ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS CIUDAD DE MÉXICO

UNIVERSIDAD DEL ROSARIO

ESCUELA DE INGENIERÍA Y CIENCIAS
DEPARTAMENTO DE INGENIERÍA MECATRÓNICA

KAG- Kinetics Analysis of Gait

KEVIN ROGERS CARDENAS MOGOLLON A01679962 IMD
SCARLETT JAZMÍN ZÚÑIGA RAMÍREZ A01333215 ITS

CLIENTE DEL PROYECTO:

CENTRO DE INVESTIGACIÓN EN MICROSISTEMAS Y BIODISEÑO (CIMB)
ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO - UNIVERSIDAD DEL ROSARIO

ASESOR:

Dr. LUIS EDUARDO RODRIGUEZ CHEU
Dr. MARTIN ROGELIO BUSTAMANTE BELLO

MÉXICO D.F. a 22 de noviembre del 2018

INDICE PRINCIPAL

INDICE DE FIGURAS

INDICE DE TABLAS

CAPITULO 1. INTRODUCCIÓN

1.1 PROBLEMATICA

El análisis del movimiento humano y más concretamente de la marcha, ha sido un tema de mucho interés desde tiempos remotos, permitiendo el desarrollo de diferentes métodos para su estudio. El avance tecnológico computacional ha permitido obtener modernos y sofisticados sistemas para el análisis del movimiento humano. El análisis cuantitativo de la marcha actualmente se reconoce como una herramienta de investigación y enseñanza, además de tener diversas aplicaciones en el campo clínico para el estudio y tratamientos de sus alteraciones, generado resultados positivos al facilitar la comprensión de los mecanismo subyacentes de las alteraciones generadas por patologías neurológicas, musculoesqueléticas o trastornos psiquiátricos como lo son la parálisis cerebral, espina bífida, apraxia, hipertonía muscular, entre otras [1].

La marcha humana corresponde a una secuencia de movimientos coordinados y alternantes los cuales nos permiten desplazarnos. Es un proceso bastante complejo que requiere el adecuado funcionamiento e interacción de diferentes estructuras tales como un sistema de control a cargo del SNC (Sistema Nervioso Central), palancas que provean el movimiento correspondiente a huesos y fuerzas para mover las palancas a cargo del sistema muscular, cualquier alteración en estos niveles puede causar alteraciones en la marcha. Las aplicaciones del laboratorio de marcha en el campo clínico se remontan desde la década de los 90 en Norteamérica dando origen al primer laboratorio de análisis de movimiento en el Hospital Shriners de San Francisco, con el fin de estudiar las alteraciones biomecánicas de niños con parálisis cerebral.

Las actividades durante la marcha ocurren simultáneamente a diferentes niveles articulares y en diferentes planos, por ende, resulta bastante complejo captar todos los elementos con un simple análisis observacional. Por otro lado, las alteraciones son difíciles de identificar debido a que las compensaciones que el paciente emplea en su

marcha pueden ser enmascaradas. Por este motivo los sistemas de análisis de movimiento han ganado un gran campo en la aplicación clínica del estudio de alteraciones del patrón de marcha, identificación y definición de anormalidades, orientar los posibles tratamientos, cuantificación de resultados y seguimiento de la evolución a lo largo del tiempo. No solo su aplicabilidad ha sido en el ambiente clínico ha logrado convertirse en una herramienta muy útil en el rendimiento de los deportistas generando una retroalimentación cualitativa permitiendo de este modo que el atleta pueda mejorar sin verse afectado por la presencia de lesiones a corto o largo plazo, permitiendo evaluar alteraciones en la marcha con la estimación del conjunto de la cinemática de la rodilla.

1.2 JUSTIFICACIÓN

Actualmente el análisis de la marcha humana se lleva a cabo con un sistema de rastreo de movimiento con cámaras en un laboratorio (Kinescan, entre otros), el principal inconveniente que presenta llevar a un atleta o un paciente con trastornos en la marcha es la limitación de su rendimiento debido a que la marcha de un corredor es menos funcional en una cinta rodante en comparación con las características de la marcha en la pista, si nos referimos a un paciente con trastornos en la marcha no es muy óptimo alejarlo del ambiente clínico por lo que actualmente existen hospitales o clínicas en donde dentro de sus instalaciones tiene laboratorios destinados para el análisis de movimiento. El uso de este sistema tiene la ventaja de poderse implementar como un sistema estándar debido a su alta precisión, aunque este sistema representa un gran gasto monetario además que se realiza en un entorno muy artificial.

Una solución actual para no depender del uso del sistema anteriormente descrito es con la vinculación de sensores inerciales debido a que las ventajas de un sistema inercial son: Su bajo costo en comparación a un sistema de análisis de movimiento, un consumo de baja energía durante su funcionamiento, el usuario no se ve limitado al estar en un entorno artificial debido a que es posible trabajar en un laboratorio de formación real o incluso durante la competencia y además tiene una amplia

aplicabilidad en diferentes áreas como lo son la ingeniería biomédica, robótica, realidad virtual y para el mejor de los casos el seguimiento en tiempo real de la orientación de las partes del cuerpo humano en un espacio tridimensional 3D.

1.3 OBJETIVOS

Objetivo general

Diseñar un sistema portátil de comunicación y procesamiento de parámetros fisiológicos para la implementación sobre un exoesqueleto, permitiendo tener un seguimiento de los parámetros de la marcha.

Objetivos específicos:

- Diseñar una tarjeta que permita la adquisición de datos de diversos sensores.
- Diseñar un sistema de comunicación universal para la adquisición y transmisión de datos.
- Realizar un algoritmo de lectura y procesamiento de datos para la estimación del rango articular de la rodilla.

1.4 ALCANCES Y ENTREGABLES DEL PROYECTO

Presentar una tarjeta previamente diseñada, que funcionará como nodo para cada uno de los puntos de referencia en el análisis de la marcha ubicados con base al sistema de coordenadas conjuntas (JCS - Joint Coordinate System) [2], de donde se adquieren los valores medidos por los sensores de unidad inercial , para posteriormente ser transmitidos mediante el protocolo de comunicación CAN hacia el nodo central el cual se encargará de procesar los datos con el fin de extraer la señal característica del rango articular de la rodilla en el periodo de flexión y extensión. Se presentará prueba de la obtención de estos datos, así como prueba de la comunicación entre nodos y un nodo central.

1.5 IMPACTOS DEL PROYECTO

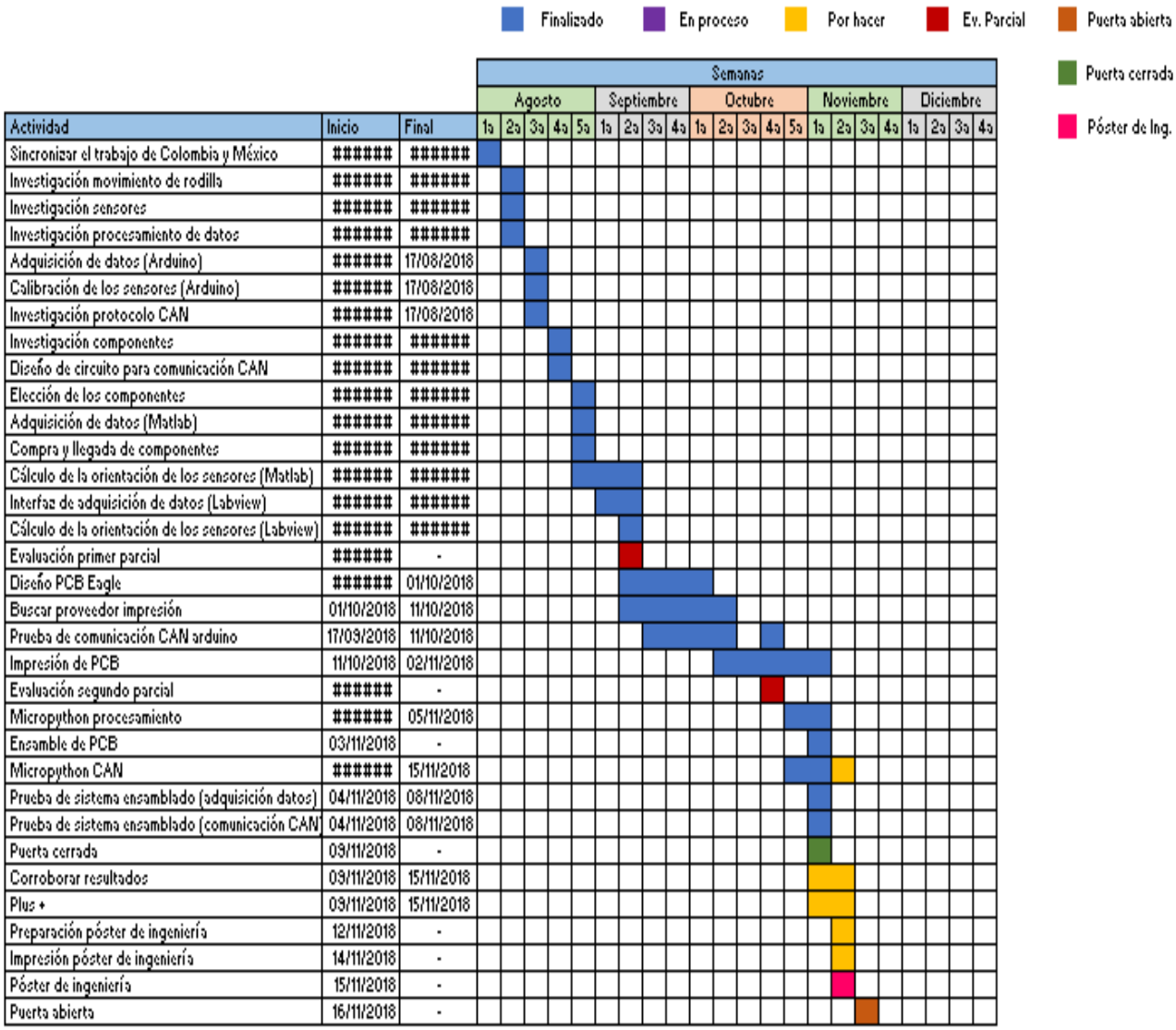
La creación y vinculación de sistemas que permitan el análisis del movimiento humano específicamente la marcha humana permite mejorar la calidad de vida de muchas personas que han sufrido alguna enfermedad neuromuscular, además de incentivar el estudio y la enseñanza de los conceptos biomecánicos que abarcan la marcha humana no solo en personas con patologías sino en personas sanas incluso en deportistas de alto rendimiento. Aunque actualmente existen varios sistemas de medición para el análisis de marcha para la adquisición de datos los cuales se basan en un sistemas cerrado de cámaras se propone usar un sistema de adquisición basado en el uso de sensores inerciales debido que su bajo costo, consumo de baja energía, portabilidad, fácil manipulación y principalmente no existe limitación alguna por lo que es posible trabajar con este sistema en cualquier ambiente debido que no necesita un sistema de referencia externo.

1.6 DESCRIPCIÓN DE LA EMPRESA O ENTIDAD

El Centro de Investigación en Microsistemas y Biodiseño se encarga de desarrollar diversos proyectos enfocados en el área biomédica, automotriz, telecomunicaciones, inteligencia artificial y biosistemas. Es un centro de investigaciones del Tecnológico de Monterrey Campus Ciudad de México, actualmente tiene proyectos como ADMAS (automotriz), ICARUS (biomédica), C2C (telecomunicaciones).

1.7 PLAN DE ACTIVIDADES

Tabla 1. 1 PLAN DE ACTIVIDADES



Actividad	Inicio	Final
Sincronizar el trabajo de Colombia y México	######	######
Investigación movimiento de rodilla	######	######
Investigación sensores	######	######
Investigación procesamiento de datos	######	######
Adquisición de datos (Arduino)	######	17/08/2018
Calibración de los sensores (Arduino)	######	17/08/2018
Investigación protocolo CAN	######	17/08/2018
Investigación componentes	######	######
Diseño de circuito para comunicación CAN	######	######
Elección de los componentes	######	######
Adquisición de datos (Matlab)	######	######
Compra y llegada de componentes	######	######
Cálculo de la orientación de los sensores (Matlab)	######	######
Interfaz de adquisición de datos (Labview)	######	######
Cálculo de la orientación de los sensores (Labview)	######	######
Evaluación primer parcial	######	-
Diseño PCB Eagle	######	01/10/2018
Buscar proveedor impresión	01/10/2018	11/10/2018
Prueba de comunicación CAN arduino	17/09/2018	11/10/2018
Impresión de PCB	11/10/2018	02/11/2018
Evaluación segundo parcial	######	-
Micropython procesamiento	######	05/11/2018
Ensamble de PCB	03/11/2018	-
Micropython CAN	######	15/11/2018
Prueba de sistema ensamblado (adquisición datos)	04/11/2018	08/11/2018
Prueba de sistema ensamblado (comunicación CAN)	04/11/2018	08/11/2018
Puerta cerrada	09/11/2018	-
Corroborar resultados	09/11/2018	15/11/2018
Plus +	09/11/2018	15/11/2018
Preparación póster de ingeniería	12/11/2018	-
Impresión póster de ingeniería	14/11/2018	-
Póster de ingeniería	15/11/2018	-
Puerta abierta	16/11/2018	-

1.8 ANÁLISIS FODA

Figura 1. 1 ANALISIS FODA

CAPITULO 2. ANTECEDENTES

2.1 ANTECEDENTES DE LA SOLUCIÓN

Para el análisis de la marcha existen distintos sistemas como lo es el Kinescan el cual es un sistema fijo, otro sistema es el Xsens el cual es un sistema portátil. A continuación, mostramos las características de estos sistemas, así como las ventajas de nuestro sistema ante estos.

Kinescan:

Es un sistema de video digital con preprocesado, consta de 4 cámaras las cuales con ayuda de indicadores que se colocan de manera estratégica definidos por el JCS,

puede saber de manera automática y en tiempo real las posiciones de los diferentes segmentos corporales y articulaciones, y los valores de las variables cinemáticas y dinámicas. Este sistema además puede realizar un análisis en 3 dimensiones del movimiento completo de la persona; para poder tener un este análisis se requiere de un tiempo mínimo de 30 minutos de prueba. A diferencia de nuestro sistema, el Kinescan no es un sistema portátil, y se debe realizar la prueba en un entorno cerrado, donde se encuentren colocadas las 4 cámaras de manera estratégica, además nuestro sistema es considerablemente más barato, ya que el costo del sistema aproximado es de $259,000.00, mientras que el nuestro es de $11,003.94 aproximadamente.

Xsens:

Es un sistema de análisis de movimiento, consta de sensores inerciales que te permite saber la posición de los diferentes segmentos corporales establecidos por el JCS, además de brindarte la posición del cuerpo te muestra las gráficas cinemáticas del movimiento, así como mostrar el gráfico en 3 dimensiones del movimiento. Es un sistema portátil, lo que te permite calcular el movimiento en cualquier entorno sin necesidad de estar en un cuarto cerrado como el proyecto presentado previamente. Aunque Xsens tiene una esencia parecida a KAG, sigue presentando una ventaja económica nuestro proyecto, ya que este sistema tiene un costo de $274,000.00 sin considerar el costo del software, mientras que el nuestro es de $11,003.94 aproximadamente. Por ello nuestro sistema representa una ventaja para el usuario.

MoCap – Nexus:

Es un sistema de captura de movimiento, que consta de 15 cámaras colocadas estratégicamente en un cuarto cerrado, que, con la ayuda de indicadores colocados en el cuerpo según el JCS, puede realizar un análisis completo del movimiento, así como graficar en 3 dimensiones el movimiento realizado. Este sistema, al igual que el Kinescan, es un sistema no portátil, y debe ser utilizado en un entorno cerrado. Otra

desventaja de este sistema es la calibración de las cámaras, ya que tardan alrededor de 15 minutos en calibrarse, a diferencia de nuestro sistema que tarda alrededor de 3 minutos. Además de la rapidez de calibración, nuestro sistema tiene ventaja sobre MoCap en términos económicos y de portabilidad.

CAPITULO 3. ESTADO DEL ARTE

La medición del rango de movimiento (ROM) ha sido un importante requerimiento en varios campos de la medicina como los son la ortopedia y los diferentes campos de la rehabilitación, además de ser utilizados en exámenes clínicos rutinarios.

El dispositivo más comúnmente utilizado dentro del área hospitalaria para la medición del rango de movimiento es el goniómetro o transportador de ángulos (figura 3.1), es un instrumento de medición de ángulos (agudos 90°>, llanos 180° o obtusos > 180°) con forma de semicírculo o circulo graduado (de grado en grado), en 180° o 360°.

Figura 3. 1 Medición del ángulo de la rodilla a través de un goniómetro

Pese a que el goniómetro es instrumento practico y de alta precisión, al momento de la medición de este rango articular en el paciente o usuario debe encontrarse en una posición estática y además hay un agente externo quien realiza la medición, en caso de que se necesite verificar el rango articular en un movimiento activo al utilizar este

instrumento de medición estaríamos limitando y atrofiando el movimiento natural con el cual contamos los seres humanos.

Es debido a esta necesidad que surgen nuevas tecnologías las cuales utilizan la cinemática la cual se encarga del estudio del movimiento de los cuerpos sin atender a las causas que lo producen o modifican, centrándose únicamente en las magnitudes que los componen como los son: posición, velocidad y aceleración.

El método mas común para la adquisición de datos cinemáticos utiliza un sistema de imágenes el cual captura el movimiento, a través de unos marcadores los cuales son fijados en un sujeto en movimiento, posteriormente se realiza una obtención de coordenadas por medio de un sistema digital manual o automático. Estas coordenadas son procesadas para obtener las variables cinemáticas que describen movimientos articulares o segmentarios.

Los sistemas mas comunes para la adquisición de estas coordenadas utilizan video, video digital o dispositivos de carga acoplada (CCD) (figura 3.2), estos dispositivos graban el movimiento usando la luz del ambiente o la luz que es emitida a través de marcadores reflectivos. (figura 3.3)

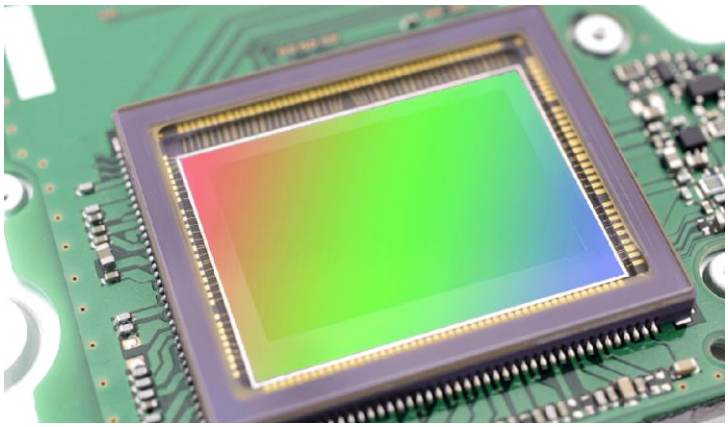


Figura 3. 2 Sensor CCD

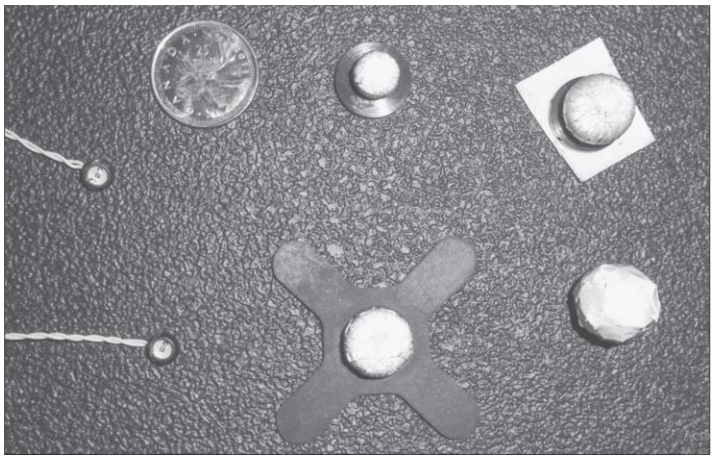


Figura 3. 3 Marcadores utilizados en la adquisición de datos cinemáticos

Las cámaras utilizadas cuentan con su propia fuente de luz la cual será reflejada por medio de los marcadores generando un contraste sobre el marcador para poderlo diferenciar sobre la piel, ropa y el fondo de área donde se esta realizando el estudio.

Para la captura del movimiento inicialmente se utilizo un sistema de dos cámaras sin embargo se evidencio que los marcadores en algunas circunstancias eran bloqueados por alguna parte del cuerpo o perdidos del campo de visión de la cámara es debido a estas situaciones que se estandarizó un sistema de multicámara (figura 3.4)

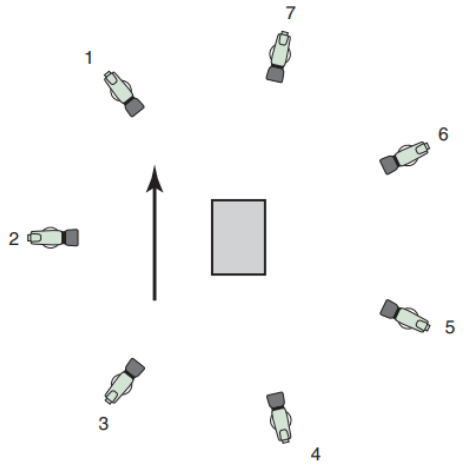


Figura 3. 4 Sistema de multicámara (7 cámaras)

A pesar de que esta tecnología se ha convertido en una importante herramienta para el estudio biomecánico, también es una tecnología de un alto costo y además es usada en un espacio muy limitado ocasionado que muy poca gente pueda acceder a esta tecnología. Es por esta

razón que ingresa otra tecnología al estudio cinemático y que tiene como base el uso de sensores inerciales (figura 3.5).

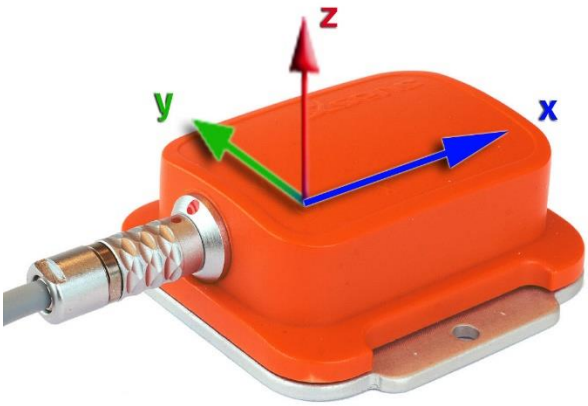


Figura 3. 5 Sensores inerciales

La estimación del movimiento a través de los sensores inerciales es debido a su composición ya que son dispositivos electrónicos los cuales cuentan con un acelerómetro y giroscopio tridimensional permitiendo obtener información en todo un campo de tres dimensiones y por ende tener un plano completo del movimiento.

CAPITULO 4. DESARROLLO

4.1 DESARROLLO

Debido a que en este proyecto están involucradas distintas ingenierías, escuelas y países, se comenzó por realizar una junta el 4 de agosto de 2018, en esta junta se especificaron los diferentes alcances que cada institución había logrado en términos del proyecto (exo-esqueleto, detección de la marcha). Posteriormente se realizaron investigaciones sobre el movimiento de la rodilla por parte del Instituto Tecnológico de Monterrey, para poder regularizarse en temas de la biomecánica del movimiento de la marcha (Anexo I); posteriormente se realizaron investigaciones sobre los distintos componentes que podrían utilizarse (Tabla 4.1), como lo sensores para la

detección del ángulo de la rodilla, así como de los filtros correspondientes para el procesamiento de estos datos.

Segunda etapa: se realizó la adquisición de datos de los sensores inerciales con los IDE Arduino y LabVIEW. Posteriormente a estos datos se les aplicó un preprocesamiento utilizando un Arduino UNO y el IDE, el cual consiste en la calibración de los sensores inerciales junto con la aplicación de un filtro pasa bajas.

Tercera etapa: se realizó una investigación, paralelamente se utilizó herramientas como Matlab para el desarrollo y posteriormente la aplicación de un Filtro Kalman sobre los sensores inerciales con el objetivo de encontrar y describir la orientación , sobre la comunicación mediante protocolo CAN (Anexo III), y se diseñó el circuito correspondiente a esta comunicación (figura 4.3), para finalizar esta etapa se realizaron pruebas de la comunicación con distintos módulos CAN, utilizando estos módulos como esclavos, asignados a un maestro.

Cuarta etapa: se realizó el procesamiento de los datos en el maestro, que luego de realizar un estudio se escogió la pyboard, una placa oficial diseñada por micropython utilizando un microcontrolador STM32F405, el cual se utilizará como maestro en la comunicación por protocolo CAN y además se aplicará el Filtro de Kalman a los datos adquiridos por medio de esta comunicación.

Para finalizar el proyecto se realizaron pruebas con las tarjetas impresas y el microcontrolador asignado a ser maestro.

4.2 RECURSOS Y ANALISIS ECONOMICO

Tabla 4. 1 Componentes seleccionados y costos

Materiales	Proveedor	Cantidad	Costo/U dlls	Costo T dlls	
Integrados					
ATMEGA328P	Newark	5	3.2	16	
MCP2515	Newark	5	3.04	15.2	
MCP2551	Newark	5	2.57	12.85	
TCAN337	Newark	3	3.59	10.77	
PYBOARD	Mouser	2	58.44	116.88	
MPU6050	Newark	4	8.12	32.48	
Resistencias					
10K	AG	12	0.144	1.728	
47K	AG	6	0.082	0.492	
100	AG	5	0.144	0.72	
120	AG	5	0.144	0.72	
Tarjetas					
ARDUINO UNO	Mouser	2	28.6	57.2	
CAN-BUS SHIELD	Mouser	2	31.85	63.7	
CAN MODULE	Mouser	3	4.94	14.82	
Condensadores					
33pF	AG	10	0.13	1.3	
560pF	AG	5	0.196	0.98	
100nF	AG	10	0.602	6.02	
Otros					
MICRO-SWITCH	AG	5	1.83	9.15	
OSCILADOR (16MH	NEWARK	10	1.37	13.7	
LED	AG	12	1.37	16.44	
PCB	Seeedstudio	10	3.36	33.6	Total MXN
				424.75	8070.25

Para la realización de este proyecto se optó por utilizar un microcontrolador ATMEGA328P debido a su facilidad de programación ya que es posible utilizar un Arduino el cual se conectará a una tarjeta previamente diseñada para posteriormente cargar el respectivo programa utilizando como programador el IDE de Arduino, junto a este se tiene pensado utilizar dos softwares de alto nivel como lo son Labview y Matlab los cuales cumplirán la función de adquisición y procesamiento respectivamente. Debido a que los datos adquiridos se enviaran usando el protocolo de comunicación CAN para posteriormente ser adquiridos por un nodo central el cual se utilizara como herramienta de programación el IDE de Arduino ya que actualmente existen librerías para tarjetas desarrolladas para trabajar con comunicación CAN por medio de un arduino y una tarjeta externa (CAN-BUS shield).

4.3 DISEÑO

Para que se logre conseguir el objetivo del proyecto, lo primero que se debe definir es la posición de los sensores inerciales para que la lectura sea eficiente (figura 4.1); una vez teniendo la posición de los sensores, podemos decidir cómo se conformará nuestro sistema (nodo/maestro) (figura 4.2). Más adelante se explicará esto a mayor detalle.

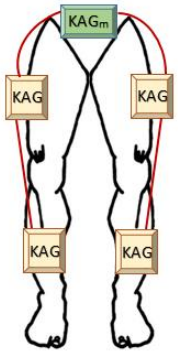


Figura 4. 1 Posición de sensores inerciales

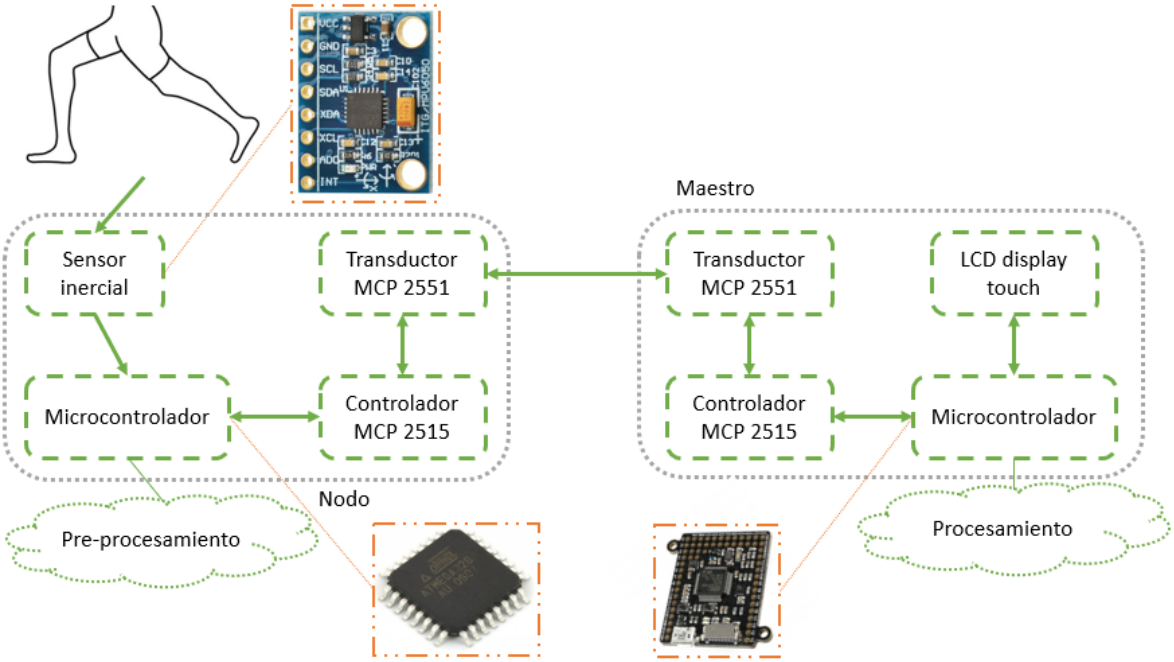


Figura 4. 2 Diseño general del sistema

Debido a que actualmente existen varios componentes electrónicos que permiten una comunicación mediante el protocolo CAN, los cuales nos permitieron tener un acercamiento como lo son el módulo MCP2515 o la tarjeta CAN-BUS shield V1.2 los cuales permiten comunicarse por medio de una interfaz SPI a través de una placa de microcontrolador de código abierto como lo es el Arduino UNO, se utilizó esta placa debido a que la tarjeta CAN-BUS viene prediseñada para poderse acoplar sobre el Arduino UNO formando un solo sistema en donde es posible utilizar los pines de la tarjeta como entradas o salidas del Arduino UNO, de igual forma se usó la interfaz SPI para el uso del módulo MCP2515. Sin embargo, al momento de formar un único sistema en donde está involucrado la tarjeta CAN-BUS, el Arduino y el sensor MPU6050 encontramos un sistema poco eficiente en cuanto al tamaño y el costo.

Debido a la baja eficiencia de tamaño y costo descrita anteriormente se optó por diseñar un sistema el cual contará con la misma funcionalidad y además funcione como un sistema portable en donde no existiera la necesidad de conexiones externas o el uso de cables los cuales pueden generar ruido, buscando un sistema compacto con una gran eficiencia en cuanto al tamaño logrando modificar la dimensión con base a las necesidades, todo esto teniendo en cuenta que no se exceda el costo total

Para poder realizar de manera adecuada una comunicación mediante protocolo CAN, requerimos de un módulo capaz de soportar este protocolo, así como un transceptor que te permita la codificación y decodificación de datos. Por ello se diseñó un circuito donde se conectará el transductor, el controlador CAN y el microcontrolador. La figura 4.3 muestra el diseño realizado.

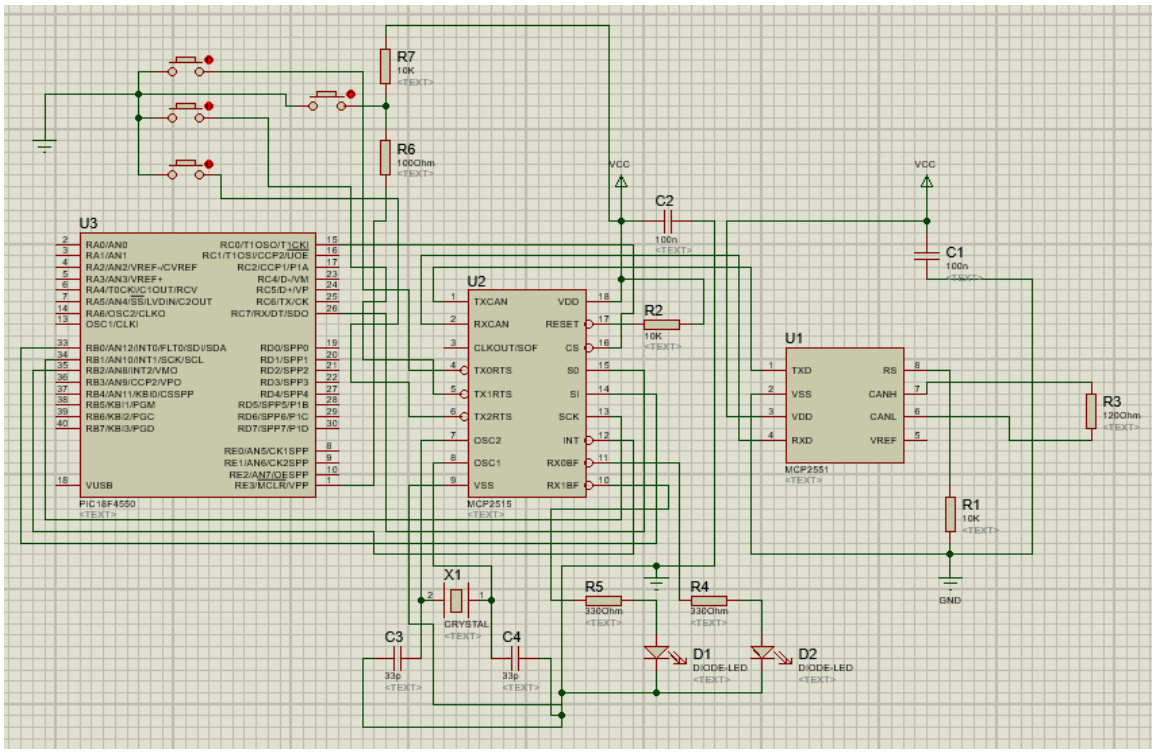


Figura 4. 3 Diseño del sistema en Proteus

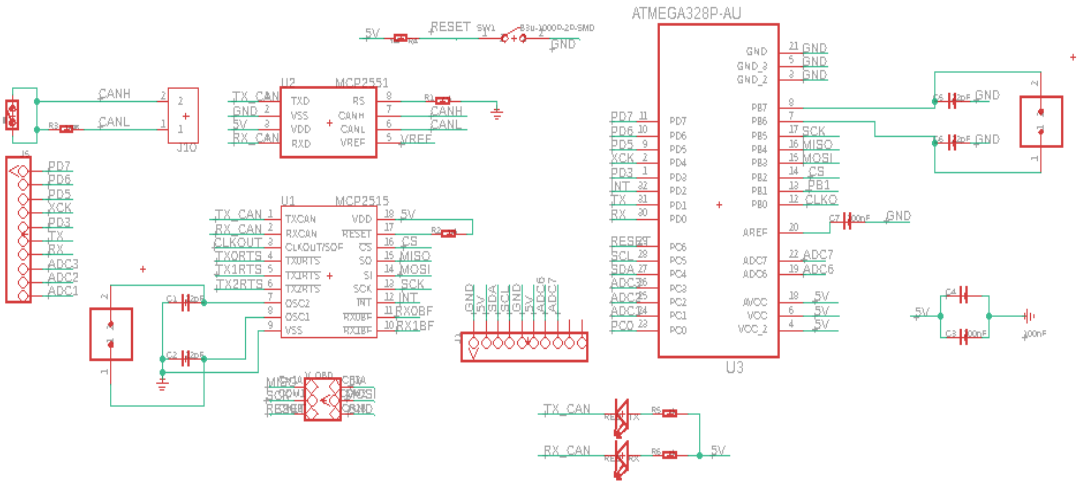


Figura 4. 4 Diseño del sistema en Eagle

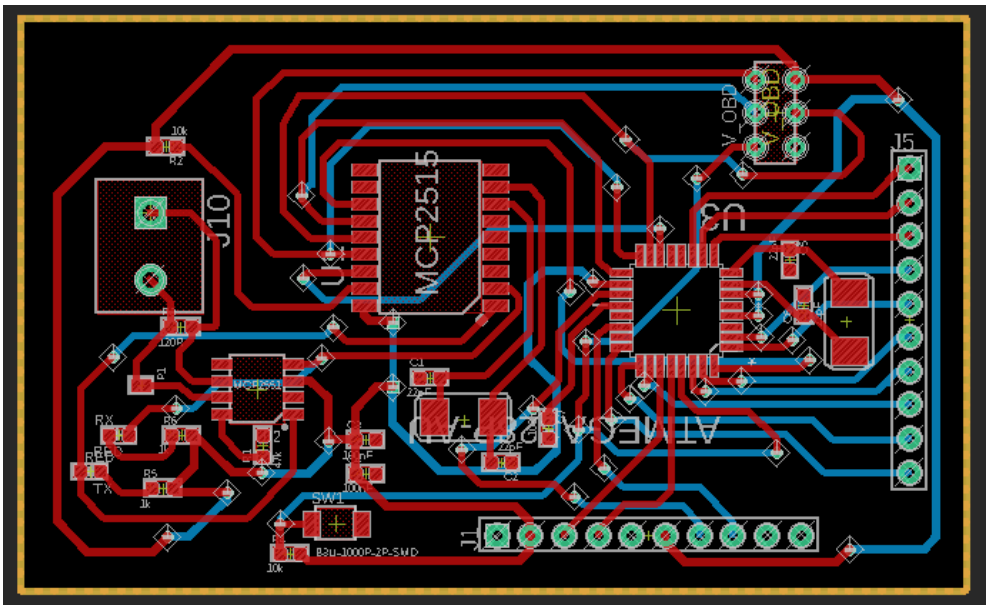


Figura 4. 5 Diseño final del sistema (PCB)

4.4 ADQUISICIÓN DE DATOS

Para poder comenzar a adquirir los datos del sensor IMU (A nexo II) fue necesario utilizar la comunicación I2C en donde nuestro maestro es un Arduino UNO y el esclavo el integrado MPU-6050 debido a que esté integrado sólo transmite información por medio de este protocolo. Se utilizó el IDE de Arduino como prueba inicial por su amplia documentación y fácil manipulación, luego de la adquisición de los datos se observa que los valores no son consecuentes debido a que los datos sufren cambios aleatorios sin que el sensor se vea expuesto a movimientos externos, por ende, el sensor se induce a un preprocesamiento de los datos.

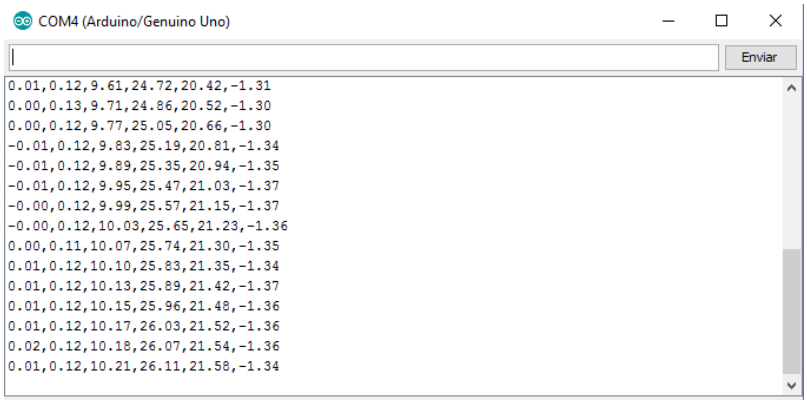


Figura 4. 6 Valores del sensor previos al preprocesamiento

4.5 PREPROCESAMIENTO

El preprocesamiento aplicado a los datos adquiridos consta en una calibración, la cual consiste en dejar el sensor en una posición estática y en una posición en donde el eje z del sensor apunte a la gravedad, con el fin de modificar el valor de referencia que tiene el sensor para la medición en donde leyendo los valores del sensor vamos retroalimentando al sistema y aproximando este valor a 0 o a 9.78 (valor correspondiente a la fuerza de gravedad en la Ciudad de México [6]). Para este punto del proyecto manejábamos un baud rate de 115200, la lectura del puerto serial con los valores obtenidos del sensor antes y después de la calibración los podemos observar en la figura 2.4 y 2.5.

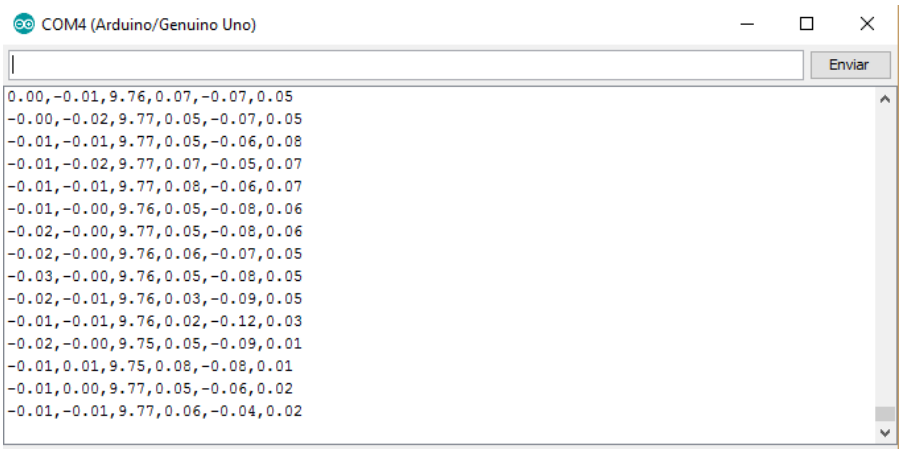


Figura 4. 7 Valores del sensor posteriores al ajuste gravitacional

Posteriormente, se aplicó un filtro bajas realizando un promedio de los datos, esto para eliminar los picos de amplitud que pudieran generar ruido en la señal final. Por último, se realizaron mediciones de lectura con el sensor ya calibrado, se graficaron en Arduino utilizando la herramienta Serial Plotter la cual nos permite ver los valores que son enviados por comunicación serial de manera gráfica y en tiempo real. Podemos observar los gráficos en la figura 4.8, en ésta podemos observar que el valor del acelerómetro en X, acelerómetro en Y, acelerómetro en Z, giroscopio en X, giroscopio en Y y giroscopio en Z.

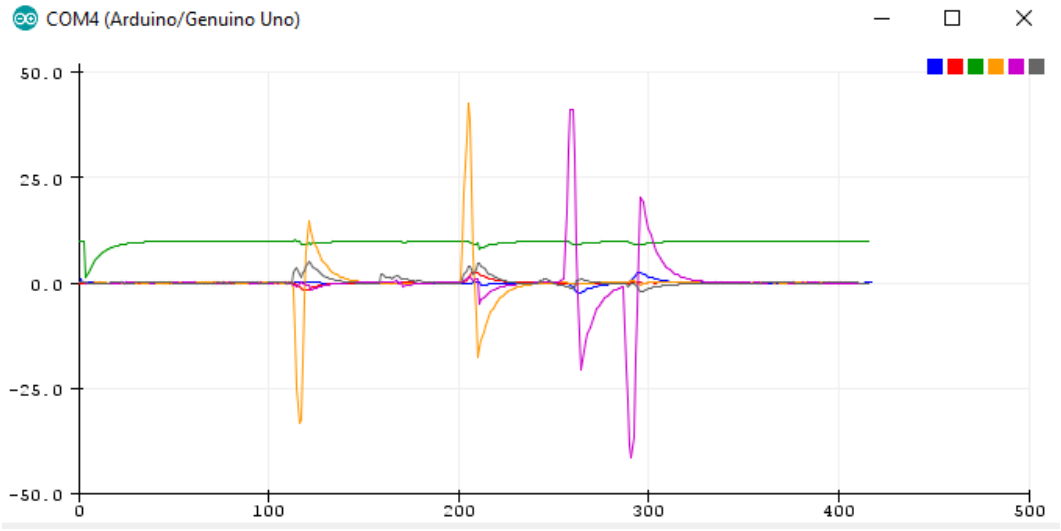


Figura 4. 8 Lectura de los sensores inerciales

Al graficar la señal de salida en tiempo real se observó un pequeño retardo, lo que hacía que fuera posible que se pudieran perder, como solución inicial se propuso manejar una baudrate de 115200 con el fin de mejorar la velocidad de adquisición, aunque esto mejoró el tiempo no tenía cambios muy significativos, debido a esto se decidió hacer una migración a LabVIEW ya que este entorno de programación está diseñado para sistemas de instrumentación en tiempo real, debido a los módulos que LabVIEW utilizaba para la comunicación serial se manejan a 9600 baudios, se decidió cambiar el baudrate a 9600; además de realizar la medición de los sensores y aplicar el filtro pasa bajas, se diseñó una interfaz para poder observar en tiempo real la respuesta del sensor. En la figura 4.11 podemos observar la respuesta del sensor en la interfaz diseñada.

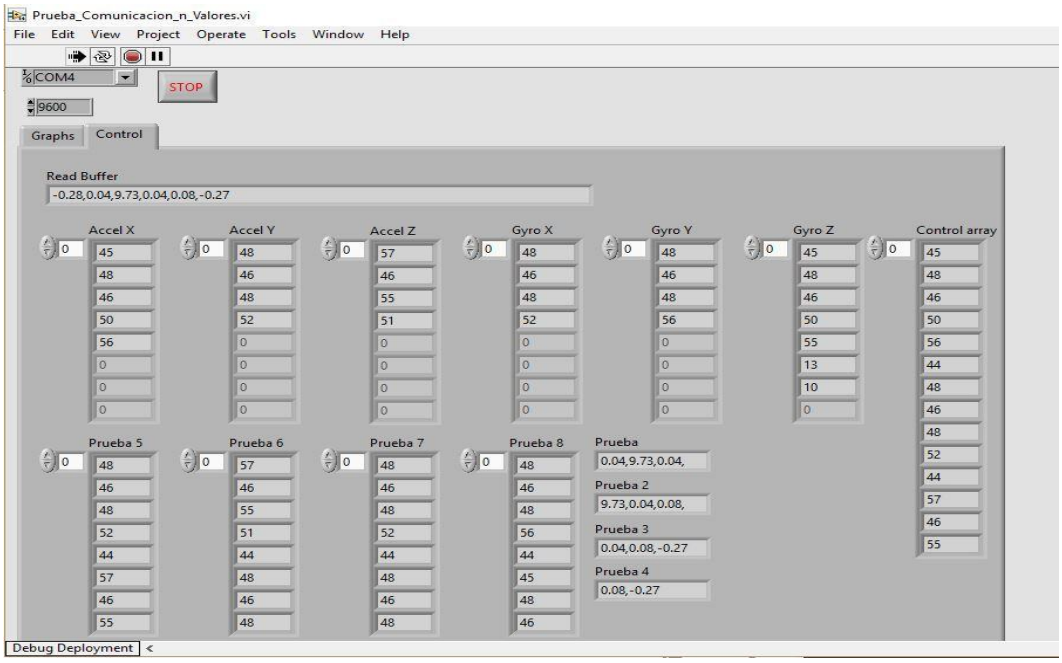


Figura 4. 9 Adquisición de datos en la interfaz desarrollada en LabView

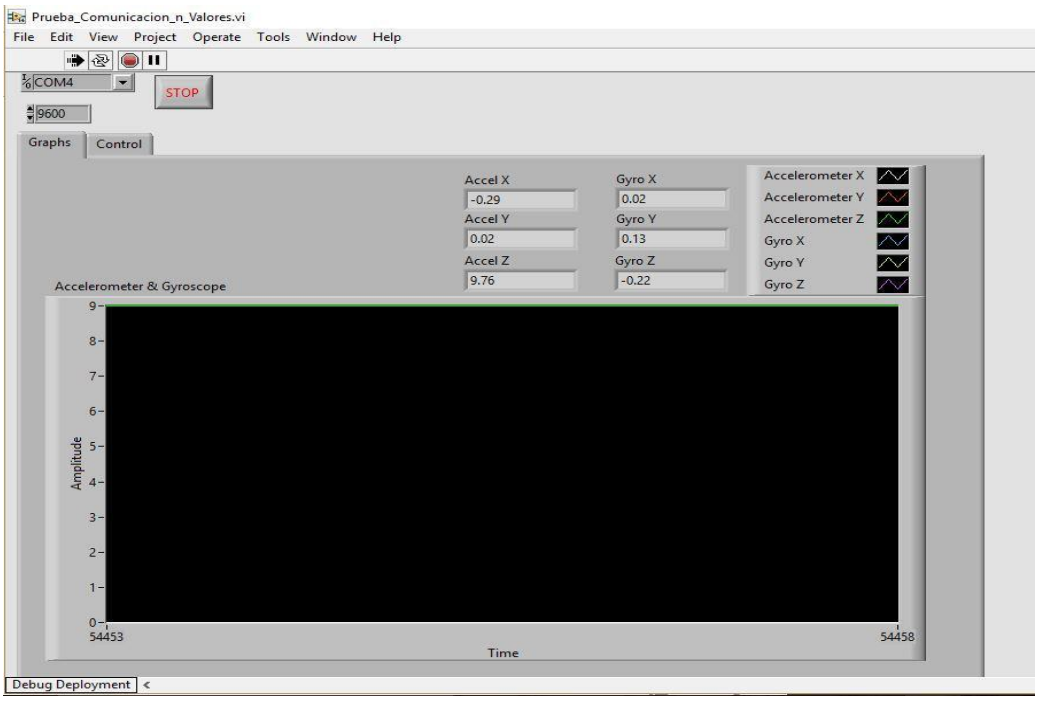


Figura 4. 10 Interfaz de LabView para la observación de la respuesta del sensor

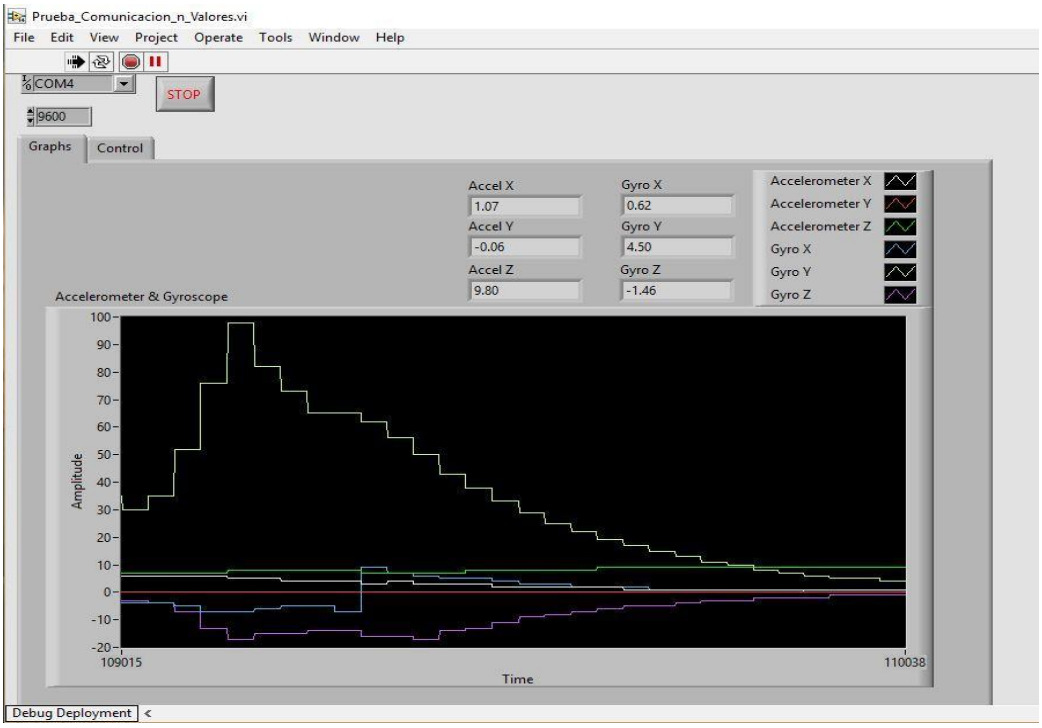


Figura 4. 11 Respuesta del sensor previamente al Filtro de Kalman

Aunque el IDE mejoró la respuesta en tiempo real del sistema, como se busca crear un sistema portátil en el análisis de la marcha LabVIEW no es una solución viable, por esa razón el procesamiento de los datos optó por realizarse en Matlab como previamente se había considerado, con un baudrate de 115200.

4.6 COMUNICACIÓN CAN

Arduino - Esclavo:

Para las primeras pruebas de comunicación CAN se utilizó el CAN-BUS Shield v1.2, éste se conectó a un Arduino UNO por medio de SPI para poder realizar la programación, se conectaron los pines de interrupción entre el Arduino y el MCP 2515 (controlador con interfaz SPI), esto para que la activación se realizará cuando el buffer de CAN tuviera información.

Las pruebas con estos módulos arrojaron buenos resultados, ya que se pudieron conectar los dos CAN-BUS Shield que se tenían. Posteriormente, se decidió comprar más módulos de CAN para simular una red más completa del proyecto. Se compraron

tres módulos de CAN v.2.0b, para poder simular la red completa. Las pruebas de comunicación entre los módulos de CAN v.2.0b fueron exitosas, ya que se logró la transmisión y recepción de datos sin pérdida entre un maestro y dos nodos, pero cuando se trató de realizar una comunicación entre los distintos módulos de CAN, se logró una conexión exitosa, pero no se logró enviar la información de un módulo a otro. En la tabla 4.1 se pueden observar las diferencias de cada módulo.

Tabla 4. 2 Diferencias entre los módulos de CAN

Componente	*CAN-BUS Shield v.1.2*	*CAN v.2.0b*
Transceiver	MCP 2551	TJA 1050
Controlador	MCP 2515	MCP 2515
Oscilador	Cristal 16 MHz	Cristal 8 MHz

Observando la tabla se puede saber que las diferencias más importantes entre cada módulo son el transductor y el oscilador; comparando las hojas de especificación de cada uno de los transductores nos podemos dar cuenta que ambos son transductores de alta velocidad de CAN, y que la diferencia entre estos es el nivel de voltaje que manejan (figura 4.12), aun así como presentan una diferencia mayor o igual a 1.5 V entra dentro del protocolo CAN, por lo que se puede saber que este no es el problema de la conexión, esto lleva a concluir que el problema se debe al oscilador, ya que se manejan a distintas frecuencias y la sincronización no se pueden presentar de manera adecuada, por lo que, aunque la conexión se genera de manera exitosa, el receptor no podrá recibir de manera correcta la trama de datos.

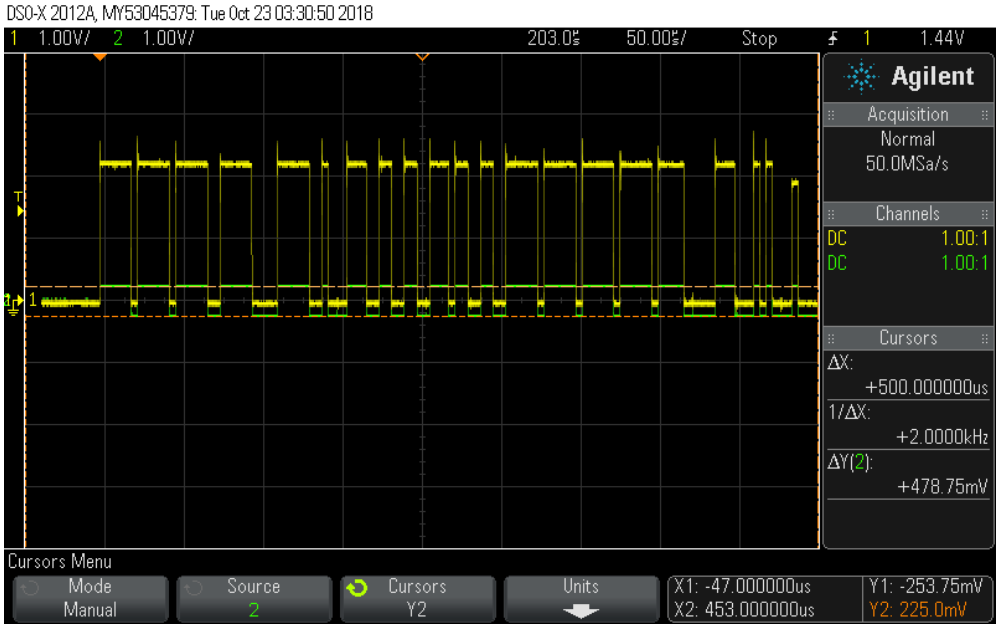


Figura 4. 12 Diferencia de voltaje entre CAN-H y CAN-L, módulo de CAN y shield de CAN

Posterior a la corrección de este error se realizaron pruebas de comunicación, en las que se pudo medir los bits dominantes y recesivos de uno de los nodos hacia el receptor (figura 4.13), así como se pudo observar que el transductor de CAN es un transductor de alta velocidad (Anexo III).

En la figura 4.14 se puede observar los bits enviados y, se puede observar que el mensaje recibido y enviado tienen el mismo diferencial de voltaje (1.825 V) que cumple con el protocolo CAN para ser considerado un bit dominante, el mismo periodo de la trama de datos (500 us) y el mismo número de bits dominantes y recesivos.

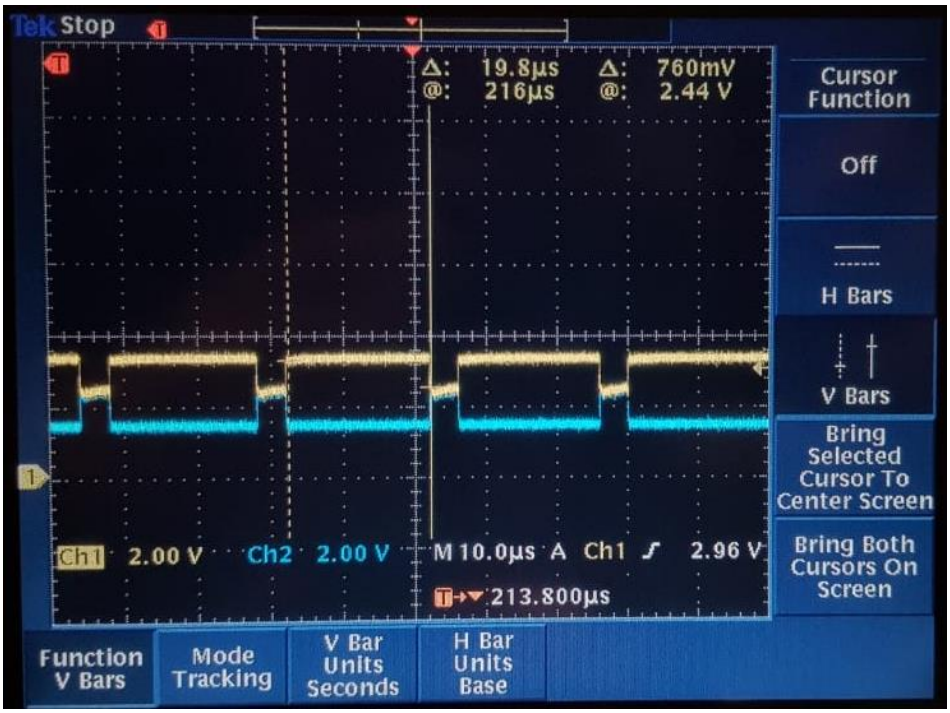


Figura 4. 13 CAN-H y CAN-L de MCP2551 (alta velocidad)

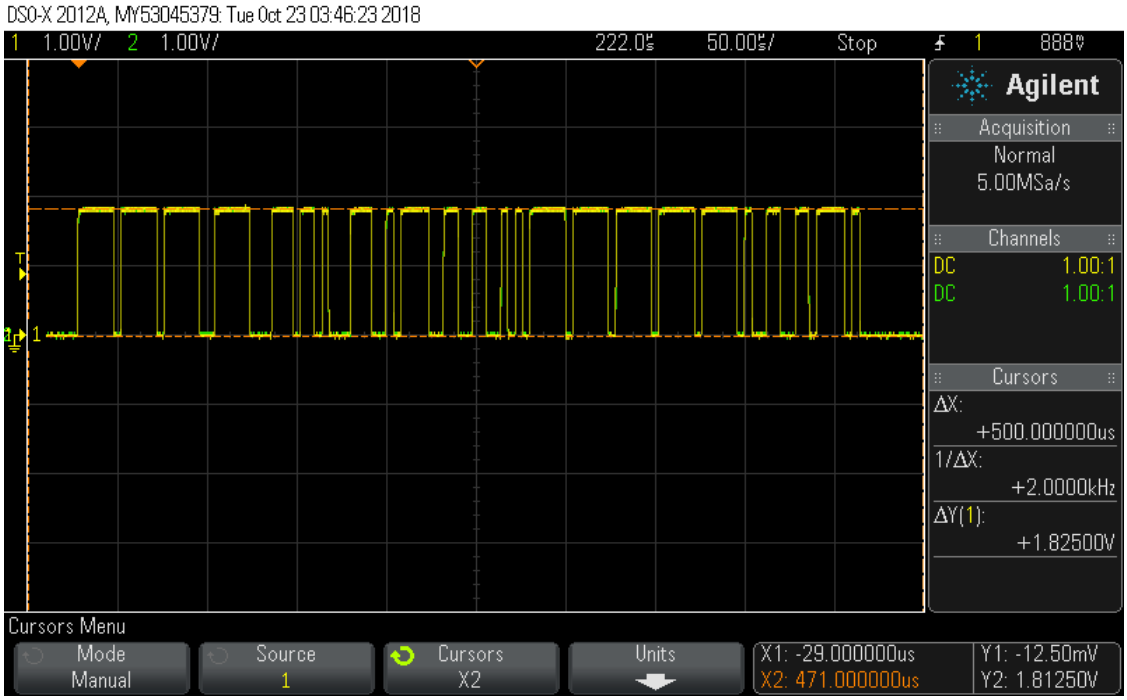


Figura 4. 14 Diferencial de voltaje entre CAN-H y CAN-L

CAPITULO 5. RESULTADOS Y ANALISIS

5.1 ADQUISICIÓN DE DATOS EN CONJUNTO

En las primeras pruebas realizadas con más de un nodo se presentaron errores en el envío de los datos, para corregir este error se priorizaron los diferentes nodos, y se modificó la velocidad de transmisión y recepción de los distintos nodos (250KBPS), esto para que no hubiera pérdida de datos del sensor debido a su velocidad, así como los retardos colocados en el código para la correcta transmisión de los datos, en la figura 5.1 se puede observar que los mensajes enviados simultáneamente se intercalan para ser recibidos por el receptor.

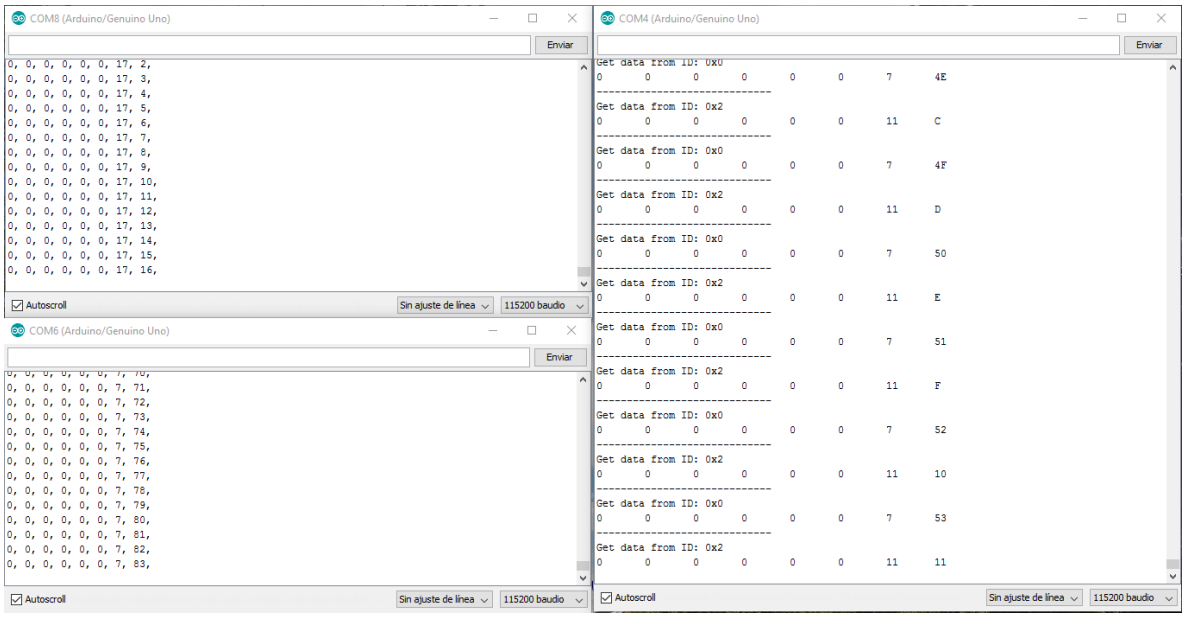


Figura 5. 1 Prueba nodo receptor y dos nodos transmisores

En las primeras pruebas realizadas por CAN en Arduino, la información del sensor era enviada en seis paquetes distintos (los tres valores del acelerómetro y los tres valores del giroscopio), pero esto desaprovechaba el espacio de data que maneja el protocolo CAN para ser transmitida y recibida (8 bytes), por lo que se decidió enviar cada valor en un byte, para que así pudiera enviarse los datos del sensor en un solo paquete.

Micropython - Maestro:

La Pyboard es una tarjeta muy completa ya que consta de un microcontrolador STM32F405RG, el cual consta de un chip ALU y FPU, el cual nos permite hacer operaciones matemáticas complejas, esto para poder calcular el ángulo de la rodilla; además, es una tarjeta que tiene incluida el controlador del protocolo CAN, lo que nos permite que funcione como maestro de nuestro sistema.

La memoria flash de la pyboard es detectada como una memoria externa al ser conectada a una computadora, en su memoria flash están guardados dos archivos .py que corresponden al programa para iniciar (boot.py (declarar librerías)), y el programa que correrá con el código programable por el usuario (main.py). Estos archivos mencionados pueden ser programados por el usuario para que la pyboard cumpla un objetivo específico.

El uso de la pyboard fue un reto para el equipo, debido a que jamás la habían utilizado. Para poder comenzar a programar la pyboard, lo primero que se realizó fue la conexión con la tarjeta y su interfaz en lenguaje Python, para conectarnos a su interfaz se utilizó una terminal serial que pudiera conectarse a su puerto COM, en este caso Putty fue utilizado a 115200 velocidad de la tarjeta (figura 5.2).

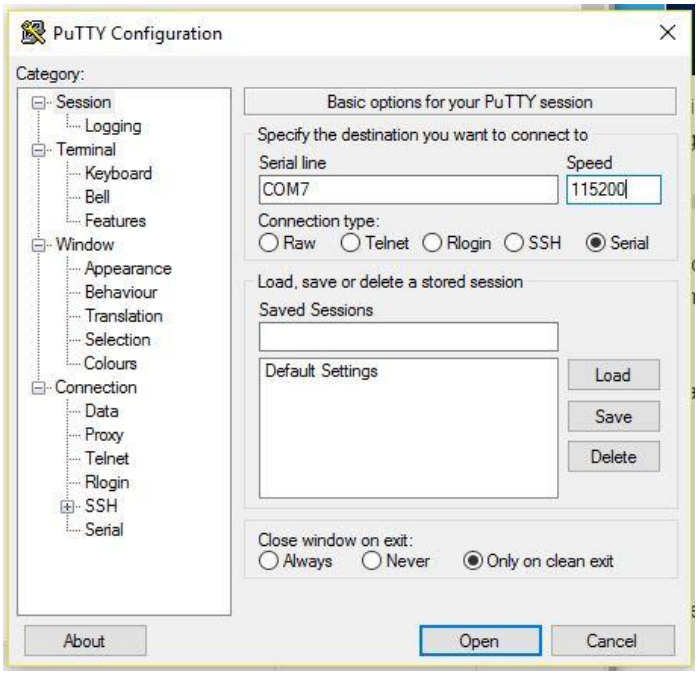


Figura 5. 2 Putty terminal serial

Al ingresar a la terminal se pudieron observar algunos comandos de ayuda para conocer y familiarizarse con la interfaz.

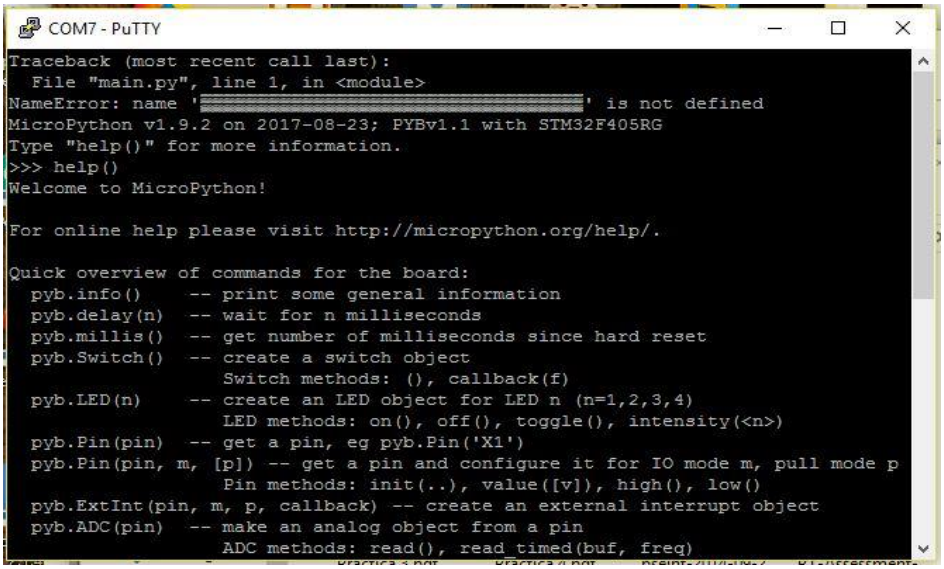


Figura 5. 3 Interfaz pyboard comandos de ayuda

Para la comunicación por CAN en la pyboard se presentaron muchos problemas de comunicación, se comenzó probando un arduino con la pyboard ya que el código de CAN estaba funcionando correctamente en las tarjetas de arduino. Para que está

conexión fuera exitosa se inicializó el CAN 1 de la pyboard con el siguiente fragmento de código:

```
can = CAN(1)
can.init(CAN.NORMAL, extframe=True, prescaler=21, sjw=1, bs1=5, bs2=2)
```

En este se especifica que se utiliza el CAN 1, debido a los puertos físicos en la pyboard, se coloca modo normal de operación, posibilidad de leer identificadores extendidos (29 bits), un prescaler de 21, y un bs1 de 5 y un bs2 de 2, estos últimos obtenidos de los siguientes cálculos:

$$Baudrate = \frac{1}{bit\ time} = 250KBPS \rightarrow bit\ time = \frac{1}{250KBPS} = 4us$$

$$TQ = Time\ sync = 500ns$$

$$TQ = \frac{prescaler}{PCLK1} = \frac{prescaler}{42MHz} = 500ns \rightarrow prescaler = 21$$

$$bit\ time = TQ * (1 + BS1 + BS2) = 4us = 500ns * (1 + BS1 + BS2) \rightarrow BS1 = 5, BS2 = 2$$

Al enviarse una serie de tres datos por el CAN 1 de la pyboard hacía un transductor y posteriormente al CANH y CANL del Arduino, se observó que los datos no se enviaban correctamente y aparecía un error en la interfaz (figura 5.4), por lo que se decidió analizar el estado del nodo CAN (figura 5.5).

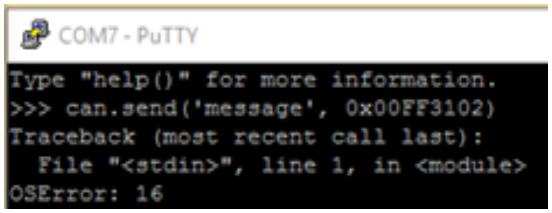


Figura 5. 4 Error de transmisión micropython

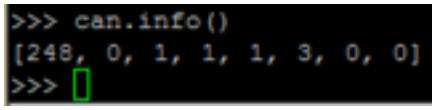


Figura 5. 5 Información del estado del nodo CAN 1

Se observó (según la documentación del micropython), que los datos que nos interesan de la trama son el primer valor que equivale al valor del TEC (Anexo III), y el sexto valor que equivale al número de mensajes esperando ser transmitidos. Con esto se puede observar que de los 3 datos enviados ninguno fue transmitido exitosamente, además, por el TEC de nuestro nodo se puede decir que está en Estado de Error Pasivo, lo que no saturará la red, pero no nos permitirá tener una comunicación totalmente exitosa dentro de ella.

El equipo se dio cuenta que el error se encontraba en el PCLK que estaba establecido para el CAN 1, ya que en lugar de que el valor fuera de 42MHz era de 105MHz, por lo que no se podía establecer un baudrate de 250KBPS como se requería; al cambiar esto se logró tener una transmisión exitosa con el Arduino desde la pyboard (figura 5.6).

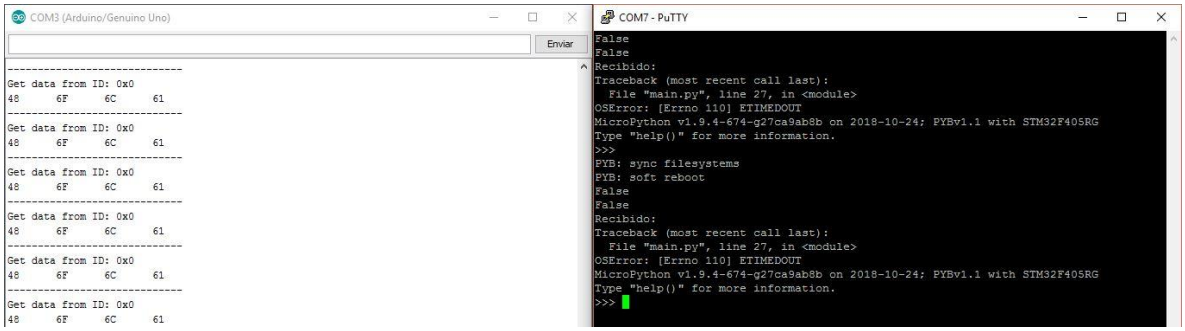


Figura 5. 6 Transmisión mycropython a arduino

Posteriormente se hizo un código para poder enviar y recibir datos simultáneamente, este código se probó con el código de transmisión y recepción del Arduino, pero la recepción no se realizó con éxito. En la figura 5.7 se puede observar que la transmisión la realiza exitosamente, pero en la figura 5.8 se observa que la recepción nos regresa un error en la interfaz.

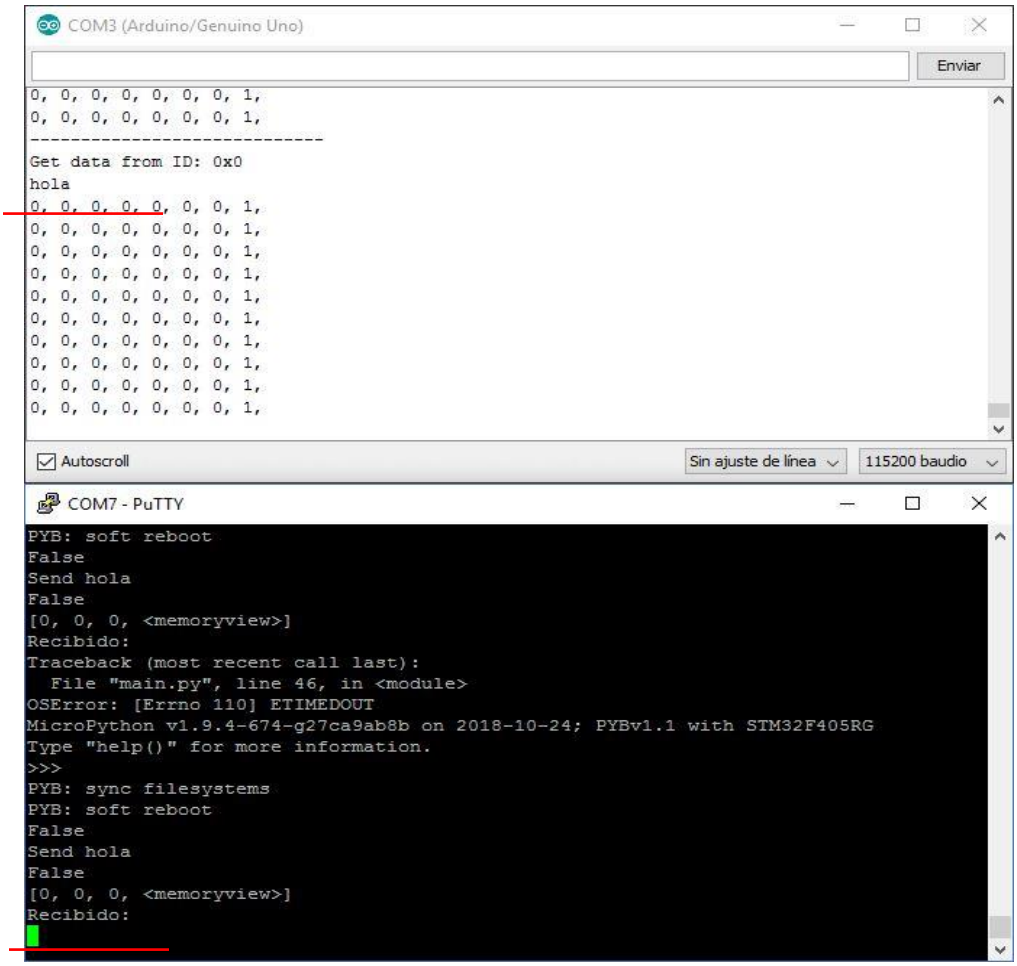


Figura 5. 7 Transmisión y recepción de datos, micropython y Arduino

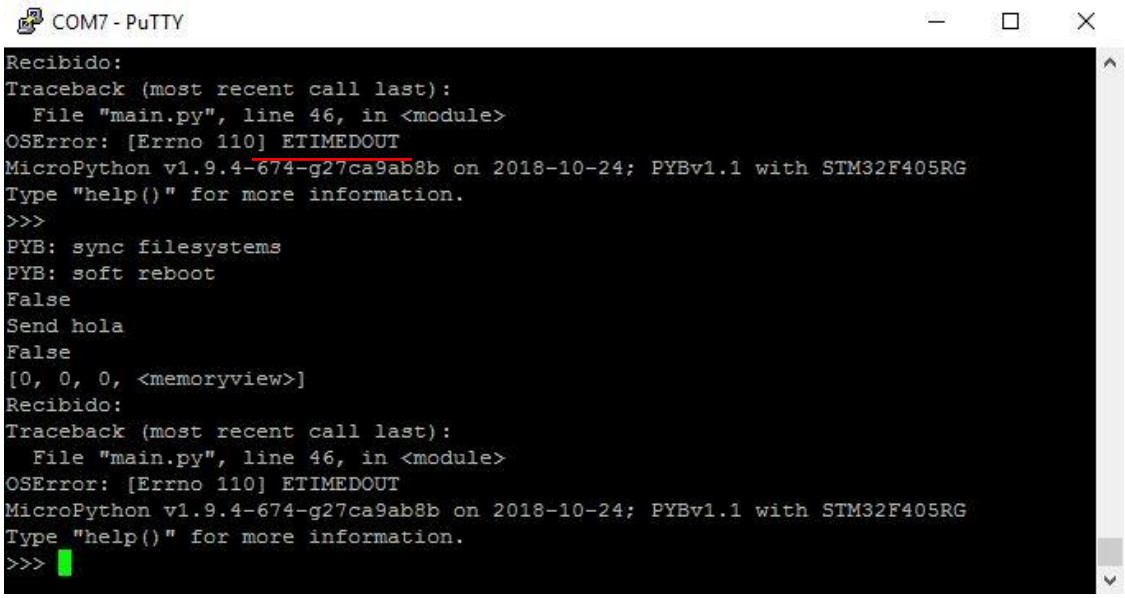


Figura 5. 8 Recepción fallida micropython

Aunque la conexión se había establecido exitosamente, se decidió verificar que la inicialización del CAN se hiciera correctamente, por esa razón el equipo decidió medir el tiempo de bit, para confirmar que se tuvieran los mismos valores prácticos y teóricos. En la figura 5.9 se puede observar que se tienen los mismos valores prácticos y teóricos.

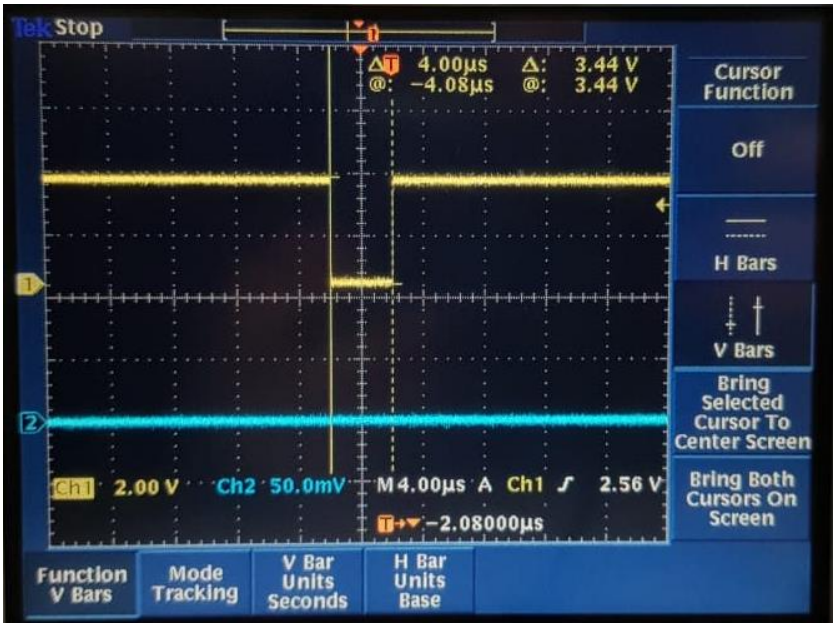


Figura 5. 9 Bit time de pyboard CAN 1

Se cree que la falta de recepción se puede deber a la configuración de algún filtro erróneo, que puede provocar que el nodo de Arduino no se pueda comunicar de manera correcta con el nodo de la pyboard, impidiendo que el mensaje llegué al buffer de recepción. La falta de documentación no ha permitido resolver este problema de conexión tan rápidamente, sin embargo, este se puede resolver realizando diferentes pruebas de transmisión y recepción.

5.2 APLICACIÓN DEL FILTRO DE KALMAN

Para la última etapa del proyecto, se realizó la etapa de procesamiento de datos en donde se utilizó el Filtro de Kalman, una potente herramienta matemática conformándose básicamente por ecuaciones matemáticas mayormente conformadas por vectores y matrices, al momento de trabajar con sensores inerciales nos

encontramos con valores de aceleración y rotación, ocasionalmente las personas tienden a semejar la orientación con la rotación, diferenciándose entre sí en que la rotación es el valor que adquirimos con el giroscopio, sin embargo la orientación que es el objetivo al que queremos llegar, que es el desplazamiento de un punto en el espacio con respecto a un sistema de coordenadas global, el FK necesita previamente el ingreso de los datos adquiridos y preprocesados del acelerómetro y giroscopio, para posteriormente ser fusionados y de esta manera encontrar los ángulos que permiten describir la orientación de un objeto en tres dimensiones, los cuales conocidos universalmente como Roll, Pitch y Yaw, permitiéndonos de esta manera conocer el desplazamiento del sensor sobre su eje, aunque actualmente existen diversos formas de fusionar sensores, los actualmente existentes tienen como finalidad combinar valores de una o varias fuentes de información por lo que en nuestra aplicación se busca combinar sensores independientes para derivar información que no está disponible en un solo sensor esto es posible con el uso de un Filtro Kalman, una potente herramienta que se usa con la finalidad de filtrar y predecir sistemas lineales funcionando como un de tipo predictor-corrector.

Respecto a lo anteriormente mencionado se optó por trabajar sobre dos plataformas que trabajan con softwares diferentes, dichas plataformas lo fueron Matlab y Micropython.

Inicialmente se utilizó Matlab, como se nombró anteriormente este software nos permite ejecutar diversas operaciones matemáticas de alto nivel de manera sencilla, eficiente y altamente confiable, además de que el FK maneja operaciones directas de matrices y vectores, además de incluir funciones que facilitan el procesamiento de datos es por este motivo que se trabajó con esta herramienta matemática, los datos que se adquirieron y preprocesaron en Arduino fueron enviados por comunicación serial a Matlab, para posteriormente procesar y graficar los datos en tiempo real, otra ventaja que cabe mencionar es que el filtro de Kalman al trabajar con el valor anterior para la predicción de un valor futuro, genera una amplia ventaja al momento de trabajar con los datos en tiempo real, en la figura 5.10 se puede evidenciar el resultado

final luego de aplicar el FK a los valores de los sensores inerciales, en donde se ven representados los ángulos de orientación en radianes esto debido a que todas las operaciones matemáticas se realizaron en radianes para un mejor rendimiento matemático.

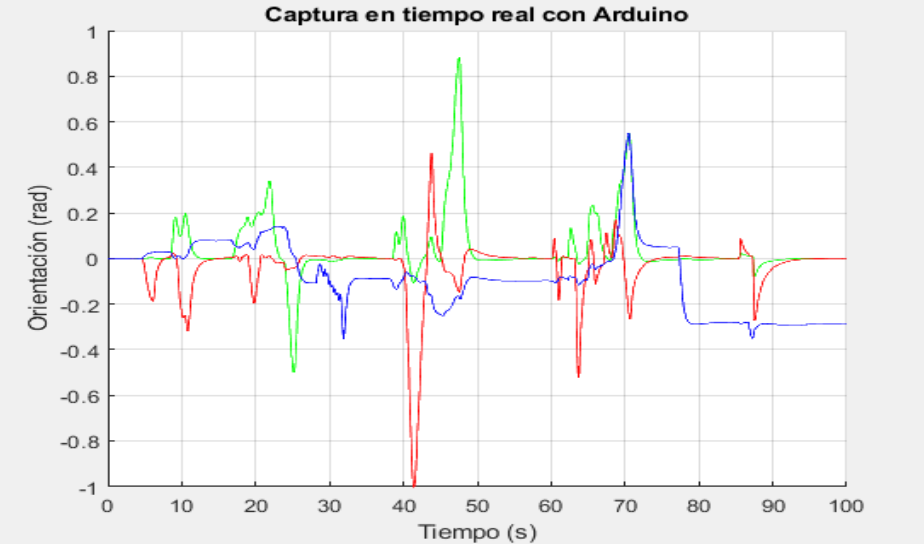


Figura 5. 10 Respuesta del sensor con el filtro de Kalman aplicado, MATLAB

Aunque se realizó el procesamiento de los datos en Matlab no se optó por finalizar el trabajo en esta etapa debido a que no se cumplía con un sistema totalmente portátil, debido a la necesidad de que hubiera una conexión externa y directa a la computadora, es por este motivo que posteriormente a esta etapa se dio el paso del uso de Matlab a Micropython para la etapa de procesamiento.

Micropython consiste en un software basado en la implementación ágil y eficiente del lenguaje de programación Python 3, el cual incluye un subconjunto de la biblioteca estándar de Python y está optimizado para ejecutarse en microcontroladores y en entornos restringidos, la tarjeta que se utilizó para la adquisición y procesamiento de datos en donde se utiliza Micropython es la Pyobard 1.1 siendo esta la placa oficial de micropython que cuenta con un soporte completo de funciones totalmente compatibles con este software, los datos fueron adquiridos por la Pyobard utilizando

el protocolo de comunicación CAN, el siguiente paso luego de poder establecer dicha comunicación entre los sensores y la pyobard se procedió a la replicación del código realizado en Matlab del Filtro Kalman , debido a que ambas plataformas manejan lenguajes de programación diferentes, como ya se nombró esta tarjeta cuenta con una gran variedad de funciones las cuales nos permitieron realizar operaciones matemáticas necesarias para el desarrollo del algoritmo característico del Filtro de Kalman, sin embargo no cuenta con funciones para el manejo de matrices y vectores, es por este motivo que se utilizaron módulos de representación y manipulación matricial, estos módulos son externos a los que maneja predeterminadamente la pyobard.

En la figura 5.11 se observa los resultados obtenidos en la primera prueba que se realizó para corroborar que el procesamiento estaba resultado similar al ejecutado en Matlab en donde leyendo los valores del acelerómetro que internamente tiene la Pyobard para posteriormente procesarlos utilizando un valor base en el giroscopio se calculó el FK, obteniendo como resultado la lectura, los parámetros característicos del FK y el diferencial de tiempo que tarda en realizar la ejecución del FK.

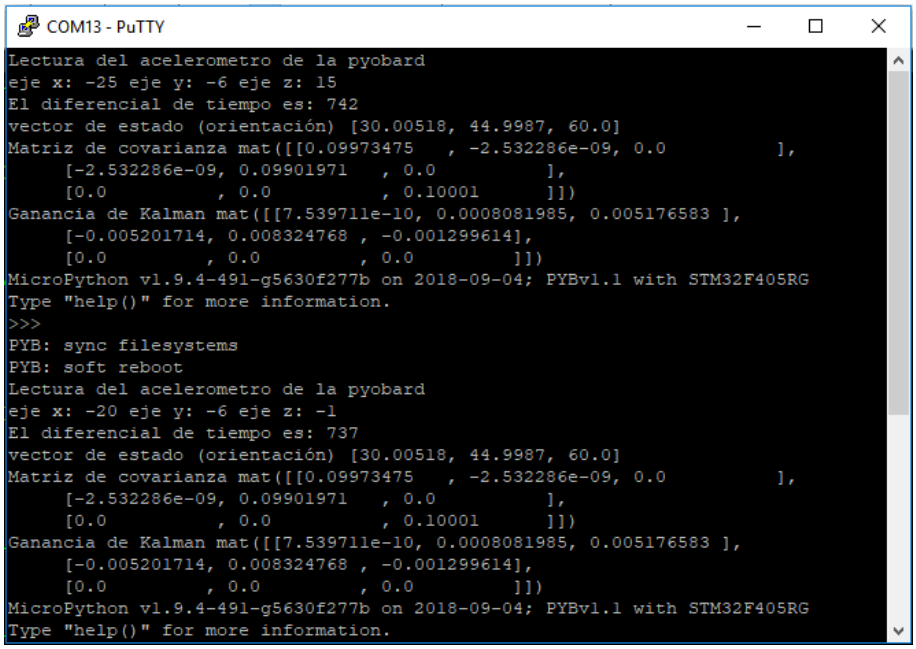


Figura 5. 11 Parámetros características del Filtro de Kalman

Finalizando la elaboración de este algoritmo utilizando Micropython y realizando pruebas se procedió a validar los resultados obtenidos con Matlab, en donde se realizaron las mismas pruebas que se hicieron en la Pyobard, para esto se transmitieron los datos del acelerómetro de la pyobard a Matlab y dejando el mismo valor base del giroscopio los resultados obtenidos se muestran en la figura 5.12, en donde se puede observar que los valores de los parámetros característicos del FK coinciden.

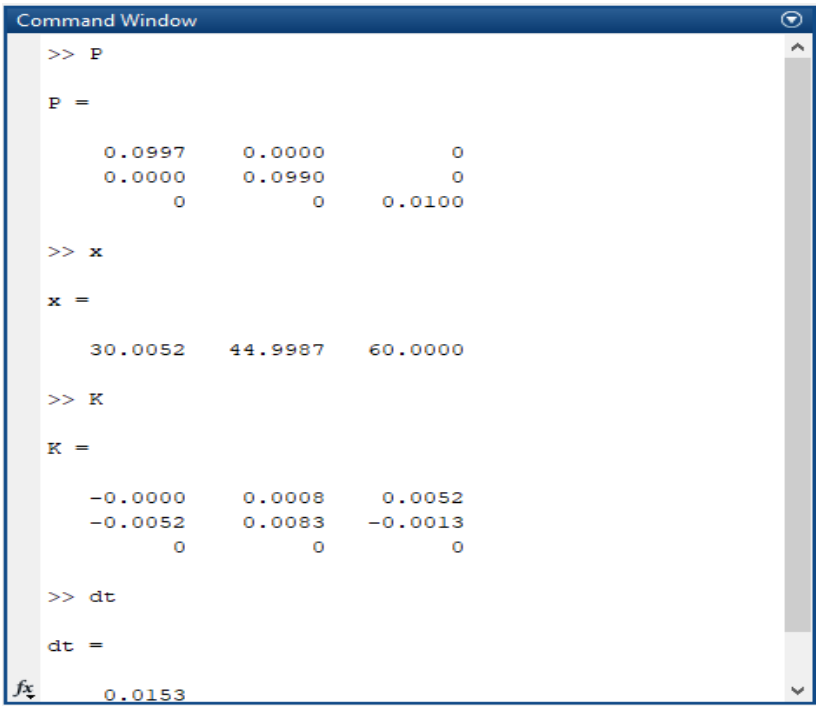


Figura 5. 12 Validación de los parámetros característicos del FK en Matlab

Sin embargo, la clara diferencia se encuentra en el diferencial de tiempo debido a que el tiempo que tarda Matlab en ejecutar el FK es de 15.3 ms y el tiempo en la Pyobard es de 737 ms generando un diferencial de 721.7 ms.

CAPITULO 6. CONCLUSIONES

6.1 CONCLUSIONES

Se logra diseñar, programar e implementar una tarjeta para la lectura de diversos sensores. Para fin de este proyecto fue utilizado un sensor inercial, cuyo tiempo de respuesta total (transmisión, recepción, procesamiento) fue de 738 ms, y cuyo ángulo resultante se puede observar mediante comunicación serial del nodo maestro hacía la computadora.

El algoritmo implementado para el cálculo del ángulo respecto a los sensores inerciales en el micropython logró un tiempo de proceso de 737 ms, mayor al tiempo conseguido por el procesamiento en Matlab, cuyo tiempo fue de 15.3 ms, este tiempo puede afectar al desempeño del sistema, ya que la velocidad del sensor y la velocidad de transmisión de datos es mayor a la de procesamiento, por lo que el sistema se vuelve propenso a fallar por pérdida de datos. Este es ya un problema mencionado anteriormente, y se busca solucionar con la optimización del código en micropython.

Al finalizar las etapas del proyecto, se procedió a estimar la precisión del algoritmo para obtener el rango articular de la rodilla, en la primera prueba se colocó cada sensor inercial sobre una base plana y sólida, de manera que generaban un ángulo de 180° entre ellos, este ángulo se midió con un transportador, los resultados de esta prueba variando el ángulo se pueden apreciar en la figura 6.1, permitiendo corroborar el algoritmo de estimación. En la segunda prueba se ubicaron los sensores inerciales sobre el brazo y el antebrazo como se observa en la figura 6.2, midiendo así el ángulo entre estos dos, el cual corresponde al rango articular del codo. Se utilizó Matlab para el procesamiento de los datos y la medición en tiempo real, en donde se obtuvieron buenos resultados ya que el sistema presenta una estimación del rango articular de la rodilla con un error no mayor a 3°; de igual manera se utilizó micropython, se compararon las estimaciones obtenidas por Matlab y este, en las cuales se pudieron observar resultados similares, sin embargo se generó un código que permitiera contar los valores similares dentro de un rango prestablecido generando coincidencias bastantes significativas, aunque con una pequeña pérdida de datos.

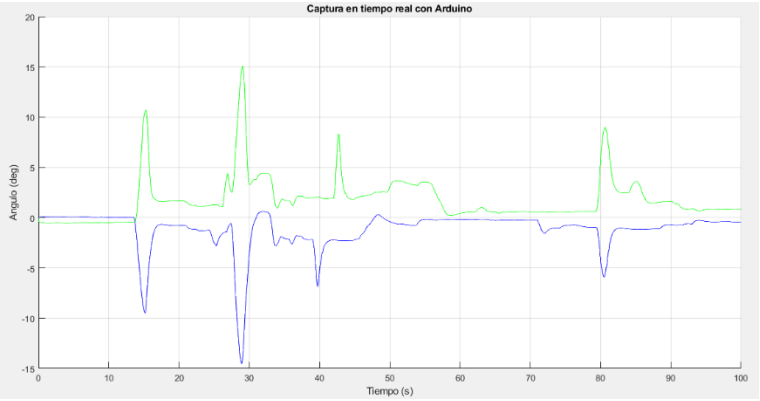


Figura 6. 1 Comprobación de sensores inerciales

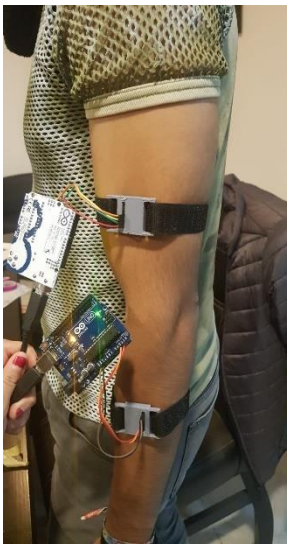


Figura 6. 2 Prueba de sensores inerciales

6.2 DILEMAS ÉTICOS

El exo-esqueleto en el cual se empleará el sistema, tiene como objetivo mejorar el movimiento en la persona que lo utilice, pero este uso de “mejora” podría llegar a causar anormalidades en su forma de caminar. El análisis del ángulo de la rodilla ayudaría a detectar esas anormalidades, pero el mismo exo-esqueleto podría ser el que las causara.

Además, si el procesamiento de los datos para encontrar el ángulo, demorara mucho tiempo, no sólo podrían perderse datos importantes que provocara la mala interpretación de los datos, si no que aumentaría el tiempo aplicado al filtro de kalman, por lo que podría tener aproximaciones erróneas. Para evitar que el tiempo de procesamiento aumente, se agregaran funciones al código, para acortar el tiempo de procesamiento del filtro de kalman.

6.3 TRABAJO FUTURO

Como se mencionó antes, los filtros de CAN de la pyboard se deben configurar de manera correcta para generar una comunicación exitosa, esto debe realizarse con pruebas de transmisión y recepción para poder implementar el sistema de manera completa con el micropython como maestro. Junto a esto se busca corroborar los resultados obtenidos con un sistema de visualización de cámaras, como lo es el MoCap.

Además, se sabe que la comunicación serial a la computadora para la observación de la estimación del ángulo, es muy poco eficiente para un sistema que se desea completamente portátil, por lo que se decidió agregar una comunicación inalámbrica con un web server para que los datos puedan permanecer en la nube, esto para su fácil accesibilidad y liberación de memoria.

BIBLIOGRAFÍA

[1]Jakob Carolin, Kugler Patrick. (). Estimation of the Knee Flexion-Extension Angle During Dynamic Sports Motions Using Body-worn Inertial Sensors. Institute of Sport Science and Sport. [Revisado el: 17 de agosto del 2018].

[2]E.S. Grood, W.J. Suntay. (1983). A joint Coordinate System for the Clinical Description of Three-Dimensional Motions: Application to the knee. University of Cincinnati Medical Center. [Revisado el: 10 de septiembre de 2018]. Recuperado de: http://biomechanical.asmedigitalcollection.asme.org/article.aspx?articleid=1396188

[3]Martínez R. Adrián, García M. Javier. (). Introducción a CAN bus: Descripción, ejemplos y aplicaciones de tiempo real. Universidad Politécnica de Madrid. [Revisado el: 17 de agosto de 2018].

[4]Vilela José M, Parra G. Francisco. (2011). Kinescan/IBV v11: Valoración biomecánica en tiempo real. Instituto de biomecánica de Valencia. [Revisado el: 10 de septiembre de 2018]. Recuperado de: https://www.ibv.org/productos-y-servicios/productos/aplicaciones-biomecanicas/kinescanibv-v2011-sistema-de-analisis-de-movimientos-3d-en-tiempo-real

[5]Licitaciones. (20). Mantenimiento y actualización del sistema de análisis de movimientos Kinescan IBV 6c. [Revisado el: 11 de septiembre de 2018]. Recuperado de: https://www.licitaciones.es/concursos-publicos/mantenimiento-y-actualizaci%C3%B3n-del-sistema-de-an%C3%A1lisis-de-movimientos-kinescan-ibv-6c?uuid=1a5778de-c623-11e7-860f-002655ffd6c8

[6]Dr. Hernández I, Luis Ignacio. (2012). Aceleración de la gravedad. [Revisado el: 13 de agosto de 2018]

[7]Pampliega Ruiz, David. (2010). El Filtro Kalman. [Revisado el: 13 de agosto de 2018]. Recuperado de: http://bibing.us.es/proyectos/abreproy/11611/fichero/Memoria%252FAp%C3%A9ndice+B+-+El+Filtro+de+Kalman.pdf+

[8]Damien P, George. (2018). MicroPython Documentation. [Revisado el: 12 de noviembre de 2018]. Recuperado de: https://media.readthedocs.org/pdf/micropython-pfalcon/latest/micropython-pfalcon.pdf

[9]Gaujal, Bruno. Navet, Nicolas. (2005). Fault Confinement Mechanisms on CAN: Analysis and Improvements. [Revisado el: 12 de noviembre de 2018]. Recuperado de: http://nicolas.navet.eu/publi/ieee_tvt_05.pdf

ANEXOS

AI COMUNICACIÓN CAN

El protocolo CAN es un protocolo de comunicación usado en sistemas que requieren de una transmisión en tiempo real. Es un sistema confiable, que puede detectar errores en la trama, así como brindarte fiabilidad de transmisión de datos.

Este protocolo presenta una alta integridad en la transmisión de datos, algunas de sus ventajas sobre otros BUS de comunicación son:

- Detección de errores en la transmisión, así como la capacidad de retransmisión de las tramas erróneas.
- Capacidad de detectar si el origen del error se debe a un problema en la transmisión o se debe a un nodo de comunicación, en caso de que se deba a un nodo de comunicación, este protocolo tiene la capacidad de desconectar este nodo para evitar la saturación en la red.
- Buena latencia en la transmisión de datos y priorización en los mensajes.
- Escalabilidad en una red CAN, ya que te permite conectar y desconectar nodos a la red sin modificar los nodos existentes. Tiene una capacidad de contener hasta 110 nodos.

Este protocolo está especificado en el estándar ISO 11898[3]; la norma ISO 11898-2 especifica que el protocolo CAN de alta velocidad puede alcanzar una velocidad de hasta 1MB/s, la norma ISO 11898-3 estandariza que el protocolo CAN de baja velocidad es tolerante a fallos. En la tabla I.1 podemos observar las velocidades respecto a diferentes distancias.

Tabla I. 1 Velocidades de transmisión de CAN respecto a diferentes distancias

Longitud del bus [m]	Velocidad [bit/s]	Tiempo de bit [us]
30	1 M	1
50	800 K	1.25
100	500 K	2

250	250 K	4
500	125 K	8
1000	50 K	20
2500	20 K	50
5000	10 K	100

Los módulos de comunicación CAN se componen de un controlador y un Transmisor/Receptor (Tx/Rx, transceptor). El controlador se encarga de gestionar las tramas de CAN, así como de detectar si se presentan errores en la transmisión o en otros nodos, además de poder detectar colisiones. El transceptor está encargado de codificar y decodificar los mensajes en el bus, de la sincronización en la transmisión, además de controlar los niveles de la señal y el acceso al medio.

CAPA FÍSICA

Para la topología son necesarios dos cables trenzados con una impedancia de 120 ohms. Las señales que se transmiten se denominan como CAN_H y CAN_L dependiendo del voltaje entre las mismas, si ambas tienen el mismo nivel de tensión se dirá que el bus se encuentra en modo recesivo, de manera contraria, si tienen entre ambos una diferencia de voltaje de al menos 1.5V se dirá que el bus se encuentra en modo dominante (figura I.1 y figura I.2); estos modos de comunicación proporcionan una mayor protección frente interferencias electromagnéticas, ya que si estas señales son sometidas a una influencia electromagnética, aunque se verá afecta la variación de la señal en los cables, la diferencia de voltaje se mantendrá constante.

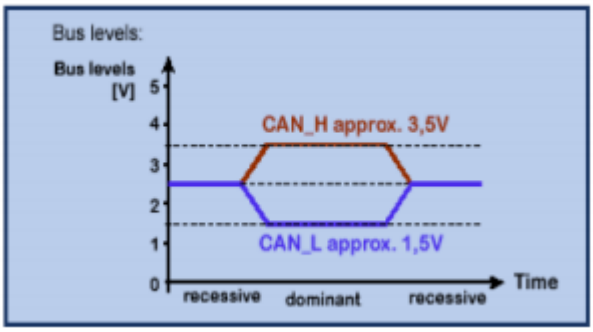


Figura I. 1 Modo dominante y recesivo, bus CAN, alta velocidad

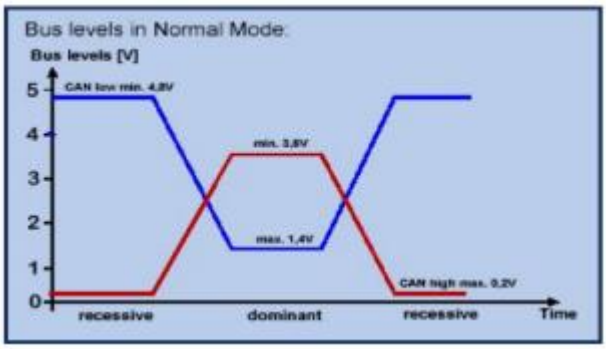


Figura I. 2 Modo dominante y recesivo, bus CAN, baja velocidad

CONTROL DE ACCESO

Para evitar posibles colisiones en la comunicación, se tiene un control en el acceso, este acceso está dado por CSMA/CD+CR (Carrier Sense, Multiple Access/Collision Detection + Collision Resolution) (Acceso Múltiple con detección de portadora, detección de colisión más resolución de colisión).

En la transmisión, en el modo dominante los valores de los bits se representan como "0", en el modo recesivo los valores de los bits se representan como "1", cuando el nodo está en reposo este se mantiene en modo recesivo. Al inicio de la trama se envía el identificador del nodo con bits recesivos y dominantes, debido a que los dominantes prevalecen sobre los recesivos, serán estos quienes puedan detectar las colisiones. Si dos nodos tratan de transmitir al mismo tiempo, el nodo que tenga una trama de bits

dominantes detectar la posible colisión, detendrá su transmisión e intentará la transmisión nuevamente cuando el otro nodo haya terminado de transmitir.

FORMATO DE LAS TRAMAS CAN

El protocolo CAN maneja distintos formatos de trama, cada una con una función específica dentro de la comunicación:

- *Tramas de datos:* Tramas utilizadas para el paso de información entre nodos, puede ser enviada o recibida por uno o más nodos.
- *Tramas de error:* Tramas utilizadas cuando algún nodo de la red detecta un error en los mensajes recibidos de otro nodo.
- *Tramas de petición remota:* Tramas utilizadas para solicitar enviar la información a otro nodo. En esta trama se envía el identificador del nodo que requiere el envío de información, cuando este nodo la recibe envía una trama de datos.
- *Tramas de sobrecarga:* Tramas utilizadas para provocar que el bus agregue un retardo entre las tramas. Este tipo de tramas se utiliza cuando un nodo está sobrecargado.

TRAMA DE DATOS

Esta trama tiene la capacidad de mandar hasta 8 bytes de información. La trama tiene el siguiente formato.

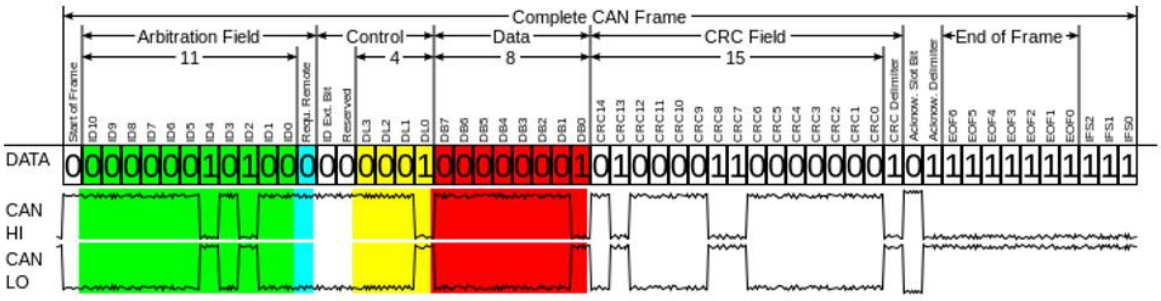


Figura I. 3 Formato de trama de datos

Las tramas de datos comienzan siempre con un bit dominante, para indicar el inicio de una trama, esto con la finalidad de sincronizar los nodos.

- Arbitration Field: Contiene el identificador, puede ser en un formato de 11 bits o el formato extendido de 29 bits. También incluye la prioridad de la trama de red, mientras el nodo central sepa la prioridad de todos los nodos puede determinar el orden en que estas tramas serán transmitidas y recibidas; una trama CAN con identificador más bajo (mayor número de bit dominantes en las primeras posiciones) es más prioritario que un identificar más alto.
- Request Remote: Al terminar el campo del identificador continua un bit que diferenciara si termino trama de datos (bit dominante) o si se refiere a una trama de petición remota (bit recesivo).
- Control: Los dos primeros bits se reservan para un uso futuro, posteriormente hay 4 bits que definen el tamaño del campo de datos.
- Data: El campo de control puede abarcar hasta 8 bytes de información.
- CRC Field: El campo de Código de Redundancia Cíclica consta de 16 bits, los primeros 15 bits se encargan de la integridad de los datos, donde comprueba que la transmisión no haya presentado errores, mientras que el último bit establece el fin del campo CRC, con un bit recesivo.
- Acknowledge: El campo de reconocimiento consta de 2 bits. El primer bit se utiliza para verificar que el nodo haya recibido el mensaje correctamente, el nodo transmisor coloca un bit recesivo, cuando el otro nodo lo recibe coloca un bit dominante para indicar que se ha recibido correctamente el mensaje. El segundo bit se utiliza para indicar cuando el ACK termina, con un bit recesivo.
- End of Frame (EOF): Por último, se encuentra el fin de trama que consiste en 7 bit recesivos, el cual indica el fin de la trama.
- Después del EOF se colocan 3 bits recesivos reglamentarios, lo que separa las tramas enviadas.

TRAMA DE ERROR

El protocolo CAN maneja una norma que obliga a la trama a tener un relleno de bits obligatorio, en cada secuencia de cinco bits con el mismo valor se debe introducir un bit de valor inverso (figura I.5). La trama de error rompe con esta norma, lo que permite que se diferencie de una trama de datos y una trama de error. Después de romper esta norma dentro una trama se presentan los siguientes campos (figura I.4).

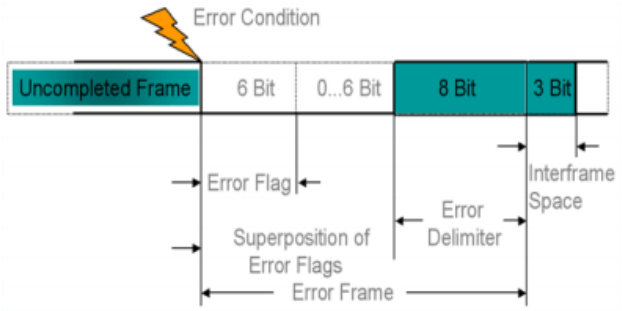


Figura I. 4 Formato trama de error

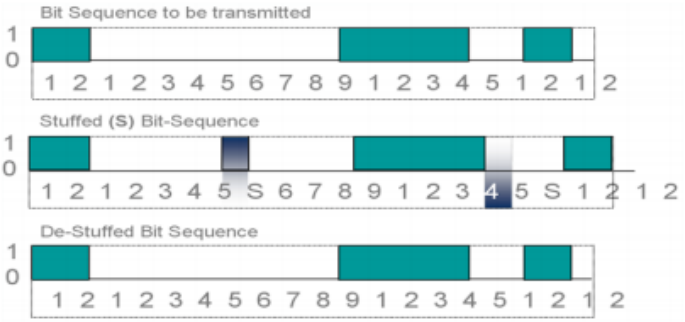


Figura I. 5 Norma de 5 bits sucesivos

- Error Flag: El indicador de error depende de los diferentes estados del nodo en el que se encuentra.

    - Nodo en estado Activo: Cuando el nodo se encuentra en estado activo el error generado será un “indicador de error activo”, lo que provocará que los nodos de la red puedan dejar de transmitir, cómo logramos hacer esto, es colocando 6 bits dominantes sucesivos, lo que provocará que se rompa la norma en los demás nodos y provoque tramas de error.

Se pueden colocar de 6 a 12 bits dominantes, según se quiera interrumpir la transmisión en los otros nodos.

- Nodo en estado Pasivo: Cuando el nodo se encuentra en estado pasivo el error generado será un "indicador de error pasivo", con este error no se interrumpirá la transmisión de los otros nodos. Se colocan 6 bits recesivos seguidos, por lo que interrumpirá la transmisión solo en ese nodo.

- Error Delimiter: El delimitador de error consta de 8 bits recesivos, con los cuales se puede restablecer la comunicación limpiamente después del error.

El protocolo CAN, además maneja unos contadores (TEC [Transmitter Error Counter] y REC [Receiver Error Counter]) para poder determinar la calidad de la comunicación; estos contadores aumentan en 8 puntos cuando la transmisión o recepción de una trama de datos no es realizada exitosamente, si se llega a realizar exitosamente estos contadores reducen su valor en 1, a menos que se encuentren en 0 ya que se mantendrían en ese valor.
Estos contadores nos ayudan a saber en cuál de los tres diferentes estados de nuestro error nos encontramos:

- Error Active: (REC<128 y TEC<128) Este estado se presenta cuando la comunicación presenta muy pocos errores o interferencias, esto puede deberse a problemas en la inicialización del CAN, por otro lado, si este estado es inducido forzosamente al colocar 6 bits dominantes sucesivos en CRC, la comunicación de los diferentes nodos de la red se desactivará entrando en un nodo de estado activo, para evitar que la red se sature. Este último caso puede deberse a que ninguno de los mensajes se está transmitiendo o recibiendo correctamente.

- Error Pasive: (REC>127 o TEC>127 y TEC<= 255) Este estado se presenta cuando la comunicación es intermitente, puede estar recibiendo y

transmitiendo con intervalos de falla. Esto hace que el nodo entré en estado pasivo, lo que permite que este nodo sea interrumpido, sin interrumpir la comunicación de los demás nodos.

- Bus-Off: (TEC>255) Este estado se presenta cuando el nodo no envía ni transmite la trama de datos; en este caso es necesario inducir el estado de error activo, para evitar que la red se sature.

TRAMA DE SOBRECARGA

La trama de sobrecarga tiene casi el mismo formato que la trama de error activo, la diferencia está en que la trama de sobrecarga sólo puede presentarse en el espacio entre tramas. A la trama de sobrecarga se anexan dos campos más.

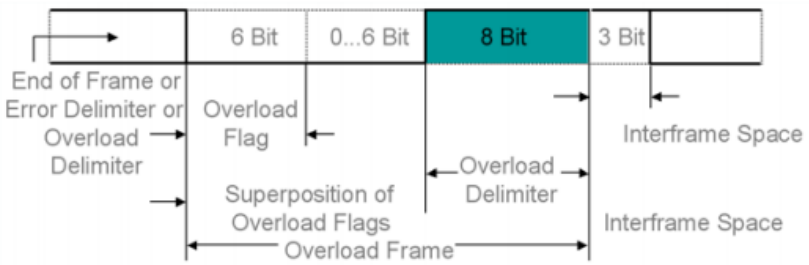


Figura I. 6 Formato de trama de sobrecarga

- Overload Flag: El indicador de sobrecarga consta de 6 bits dominantes que pueden ser seguidos de bit generados por otros nodos; este campo tiene un máximo de 12 bits dominantes.
- Overload Delimiter: El delimitador de sobrecarga es de 8 bits recesivos e indica el final de la trama de sobrecarga.

AII FILTRO DE KALMAN

El filtro Kalman es una herramienta matemática muy potente que juega un papel muy importante cuando se trabaja con medidas del mundo real. Fue inventado por Rudolph Emil Kalman a finales de la década de 1950, con la finalidad de filtrar y predecir

sistemas lineales. Este algoritmo matemático es un conjunto de ecuaciones matemáticas que implementan un estimador óptimo del tipo predictor-corrector, el cual procesa todas las medidas disponibles, sin importar su precisión con el fin de estimar el valor actual de las variables de interés. Es importante tener un buen conocimiento del sistema y los dispositivos dinámicos de medición debido a que dentro de las ecuaciones de trabajo se requiere información que es necesaria conocer o determinar.

El KF es un algoritmo de procesamiento de datos por lo que su aplicación es mediante un software por lo que es necesario trabajar sobre medidas discretas en vez de trabajar en tiempo continuo. Las ecuaciones matemáticas que describen al KF se observan en la Tabla II.1

Tabla II. 1 Algoritmo del Filtro de Kalman

	Algoritmo KF$(\mathbf{x}_{t-1}, \mathbf{P}_{t-1}, \mathbf{u}_t, \mathbf{z}_t)$**:**
	Etapa de Predicción:
1.-	Proyección del estado hacia adelante.
	$\overline{\mathbf{x}}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t$
2.-	Proyección de la covarianza del error hacia adelante.
	$\overline{\mathbf{P}}_t = \mathbf{A}_t\mathbf{P}_{t-1}\mathbf{A}_t^T + \mathbf{R}_t$
	Etapa de Actualización:
3.-	Cómputo de la ganancia de Kalman.
	$\mathbf{K}_t = \overline{\mathbf{P}}_t\mathbf{C}_t^T(\mathbf{C}_t\overline{\mathbf{P}}_t\mathbf{C}_t^T + \mathbf{Q}_t)^{-1}$
4.-	Actualización del estado con la medida $(\mathbf{z}_t)$
	$\mathbf{x}_t = \overline{\mathbf{x}}_t + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}_t\overline{\mathbf{x}}_t)$
5.-	Actualización de la covarianza del error
	$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{C}_t)\overline{\mathbf{P}}_t$
6.-	Devuelve $\mathbf{x}_t, \mathbf{P}_t$

El principio fundamental del KF es inicialmente generar una creencia o confianza en un punto específico del tiempo, dicho de manera matemática es la media x, y covarianza P. Estos valores son estimados de acuerdo a los valores de entrada en un

estado anterior t-1, en donde se actualiza dicha creencia con base a las señales de control (u) y las observaciones en el instante de tiempo.

El KF está estructurado en tres etapas, Etapa de predicción, Etapa de actualización y una posible adición de nuevas características.

1. Etapa de predicción: en esta etapa se proyecta la creencia en el instante de tiempo actual, a través de la odometría la cual consiste en la predicción de la posición relativa de un sistema.
2. Etapa de actualización: En esta etapa se consideran las características observadas, gracias a la estimación de la posición proporcionada por la etapa de predicción, es posible determinar donde debería estar la característica, lo cual permite corregir el valor de entrada.
3. En la etapa de adición de nuevas características al mapa. este proceso se lleva a cabo usando información acerca de la posición actual y añadiendo información sobre la relación entre las nuevas y las viejas características.

Kt es la matriz que representa a la ganancia de Kalman, la cual indica la confianza en las características observadas, empleando la incertidumbre de las mismas junto con una medida de la calidad de los datos proporcionados por los instrumentos de medición, un ejemplo es en el caso de trabajar con un sensor láser y la odometría, si los datos del sensor láser no son buenos y la odometría es buena, se tendrá que dar más peso a la odometría, lo que genera un decremento en el valor de la ganancia de Kalman, si se presentara situación contraria el valor de la ganancia de Kalman se elevaría y tendría más peso los valores del sensor laser.

AIII CÓDIGO DE LECTURA DE DATOS

MATLAB

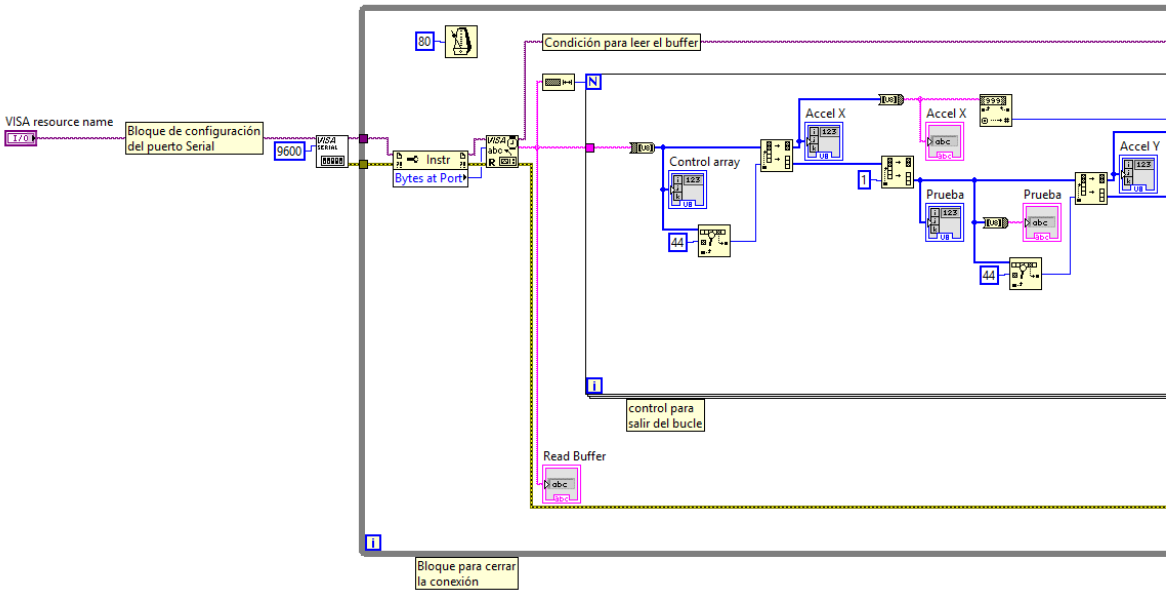
```
clear all
close all
clc
```

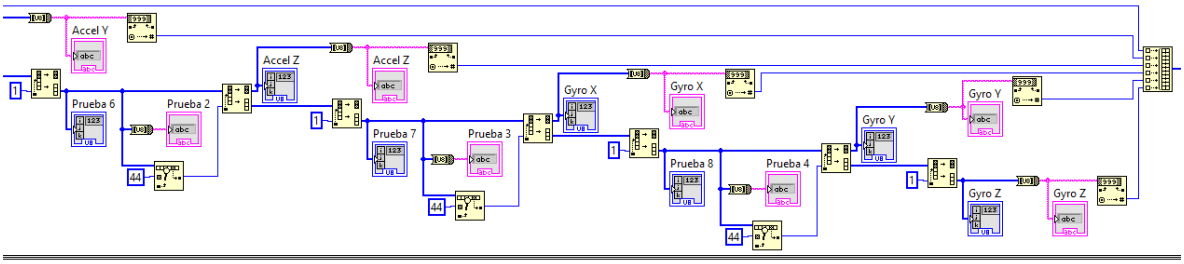
```
% Se borran los datos previos
% borrando cualquier rastro anterior que quede en el puerto COM
delete(instrfind({'Port'},{'COM7'}));
% se crea un nuevo puerto COM
s = serial('COM7','BaudRate',9600); %115200);
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
% Abrir Puerto
fopen(s);
% Parametros de medidas
tmax = 100; % tiempo de captura
rate = 18; % resultado experimental
% Se prepara la figura
f = figure('Name','Captura');
a = axes('Xlim',[0 tmax]);
l1 = line(nan,nan,'Color','g');
l2 = line(nan,nan,'Color','r');
l3 = line(nan,nan,'Color','b');
l4 = line(nan,nan,'Color','g');
l5 = line(nan,nan,'Color','r');
l6 = line(nan,nan,'Color','b');


xlabel('Tiempo (s)')
ylabel('gravedad (m/s^2) - velocidad angular (deg/s)')
title('Captura de voltaje en tiempo real con Arduino')
grid on
hold on

% Inicialización
ax = zeros (1,tmax*rate);
```

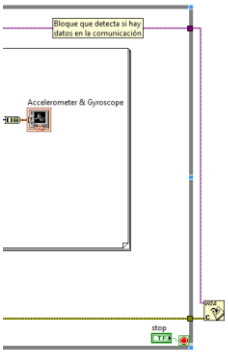
```
ay = zeros (1,tmax*rate);
az = zeros (1,tmax*rate);
gx = zeros (1,tmax*rate);
gy = zeros (1,tmax*rate);
gz = zeros (1,tmax*rate);
l_ax = zeros (1,tmax*rate);
l_ay = zeros (1,tmax*rate);
l_az = zeros (1,tmax*rate);
i = 1;
t1 = 0; t2 = 0; dt = 0;
% inicialización de los parámetros de entrada del filtro kamlan
x = [0 0 0]; % vector de estado
p = [0.1 0.1 .01]; % vector con el valor de covarianza = 0.1
P = diag(p); % formación de la matriz de covariaza
% Ejecución del ciclo de adquisición de datos el cual se cronometrará
tic
while t1 <= tmax
t1 = toc;
% lectura del puerto serial
a = fscanf(s,'%f,%f,%f,%f,%f,%f');
ax(i) = a(1);
ay(i) = a(2);
az(i) = a(3);
gx(i) = a(4);
gy(i) = a(5);
gz(i) = a(6);
l_ax(i) = x(1);
l_ay(i) = x(2);
l_az(i) = x(3);
% construir la grafica
x_tiempo = linspace(0,i/rate,i);
```

```
%     set(l1,'YData',ax(1:i),'XData',x_tiempo);
%     set(l2,'YData',ay(1:i),'XData',x_tiempo);
%     set(l3,'YData',az(1:i),'XData',x_tiempo);
% SOLO ACTUALICE EL ULTIMO VALOR
set(l4,'YData',l_ax(1:i),'XData',x_tiempo);
set(l5,'YData',l_ay(1:i),'XData',x_tiempo);
set(l6,'YData',l_az(1:i),'XData',x_tiempo);
drawnow
end

% Observamos el valor del cronometro
clc;
fprintf('%g s de captura a %g cap/s \n',t2,i/t2);

% limpiamos el puerto serial
fclose(s);
delete(s)
```

LabVIEW







ARDUINO

```
// Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h

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

// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estará implicito
MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
```

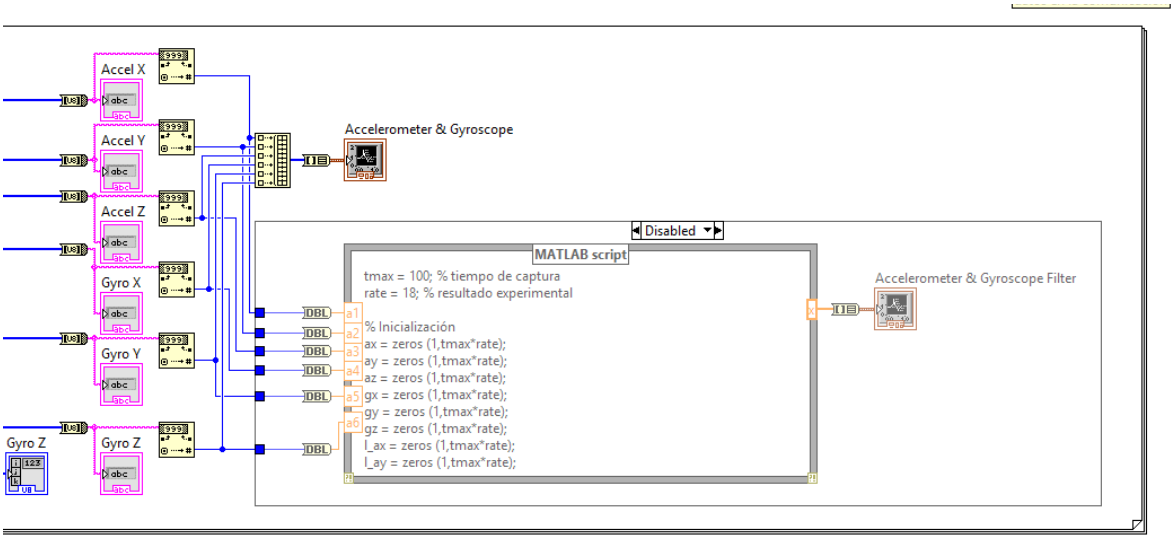
```
int ax, ay, az;
int gx, gy, gz;
const int calibracion = 2;

// Inicialización de variables para el filtro pasa bajos
long f_ax, f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx, f_gy, f_gz;
int p_gx, p_gy, p_gz;
int cont = 0;

// Variables para la adquisicioón del offset y calibracion
int ax_of, ay_of, az_of;
int gx_of, gy_of, gz_of;

void setup() {
pinMode(calibracion, INPUT);
Serial.begin(9600);    //Iniciando puerto serial
Wire.begin();          //Iniciando I2C
sensor.initialize();    //Iniciando el sensor

//  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");
//  else Serial.println("Error al iniciar el sensor");

// Lectura del valor de los offsets
ax_of = sensor.getXAccelOffset();
ay_of = sensor.getYAccelOffset();
az_of = sensor.getZAccelOffset();
gx_of = sensor.getXGyroOffset();
gy_of = sensor.getYGyroOffset();
gz_of = sensor.getZGyroOffset();
```

```
/*Serial.println("Valores Offsets : ");
Serial.print(ax_of); Serial.print("\t");
Serial.print(ay_of); Serial.print("\t");
Serial.print(az_of); Serial.print("\t");
Serial.print(gx_of); Serial.print("\t");
Serial.print(gy_of); Serial.print("\t");
Serial.print(gz_of); Serial.print("\t");

}

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

// Filtrar las lecturasjappo
f_ax = f_ax - (f_ax >> 3) + ax;
p_ax = f_ax >> 3;

f_ay = f_ay - (f_ay >> 3) + ay;
p_ay = f_ay >> 3;

f_az = f_az - (f_az >> 3) + az;
p_az = f_az >> 3;

f_gx = f_gx - (f_gx >> 3) + gx;
p_gx = f_gx >> 3;

f_gy = f_gy - (f_gy >> 3) + gy;
p_gy = f_gy >> 3;
```

```
f_gz = f_gz - (f_gz >> 3) + gz;
p_gz = f_gz >> 3;
//
float ax_n = p_ax * (9.78 / 16384.0);
float ay_n = p_ay * (9.78 / 16384.0);
float az_n = p_az * (9.78 / 16384.0);
float gx_n = p_gx * (250.0 / 32768.0);
float gy_n = p_gy * (250.0 / 32768.0);
float gz_n = p_gz * (250.0 / 32768.0);

//Calculo de los angulos de inclinacion:
//  float accel_ang_x=atan(ax_n/sqrt(pow(ay,2) + pow(az,2)))*(180.0/3.14);
//  float accel_ang_y=atan(ay_n/sqrt(pow(ax,2) + pow(az,2)))*(180.0/3.14);
//  float accel_ang_z=atan(az_n/sqrt(pow(ax,2) + pow(ay,2)))*(180.0/3.14);

//Mostrar las lecturas separadas por un [tab]
//   Serial.print("promedio:"); Serial.print("\t");
Serial.print(ax_n); Serial.print(",");
Serial.print(ay_n); Serial.print(",");
Serial.print(az_n); Serial.print(",");
Serial.print(gx_n); Serial.print(",");
Serial.print(gy_n); Serial.print(",");
Serial.println(gz_n);

//Cada 100 lecturas corregir el offset
if (cont==5){
if (p_ax>0) ax_of--;
  else {ax_of++;}
if (p_ay>0) ay_of--;
  else {ay_of++;}
```

```
if (p_az-16384>0) az_of--;
else {az_of++;}
sensor.setXAccelOffset(ax_of);
sensor.setYAccelOffset(ay_of);
sensor.setZAccelOffset(az_of);
if (p_gx>0) gx_of--;
else {gx_of++;}
if (p_gy>0) gy_of--;
else {gy_of++;}
if (p_gz>0) gz_of--;
else {gz_of++;}
sensor.setXGyroOffset(gx_of);
sensor.setYGyroOffset(gy_of);
sensor.setZGyroOffset(gz_of);
cont=0;
}
cont++;
delay(80);
}
```

LabVIEW



MICROPYTHON

```
import pyb
import math
import umatrix
import untime
import ulinalg
# main.py -- put your code here!
# Lectura del acelerometro de la pyobard
lec = pyb.Accel()
accel_1 = [lec.x(),lec.y(),lec.z()]
print ('Lectura del acelerometro de la pyobard')
print('eje x:',lec.x(),'eje y:',lec.y(),'eje z:',lec.z())
# main.py -- put your code here!
# boot.py -- run on boot-up
# can run arbitrary Python, but best to keep it minimal
T1 = utime.ticks_ms()
# VARIABLES DE ENTRADA
accel = ([0,0,9.78])
gyro = ([100,25,50])
dt = 0
P = umatrix.matrix([[0.1,0,0],[0,0.1,0],[0,0,0.1]], dtype = float) # Varianza del vector de estado (Matriz de covarianza)
x = ([30,45,60]) # Vector de estado
# def orientacion(accel,gyro,x,P,dt):
# norm_accel = 0 ; norm_gyro = 0
gravity = 9.78
vel_angular = 180/math.pi  # Cambio de unidades de grados a radianos de los valores del giroscopio
th_accel = 0.25  # Error maximo permitido para el acelerometro
th_gyro = 0.7 # Error maximo permitido para el giroscopio
```

```
gyro_rot = ([0,0,0])
for i in range(3):
# Se divide los vectores sobre la magnitud para posteriormente calcular la norma de los vectores
accel[i] = accel[i] / gravity
gyro[i] = gyro[i] / vel_angular
# Calculo de la norma de los vectores
norm_accel = math.sqrt((accel[0] ** 2) + (accel[1] ** 2) + (accel[2] ** 2))
norm_gyro  = math.sqrt((gyro[0] ** 2) + (gyro[1] ** 2) + (gyro[2] ** 2))
norm_accel = math.fabs(norm_accel - 1)
# Condicion para determinar si el sensor o la predicción debe tener mas ruido
# 1 (true) y 0(false)
if norm_accel < th_accel and norm_gyro < th_gyro:
motion = 0
else:
motion = 1
# Matrix de Rotación
Rz_Ry = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
Rot   = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
Rx = umatrix.matrix([[1,0,0],[0,math.cos(x[0]),math.sin(x[0])],[0,-
math.sin(x[0]),math.cos(x[0])]])
Ry = umatrix.matrix([[math.cos(x[1]),0,-
math.sin(x[1])],[0,1,0],[math.sin(x[1]),0,math.cos(x[1])]])
Rz = umatrix.matrix([[math.cos(x[2]),math.sin(x[2]),0],[-
math.sin(x[2]),math.cos(x[2]),0],[0,0,1]])
[m,n] = Rot.shape
# Ciclo para el producto entre la matriz Rz y Ry
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
Rz_Ry[i,j] +=  Rz[i,k] * Ry[k,j]
```

```
for i in range (0,m):
for j in range (0,n):
for p in range (0,n):
Rot[i,j] +=  Rz_Ry[i,p] * Rx[p,j]
# Ecuación para el sistema dinamico (sistema no lineal Giroscopio) - Entrada del
FILTRO KALMAN
# Estimación inicial para el estado t = 0
# A es la matriz para el sistema dinamico
A = umatrix.matrix([[1,0,0],[0,1,0],[0,0,1]] , dtype = float)
# Producto entre la matriz de rotación y el vector del gyroscopio(sistema dinamico)
# Permite cambiar la referencia al sistema global de coordenadas (gravedad)
for i in range(0,m):
for k in range (0,n):
gyro_rot[i] += Rot[i,k] * gyro[k]
# Sumatoria de los valores medidos por el giroscopio (integral) dt * dθ/dt obtencion
del angulo
u = ([0,0,0])
for i in range (3):
u[i] = gyro_rot[i] * dt
#                                                    ECUACIONES DE
PREDICCIÓN
# 1. ECUACION x = A * x  + u
x_t = ([0,0,0])
for i in range(0,m):
for k in range (0,n):
x_t[i] += A[i,k] * x[k]
for i in range (3):
x[i] = x_t[i] + u[i]
# 2. ECUACION P(t) = A * P(t-1) * A' + Q
q = 0.00001
# Q es la matriz de covarianza de ruido para la etapa de predicción
```

```
Q = umatrix.matrix([[q,0,0],[0,q,0],[0,0,q]])
# Producto A * P(t-1)
P_1 = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]], dtype = float)
P_2 = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]], dtype = float)
ban = 0
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
P_1[i,j] +=  A[i,k] * P[k,j]

# Producto A * P(t-1) * A'
T = umatrix.matrix.transpose(A)
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
P_2[i,j] +=  P_1[i,k] * T[k,j]
# Resultado
P = P_2 + Q

# Vector de Medicion y su matriz de covarianza de ruido (R)
z = accel
# al ser la medicion de sensor se le da un valor mas alto al ruido que en la etapa de
predicción
r = 0.01
R = umatrix.matrix([[r,0,0],[0,r,0],[0,0,r]])
if motion == 1:
R = R * 1000

#ECUACION DE MEDIDA
```

```
# Es necesario calcular el jacobiano debido a que estamos trabajando con un sistema
no lineal
# H es la predicción del sensor
H = ([-math.sin(x[1]),(math.sin(x[0]) * math.cos(x[1])),(math.cos(x[0]) *
math.cos(x[1]))])
J = umatrix.matrix([[0,-math.cos(x[1]),0],[(math.cos(x[0])*math.cos(x[1])),(-
math.sin(x[1])*math.sin(x[0])),0],[(-math.cos(x[1])*math.sin(x[0])),(-
math.sin(x[1])*math.cos(x[0])),0]])
# Se utiliza la matriz jacobiana en EFK (sistema dinamico - no lineal)
# Se utiliza la matriz jacobiana para relacionar posicion con la rotación
#  y es el error en la medicion de los sensores
y = ([0,0,0])
for k in range (3):
y[k] = z[k] - H[k]
# Calculo de la ganancia de kalman (K) y la matriz de covarianza aposteriori (t+1)
# La ganancia de Kalman minimiza la correlación de error a-posteriori
# 1. ECUACION S = J * P(t) * J' + R
S = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
S1 = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
# Producto J * P(t+1)
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
S[i,j] +=  J[i,k] * P[k,j]


# Producto J * P(t+1) * J'
T_J = umatrix.matrix.transpose(J)
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
S1[i,j] +=  S[i,k] * T_J[k,j]
```

```
# Suma + R
S = S1 + R

# 2. ECUACION K = (P(t) * J') / S
KALMAN = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
K = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
KALMAN[i,j] +=  P[i,k] * T_J[k,j]

D = ulinalg.det_inv(S)
det = D[0]
det = 1 / det
# Calculo de la adjunta de la matriz S
# Primero se calcula la matriz de cofactores
a = ((S[1,1] * S[2,2]) - (S[1,2] * S[2,1]))
b = (-(S[1,0] * S[2,2]) - (S[1,2] * S[2,0]))
c = ((S[1,0] * S[2,1]) - (S[1,1] * S[2,0]))
d = (-(S[0,1] * S[2,2]) - (S[0,2] * S[2,1]))
e = ((S[0,0] * S[2,2]) - (S[0,2] * S[2,0]))
f = (-(S[0,0] * S[2,1]) - (S[0,1] * S[2,0]))
g = ((S[0,1] * S[1,2]) - (S[0,2] * S[1,1]))
h = (-(S[0,0] * S[1,2]) - (S[0,2] * S[1,0]))
i = ((S[0,0] * S[1,1]) - (S[0,1] * S[1,0]))
S = umatrix.matrix([[a,b,c],[d,e,f],[g,h,i]],dtype = float)
T_S = umatrix.matrix.transpose(S)
S = T_S * det

# Calculo de la ganancia de Kalman
for i in range (0,m):
```

```
for j in range (0,n):
for k in range (0,n):
K[i,j] +=  KALMAN[i,k] * S[k,j]

# 3.ECUACION P(t+1) = (I - K * J) * P(t) --- Varianza del vector de estado en la etapa de corrección
I = umatrix.matrix([[1,0,0],[0,1,0],[0,0,1]], dtype = float)
K1 = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]], dtype = float)
P_nuevo = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]], dtype = float)

# Producto K * J
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
K1[i,j] +=  K[i,k] * J[k,j]
# Resta I - K * J
I = I - K1
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
P_nuevo[i,j] +=  I[i,k] * P[k,j]
P = P_nuevo
# 4.ECUACION x(t+1) = x + K * (y) --- Evaluacion del estado a-posteriori (Actualizacion del vector de estado)
# Producto K * y
new = ([0,0,0])
for i in range(0,m):
for k in range (0,m):
new[i] += K[i,k] * y[k]
# Suma
for i in range (3):
```

```
x[i] = x[i] + new[i]

# Se utiliza la matriz de rotación para alinear nuevamente el vector de estado con
base a una referencia global
# Matrix de Rotación
Rz_Ry = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
Rot   = umatrix.matrix([[0,0,0],[0,0,0],[0,0,0]],dtype = float)
Rx = umatrix.matrix([[1,0,0],[0,math.cos(x[0]),math.sin(x[0])],[0,-
math.sin(x[0]),math.cos(x[0])]])
Ry = umatrix.matrix([[math.cos(x[1]),0,-
math.sin(x[1])],[0,1,0],[math.sin(x[1]),0,math.cos(x[1])]])
Rz = umatrix.matrix([[math.cos(x[2]),math.sin(x[2]),0],[-
math.sin(x[2]),math.cos(x[2]),0],[0,0,1]])
[m,n] = Rot.shape
# Ciclo para el producto entre la matriz Rz y Ry
for i in range (0,m):
for j in range (0,n):
for k in range (0,n):
Rz_Ry[i,j] +=  Rz[i,k] * Ry[k,j]
for i in range (0,m):
for j in range (0,n):
for p in range (0,n):
Rot[i,j] +=  Rz_Ry[i,p] * Rx[p,j]

# Calculo de la aceleracion lineal
n = ([0,0,1])
accel_lineal = ([0,0,0])
for i in range(0,m):
for k in range (0,m):
accel_lineal[i] += Rot[i,k] * accel[k]
```

```
for i in range (3):
accel_lineal[i] = accel_lineal[i] - n[i]
for i in range (3):
accel_lineal[i] = accel_lineal[i] * gravity
T2 = utime.ticks_ms()
dt = T2 - T1
print ('El diferencial de tiempo es:',dt)
print('vector de estado (orientación)', x)
print('Matriz de covarianza',P)
print('Ganancia de Kalman',K)
#pyb.main('main.py') # main script to run after this one
#pyb.usb_mode('CDC+MSC') # act as a serial and a storage device
#pyb.usb_mode('CDC+HID') # act as a serial device and a mouse
```

AIV CÓDIGO DE COMUNICACIÓN CAN

ARDUINO

```
// Transmisión Sensores
#include <mcp_can.h>
#include <SPI.h>
#include "MPU6050.h"
#include "I2Cdev.h"
#include "Wire.h"
// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estará implicito
MPU6050 sensor;
unsigned long int canID = 0x00; //0x00FF3101; //0x0001;
int calib = 0;
// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;
```

```
const int calibracion = 2;
// Inicialización de variables para el filtro pasa bajos
long f_ax, f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx, f_gy, f_gz;
int p_gx, p_gy, p_gz;
int cont = 0;
// Variables para la adquisicioón del offset y calibracion
int ax_of, ay_of, az_of;
int gx_of, gy_of, gz_of;
const int SPI_CS_PIN = 10; //CS pin
MCP_CAN CAN(SPI_CS_PIN);                    // Set CS pin
void setup()
{
Serial.begin(115200);
Wire.begin();          //Iniciando I2C
sensor.initialize();    //Iniciando el sensor
// Lectura del valor de los offsets
ax_of = sensor.getXAccelOffset();
ay_of = sensor.getYAccelOffset();
az_of = sensor.getZAccelOffset();
gx_of = sensor.getXGyroOffset();
gy_of = sensor.getYGyroOffset();
gz_of = sensor.getZGyroOffset();
while (CAN_OK != CAN.begin(CAN_250KBPS))            // init can bus : baudrate
= 500k
{
Serial.println("CAN BUS Shield init fail");
Serial.println(" Init CAN BUS Shield again");
delay(100);
}
```

```
Serial.println("CAN BUS Shield init ok!");
}
void loop()
{
signed char stmp[8] = {0, 0, 0, 0, 0, 0, 0, 0};
// Leer las aceleraciones y velocidades angulares
sensor.getAcceleration(&ax, &ay, &az);
sensor.getRotation(&gx, &gy, &gz);
// Filtrar las lecturas/*
f_ax = f_ax - (f_ax >> 3) + ax;
p_ax = f_ax >> 3;
f_ay = f_ay - (f_ay >> 3) + ay;
p_ay = f_ay >> 3;
f_az = f_az - (f_az >> 3) + az;
p_az = f_az >> 3;
f_gx = f_gx - (f_gx >> 3) + gx;
p_gx = f_gx >> 3;
f_gy = f_gy - (f_gy >> 3) + gy;
p_gy = f_gy >> 3;
f_gz = f_gz - (f_gz >> 3) + gz;
p_gz = f_gz >> 3;
//
float ax_n = p_ax * (9.78 / 16384.0);
float ay_n = p_ay * (9.78 / 16384.0);
float az_n = p_az * (9.78 / 16384.0);
float gx_n = p_gx * (250.0 / 32768.0);
float gy_n = p_gy * (250.0 / 32768.0);
float gz_n = p_gz * (250.0 / 32768.0);//*/
if(calib == 0){
for(int i = 0; i < 100; i++){
calib = 1;
```

```
if (cont==5){
if (p_ax>0) ax_of--;
  else {ax_of++;}
if (p_ay>0) ay_of--;
  else {ay_of++;}
if (p_az-16384>0) az_of--;
  else {az_of++;}
sensor.setXAccelOffset(ax_of);
sensor.setYAccelOffset(ay_of);
sensor.setZAccelOffset(az_of);
if (p_gx>0) gx_of--;
  else {gx_of++;}
if (p_gy>0) gy_of--;
  else {gy_of++;}
if (p_gz>0) gz_of--;
  else {gz_of++;}
sensor.setXGyroOffset(gx_of);
sensor.setYGyroOffset(gy_of);
sensor.setZGyroOffset(gz_of);
cont=0;
}
//Serial.println("Valores: ");
//Serial.print(ax_n); Serial.print(", ");
//Serial.print(ay_n); Serial.print(", ");
//Serial.print(az_n); Serial.print(", ");
//Serial.print(gx_n); Serial.print(", ");
//Serial.print(gy_n); Serial.print(", ");
//Serial.print(gz_n); Serial.println("");
cont++;
delay(80);
}
```

```
}
Serial.println("Valores: ");
Serial.print(ax_n); Serial.print(", ");
Serial.print(ay_n); Serial.print(", ");
Serial.print(az_n); Serial.print(", ");
Serial.print(gx_n); Serial.print(", ");
Serial.print(gy_n); Serial.print(", ");
Serial.print(gz_n); Serial.println("");
stmp[0] = ax_n;
stmp[1] = ay_n;
stmp[2] = az_n;
stmp[3] = gx_n;
stmp[4] = gy_n;
stmp[5] = gz_n;
//Enviar Node 1
CAN.sendMsgBuf(canID, 0, 8, stmp);
delay(50);                    // send data per 50ms
for(int x = 0; x<8; x++){
Serial.print(stmp[x]); Serial.print(", ");
}
Serial.println("");
}
```

```
//Código prueba recepción/transmisión
#include <SPI.h>
#include "mcp_can.h"
const int SPI_CS_PIN = 10; //CS pin
MCP_CAN CAN(SPI_CS_PIN);                    // Set CS pin
void setup()
{
Serial.begin(115200);
```

```
while (CAN_OK != CAN.begin(CAN_250KBPS)) //500KBPS))            // init can
bus : baudrate = 500k
{
Serial.println("CAN BUS Shield init fail");
Serial.println(" Init CAN BUS Shield again");
delay(100);
}
Serial.println("CAN BUS Shield init ok!");
}
void loop()
{
unsigned char len = 0;
unsigned char buf[8];
unsigned char stmp[8] = {0, 0, 0, 0, 0, 0, 0, 0};
//Rx - En caso de que el buffer esté lleno
if(CAN_MSGAVAIL == CAN.checkReceive())          // check if data coming
{
CAN.readMsgBuf(&len, buf);    // read data,  len: data length, buf: data buf
unsigned long canId = CAN.getCanId();
Serial.println("-----------------------------");
Serial.print("Get data from ID: 0x");
Serial.println(canId, HEX);
for(int i = 0; i<len; i++)    // print the data
{
switch(buf[i]){
case 32:
Serial.print(" ");
break;
case 97:
Serial.print("a");
break;
```

```
case 98:
Serial.print("b");
break;
case 99:
Serial.print("c");
break;
case 100:
Serial.print("d");
break;
case 101:
Serial.print("e");
break;
case 102:
Serial.print("f");
break;
case 103:
Serial.print("g");
break;
case 104:
Serial.print("h");
break;
case 105:
Serial.print("i");
break;
case 106:
Serial.print("j");
break;
case 107:
Serial.print("k");
break;
case 108:
```

```
Serial.print("l");
break;
case 109:
Serial.print("m");
break;
case 110:
Serial.print("n");
break;
case 111:
Serial.print("o");
break;
case 112:
Serial.print("p");
break;
case 113:
Serial.print("q");
break;
case 114:
Serial.print("r");
break;
case 115:
Serial.print("s");
break;
case 116:
Serial.print("t");
break;
case 117:
Serial.print("u");
break;
case 118:
Serial.print("v");
```

```
break;
case 119:
Serial.print("w");
break;
case 120:
Serial.print("x");
break;
case 121:
Serial.print("y");
break;
case 122:
Serial.print("z");
break;
default:
Serial.print("-");
break;
}
}
Serial.println();
}
//Tx - En caso de que no haya que recibir
stmp[7] = stmp[7]+1;
if(stmp[7] == 100)
{
stmp[7] = 0;
stmp[6] = stmp[6] + 1;
if(stmp[6] == 100)
{
stmp[6] = 0;
stmp[5] = stmp[6] + 1;
}
```

```
}
CAN.sendMsgBuf(0xFE01, 0, 8, stmp);
delay(100);                    // send data per 100ms
for(int i = 0; i < 8; i++){
Serial.print(stmp[i]); Serial.print(", ");
}
Serial.println(" ");
}
```

MICROPYTHON

```
# boot.py -- run on boot-up
# can run arbitrary Python, but best to keep it minimal
import machine
import pyb
from pyb import CAN
#pyb.main('main.py') # main script to run after this one
#pyb.usb_mode('VCP+MSC') # act as a serial and a storage device
#pyb.usb_mode('VCP+HID') # act as a serial device and a mouse
# main.py -- put your code here!
i = 0
while i < 2:
pyb.LED(1).on()
pyb.delay(500)
pyb.LED(2).on()
pyb.delay(500)
pyb.LED(1).off()
pyb.delay(500)
pyb.LED(2).off()
pyb.delay(500)
i = i + 1
#
def cb0(bus, reason):
```

```
print('cb0')
if reason == 0:
print('pending')
if reason == 1:
print('full')
if reason == 2:
print('overflow')
#
can = CAN(1) #, CAN.LOOPBACK)
can.deinit()
can.clearfilter(0)
can.clearfilter(1)
can.initfilterbanks(14)
can.init(CAN.NORMAL, extframe=False, prescaler=21, sjw=1, bs1=5, bs2=2)
#extframe=True for 29bit identifier, PCL1=42000000 (42MHz,
23.8095nsec),250kbps= 42MHz/(21*(1+5+2))
can.setfilter(1, CAN.MASK32, 0, (0x0, 0x0)) #Aplicar filtro para recibir de
cualquier ID
print(can.any(0))
print(can.any(1))
while True:
print(can.any(0))
print(can.any(1))
print(can.rxcallback(0,cb0))
print(can.rxcallback(1,cb0))
print(can.info())
can.send('hola', 0x00)
print(can.info())
pyb.delay(1000)
print(can.info())
buf1 = bytearray(8)
```

```
lst1 = [0, 0, 0, memoryview(buf1)]
print(can.info())
can.recv(1, lst1, timeout=5000)
print(can.info())
pyb.delay(1000)
print(can.info())
print(lst1)
buf2 = bytearray(8)
lst2 = [0, 0, 0, memoryview(buf2)]
print(can.info())
can.recv(0, lst2, timeout=5000)
print(can.info())
pyb.delay(1000)
print(can.info())
print(lst2)
```