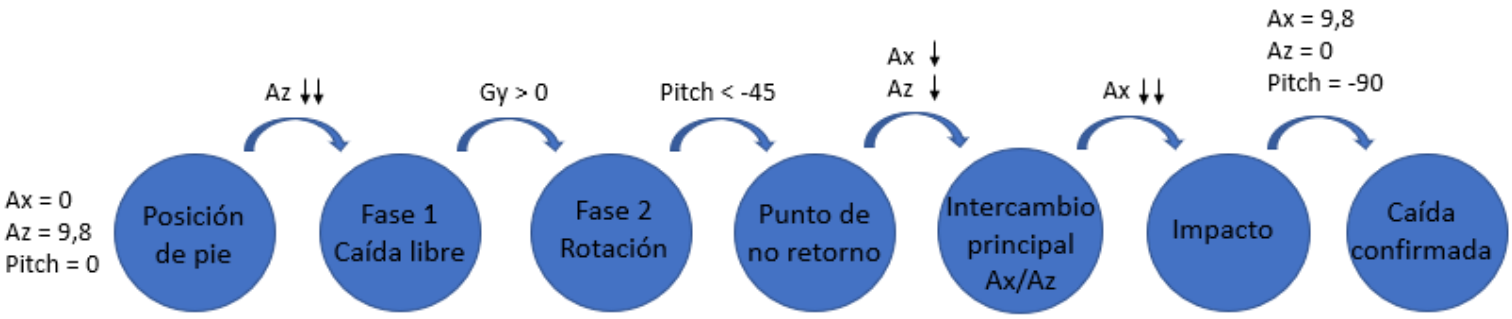


Figura 32. Patrón de Fall

7.4 SIMPLIFICACIÓN

A la hora de implementar un algoritmo que sea capaz de detectar cualquier tipo de caída humana, sería conveniente realizar una simplificación de estos modelos que nos permita aunar las características comunes de todos ellos para detectar de manera inequívoca la existencia de una caída por parte del usuario.

Hay que tener en cuenta que existen distintos modelos de caídas, por ejemplo, si alguien se desmaya y su cuerpo cae a plomo al suelo, no nos encontraríamos en ninguno de los 3 casos analizados, sino en una mezcla de varios, ya que los casos analizados simplemente son casos ideales que nos sirven para hacernos una idea de cómo varían los datos estudiados en varios tipos de caídas.

Tenemos que considerar la posibilidad de caer hacia la izquierda o hacia la derecha, o incluso de manera diagonal, es decir combinando Pitch y Roll y las aceleraciones Ax y Ay. Por ese motivo, emplearemos una nueva formulación utilizando valores absolutos y sumas matemáticas para obtener un patrón generalizado para detectar cualquier tipo de caída.

Además, hay que recordar que hay valores que pueden estar "saltados", sobre todo los valores de los impactos, ya que muestreamos los valores cada 0,1 segundos y estos ocurren en instantes de tiempos muy concretos. También es posible que tras un impacto ocurran imprecisiones debido a la estabilización del sensor. Por tanto, no sería oportuno incluir valores muy concretos para los impactos en el patrón de análisis, ya que aparte de ser los valores más variables, pueden no quedar bien registrados.

Lo que sí es seguro es que el cuerpo parte de una posición de pie en la que el cuerpo está más o menos erguido y a partir de ahí cuando sobrepasa los 45º respecto de la vertical existe un intercambio de aceleraciones entre los ejes Ax y Ay con el eje Az, mientras se mantiene una posición del cuerpo que sigue superando los 45º respecto de la vertical. Finalmente, se produce el impacto contra el suelo, donde podremos apreciar un despunte en la suma de aceleraciones en Ax y Ay. Cuando el cuerpo está ya casi tumbado, la aceleración de la gravedad y el posicionamiento confirman que se ha producido una caída.

Con todo, podemos resumir el patrón que aúne los distintos tipos de caídas analizados en la siguiente imagen.

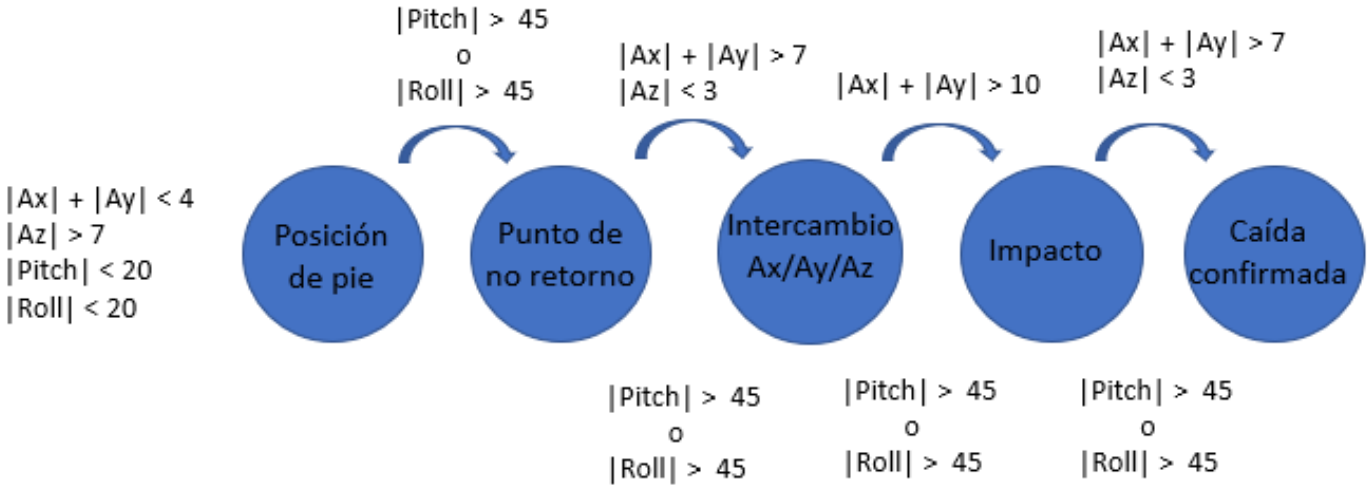


Figura 33. Patrón simplificado

Cabe destacar de nuevo que se espera que este patrón responde a las caídas analizadas o similares ya que los parámetros han sido ajustados según las muestras tomadas para que se detecten la mayoría de las caídas, pero sin registrar falsos positivos.

Dado que en el estudio sí hemos distinguido entre caídas hacia la izquierda, derecha, delante o atrás, impondremos como condición extra en el algoritmo que en el intercambio de aceleraciones al menos la mitad de la aceleración sea registrada por el eje al que debería corresponder según la dirección de caída.

El sentido de la caída quedará registrado en el momento en el que pitch o roll superen los 45º en el ángulo de caída. Así podremos facilitar más información sobre la caída a las asistencias sanitarias que reciban el aviso de alerta. El código del algoritmo puede encontrarse en el anexo.

Capítulo 8. SISTEMA PARA ROBOTS

En este apartado planteo un posible uso del algoritmo que sirva de ayuda para prevenir la caída de un robot bípedo durante su aprendizaje. Para ayudar a levantar robots bípedos durante el entrenamiento, el profesor Kim Joohyung de la Universidad de Illinois en Urbana-Champaign, quien busca un sistema para entrenar eficientemente a su Robotis OP2, propuso utilizar otro robot, apodado como Nannybot (robot cuidador o ayudante), que pudiera de alguna manera seguir al robot entrenado para que cuando este se cayera, pudiera levantarlo (Joohyung, 2020).

Siguiendo esa idea, propongo un diseño de robot que pueda seguir al robot entrenado aprovechando el sensor de movimiento incorporado en el robot andante, y que cuando el algoritmo detecte una caída durante el entrenamiento, el Nannybot pueda evitar la caída mediante un sistema de tiraje con poleas. De esta manera ahorramos tiempo, esfuerzo humano y evitamos el posible daño del robot entrenado.

La idea es desarrollar un sistema autónomo con el que poder dejar ambos robots en una sala durante el tiempo que queramos y que el robot entrenado vaya recopilando la información necesaria para desarrollar su algoritmo de estabilidad mediante robot learning.

Para el manejo del Nannybot, aunque es posible utilizar AprilTags para crear un sistema totalmente autónomo, como estamos ante el primer prototipo se usará un control manual controlado por el usuario para verificar el correcto movimiento del Nannybot.

La fabricación de este robot tuvo su inicio a comienzos de 2020 pero la situación excepcional de pandemia mundial causada por el covid-19 provocó el cierre al acceso de laboratorios en las universidades y por tanto no se pudo finalizar. No obstante, traigo aquí todo el trabajo teórico y la información documentada hasta el momento.

8.1 OBJETIVOS DEL SISTEMA

A diferencia del robot bípedo, el robot de ayuda se sostendrá con ruedas, ya que queremos que este tenga la mayor estabilidad posible. Propongo construir un robot que pueda seguir al robot andante y prevenir completamente la caída de este.

El robot entrenado tiene que caer forzosamente hasta cierto ángulo para que pueda detectar que se ha caído y así poder registrar la información de sus movimientos en el algoritmo de aprendizaje. Sin embargo, no queremos que termine de caer para evitar su dañarlo.

Después de que el robot entrenado caiga y el robot ayudante haya prevenido la caída, el robot de ayuda levantará al entrenado para poder seguir con su entrenamiento.

El robot de ayuda no debe ser muy grande, usaremos unas medidas de 80x80x60 cm ya que este diseño es específico para el modelo OP2. La intención es que, si el modelo aquí propuesto funciona, sirva de referencia para escalarlo para robots más grandes. Aun así, la solución propuesta intenta ser un modelo lo más genérico posible, de manera que otros robots de tamaño similar puedan ser entrenados con el mismo sistema.

8.2 LISTA DE REQUISITOS A ALTO NIVEL

- El Nannybot debe ser capaz de seguir al robot Robotis OP2, el cual tiene una velocidad máxima analizada en laboratorio de 24 cm/s, bajo el control de una persona o un sistema autónomo.
- El Nannybot debe poder levantar al Robotis OP2, el cual tiene un peso de 3kg.
- El Nannybot no debe interferir con el desplazamiento regular del Robotis OP2
- El Nannybot tiene que ser compatible con robots bípedos de características similares al Robotis OP2

8.3 DISEÑO

El diseño del Nannybot está basado en los sistemas con arnés para bebés que están aprendiendo a caminar. En la siguiente imagen tenemos un ejemplo.

Figura 34. Ejemplo de arnés para bebés [1]

Realizando una analogía, queremos que el Nannybot sea como la madre que ayuda a su hijo a que aprenda a caminar y el robot andante es como el bebe. Así, necesitamos un sistema móvil, como la madre que sigue al hijo, y un sistema de tiraje, como el arnés que levanta al niño cuando se cae.

La figura siguiente muestra el diseño planteado del Nannybot en Autocad desde varias perspectivas. Como era de esperar el Nannybot está dividido en dos módulos, el chasis y el sistema de levantamiento. El chasis es el encargado de seguir al robot andante y para que este permanezca dentro de sus límites, rodeándolo en todo momento, mientras que el sistema de levantamiento se encarga de levantar al robot cuando este detecta una caída.

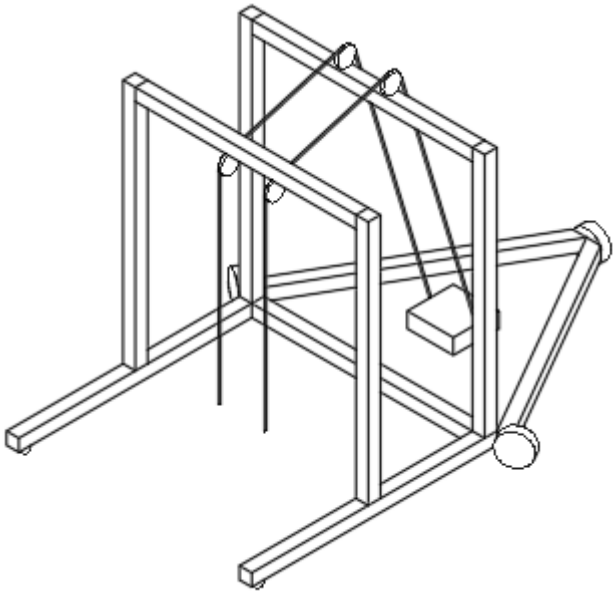


Figura 36. Vista frontal del Nannybot en 3D

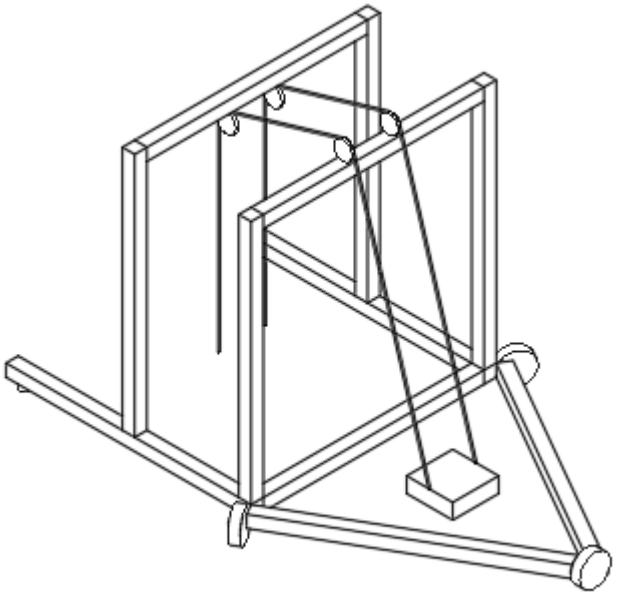


Figura 35. Vista posterior del Nannybot en 3D

El chasis se compone de unas barras laterales al nivel del suelo que rodean al robot entrenado para que este permanezca en todo momento en el centro del Nannybot, que es la zona de seguridad. Estas barras están compuestas de un material sólido pero ligero, capaces de equilibrar al Nannybot durante el balanceo del robot entrenado mientras se levanta y de soportar las poleas del sistema de levantamiento.

El chasis también incorpora 3 ruedas omnidireccionales en forma de triángulo, esto permite que el Nannybot pueda moverse en todas las direcciones para seguir al robot entrenado.

Utilizamos un motor paso a paso en el sistema de levantamiento para tirar de las cuerdas o hilos situados sobre las poleas que levantan al robot entrenado. El motor se sitúa sobre el chasis para que el centro de masa del Nannybot permanezca cercano al suelo.

Además, el modelo propuesto incluye un recubrimiento de gomaespuma para no dañar al robot andante en caso de que este impacte con alguna de las paredes del robot ayudante.

A continuación, podemos ver el alzado, planta y perfil del diseño con las medidas.

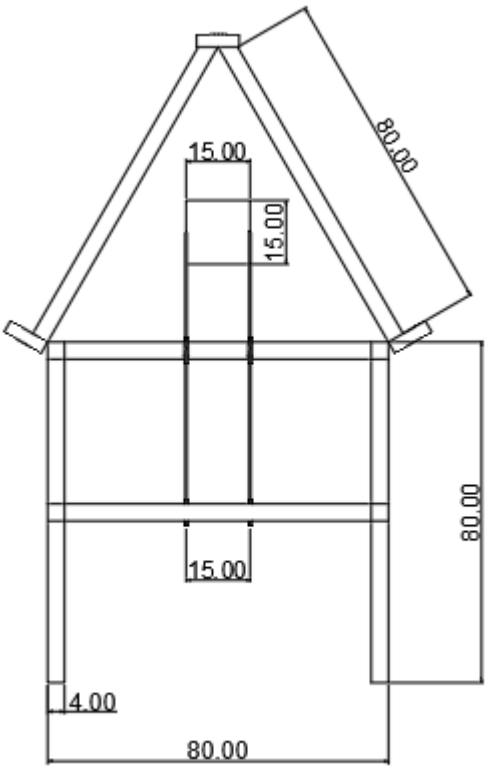


Figura 38. Planta del Nannybot

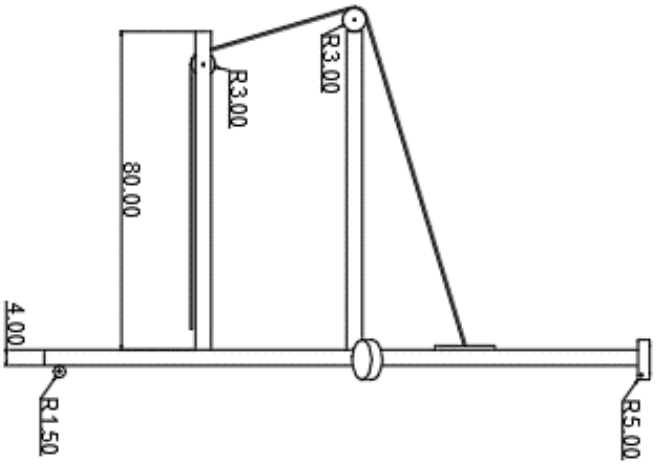


Figura 37. Lateral del Nannybot

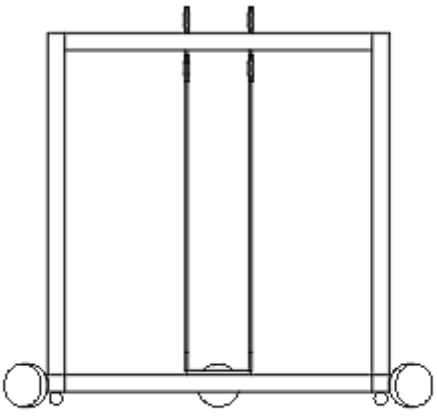


Figura 39. Alzado del Nannybot

8.4 *DIAGRAMA DE BLOQUES*

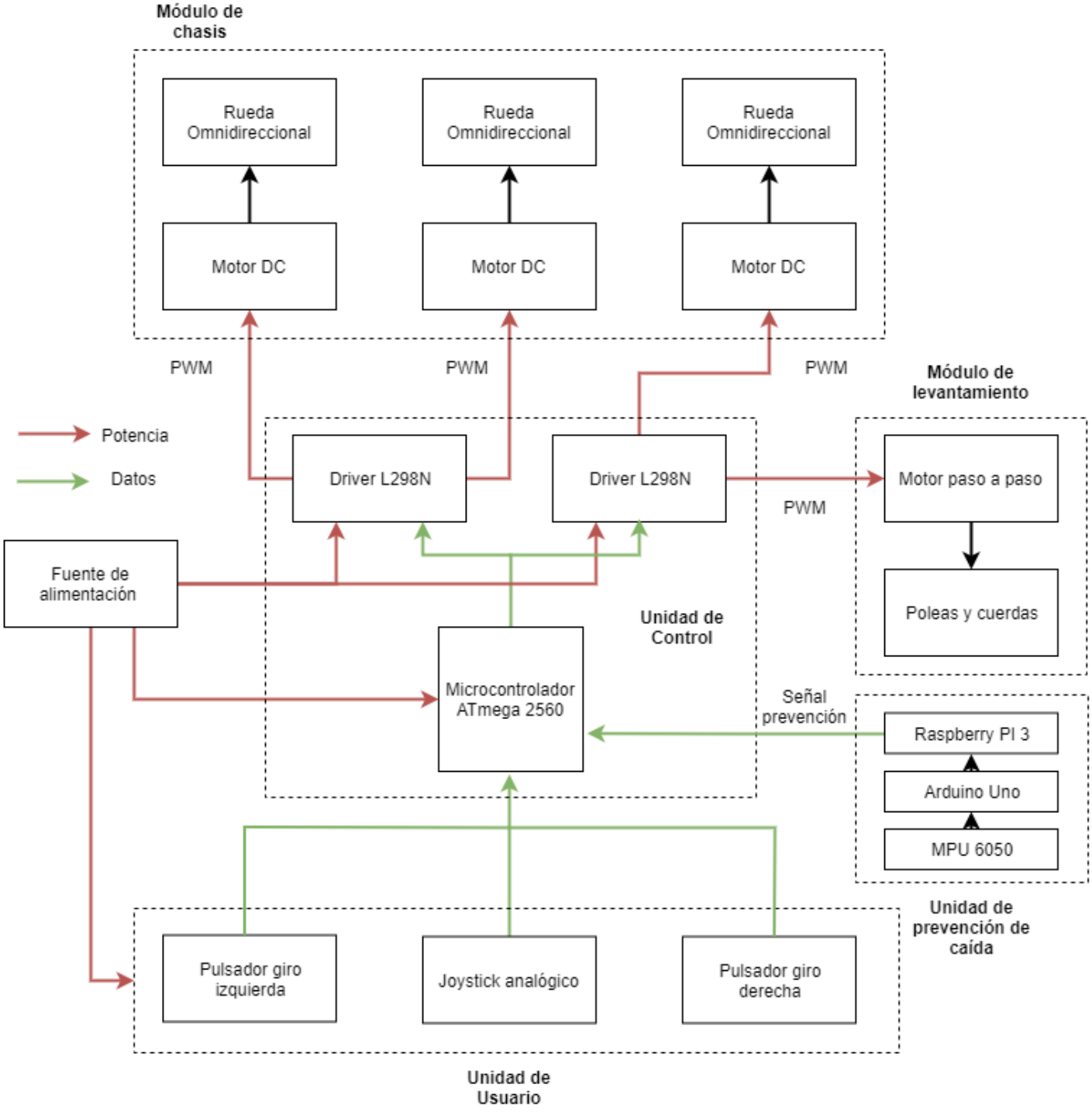


Figura 40. Diagrama de Bloques del Nannybot

Para controlar el Nannybot utilizaremos una unidad de usuario que envía señales de I/0 a la Unidad de Control para manejar el módulo de chasis. Por otro lado, la unidad de prevención de caída manda señales a la unidad de control para utilizar el módulo de levantamiento.

8.5 FUNCIONALIDAD DE LOS COMPONENTES

8.5.1 UNIDAD DE USUARIO

La unidad de usuario permite transmitir comandos al microcontrolador. Está formado por el joystick y los pulsadores.

Joystick y pulsadores

Con el joystick se controlará el movimiento frontal y lateral del chasis, mientras que con los pulsadores ofrecemos al Nannybot la posibilidad de rotar sobre sí mismo a izquierda o derecha. Así, podremos mantener al robot andante siempre en el centro y mirando hacia adelante, de lo contrario los cables controlados por las poleas podrían cruzarse e interferiríamos en el desplazamiento regular del robot andante, lo cual es uno de los requisitos para tener en cuenta.

El joystick es el encargado del movimiento en los ejes x e y, y permite seguir al robot entrenado indicando una velocidad cada eje. En la figura anterior el Nannybot se desplazaría hacia adelante y ligeramente a la izquierda.

Los botones son los encargados de los movimientos de rotación. Cada pulsador tiene asociado un movimiento de giro hacia un lado para poder rotar a la vez que el robot entrenado

8.5.2 UNIDAD DE PREVENCIÓN DE CAÍDAS

Incorpora el sensor MU 6050, la Arduino Uno y la Raspberry PI 3 utilizados para el desarrollo del algoritmo. Esta unidad recoge los datos del sensor y procesa el algoritmo desarrollado para enviar la señal de prevención de caídas a la unidad de control para que active el sistema de levantamiento.

8.5.3 UNIDAD DE CONTROL

La unidad de control recibe señales desde la unidad de usuario y envía señales a los motores del módulo de chasis y al motor paso a paso del módulo de levantamiento. Está alimentado por la tensión de la red y consiste en un microcontrolador y 2 drivers de motor. El microcontrolador recibe señales del usuario y del algoritmo de prevención de caídas para desplazar al Nannybot y activar el sistema de levantamiento mediante se señales PWM.

Microcontrolador ATmega 2560

Para construir el Nannybot utilizamos 3 motores DC y 1 motor paso a paso. Para controlar la dirección y velocidad de los motores, así como el sistema de recogida, necesitamos al menos 2 pines PWM para cada motor, es decir un total de 8. Además, si queremos integrar la unidad de usuario y la unidad de prevención de caídas necesitaremos al menos 2 pines analógicos de input, 2 pines digitales de I/0 para los pulsadores y los pines RX y TX. Tras tener en consideración estos requisitos, la elección del microcontrolador ATmega 2560 parece acertada, ya que este incorpora 15 pines PWM, 54 pines digitales de I/O, 16 pines de input analógicos y 18 de RX y TX.

Driver L298N

Como la mayoría de los microcontroladores, el ATmega 2560 no tiene la capacidad necesaria para mover los 4 motores utilizados. Por tanto, es necesario incorporar un módulo controlador de motores como el L298N que proporcione la corriente necesaria para el control de los motores DC y del motor paso a paso. Como el modelo L298N tiene una potencia de 25 watios y puede manejar 2 motores, será suficiente con incluir dos controladores de este tipo, uno para controlar 2 motores DC y otro para controlar 1 motor DC y el motor paso a paso.

8.5.4 MÓDULO DE CHASIS

En este módulo se localizan los motores DC y las ruedas omnidireccionales. El Nannybot cuenta con una estructura triangular en la parte trasera del chasis donde 3 ruedas omnidireccionales controlan el movimiento del Nannybot, mientras que en la parte delantera se encuentran 2 ruedas giratorias sin propulsión, diseñadas únicamente para sujetar el chasis por la parte frontal y permitir el desplazamiento propulsado por las ruedas omnidireccionales.

Motores DC

Los 3 motores DC reciben las señales PWM de la unidad de control para girar las 3 ruedas omnidireccionales en cualquier dirección deseada y poder mover el chasis. Estos motores deben tener una capacidad individual de al menos 180 rpm para poder seguir al robot andante OP2 en cualquier dirección y velocidad con la que se mueva.

Los 3 motores deben permitir el giro del Nannybot alrededor del robot entrenado manteniéndolo en el centro de giro, teniendo en cuenta que el OP2 realiza un giro completo en aproximadamente 8 segundos. Además, deben ser capaces de mover al Nannybot cuando este está levantando con el sistema de levantamiento al robot entrenado, es decir, tienen que poder moverse aun sosteniendo en la estructura del Nannybot 3kg.

Ruedas omnidireccionales

Las 3 ruedas motorizadas son omnidireccionales para permitir seguir al robot entrenado sin importar en qué dirección se mueva. Las ruedas omnidireccionales, propulsadas por los motores DC, tienen la misma forma que unas ruedas convencionales, pero además poseen una serie de rodillos cilíndricos adosados que permiten el deslizamiento perpendicular respecto del movimiento de giro normal de la rueda. Esto permite que si una rueda gira y otra se queda estática porque no puede ir en la dirección deseada, los rodillos permiten el deslizamiento de la rueda estática sin bloquear el movimiento.

8.5.5 MÓDULO DE LEVANTAMIENTO

El módulo de levantamiento recibe las señales de la unidad de control y utiliza el motor paso a paso para levantar al robot entrenado mediante el uso de cuerdas y poleas.

Motor paso a paso

El motor paso a paso está instalado en la parte trasera del chasis y se conecta a dos cuerdas. Estas cuerdas tiran a la vez con la ayuda de poleas para levantar al robot entrenado cuando se va a caer. Cuando la unidad de prevención de caídas detecte la caída del robot, se enviará una señal de aviso a la unidad de control para activar el motor paso a paso, que tirará de las cuerdas para levantar al robot entrenado y lo volverán a colocar en la posición de entrenamiento. El motor paso a paso debe ser capaz de levantar aproximadamente 5kg a unos 3cm del suelo. Es conveniente dejar un margen de 2kg para contar con la aceleración inicial a la hora de levantarlo.

Poleas

Existen 2 pares de poleas situadas en la parte superior de las ruedas delanteras y en la parte superior del robot andante para poder levantar del arnés del robot andante cuando este se caiga. Sobre estas 4 poleas irán las cuerdas que conectan el arnés del robot con el motor paso a paso. Las poleas deben permitir el movimiento de recogida de la cuerda y también deben ser capaces de soportar aproximadamente 5kg para contar con margen a la hora de levantar al robot.

Cuerdas

Se usan 2 cuerdas atadas a un arnés que lleva el robot entrenado y al motor paso a paso. Mediante el uso de las poleas, conseguimos que el último tramo vertical esté en la misma dirección que el robot entrenado. Así, cuando se activa el motor paso a paso, el robot entrenado es levantado hacia arriba.

Las cuerdas tienen que ser lo suficientemente largas para poder atarse al arnés del robot entrenado y además permitir cierta holgura, ya que para detectar la caída necesitamos que el robot llegué a un ángulo de 30 grados. Las cuerdas tienen que ser capaces de soportar al menos 5kg.

Además, necesitamos que haya cierto margen para que en caso de que el robot se descuadre un poco de la zona central del Nannybot, estas no interfieran en el movimiento regular del robot. Por otro lado, no ha de ser demasiado larga sino podría enredar al propio robot entrenado.

8.6 CÁLCULOS

Los motores del chasis han de tener la potencia suficiente para seguir al robot entrenado tanto en el movimiento de rotación como al correr a máxima velocidad. Para ello, calculo las RPM mínimas de los motores

8.6.1 RPM PARA ALCANZAR LA ROTACIÓN DEL ROBOTIS OP2

Conociendo la estructura del Nannybot podemos calcular el radio R de giro que va desde la posición central del Nannybot donde se sitúa el Robotis OP2 hasta la rueda más externa del chasis.

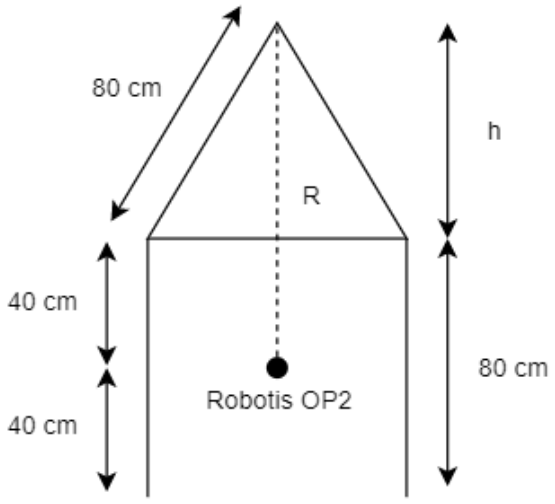


Figura 41. Radio de giro del Nannybot

Sabiendo que h es la altura del triángulo equilátero de lado 80 cm, podemos saber su medida

$$h = 80 \cdot \frac{\sqrt{3}}{2} = 69{,}28\ cm$$

Por tanto, ya conocemos el radio de giro R del Nannybot

$$R = 40 + 69{,}28 = 109{,}28\ cm$$

También sabemos que el Nannybot utiliza unas ruedas omnidireccionales de 5 cm de radio y que el modelo Robotis OP2 gira a 8RPM

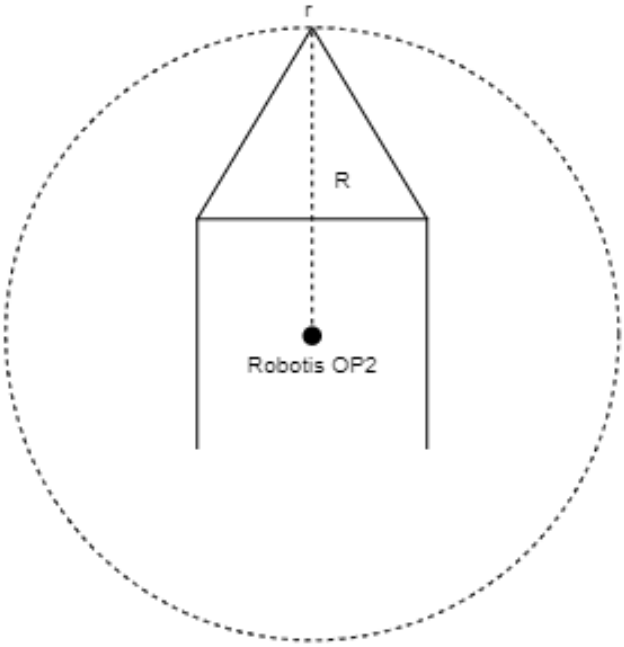


Figura 42. Circunferencia de giro del Nannybot

La relación entre las revoluciones del Robotis OP2 y las de la rueda del Nannybot es

$$Relación = \frac{2 \cdot \pi \cdot 109{,}28\ cm}{2 \cdot \pi \cdot 5\ cm} = 21{,}836$$

Por tanto, las RPM mínimas que deben alcanzar los motores de las ruedas son

$$RPM_{mín} = 8\ \cdot 21{,}836 = 174{,}69\ RPM$$

8.6.2 RPM PARA ALCANZAR LA MÁXIMA VELOCIDAD DEL ROBOTIS OP2

En primer lugar, hemos de tener en cuenta que la parte del chasis posterior del Nannybot, donde se sitúan las ruedas omnidireccionales, forma un triángulo equilátero. De esta manera, el Nannybot es capaz de dirigirse en cualquier dirección utilizando esta estructura triangular

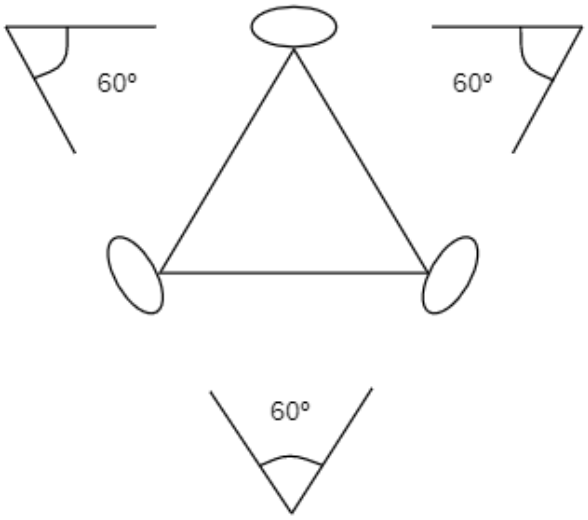


Figura 43. Estructura triangular del chasis

Para calcular las RPM necesarias para alcanzar la máxima velocidad del robot andante, nos ponemos en el peor de los casos en el que la dirección del movimiento del Robotis OP2 es perpendicular a una de las ruedas y con cualquiera de los dos sentidos, ya que por ángulos complementarios nos encontramos en el mismo caso.

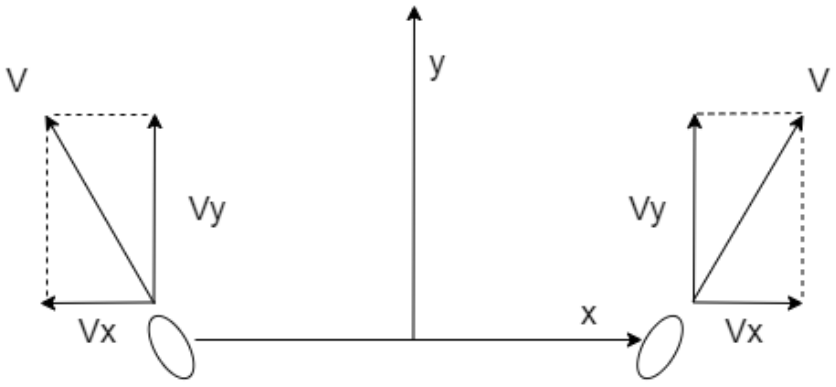


Figura 44. Vectores de velocidad del Nannybot

Sabiendo que la velocidad máxima del Robotis OP2 es de 24 cm/s, podemos calcular para este caso las RPM que han de alcanzar las ruedas del Nannybot. Para ello, primero calculamos la velocidad lineal que deben alcanzar las ruedas.

$$\vec{V_x} + \vec{V_x} = 0$$

$$\vec{V_y} + \vec{V_y} = 0{,}24\ m/s$$

$$2 \cdot \vec{V} \cdot sen(60) = 0{,}24\ m/s$$

$$\vec{V} = 0{,}138\ m/s$$

Entonces, podemos averiguar la velocidad angular en radianes por segundo y en RPM

$$\vec{\omega} = \frac{\vec{V}}{r} = \frac{0{,}138\ m/s}{0{,}05\ m} = 2{,}772\ rad/s$$

$$RPM_{mín} = \frac{2{,}772\ rad/s}{2\pi/60} = 26{,}47\ RPM$$

Como las RPM necesarias para alcanzar la rotación del Robotis OP2 son mayores que las necesarias para alcanzar la máxima velocidad del robot andante, los motores deben tener como mínimo 174,69 RPM. Este valor es aproximado a 180 RPM para contar con un margen de tolerancia.

8.7 ESTIMACIÓN ECONÓMICA

Teniendo en cuenta los recursos empleados se puede estimar el coste económico de la construcción del Nannybot.

Recurso	***Precio (€)***
3 Motor DC 180 RPM 12V	101,67
3 Ruedas omnidireccionales 100mm ø	24,99
Arduino Mega 2560	13,99
2 Drivers L298N	8,99
Motor paso a paso	22,99
Joystick y pulsadores	4,75
Chasis	19.99
Poleas y cuerdas	14.99
Unidad de prevención de caídas	59,42
Ingeniero de Telecomunicaciones (10h)	200
Total	**471,78 €**

Tabla 17. Estimación Económica del Nannybot

Capítulo 9. SISTEMA PARA PERSONAS

En este capítulo planteo un posible uso del algoritmo en un sistema para detectar las caídas en personas. En este caso, a diferencia del sistema para robots que pretendía prevenir la caída, el sistema detectará la caída y se dará la posibilidad al usuario de confirmar la posible caída para avisar en caso de peligro a los servicios de asistencia sanitaria.

La idea es que, en el futuro, este producto pueda estar sincronizado por Bluetooth Low Energy con los ya existentes *smartwacthes* y *smartbands*, para que se pueda confirmar las caídas por parte del usuario y avisar a los servicios de asistencia sanitaria. Es decir, esta banda pretende ser un accesorio complementario mucho más fiable para las detecciones de caídas.

Para este primer prototipo del sistema, plantearé una unidad de prevención de caída que incluirá el algoritmo programada, junto con una interfaz de usuario sencilla que permita la interacción con el usuario para corroborar que se ha producido una caída. En el caso de que no se confirme ninguna respuesta en un tiempo determinado, se procederá igualmente al aviso de asistencia.

9.1 OBJETIVOS DEL SISTEMA

El sistema debe ser capaz de detectar las caídas en las personas que incorporen el sistema junto con el algoritmo. Para ello, el usuario debe portar consigo de alguna manera el sensor MPU 6050, un microcontrolador capaz de procesar el algoritmo con los datos recibidos, y una pantalla con la que el usuario pueda interactuar.

Además, el sistema debe ser cómodo y ajustable, no todas las personas son iguale ni tienen las mismas capacidades físicas. El sistema debe poder ser utilizable en cualquier tipo de situación, ya que está pensado para situaciones cotidianas del día a día, no para condiciones de laboratorio como ocurría en el caso del sistema para robots.

Por tanto, buscamos un sistema lo más genérico posible que se adapte a las condiciones del usuario y no suponga una carga adicional a la hora de portarlo consigo mismo. Siguiendo este espíritu podemos plantear un sistema similar a las bandas o chalecos de entrenamiento deportivo utilizadas para registrar el desempeño de por ejemplo futbolistas.

9.2 LISTA DE REQUISITOS A ALTO NIVEL

- El sistema debe poder detectar las caídas de cualquier persona en condiciones cotidianas y avisar al usuario
- El sistema debe permitir la respuesta del usuario para confirmar si se ha producido una caída
- El sistema debe avisar al número de emergencia asignado en caso de que el usuario confirme la caída o de que se agote el tiempo de espera de confirmación

9.3 DISEÑO

A la hora de diseñar el sistema para personas, hemos de tener en cuenta que no está pensado para usarse en condiciones de laboratorio, como ocurría en el caso del sistema para robots, sino más bien en el día a día. Pensando en una solución cómoda y usable en distintos entornos, planteo la posibilidad de utilizar una banda o chaleco similar a las usadas por deportistas, para poder situar el sensor de posicionamiento en el pecho, que es un punto clave para determinar las caídas.

En la primera foto observamos la banda de frecuencia cardíaca HRM3-SS de Garmin, la cual detecta las pulsaciones del usuario e incorpora tecnología GPS para monitorizar sin cables las pulsaciones cardiacas y transmitirlas al dispositivo al que esté sincronizado mientras el usuario hace ejercicio. Este producto tiene una cómoda bandas ajustables para que el usuario la regule a su gusto.

Otra posibilidad es adoptar la forma como la del chaleco ergonómico *Smart Football Tracker* de Playr. Este incorpora tecnología GPS y en un Tracker autorizado por la FIFA para analizar y mejorar el rendimiento de los futbolistas. Este chaleco también se puede sincronizar con un móvil que permite sincronizar las sesiones de uso para ver rápidamente los datos de rendimiento del usuario.

Figura 45. Ejemplo de banda deportiva [17]

Figura 46. Ejemplo de chaleco deportivo [2]

En cualquiera de los dos diseños sería sencillo incluir el sensor MPU6050 para poder recopilar información sobre los movimientos del usuario y utilizar una interfaz de usuario como un *smartwatch* o un móvil para interactuar con el algoritmo.

9.4 DIAGRAMA DE BLOQUES

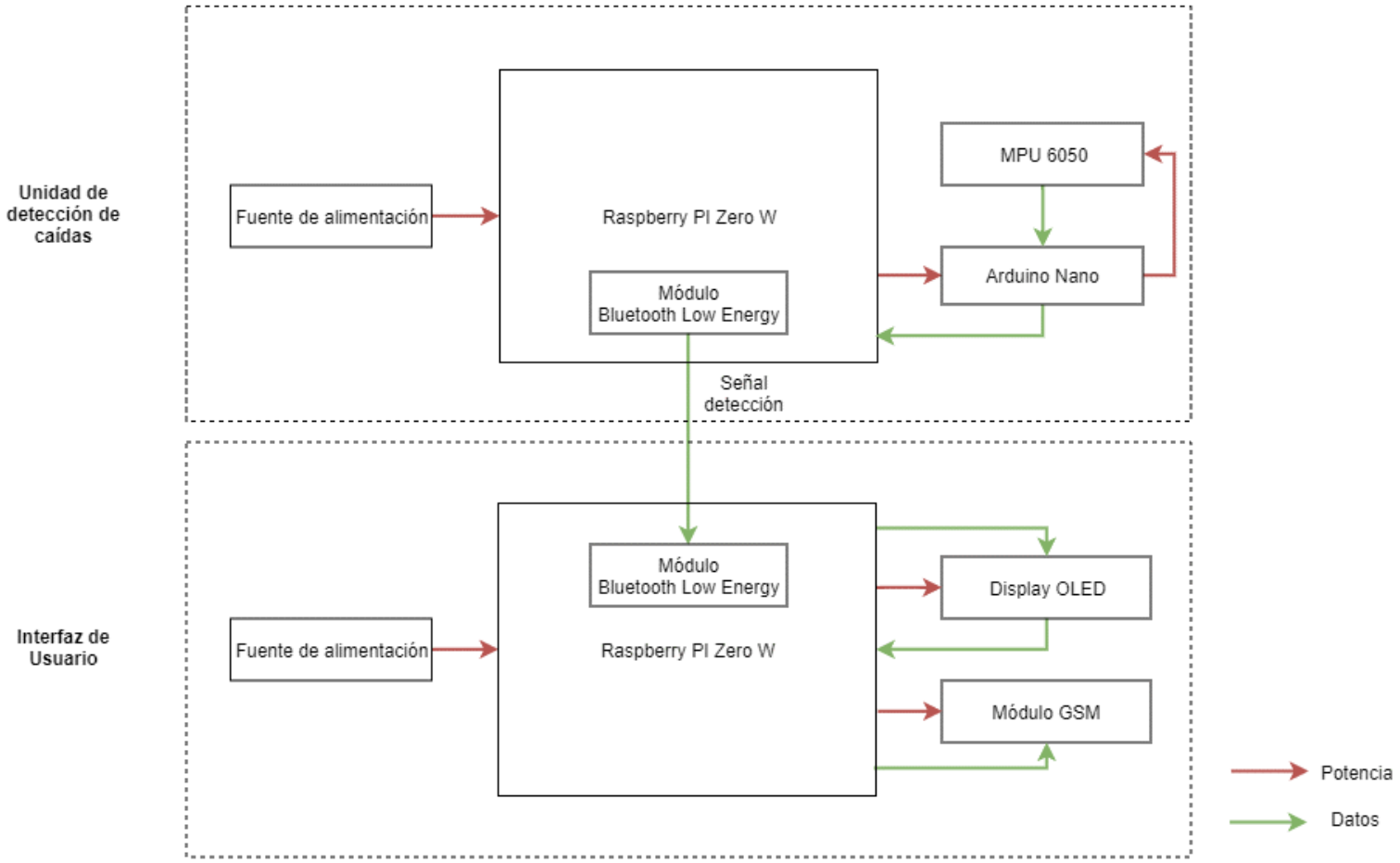


Figura 47. Diagrama de Bloques del Sistema para Personas

El sistema propuesto para personas contiene dos bloques principales, la unidad de detección de caídas y la interfaz de usuario.

La unidad de detección de caídas se compone de una Raspberry PI Zero W que procesa los datos con el algoritmo y decide cuando mandar la señal de prevención de caída a la interfaz de usuario. Para obtener los datos se usa el MPU 6050 y una Arduino nano.

La interfaz de usuario recibe la señal de detección de caída y muestra a través de un Display OLED la opción de confirmar la caída por parte del usuario. En caso de que el usuario confirme la caída o de que se agote el tiempo de espera, se activa el módulo GSM para avisar a los servicios sanitarios.

9.5 FUNCIONALIDAD DE LOS COMPONENTES

9.5.1 UNIDAD DE DETECCIÓN DE CAÍDAS

Esta unidad está integrada en el chaleco o la banda y es la encargada de detectar una caída por parte del usuario y transmitir una señal de Bluetooth Low Energy a la interfaz de usuario. Es idéntica y realiza la misma función que la que se planteó en el capítulo anterior, pero se han cambiado los componentes para que sean de un tamaño más reducido.

MPU 6050

Es el sensor utilizado para recopilar los datos necesarios que emplea el algoritmo. Los datos del sensor se leen en la Arduino Nano.

Arduino Nano

Recibe los datos del MPU 6050 y los envía a la Raspberry para que los procese en el algoritmo. Propongo utilizar este modelo de Arduino para reducir el tamaño del dispositivo ya que este alberga los suficientes pines como para conectarse a la Raspberry y al sensor.

Raspberry PI Zero W

Este modelo de Raspberry además de ser de tamaño reducido incluye un módulo de Bluetooth Low Energy capaz de enviar y recibir señales BLE. Es la encargada de procesar el algoritmo con los datos recibidos de la Arduino y de enviar la señal de detección de caída a la interfaz de usuario.

9.5.2 INTERFAZ DE USUARIO

Esta unidad también la incorpora el usuario, pero de manera separada al chaleco o banda, ya que no es necesario posicionarla en el tronco como ocurría con la unidad de prevención de caídas. Un lugar cómodo puede ser en la muñeca en forma de *smartband* o *smartwatch*.

Raspberry PI Zero W

Utilizamos el mismo modelo de Raspberry con bluetooth integrado que en la unidad de detección de caídas para recibir la señal de detección de caída y activar tanto el display OLED como el módulo GSM.

Display OLED

Esta pantalla muestra al usuario que se ha podido producir una caída ya que así lo ha detectado el algoritmo. El usuario tiene entonces la opción de confirmar la caída o desmentirla mediante unos pulsadores incorporados.

Módulo GSM

En el caso en que el usuario confirme la caída o se agote el tiempo de espera, lo cual indicaría una caída y posible pérdida de conciencia por parte del usuario, se procedería a activar el módulo GSM para avisar a los servicios sanitarios.

9.6 ESTIMACIÓN ECONÓMICA

Teniendo en cuenta los recursos empleados se puede estimar el coste económico de la construcción del sistema de detección de caídas para persona.

Recurso	***Precio (€)***
2 Raspberry PI Zero W	23,94
MPU 6050	7,99
Arduino Nano	7,99
Display OLED	6,80
Módulo GSM	18,89
Ingeniero de Telecomunicaciones (5h)	100
Total	**165.51 €**

Tabla 18. Estimación Económica del Sistema para Personas

Capítulo 10. ANÁLISIS DE RESULTADOS

Para analizar los resultados obtenidos, simulamos la implementación del algoritmo en ambos sistemas. Conectando la Arduino y el sensor con la Raspberry PI 3 que incluye el programa con el algoritmo, somos capaces de recrear un sistema portátil que simulará los sistemas propuestos.

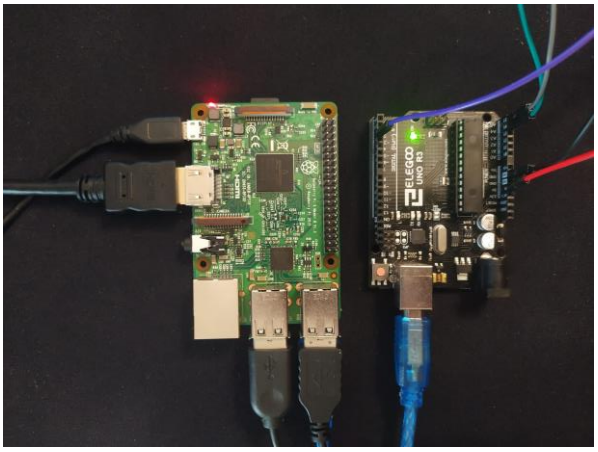


Figura 48. Conexión Arduino con Raspberry PI 3

Para ello, utilizaremos el sensor posicionado en el tronco de una persona o del Robotis OP2 e indicamos al programa con el parámetro de distinción entre robot y persona con qué sujeto estamos trabajando. Analizaremos estadísticamente los resultados obtenidos por ambas simulaciones para ver si los diseños son fiables y repasaremos las fases por los que pasa cada sistema.

Además, utilizaremos una base que soporte el sensor en el tronco de los usuarios para situarlo en la misma posición que cuando se obtuvieron los análisis de las caídas de robots bípedos y de personas, de esta manera garantizamos que el algoritmo funciona como deseamos.

Dado que la situación de covid-19 impide el acceso a laboratorios y por tanto el uso del Robotis OP2 que había disponible para este proyecto queda restringido, se usará un sustituto que pueda ofrecer características similares para proceder con la simulación.

10.1 SIMULACIÓN DEL SISTEMA PARA ROBOTS

Como sustituto para el análisis de caídas, se usará una estructura con las mismas dimensiones, ya que esto será suficiente para decidir si el algoritmo responde correctamente en una estructura similar a la del robot original. En este caso, utilizaremos un peluche que reúne las características necesarias para probar el algoritmo. Además, incluimos el arnés y dos hilos resistente que simulan el sistema de levantamiento para robots.

También utilizaremos un atril de madera para limitar la caída a unos 45º respecto de la vertical, de esta manera nos aseguramos de que la caída se detecta antes del tiempo necesario para activar el sistema de levantamiento.

Figura 50. Estructura similar al OP2

Figura 49. Limitador de caída

En caso de que el algoritmo diseñado que se ejecuta en la Raspeberry detecte una caída se muestra la siguiente pantalla en el monitor para indicar que en ese momento se activaría el sistema de prevención de caídas del Nannybot.

Figura 51. GUI robots

Además, por la terminal podemos observar que la conexión de la Raspberry con la Arduino es satisfactoria y cuando se detecta una caída se imprimen por pantalla la matriz de datos que se registra en la base de datos MySQL

```
Puertos Disponibles:

   [0] ttyS3: Physical Port S3 - Physical Port S3
   [1] ttyS1: Physical Port S1 - Physical Port S1
   [2] ttyS2: Physical Port S2 - Physical Port S2
   [3] ttyS0: Physical Port S0 - Physical Port S0
   [4] ttyUSB1: USB-to-Serial Port (cp210x) - CP2102 USB to UART Bridge Controller

Pre-setting RTS: Success

Abriendo ttyUSB1: User-Specified Port - User-Specified Port: true

Lectura mediante eventos

Escuchando cualquier cantidad de datos disponible

Calibrando el sensor durante 15 segundos ...
[Data [ax=-0.72, ay=-0.9, az=9.85, gx=-0.64, gy=-0.37, gz=-0.32, yaw=-6.41, pitch=-2.5
```

Figura 52. Consola en la simulación para robots

Al contar con una base de datos que registra las matrices de datos de las caídas podemos usar esta información para presentar los datos gráficamente. Esto puede ser de utilidad en el futuro para crear aplicaciones en *smartwatches* y *smartbands* que analicen las caídas producidas. Las gráficas que veremos a continuación están representadas en Angular a partir de los datos de estas caídas.

10.1.1 CONTRASTE DE HIPÓTESIS

Para verificar que el porcentaje de casos detectados es adecuado para el algoritmo desarrollado, planteamos el siguiente contraste de hipótesis. En el caso del sistema para robots, como el algoritmo no es demasiado complejo, comprobaremos si podemos afirmar que el algoritmo es fiable en al menos un 90% al 5% de significación.

$$H_0: P < 0{,}9$$

$$H_1: P \geq 0{,}9$$

Si la población es de carácter dicotómico como en este caso, o el algoritmo detecta la caída o no la detecta, el parámetro de proporción "p" es igual a la media de la población. Además, esta prueba solamente es válida cuando el tamaño de la muestra es suficientemente grande, es decir n>30.

En el experimento llevado a cabo, hemos obtenido de un total de 50 muestras de caídas, que 49 han sido registradas correctamente como caídas, mientras que 1 no lo ha sido. Si la hipótesis nula es cierta, podemos plantear el siguiente estadístico de contraste que sigue una ley normal (0;1) (Martínez de Ibarreta Zorita, y otros, 2017).

$$z = \frac{proporción\ muestral - p^0}{(\frac{\sigma}{\sqrt{n}})} = \frac{proporción\ muestral - p^0}{\sqrt{\frac{p^0 \cdot (1 - p^0)}{n}}}$$

Aplicando los valores adecuados a los parámetros, obtenemos el valor:

$$z = \frac{0{,}98 - 0{,}9}{\sqrt{\frac{0{,}9 \cdot (1 - 0{,}9)}{50}}} = 1{,}886$$

Como para el valor anterior obtenemos un p-valor de 0,0296, que es menor que el nivel de significación de 0,05, podemos rechazar la hipótesis nula y afirmar que el algoritmo para el caso del sistema para robots es fiable en al menos un 90% bajo las condiciones en las que se ha desarrollado.

10.1.2 RESULTADOS OBTENIDOS

Con estos resultados podemos verificar que el algoritmo es capaz de detectar en un 90% de los casos el inicio de la caída para poder mandar una señal que active el sistema de levantamiento del robot de ayuda. Faltaría por comprobar cuando hubiera un prototipo del robot de ayuda que este es capaz de reaccionar con un tiempo de respuesta que se ajuste a las necesidades, es decir, antes de que el robot se caiga.

Figura 53. Resultados Robots

En la siguiente página podemos ver dos gráficas, la primera con las medias de las aceleraciones Ax, Ay, Az de las caídas hacia delante registradas en la simulación, y la segunda con la media del posicionamiento de Pitch y de Roll. Estas gráficas muestran los valores de aceleración (m/s^2) y posicionamiento (θ) respecto de las 15 muestras obtenidas, es decir, en un periodo de 1,5 segundos.

Se han seleccionado únicamente las caídas hacia delante para que los resultados sean más fieles y podamos analizar los resultados, de lo contrario habría que haber empleado módulos para calcular las medias y esto distorsionaría los resultados.

En la primera gráfica podemos observar cómo los valores de las aceleraciones de Ax y Az caen ligeramente pero no en exceso. Como se ha limitado la simulación a registrar un máximo de 45º para poder demostrar si se detecta antes el paso de los 30º, como esperábamos, el intercambio principal de aceleraciones no ha ocurrido todavía. Además, como las caídas se producen hacia delante la aceleración en Ay apenas varía.

En la segunda gráfica observamos como era de esperar que Roll permanece casi constante mientras que Pitch desciende desde un valor aproximado a 0 hasta -35. Esto verifica que la mayoría de los resultados de las caídas hacia delante simuladas responden fielmente al algoritmo como queríamos demostrar.

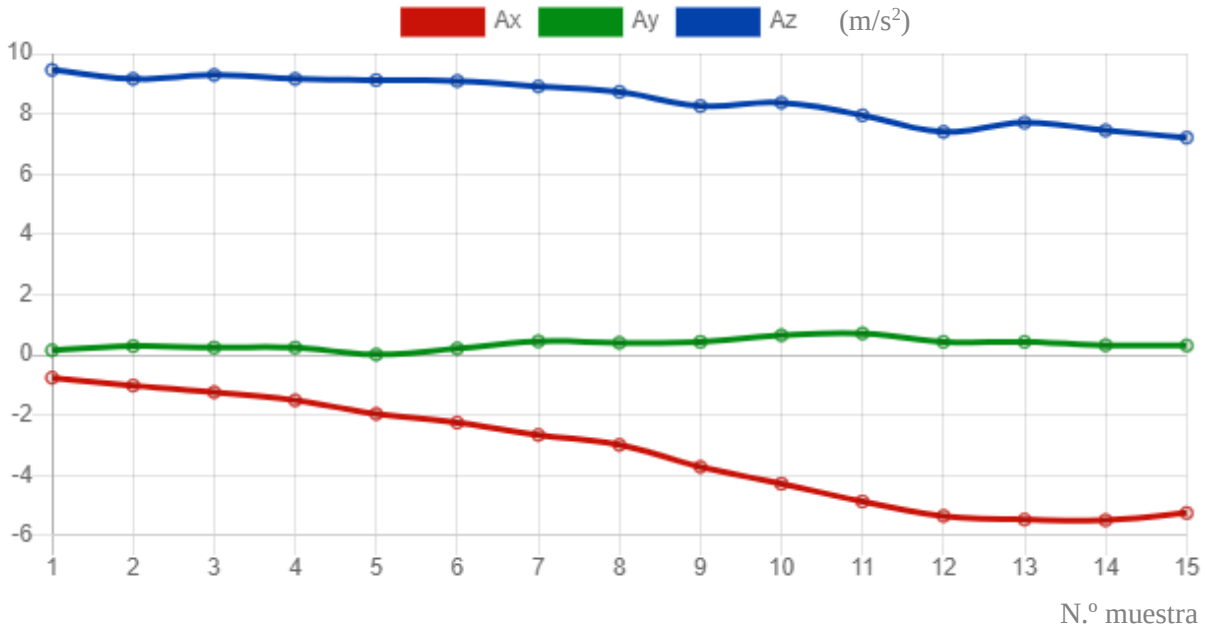


Figura 55. Aceleraciones medias en caídas frontales para Robots

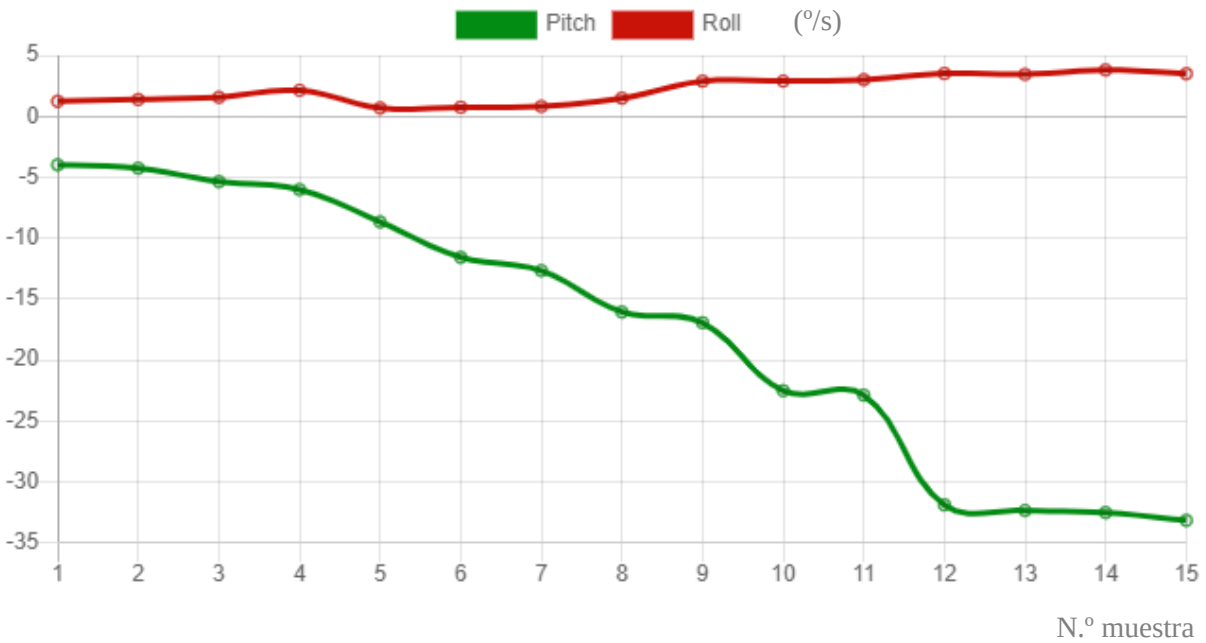


Figura 54. Posicionamiento medio en caídas frontales para Robots

10.2 SIMULACIÓN DEL SISTEMA PARA PERSONAS

En este caso, utilizaremos una esterilla como protección para no hacernos daño contra el suelo a la hora de simular las caídas. Realizaremos caídas a propósito de igual manera que hicimos para obtener los datos para analizar el patrón seguido.

En caso de detectar una caída se mostrará la GUI que simula la interfaz de usuario del sistema para personas. Esta indicará al usuario que se ha producido una caída y en qué sentido. Además, se ofrece la posibilidad de confirmar la caída o de desmentirla. Además, se muestra una cuenta atrás de 15 segundos por si el usuario queda inconsciente y no puede contestar.

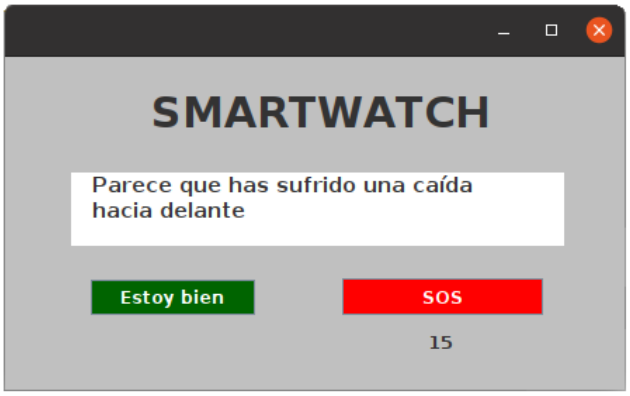


Figura 58. GUI personas

En caso de que el usuario pulse el botón de "Estoy bien" no ocurre nada, simplemente se muestra un mensaje de alivio por parte del *smartwatch.*

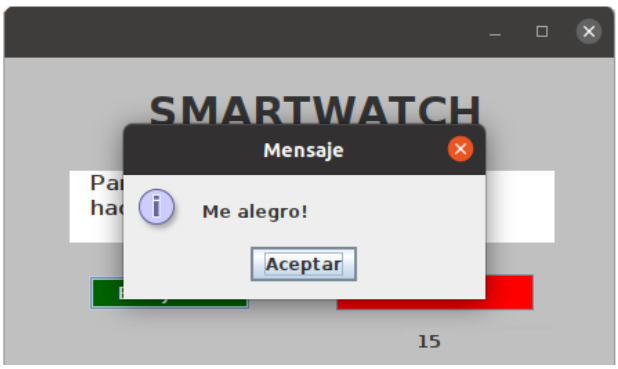


Figura 59. GUI personas "Estoy bien"

Si por el contrario se pulsa el botón de ayuda “SOS” o finaliza la cuenta atrás, se simula un mensaje de aviso a los servicios sanitarios.

Figura 60. GUI personas "SOS"

Además, por la consola se imprime por pantalla los estados del algoritmo por los que está pasando la caída y cuando llega al final nos indica que se ha producido la caída, la dirección, y la matriz de datos que se guarda en la base de datos MySQL.

```
1
1
2
2
3
3
3
4
ALERTA! Se ha producido una CAIDA hacia delante
[Data [ax=1.07, ay=-0.26, az=10.01, gx=-0.11, gy=-0.03, gz=-0.34, yaw=3.46
```

Figura 61. Consola en la simulación para personas

10.2.1 CONTRASTE DE HIPÓTESIS

Mientras que en el sistema para robots la complejidad reside en el robot de ayuda más que en el algoritmo, en el sistema para personas ocurre lo contrario, la complejidad reside en el algoritmo y el sistema de la banda es más sencillo. Por ello vamos a realizar el mismo contraste de hipótesis, pero con una fiabilidad del 80% al 5% de significación.

$$H_0: P < 0{,}8$$

$$H_1: P \geq 0{,}8$$

En el experimento llevado a cabo, hemos obtenido de un total de 50 muestras de caídas, que 45 han sido registradas correctamente como caídas, mientras que 5 no lo han sido. Si la hipótesis nula es cierta, podemos plantear el mismo estadístico de contraste que el anterior, que sigue una ley normal (0;1). Aplicando los valores adecuados a los parámetros, obtenemos el valor:

$$z = \frac{0{,}98 - 0{,}9}{\sqrt{\frac{0{,}9 \cdot (1 - 0{,}9)}{50}}} = 1{,}767$$

Como para el valor anterior obtenemos un p-valor de 0,0385, que es menor que el nivel de significación de 0,05, podemos rechazar la hipótesis nula y afirmar que el algoritmo para el caso del sistema para personas es fiable en al menos un 80% bajo las condiciones en las que se ha desarrollado.

10.2.2 RESULTADOS OBTENIDOS

Repitiendo el mismo proceso para la simulación de caídas en personas observamos como existen más casos no registrados que en el caso anterior dada la mayor complejidad que existe en el caso de identificar caídas para personas que para robots bípedos. Aun así, la simulación responde de manera bastante fiable y podemos afirmar que en un 80% de los casos el algoritmo detecta las caídas simuladas.

Figura 62. Resultados Personas

Analizando las mismas dos gráficas que en el caso anterior podemos ver datos y relaciones mucho más interesantes dado el incremento de complejidad en el algoritmo. Por ejemplo, en las aceleraciones seguimos viendo como Ay sigue siendo casi 0 de manera constante mientras que Ax y Az realizan el intercambio completo de la aceleración de la gravedad como era de esperar.

En la segunda gráfica apreciamos como Roll sigue la misma tendencia que en la gráfica anterior, pero Pitch llega a reducirse mucho más hasta casi -80º ya que ahora no existe ninguna limitación del ángulo de caída en esta simulación.

Un detalle interesante a tener en cuenta es que se ha utilizado el mismo color rojo para Roll y Ax así como verde para Pitch y Ay. Esto se debe a que el ángulo de giro de Roll es sobre el eje de Ax y el ángulo de giro de Pitch es sobre el eje de Ay. Esto nos permite ver la relación de cómo mientras la aceleración de un eje permanece constante, como en Ay, el giro sobre ese eje cambia significativamente, en este caso Pitch. También encontramos el ejemplo inverso con Ax y Roll en el que mientras Roll permanece constante y Ax cambia significativamente. Esto se debe a que mientras un cuerpo rota sobre un eje en una caída, es más difícil que se modifique las aceleraciones perpendiculares a la dirección de giro.

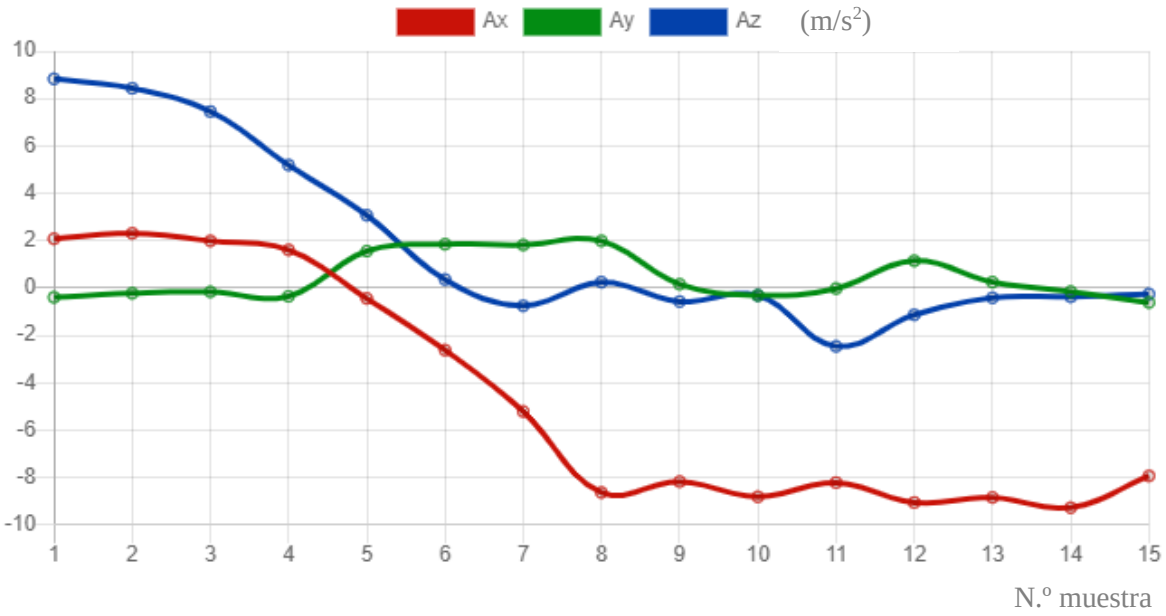


Figura 63. Aceleraciones medias en caídas frontales para Personas

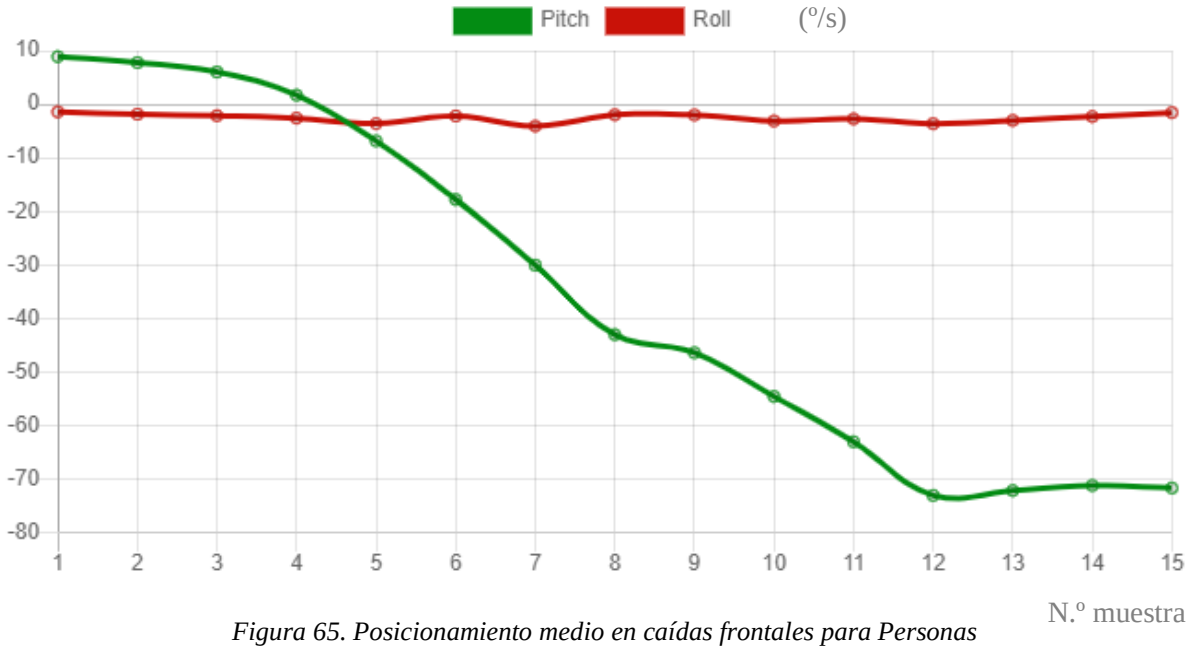


Figura 65. Posicionamiento medio en caídas frontales para Personas

Capítulo 11. CONCLUSIONES Y TRABAJOS FUTUROS

11.1 CONCLUSIONES

En este proyecto nos proponíamos diseñar un algoritmo capaz de detectar caídas en estructuras bípedas como robots o personas y plantear un sistema específico para cada caso con el que, utilizando el algoritmo diseñado, pudiéramos prevenir las caídas de los robots bípedos entrenados para poder entrenarlos más rápidamente y detectar las caídas en personas para avisar a los servicios sanitarios y salvar vidas.

Este trabajo ha alcanzado los objetivos propuestos de manera satisfactoria y se ha permitido verificar que con un sensor acelerómetro y giroscopio como el MPU 6050 situado en el tronco de una estructura bípeda, es posible desarrollar un algoritmo capaz de detectar caídas para incrementar la fiabilidad de los sistemas actuales en combinación con los sistemas propuestos.

Debido al covid-19 no se ha podido finalizar la fabricación de los sistemas planteados, por ello, se ha recurrido a una simulación de ambos para poder verificar la fiabilidad del algoritmo propuesto bajo las condiciones en las que se ha desarrollado.

Este proyecto plantea en este primer acercamiento con un algoritmo de detección de caídas y dos sistemas propuestos, un punto de partida en dos campos muy importantes como son la robótica y la salud, que pueden ver una mejora en el rendimiento de los tiempos de entrenamiento de robots bípedos y en el alto número de muertes causadas por caídas accidentales.

11.2 TRABAJOS FUTUROS

Como trabajos futuros, sería interesante construir los sistemas aquí planteados y refinar el algoritmo con experimentos realizados en los propios sistemas en vez de con simulaciones como las que se han utilizado en este trabajo. De esta manera podríamos, por ejemplo, medir los tiempos de reacción del robot de ayuda para activar el sistema de levantamiento o la precisión del algoritmo en todo tipo de situaciones.

La manera ideal para probar el algoritmo en las personas es utilizando el sistema durante un largo periodo de tiempo para comprobar que funciona correctamente y que no se detectan falsos positivos. Por otro lado, para probarlo en otros robots es necesario medir con precisión el ángulo de caída con el que se alcanza el punto de no retorno en la caída, que variará dependiendo de cada modelo.

También, se podrían sacar modelos industrializados de los sistemas aquí propuestos para adaptar el tamaño del sensor y de los elementos empleados para la comodidad del usuario. Los sistemas aquí planteados son prototipos que cumplen con la funcionalidad deseada de implementar el algoritmo y ayudar al robot o al usuario que por supuesto pueden evolucionar y ser más sofisticados.

Finalmente, se podría aprovechar la base de datos que registra las caídas detectadas para crear una aplicación o web a través de la cual, mediante protección de usuario y contraseña, los servicios médicos tengan acceso al historial de las caídas para obtener información del paciente. La aplicación y el sistema podría utilizarse en combinación con *smartbands* y *smartwatches* para complementar los sistemas actuales.

Capítulo 12. BIBLIOGRAFÍA

[1] Amazon. (s.f.). *Amazon España*. Obtenido de https://www.amazon.es/Felly-Seguridad-Ajustable-Aprendizaje-Protecci%C3%B3n/dp/B07PPDY1QW/ref=asc_df_B07PPDY1QW/?tag=googshopes-21&linkCode=df0&hvadid=297958066227&hvpos=&hvnetw=g&hvrand=5281218284742980795&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocin

[2] Amazon. (s.f.). *Amazon España*. Obtenido de https://www.amazon.es/CATAPULT-SmartCoach-Dispositivo-Seguimiento-Aplicaci%C3%B3n/dp/B07F3C6W5P/ref=asc_df_B07F3C6W5P/?tag=googshopes-21&linkCode=df0&hvadid=299981144307&hvpos=&hvnetw=g&hvrand=8879137769661268875&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&h

[3] C, C. (s.f.). *3D Warehouse*. Obtenido de https://3dwarehouse.sketchup.com/model/b95ab5c7-943d-4bb9-85fe-9a87ebc8b0fe/MPU-6050?hl=en

[4] Dejan. (Mayo de 2019). *How To Mechatronics*. Obtenido de https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/

[5] Fazecast Inc. (30 de Abril de 2020). Obtenido de https://fazecast.github.io/jSerialComm/

[6] InvenSense Inc. (19 de Agosto de 2013). *Invensense.TDK*. Obtenido de https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[7] Joohyung, K. (Enero de 2020). *NannyBot: a Robot for Legged Robot Experiments*.

[8] Kempe, V. (2011). *Inertial MEMS: Principles and Practice.* Cambridge University Press.

[9] Knight, W. (16 de Diciembre de 2004). *Newscientist*. Obtenido de https://www.newscientist.com/article/dn6809-bipedal-robot-learns-to-run/

[10] Koks, D. (2006). Explorations in Mathematical Physics: The Concepts Behind an Elegant Language. En D. Koks, *Explorations in Mathematical Physics: The Concepts Behind an Elegant Language* (págs. 181-183). Springer.

[11] Llamas, L. (2018). *Luis Llamas.* Obtenido de https://www.luisllamas.es/arduino-orientacion-imu-mpu-6050/

[12] Martínez de Ibarreta Zorita, C., Álvarez Fernández, C., Budría Rodríguez, S., Curto González, T., Escobar Torres, L. S., & Borrás Palá, F. (2017). *Modelos Cuantitativos para la Economía y la Empresa.* Madrid: EV Services.

[13] Organización Mundial de la Salud. (16 de Enero de 2018). *OMS*. Obtenido de https://www.who.int/es/news-room/fact-sheets/detail/falls

[14] Peter Ma, J. W. (8 de Mayo de 2020). *Hackster*. Obtenido de https://www.hackster.io/308536/senior-guardian-frame-018843?utm_content=buffer0bf96&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

[15] Pilla, S. D. (2009). *Slip, Trip, and Fall Prevention.* CRC Press.

[16] Pron, J. (11 de Enero de 2020). *AppOnly*. Obtenido de https://aaplonly.com/apple-watch-fall-detection-accident/

[17] Pulsómetro SinBanda. (s.f.). *Pulsómetro SinBanda*. Obtenido de https://www.pulsometrosinbanda.com/garmin-hrm-run-hrm-swim-y-hrm-tri-comparativa-bandas-frecuencia-cardiaca/

[18] Rui Wang, J. L. (2019). *Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions.* San Francisco: Cornell University.

[19] Serenocare. (17 de Abril de 2017). *Serenocare*. Obtenido de http://www.serenocare.com/sensores-prevencion-caidas-noche

[20] Vertice Vertical. (3 de Marzo de 2015). *Vertice Vertical*. Obtenido de https://www.verticevertical.com/sistemas-de-proteccion-contra-caidas/

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Este proyecto se alinea con dos de los diecisiete objetivos de desarrollo sostenible promovidos por la Organización de las Naciones Unidas. En concreto los objetivos 3, salud y bienestar, y 9, industria, innovación e infraestructuras.

Objetivo 3: Salud y Bienestar

Este objetivo sostiene que garantizar una vida sana y promover el bienestar en todas las edades es esencial para el desarrollo sostenible. En este ámbito se encuentra el sistema para personas diseñado en este proyecto que propone reducir la alta mortalidad en personas mayores de 65 años que la Organización Mundial de la Salud tasa en 646.000 muertes anuales por caídas.

Precisamente, el algoritmo diseñado aplicado al sistema de detección de caídas para personas tiene como objetivos detectar las caídas de cualquier persona en condiciones cotidianas para poder avisar al usuario en caso de que esta se produzca. Así, con la confirmación del usuario o pasado el tiempo límite, el sistema avisa a los servicios de emergencia sanitaria para garantizar la salud y el bienestar del usuario.

Objetivo 9: Industria, innovación e infraestructuras

Este objetivo sostiene que la industrialización inclusiva y sostenible, junto con la innovación y la infraestructura, pueden dar rienda suelta a las fuerzas económicas dinámicas y competitivas que generan el empleo y los ingresos. Una de las metas propuestas es modernizar para 2030 la infraestructura y reconvertir la industria para que sea sostenible.

Con este espíritu se encuentra alineado el algoritmo aplicado al sistema para robots aquí propuesto, que tiene como objetivo entrenar de manera más eficiente a robots bípedos que son capaces de desarrollar tareas realizadas por personas, permitiendo un avance en innovación que are un mundo de posibilidades para los trabajos automatizables.

ANEXO II

En este anexo se incluyen los códigos utilizados y una breve explicación sobre la utilidad de cada uno de ellos. Además, los códigos están comentados para poder entender su funcionamiento. Los códigos del algoritmo en Java han sido totalmente desarrollados por mí, mientras que los códigos en Arduino para la lectura de datos han sido reutilizados de (Llamas, 2018) y modificados para este proyecto, su autor principal es Jeff Rowberg, el cual desarrolló las librerías necesarias para la lectura del MPU 6050 en Arduino.

AGYRP.java

Es la clase principal donde se realiza la lectura de datos del sensor a través del puerto serie y se ejecuta el algoritmo para detectar una posible caída, en cuyo caso activa la GUI correspondiente al sistema para robots o al de personas y se almacenan los datos de las caídas en una base de datos MySQL. El algoritmo toma el nombre de "AGYRP" ya que son los datos utilizados en su desarrollo (Aceleraciones, velocidades de Giro, Yaw, Roll y Pitch).

```
import data.Data;
import gui.GUIpersona;
import gui.GUIrobot;
import jdbc.SQL;

import java.util.*;

import com.fazecast.jSerialComm.SerialPort;
import com.fazecast.jSerialComm.SerialPortDataListener;
import com.fazecast.jSerialComm.SerialPortEvent;

public class AGYRP {

public static void main(String[] args) {


/*  ------------------------------------------
 *  -            LECTURA DE DATOS            -
 *  ------------------------------------------
 */

ArrayList<Data> dataVector = new ArrayList<>();
```

```
//Configuración del Puerto Serie

SerialPort[] ports = SerialPort.getCommPorts();
System.out.println("\nPuertos Disponibles:\n");
for (int i = 0; i < ports.length; ++i)
	System.out.println("   [" + i + "] " + ports[i].getSystemPortName() + ": "
+ ports[i].getDescriptivePortName() + " - " + ports[i].getPortDescription());
SerialPort ubxPort = SerialPort.getCommPort("/dev/ttyUSB1");
System.out.println("\nPre-setting RTS: " + (ubxPort.setRTS() ? "Success" :
"Failure"));
boolean openedSuccessfully = ubxPort.openPort(0);
System.out.println("\nAbriendo " + ubxPort.getSystemPortName() + ": " +
ubxPort.getDescriptivePortName() + " - " + ubxPort.getPortDescription() + ": " +
openedSuccessfully);
if (!openedSuccessfully)
	return;
ubxPort.setBaudRate(115200);


//Leer Datos del sensor MPU 6050 mediante eventos
//para obtener una lista de los últimos 15 vectores de datos

//Actualizaciones de 1 vector nuevo cada 0.1 segundos

System.out.println("\nLectura mediante eventos");
System.out.println("\nEscuchando cualquier cantidad de datos disponible\n");

StringBuilder sb = new StringBuilder();

//Traducir el buffer (sb) a Data
//Meter Data en el ArrayList
//Vaciar el buffer

ubxPort.addDataListener(new SerialPortDataListener() {
@Override
public int getListeningEvents() { return
SerialPort.LISTENING_EVENT_DATA_AVAILABLE; }
@Override
public void serialEvent(SerialPortEvent event)
{
SerialPort comPort = event.getSerialPort();
byte[] newData = new byte[comPort.bytesAvailable()];
comPort.readBytes(newData, newData.length);
for (int i = 0; i < newData.length; ++i)
{
	char lastChar= (char)newData[i];
	if(lastChar == 'a') //Protocolo: a + vector datos + z
	{
		sb.setLength(0);
	}else if(lastChar == 'z')
	{
		String dataString = sb.toString();
```

```
            Data data = new Data();
            StringBuilder tempString = new StringBuilder();
            int numPosition = 1;
            for(int j=0; j<dataString.length(); j++)
            {

                  if(dataString.charAt(j) != ' ')
                  {
                        tempString.append(dataString.charAt(j));
                  }
                  else
                  {
                        double tempDouble = 0.0;
                        tempDouble = Double.parseDouble(tempString.toString());

                        if(numPosition==1)
                        {
                              data.setAx(tempDouble);
                        }else if (numPosition==2)
                        {
                              data.setAy(tempDouble);
                        }else if (numPosition==3)
                        {
                              data.setAz(tempDouble);
                        }else if (numPosition==4)
                        {
                              data.setGx(tempDouble);
                        }else if (numPosition==5)
                        {
                              data.setGy(tempDouble);
                        }else if (numPosition==6)
                        {
                              data.setGz(tempDouble);
                        }else if (numPosition==7)
                        {
                              data.setYaw(tempDouble);
                        }else if (numPosition==8)
                        {
                              data.setPitch(tempDouble);
                        }else if (numPosition==9)
                        {
                              data.setRoll(tempDouble);
                              numPosition=0;
                        }
                        numPosition++;
                        tempString.setLength(0);
                  }
            }
            if(dataVector.size()==15) {
                  dataVector.remove(0);
            }
            dataVector.add(data);
      }else
```

```
	{
		//Meter el numero sacado del protocolo en el sb
		sb.append(lastChar);
	}
}
}
});


//Esperar el tiempo de calibración del MPU 6050
//para recibir datos fiables
try {
	System.out.println("Calibrando el sensor durante 15 segundos ...");
	Thread.sleep(15000);
} catch (InterruptedException e1) {
	// TODO Auto-generated catch block
	e1.printStackTrace();
}


/*  ---------------------------------------------
 *  -               ALGORITMO                   -
 *  ---------------------------------------------
 */

int robotOrPerson = 1; //Distinción robot/persona
int n=0; //Índice para meter los datos guardados en la tabla MySQL
int registrarCaida=0; //Flag para confirmar la caída por parte del usuario
boolean algorithmFlag = true; //Flag del algoritmo en bucle infinito


while(algorithmFlag)
{
int estado = 0; //Inicializo al estado 0
int direccionCaida=0; //Direccion de la caida en grados
// Ej: 0 = derecha, 90 = adelante, 180 = izquierda, 270 = abajo
String direccion=null; //Cadena que indica la dirección

//Tiempo de espera entre cada anális
//Espero a que se renueven la mitad de los vectores de datos
try {
	Thread.sleep(700);
} catch (InterruptedException e) {
	e.printStackTrace();
}
if(dataVector.size()>=15)
{
ArrayList<Data> dataAnalysis = dataVector;

//Si aplicamos el algoritmo para robots --> Prevención de caídas
if(robotOrPerson==0)
{
//Check de inclinación máxima --> Casída de 30° en cualquier dirección
```

```
for(int k=0; k<15;k++)
{
double tempPitch = dataAnalysis.get(k).getPitch();
double tempRoll = dataAnalysis.get(k).getRoll();
if(tempPitch > 30.0 || tempPitch < -30.0 || tempRoll > 30.0 || tempRoll < -30.0)
{
	System.out.println(dataAnalysis.toString());
	GUIrobot guirobot = new GUIrobot(); //Enviar señal de prevención de caída
	guirobot.accion();
	SQL.addToDB(n, dataAnalysis); //Registrar datos en BD MySQL
	n++;
	try {
		Thread.sleep(5000); //Esperar tiempo de recuperación entre caídas
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}
}
}

//Si aplicamos el algoritmo para personas --> Detección y confirmación de caídas
if(robotOrPerson==1)
{
//Check del algoritmo para caídas en personas --> Recorrer todos los estados
definidos
int indiceCheck=15;
for(int k=0; k<15;k++)
{
	if(estado==0)//Check de pie
{
	System.out.println("0");
	double tempAx = dataAnalysis.get(k).getAx();
	double tempAy = dataAnalysis.get(k).getAy();
	double tempAz = dataAnalysis.get(k).getAz();
	double tempRoll = dataAnalysis.get(k).getRoll();
	double tempPitch = dataAnalysis.get(k).getPitch();
	if((Math.abs(tempAx) + Math.abs(tempAy)) <4 && Math.abs(tempAz)>7 &&
Math.abs(tempPitch) < 20 && Math.abs(tempRoll) < 20)
	{
		estado=1;
		indiceCheck = k;
	}
}
if(estado==1 && indiceCheck<k) //Check >45°
{
	System.out.println("1");
	double tempRoll = dataAnalysis.get(k).getRoll(); //Caída hacia
izquierda/derecha
	double tempPitch = dataAnalysis.get(k).getPitch(); //Caída hacia
delante/atrás
	if(Math.abs(tempPitch) > 45.0 || Math.abs(tempRoll) > 45.0)
	{
		estado=2;
```

```
                indiceCheck = k;
                if(Math.abs(tempPitch)>Math.abs(tempRoll))//Caída hacia
delante/atrás
                {
                        if(tempPitch>0)
                        {
                                direccionCaida=270; //Caída hacia atrás
                        }else
                        {
                                direccionCaida=90; //Caída hacia delante
                        }
                } else
                {
                        if(tempRoll>0)
                        {
                                direccionCaida=0; //Caída hacia derecha
                        }else
                        {
                                direccionCaida=180; //Caída hacia izquierda
                        }
                }
        }
}

if(estado==2 && indiceCheck<k) // Check reducción Az, aumento Ax o Ay
{
        System.out.println("2");
        double tempAx = dataAnalysis.get(k).getAx();
        double tempAy = dataAnalysis.get(k).getAy();
        double tempAz = dataAnalysis.get(k).getAz();
        double tempRoll = dataAnalysis.get(k).getRoll();
        double tempPitch = dataAnalysis.get(k).getPitch();
        if(Math.abs(tempAz)<3 && (Math.abs(tempAx)+Math.abs(tempAy))>7 &&
(Math.abs(tempPitch) > 45.0 || Math.abs(tempRoll) > 45.0))
        {
                estado=3;
                indiceCheck = k;
        }
}

if(estado==3 && indiceCheck<k)// Check golpe
{
        System.out.println("3");
        double tempAx = dataAnalysis.get(k).getAx();
        double tempAy = dataAnalysis.get(k).getAy();
        double tempRoll = dataAnalysis.get(k).getRoll();
        double tempPitch = dataAnalysis.get(k).getPitch();
        if((Math.abs(tempAx)+Math.abs(tempAy))>10 && (Math.abs(tempPitch) > 45.0
|| Math.abs(tempRoll) > 45.0))
        {
                if(direccionCaida == 0)//Caída hacia derecha
                {
                        if(tempAy > 5)
```

```
                        {
                                estado=4;
                                indiceCheck = k;
                        }
                }else if (direccionCaida == 90)//Caída hacia delante
                {
                        if(tempAx < -5)
                        {
                                estado=4;
                                indiceCheck = k;
                        }
                }else if(direccionCaida == 180)//Caída hacia izquierda
                {
                        if(tempAy < -5)
                        {
                                estado=4;
                                indiceCheck = k;
                        }
                }else if(direccionCaida == 270) //Caída hacia atrás
                {
                        if(tempAx > 5)
                        {
                                estado=4;
                                indiceCheck = k;
                        }
                }
        }
}

if(estado==4 && indiceCheck<k)//Check tumbado
{
        System.out.println("4");
        double tempAx = dataAnalysis.get(k).getAx();
        double tempAy = dataAnalysis.get(k).getAy();
        double tempAz = dataAnalysis.get(k).getAz();
        double tempRoll = dataAnalysis.get(k).getRoll();
        double tempPitch = dataAnalysis.get(k).getPitch();
        if(Math.abs(tempAz)< 3 && (Math.abs(tempAx) + Math.abs(tempAy)) >7 &&
(Math.abs(tempPitch) > 45.0 || Math.abs(tempRoll) > 45.0))
        {
                estado=5;
        }
}

if(estado==5)//Caída confirmada
{
        System.out.print("ALERTA! Se ha producido una CAIDA ");
        if(direccionCaida == 0)//Caída hacia derecha
        {
                direccion = "hacia la derecha";
                System.out.println(direccion);
        }else if (direccionCaida == 90)//Caída hacia delante
        {
```

```
            direccion = "hacia delante";
            System.out.println(direccion);
      }else if(direccionCaida == 180)//Caída hacia izquierda
      {
            direccion = "hacia la izquierda";
            System.out.println(direccion);
      }else if(direccionCaida == 270) //Caída hacia atrás
      {
            direccion = "hacia la atrás";
            System.out.println(direccion);
      }
      System.out.println(dataAnalysis.toString());
      GUIpersona gui = new GUIpersona(direccion); //Confirmación de caída
      registrarCaida=gui.accion();  //Llamar a las asistencias sanitarias
      System.out.println(registrarCaida);
      if(registrarCaida==1) //Si la caída se confirma
      {
            SQL.addToDB(n, dataAnalysis); //Registrar datos en BD MySQL
            n++;
      }
      estado=0;
      try {
            Thread.sleep(10000); //Esperar tiempo de recuperación entre caídas
      } catch (InterruptedException e) {
            e.printStackTrace();
      }
}
}
}
}
}
}
}
```

Data.java

Es la clase que define el objeto Data con el que se trabaja a lo largo de todo el algoritmo y permite almacenar los vectores de datos recibidos de la Arduino.

```
package data;

public class Data {

	//Aributos

	double ax;
	double ay;
	double az;
	double gx;
	double gy;
	double gz;
	double yaw;
	double pitch;
	double roll;

	//Constructors

	public Data() {

	}

	public Data(double ax, double ay, double az, double gx, double gy, double
gz, double yaw, double pitch,
			double roll) {
		super();
		this.ax = ax;
		this.ay = ay;
		this.az = az;
		this.gx = gx;
		this.gy = gy;
		this.gz = gz;
		this.yaw = yaw;
		this.pitch = pitch;
		this.roll = roll;
	}


	//Getters & Setters

	public double getAx() {
		return ax;
	}

	public void setAx(double ax) {
```

```
		this.ax = ax;
	}

	public double getAy() {
		return ay;
	}

	public void setAy(double ay) {
		this.ay = ay;
	}

	public double getAz() {
		return az;
	}

	public void setAz(double az) {
		this.az = az;
	}

	public double getGx() {
		return gx;
	}

	public void setGx(double gx) {
		this.gx = gx;
	}

	public double getGy() {
		return gy;
	}

	public void setGy(double gy) {
		this.gy = gy;
	}

	public double getGz() {
		return gz;
	}

	public void setGz(double gz) {
		this.gz = gz;
	}

	public double getYaw() {
		return yaw;
	}

	public void setYaw(double yaw) {
		this.yaw = yaw;
	}

	public double getPitch() {
		return pitch;
```

```
	}

	public void setPitch(double pitch) {
		this.pitch = pitch;
	}

	public double getRoll() {
		return roll;
	}

	public void setRoll(double roll) {
		this.roll = roll;
	}

	//ToString

	@Override
	public String toString() {
		return "Data [ax=" + ax + ", ay=" + ay + ", az=" + az + ", gx=" + gx
+ ", gy=" + gy + ", gz=" + gz + ", yaw="
				+ yaw + ", pitch=" + pitch + ", roll=" + roll + "]";
	}

}
```

SQL.java

Es la clase con la que introducimos los vectores de datos en la base de datos MySQL que indiquemos en la conexión a la base de datos, en este caso se llama "mydb" y tiene por usuario y contraseña "root" y "ubuntu".

```
package jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;

import data.Data;

public class SQL {

	public static void addToDB(int n, ArrayList<Data> dataRecibida){

		Connection con=null;
		PreparedStatement ps = null;

		try {

			// 1. Conexión a la BD
			con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb?serverTimezone=UTC"
,"root","ubuntu");

			// 2. Crear Statement
			ps = con.prepareStatement("INSERT INTO caidas
VALUES(?,?,?,?,?,?,?,?,?,?,?,?);");

			for(int i=0; i<15; i++)
			{
				System.out.println(dataRecibida.get(0));
				Data dataSQL = dataRecibida.get(i);
				ps.setInt(1, n);
				ps.setInt(2,i);
				ps.setDouble(3,dataSQL.getAx());
				ps.setDouble(4,dataSQL.getAy());
				ps.setDouble(5,dataSQL.getAz());
				ps.setDouble(6,dataSQL.getGx());
				ps.setDouble(7,dataSQL.getGy());
				ps.setDouble(8,dataSQL.getGz());
				ps.setDouble(9,dataSQL.getYaw());
				ps.setDouble(10,dataSQL.getPitch());
				ps.setDouble(11,dataSQL.getRoll());
```

```
                        // 3. EJectutar query SQL
                        ps.executeUpdate();
                }
        }
        catch(SQLException exc) {
                exc.printStackTrace();
        }
        catch(Exception exc) {
                exc.printStackTrace();
        }
        finally {
                try {
                        if(ps!= null)
                        ps.close();
                }
                catch(SQLException exc) {
                        exc.printStackTrace();
                }
                try {
                        if(con!= null)
                        con.close();
                }
                catch(SQLException exc) {
                        exc.printStackTrace();
                }
        }
    }
}
```

GUIpersona.java

Clase que define la GUI del sistema para personas. Es una simulación de lo que mostraría por pantalla la aplicación del *smartwatch* o *smartband* con el que se conecta el algoritmo una vez detecta la posible caída en una persona. Esta da la opción de confirmar la caída o denegarla e incluye un temporizador por si el usuario no responde.

```
package gui;

import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;

public class GUIpersona {

	private JFrame frame;
	private JTextArea textArea;
	private JLabel label;
	private JLabel labelTitulo;
	private JButton btnEstoyBien;
	private JButton btnPedirAyuda;

	final static int[] res= {0};

	int salir=0;

	public GUIpersona(String direccion) {
		frame = new JFrame();
		frame.getContentPane().setBackground(new Color(192, 192, 192));
		frame.setBackground(new Color(105, 105, 105));
		frame.setBounds(100, 100, 441, 273);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);

		labelTitulo = new JLabel("SMARTWATCH");
		labelTitulo.setFont(new Font("Dialog", Font.BOLD, 30));
		labelTitulo.setBounds(104, 23, 279, 34);
		frame.getContentPane().add(labelTitulo);

		StringBuilder mensaje= new StringBuilder("   Parece que has sufrido
una caída \n   ");
		mensaje.append(direccion);
```

```
		textArea = new JTextArea(mensaje.toString());
		textArea.setFont(new Font("Dialog", Font.BOLD, 15));
		textArea.setBounds(47, 82, 351, 52);
		frame.getContentPane().add(textArea);
		textArea.setColumns(10);

		btnEstoyBien = new JButton("Estoy bien");
		btnEstoyBien.setForeground(new Color(255, 255, 255));
		btnEstoyBien.setBackground(new Color(0, 100, 0));
		btnEstoyBien.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				System.out.println("Me has pulsado y estas bien");
				res[0]=0;
				JOptionPane.showMessageDialog(textArea, "Me alegro!");
				try {
					Thread.sleep(5000);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
				frame.dispose();
				salir=1;
			}
		});
		btnEstoyBien.setBounds(61, 158, 117, 25);
		frame.getContentPane().add(btnEstoyBien);

		btnPedirAyuda = new JButton("SOS");
		btnPedirAyuda.setForeground(new Color(255, 255, 255));
		btnPedirAyuda.setBackground(new Color(255, 0, 0));
		btnPedirAyuda.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent arg0) {
				System.out.println("Me has pulsado y estas KO");
				res[0]=1;
				JOptionPane.showMessageDialog(textArea, "Llamando a los
servicios médicos...");
				try {
					Thread.sleep(5000);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
				frame.dispose();
				salir=1;
			}
		});
		btnPedirAyuda.setBounds(240, 157, 143, 26);
		frame.getContentPane().add(btnPedirAyuda);

		label = new JLabel("15");
		label.setBounds(302, 195, 70, 15);
		frame.getContentPane().add(label);
	}
```

```
	public int accion() {
		res[0]=0;
		frame.setVisible(true);

			for(int i=15; i>-1; i--)
			{
				if(salir==1)
				{
					break;
				}
				else{
					try {
						Thread.sleep(1000);
						if(salir==1)
						{
							break;
						}
						label.setText(Integer.toString(i));
						frame.setVisible(true);
						if(i==0)
						{
							label.setText(Integer.toString(i));
							frame.setVisible(true);
							res[0]=1;
						}
					}catch (InterruptedException e) {
					e.printStackTrace();
					}
				}
			}
			JOptionPane.showMessageDialog(textArea, "Llamando a los
servicios médicos...");
		try {
			Thread.sleep(5000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		return res[0];
		}
}
```

GUIrobot.java

Clase que define la GUI del sistema para robots. Es una simulación de lo que haría el Nannybot una vez detecta la caída. Simplemente muestra el aviso de la caída detectada y simula enviar una señal de detección de caída para activar el sistema de levantamiento del Nannybot.

```
package gui;

import java.awt.Color;
import java.awt.Font;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;


public class GUIrobot {

	private JFrame frame;
	private JLabel labelTitulo;
	private JLabel labelMensaje;

	public GUIrobot() {
		frame = new JFrame();
		frame.getContentPane().setBackground(new Color(192, 192, 192));
		frame.setBackground(new Color(105, 105, 105));
		frame.setBounds(100, 100, 441, 273);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.getContentPane().setLayout(null);

		labelTitulo = new JLabel("NANNYBOT");
		labelTitulo.setFont(new Font("Dialog", Font.BOLD, 30));
		labelTitulo.setBounds(124, 26, 197, 34);
		frame.getContentPane().add(labelTitulo);

		labelMensaje = new JLabel("");
		labelMensaje.setBounds(177, 115, 68, 21);
		frame.getContentPane().add(labelMensaje);

		frame.setVisible(true);
	}

	public void accion()
	{
		JOptionPane.showMessageDialog(labelMensaje, "Activando Sistema de
Prevención de Caída...");
	}
}
```

CalibrarDMP.ino

Código que calcula los *offsets* específicos de cada MPU 6050, estos se han de incluirse en el código siguiente de Arduino para poder leer los datos del sensor correctamente.

```
// I2Cdev and MPU6050 must be installed as libraries
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

///////////////////////////////////   CONFIGURATION
//////////////////////////////
//Change this 3 variables if you want to fine tune the skecth to your needs.
int buffersize=1000;     //Amount of readings used to average, make it higher to
get more precision but sketch will be slower  (default:1000)
int acel_deadzone=8;     //Acelerometer error allowed, make it lower to get more
precision, but sketch may not converge  (default:8)
int giro_deadzone=1;     //Giro error allowed, make it lower to get more
precision, but sketch may not converge  (default:1)

// default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
//MPU6050 accelgyro;
MPU6050 accelgyro(0x68); // <-- use for AD0 high

int16_t ax, ay, az,gx, gy, gz;

int mean_ax,mean_ay,mean_az,mean_gx,mean_gy,mean_gz,state=0;
int ax_offset,ay_offset,az_offset,gx_offset,gy_offset,gz_offset;

///////////////////////////////////   SETUP
////////////////////////////////////
void setup() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
  Wire.begin();
  // COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo measured
250kHz.

  // initialize serial communication
  Serial.begin(115200);

  // initialize device
  accelgyro.initialize();

  // wait for ready
  while (Serial.available() && Serial.read()); // empty buffer
```

```
  while (!Serial.available()){
    Serial.println(F("Send any character to start sketch.\n"));
    delay(1500);
  }
  while (Serial.available() && Serial.read()); // empty buffer again

  // start message
  Serial.println("\nMPU6050 Calibration Sketch");
  delay(2000);
  Serial.println("\nYour MPU6050 should be placed in horizontal position, with
package letters facing up. \nDon't touch it until you see a finish message.\n");
  delay(3000);
  // verify connection
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" :
"MPU6050 connection failed");
  delay(1000);
  // reset offsets
  accelgyro.setXAccelOffset(0);
  accelgyro.setYAccelOffset(0);
  accelgyro.setZAccelOffset(0);
  accelgyro.setXGyroOffset(0);
  accelgyro.setYGyroOffset(0);
  accelgyro.setZGyroOffset(0);
}

////////////////////////////////   LOOP   ////////////////////////////////////
void loop() {
  if (state==0){
    Serial.println("\nReading sensors for first time...");
    meansensors();
    state++;
    delay(1000);
  }

  if (state==1) {
    Serial.println("\nCalculating offsets...");
    calibration();
    state++;
    delay(1000);
  }

  if (state==2) {
    meansensors();
    Serial.println("\nFINISHED!");
    Serial.print("\nSensor readings with offsets:\t");
    Serial.print(mean_ax);
    Serial.print("\t");
    Serial.print(mean_ay);
    Serial.print("\t");
    Serial.print(mean_az);
    Serial.print("\t");
    Serial.print(mean_gx);
    Serial.print("\t");
```

```
    Serial.print(mean_gy);
    Serial.print("\t");
    Serial.println(mean_gz);
    Serial.print("Your offsets:\t");
    Serial.print(ax_offset);
    Serial.print("\t");
    Serial.print(ay_offset);
    Serial.print("\t");
    Serial.print(az_offset);
    Serial.print("\t");
    Serial.print(gx_offset);
    Serial.print("\t");
    Serial.print(gy_offset);
    Serial.print("\t");
    Serial.println(gz_offset);
    Serial.println("\nData is printed as: acelX acelY acelZ giroX giroY giroZ");
    Serial.println("Check that your sensor readings are close to 0 0 16384 0 0
0");
    Serial.println("If calibration was succesful write down your offsets so you
can set them in your projects using something similar to
mpu.setXAccelOffset(youroffset)");
    while (1);
  }
}

/////////////////////////////////   FUNCTIONS
/////////////////////////////////////
void meansensors(){
  long i=0,buff_ax=0,buff_ay=0,buff_az=0,buff_gx=0,buff_gy=0,buff_gz=0;

  while (i<(buffersize+101)){
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    if (i>100 && i<=(buffersize+100)){ //First 100 measures are discarded
      buff_ax=buff_ax+ax;
      buff_ay=buff_ay+ay;
      buff_az=buff_az+az;
      buff_gx=buff_gx+gx;
      buff_gy=buff_gy+gy;
      buff_gz=buff_gz+gz;
    }
    if (i==(buffersize+100)){
      mean_ax=buff_ax/buffersize;
      mean_ay=buff_ay/buffersize;
      mean_az=buff_az/buffersize;
      mean_gx=buff_gx/buffersize;
      mean_gy=buff_gy/buffersize;
      mean_gz=buff_gz/buffersize;
    }
    i++;
    delay(2); //Needed so we don't get repeated measures
  }
```

```
}

void calibration(){
  ax_offset=-mean_ax/8;
  ay_offset=-mean_ay/8;
  az_offset=(16384-mean_az)/8;

  gx_offset=-mean_gx/4;
  gy_offset=-mean_gy/4;
  gz_offset=-mean_gz/4;
  while (1){
    int ready=0;
    accelgyro.setXAccelOffset(ax_offset);
    accelgyro.setYAccelOffset(ay_offset);
    accelgyro.setZAccelOffset(az_offset);

    accelgyro.setXGyroOffset(gx_offset);
    accelgyro.setYGyroOffset(gy_offset);
    accelgyro.setZGyroOffset(gz_offset);

    meansensors();
    Serial.println("...");

    if (abs(mean_ax)<=acel_deadzone) ready++;
    else ax_offset=ax_offset-mean_ax/acel_deadzone;

    if (abs(mean_ay)<=acel_deadzone) ready++;
    else ay_offset=ay_offset-mean_ay/acel_deadzone;

    if (abs(16384-mean_az)<=acel_deadzone) ready++;
    else az_offset=az_offset+(16384-mean_az)/acel_deadzone;

    if (abs(mean_gx)<=giro_deadzone) ready++;
    else gx_offset=gx_offset-mean_gx/(giro_deadzone+1);

    if (abs(mean_gy)<=giro_deadzone) ready++;
    else gy_offset=gy_offset-mean_gy/(giro_deadzone+1);

    if (abs(mean_gz)<=giro_deadzone) ready++;
    else gz_offset=gz_offset-mean_gz/(giro_deadzone+1);

    Serial.print(ready);

    if (ready==6) break;
  }
}
```

AGYRP.ino

Código principal para leer los vectores de datos del MPU 6050 en Arduino. Debe incluir los *offsets* calculados con el código anterior.

```
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#include "Wire.h"
#include "MPU6050.h"

const int mpuAddress = 0x68;  // Puede ser 0x68 o 0x69
MPU6050 mpu(mpuAddress);

int ax, ay, az;
int gx, gy, gz;

// Factores de conversion
const float accScale = 2.0 * 9.81 / 32768.0;
const float gyroScale = 250.0 / 32768.0;

#define OUTPUT_READABLE_YAWPITCHROLL
#define LED_PIN 13
#define VCC2  4 // define pin 4 or any other digial pin here as VCC2

bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 =
success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorFloat gravity;    // [x, y, z]            gravity vector
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and
gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r',
'\n' };


// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================
```

```
volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin
has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

void setup() {

   pinMode(VCC2,OUTPUT);//define a digital pin as output
   digitalWrite(VCC2, HIGH);// set the above pin as HIGH so it acts as 5V

    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue
immediately

    // initialize device
    mpu.initialize();

    while (Serial.available() && Serial.read()); // empty buffer

    // load and configure the DMP

    devStatus = mpu.dmpInitialize();

    //I supplied my own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(-32);
    mpu.setYGyroOffset(88);
    mpu.setZGyroOffset(-1);
    mpu.setZAccelOffset(1680);


    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready

        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection

        attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to
use it
```

```
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}


// Function to Print Acceleration and Angular Velocity
void printSI()
{
   Serial.print(ax * accScale);
   Serial.print(" ");
   Serial.print(ay * accScale);
   Serial.print(" ");
   Serial.print(az * accScale);
   Serial.print(" ");
   Serial.print(gx * gyroScale);
   Serial.print(" ");
   Serial.print(gy * gyroScale);
   Serial.print(" ");
   Serial.print(gz * gyroScale);
   Serial.print(" ");
}


void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();

    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too
inefficient)
```

```
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();

    // otherwise, check for DMP data ready interrupt (this should happen
frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        #ifdef OUTPUT_READABLE_YAWPITCHROLL

             // Read Accelerations and Angular Velocities
             mpu.getAcceleration(&ax, &ay, &az);
             mpu.getRotation(&gx, &gy, &gz);

             //Start of Protocol
             Serial.print("a");
             printSI();

            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

            Serial.print(ypr[0] * 180/M_PI);
            Serial.print(" ");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print(" ");
            Serial.print(ypr[2] * 180/M_PI);
            Serial.print(" ");
            Serial.print("z");
            delay(100);
            //End of Protocol

        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```