ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE MECÁNICA

CARRERA DE INGENIERÍA DE MANTENIMIENTO

"OBTENCIÓN DE LA CURVA CARACTERÍSTICA DE LA POSICIÓN DE UN EJE DE MOTOR DESDE UN ORIGEN, MEDIANTE LOS DATOS OBTENIDOS POR UN SENSOR DE TRES DIMENSIONES"

Trabajo de titulación
Tipo: Trabajo Experimental

Presentado para optar al grado académico de:

INGENIERO DE MANTENIMIENTO

AUTOR: ALAN PATRICIO DEL PINO VALENCIA

DIRECTOR: Ing. JULIO CAJARMARCA

Riobamba – Ecuador

2021

Yo, Alan Patricio Del Pino Valencia, declaro que el presente trabajo de titulación es de mi autoría y los resultados del mismo son auténticos. Los textos en el documento que provienen de otras fuentes están debidamente citados y referenciados.

Como autor asumo la responsabilidad legal y académica de los contenidos de este trabajo de titulación; El patrimonio intelectual pertenece a la Escuela Superior Politécnica de Chimborazo.

Riobamba, 01 de abril de 2021

Alan Patricio Del Pino Valencia
060499538-1

ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE MECÁNICA

CARRERA DE INGENIERÍA DE MANTENIMIENTO

El tribunal del trabajo de titulación certifica que: El trabajo de titulación: Tipo: Propuesta Experimental, **OBTENCIÓN DE LA CURVA CARACTERÍSTICA DE LA POSICIÓN DE UN EJE DE MOTOR DESDE UN ORIGEN, MEDIANTE LOS DATOS OBTENIDOS UN SENSOR DE TRES DIMENSIONES**, realizado por el señor: **ALAN PATRICIO DEL PINO VALENCIA**, ha sido minuciosamente revisado por los Miembros del Tribunal del trabajo de titulación, el mismo que cumple con los requisitos científicos, técnicos, legales, en tal virtud el Tribunal Autoriza su presentación.

	FIRMA	**FECHA**
Ing. José Antonio Granizo Ph.D. **PRESIDENTE DEL TRIBUNAL**	JOSE ANTONIO GRANIZO Firmado digitalmente por JOSE ANTONIO GRANIZO Fecha: 2021.04.01 22:24:35 -05'00'	2021-04-01
Ing. Julio Eduardo Cajamarca Villa **DIRECTOR DEL TRABAJO DE TITULACIÓN**	JULIO EDUARDO CAJAMARCA VILLA Firmado digitalmente por JULIO EDUARDO CAJAMARCA VILLA Fecha: 2021.04.01 17:01:11 -05'00'	2021-04-01
Ing. Eduardo Segundo Hernandez Dávila **MIEMBRO DE TRIBUNAL**	Firmado electrónicamente por: EDUARDO SEGUNDO HERNANDEZ DAVILA	2021-04-01

DEDICATORIA

El presente trabajo experimental lo dedico primero a Dios, por darme fuerza, apoyo e inspiración, no decaer en el intento, llenarme de salud y vida estos años.

A mis padres, por el apoyo y amor incondicional en todos estos años que tengo de vida, gracias a ustedes he podido llegar a cumplir una meta más. Sus acciones, su sacrificio y su esfuerzo, que hicieron de mí una gran persona. Les quiero mucho.

Finalmente quiero dedicar este trabajo a mis familiares y amigos, que con su apoyo moral y consejos que me han brindado en esta etapa de mi vida.

Alan

AGRADECIMIENTO

Agradezco a Dios por protegerme y bendecirme de salud y vida, poder culminar este trabajo de titulación pese a que me toco estar en momentos de pandemia y pude salir adelante estando enfermo con el virus, Dios me ha protegido y me ha dado las fuerzas para poder seguir adelante en tiempos difíciles.

Agradezco a mis padres los dos simpre han sido pilares fundamentales en mi vida, me han apoyado, ayudado y me han visto crecer y ver como culmino mis metas, me siento muy agradecido por tenerlos a mi lado, se que siempre me van a apoyar.

Adradezco a la Escuela Superior Politécnica de Chimborazo y en especial a mi escuela de Ingenieria de Mantenimiento, ya que por medio de los docentes me han podido brindar sus conocimeintos y compartir sus experiencias y poder llenarle de ese conociento y formarme profesionamente y también como persona, un agradecimiento en especial a mi director de tesis ing. Julio Cajamarca y mi miembro ing. Eduardo Hernández gracias por birindarme su apoyo y dedicación en todo el tiempo del trabajo de titulación les tengo un gran cariño y aprecio.

.

Alan

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE GRÁFICOS

LISTA DE ANEXOS

RESUMEN

El objetivo de esta investigación fue obtener la curva característica del movimiento de un eje de motor mediante los datos obtenidos por un sensor de tres dimensiones. Se seleccionó dos sensores, el primero fue un acelerómetro analógico ADXL335 con un microcontrolador Arduino Mega 2560 usando una conexión serial para comunicarse con el ordenador y el otro fue un sensor digital (IMU) WT901BLECL con conexión inalámbrica bluettoth 5.0, se tuvo que programar y calibrar el sensor ADXL335, el sensor WT901BLECL solo se tuvo que calibrar para poder utilizar, se diseñó los soportes para cada sensor y se imprimió en 3D para poder colocarlos en un eje de pruebas y ver el comportamiento de cada sensor. Para poder recolectar los datos y hacer las gráficas de cada sensor se tuvo que programar en Matlab versión R2019a. En la fase experimental se realizaron en total 12 pruebas a los sensores para ver el grado de error de cada sensor, las limitaciones que tiene, y se hizo una comparación entre los dos sensores para ver con cual se puede obtener la curva característica y el sensor con el que se pudo realizar la curva característica fue el WT901BLECL, se utilizó el método de regresión polinómica de Lagrange para obtener la ecuación de la curva porque se tuvo en las pruebas datos positivos y negativos, siendo el que más se acopla a la curva. Gracias al sensor WT901BLECL se pudo ver y analizar en qué ángulo existe una falla o cabeceo, porque puede medir ángulos, aceleración, velocidad angular y medir el campo magnético, el error del sensor ADXL335 fue del 100% y del WT901BLECL del 2%. No se debería usar sensores de baja sensibilidad ya que estos tienen un grado de medición muy deficiente.

Palabras clave: <TECNOLOGIA Y CIENCIAS DE LA INGENIERIA>, <CURVA CARACTERÍSTICA>, <ACELERÓMETRO>, <SISTEMA DE MEDICIÓN INERCIAL(IMU)>, <CALIBRAR>, <ERROR>, <REGRESIÓN POLINÓMICA>

ABSTRACT

The objective of this research work was to obtain the characteristic movement curve of a motor axis using the data acquired by a three-dimensional sensor. Two sensors were selected, the first was an ADXL335 analog accelerometer with an Arduino Mega microcontroller 2560 using a serial connection to get communication with the computer and the other was a digital sensor (IMU) WT901BLECL with Bluetooth 5.0 wireless connection. The ADXL335 sensor had to be programmed and calibrated, the WT901BLECL sensor had only to be calibrated for being able to use. The brackets for each sensor were designed and were 3D printed so that they could be positioned on a shaft of tests and observe the behavior of each sensor. In order to collect the data and make the graphs of each sensor. It was necessary to program in Matlab version R2019a. In the experimental phase a total of 12 tests were carried out on the sensors to see the degree of error of each one and the limitations that it has. A comparison between the two sensors was carried out to see which one can get the characteristic curve and the sensor with which the characteristic curve was completed was WT901BLECL. Since positive and negative data were found in the tests, The Lagrange polynomial regression method was used to obtain the equation of the curve, being this the one that is attached to the curve. Thanks to the WT901BLECL sensor it was possible to see and analyze the angle in which a fault or pitch exists, because it can measure angles, acceleration, angular velocity and the magnetic field. The ADXL335 sensor error was 100% and WT901BLECL 2%. Low sensitivity sensors should not be used because these have a high deficient level of measurement.

Keywords: <TECHNOLOGY AND ENGINEERING SCIENCES>, <CURVE FEATURE>, <ACCELEROMETER>, < INERTIAL MEASUREMENT SYSTEM (IMU)>, <CALIBRATE>, <ERROR>, <POLYNOMIC REGRESSION>

INTRODUCCIÓN

Antecedentes

Se han realizado estudios anteriores en la Escuela Politécnica Nacional del Ecuador sobre la utilización de los acelerómetros, en la implementación de un prototipo para el monitoreo sísmico debido al bajo costo de los acelerómetros, ya que normalmente los dispositivos que captan los sismos tienen un costo elevado y no se puede adquirir de forma masiva para todos, lo cual estos sensores de bajo costo permitir obtener información del movimiento sísmico al ser combinado con un placa de desarrollo Raspberry(microcontrolador), sin embargo esta propuesta se diferencia en el uso que se le dará a los sensores porque se pretende saber la posición de los ejes de las máquinas, combinado con un microcontrolador (David y Michelle, 2018).

Se encontraron otros estudios similares en la Universitat Oberta de Catalunya sobre el uso de los acelerómetros para el control de dispositivos mediante captura de movimiento con el fin de adquirir y transmitir la señal generada por el acelerómetro de tres ejes mediante un software, también implementa un módulo de comunicación entre el software de la computadora y el acelerómetro a través de una conexión mediante un cable de ethernet, siendo la diferencia que en este caso se va a utilizar una comunicación serial asíncrona para comunicar el microcontrolador con el ordenador a través de un cable USB (Villaluenga Morán, 2015).

Todas las maquinas rotativas deben estar bien alineadas para su correcto funcionamiento, se debe verificar que no estén desalineados los ejes del conjunto de máquinas (máquina conductora y la conducida), existen varios tipos de desalineamiento en los ejes: desalineamiento paralelo, desalineamiento angular, desalineamiento combinado (paralelo y angular).
Hay varios métodos para alinear los ejes: manualmente, mediante la utilización de una regla, mediante un reloj de caratula, y mediante equipo: alineador laser.

La ventaja de alinear manualmente los ejes está en el costo porque una regla viene a costar $1 y un reloj comparador $42 aproximadamente y las desventajas es que no son precisos en las mediciones y nos pueden dar valores no reales, no servirían para una buena alineación sino más bien para comprobar porque son imprecisos.

En cambio, alinear con un alineador laser se tiene una alta exactitud y rápida obtención de datos, los valores son más reales, lo cual da información más veraz sobre la posición en que se encuentran los ejes, y la desventaja es el costo del equipo debido a que son muy costosos viniendo a costar $15000 aproximadamente.

Por ende, la mayoría de las empresas no adquieren los equipos de alineación laser porque son muy costosos, y cuando realizan la alineación de manera artesanal da como consecuencias una errónea exactitud y a la larga tendrá efectos negativos a las máquinas, disminuyendo la eficiencia de las mismas.

Planteamiento del problema

Un inadecuado montaje en las máquinas rotativas causa que su funcionamiento sea ineficiente, la vida útil de las componentes se reduce significativamente produciéndose desalineaciones en los ejes de las máquinas, por lo tanto, es necesario que se tenga exactitud al momento de hacer el montaje de las máquinas para poder disminuir costos innecesarios.

Las industrias no toman en cuenta las consecuencias que puede ocasionar una mala instalación en las maquinas rotativas, provocando que a la larga se reduzca la vida útil de las mismas.

Justificación

En las industrias uno de los problemas que se da es el gasto innecesario de energía en los procesos debido a que la maquina no está instalada de una manera correcta, dejándola con una eficiencia baja, consumo de energía alto y sobre todo un desgate prematuro de los componentes rotativos.

Aumentar la eficiencia es lo que se logra con una correcta instalación de las máquinas, un consumo de energía reducido y menos desgaste en los componentes rotativos de las máquinas, mediante sensores se pretende ver la posición en que se encuentra los ejes de las máquinas, a través de mediciones, pruebas y obtenciones de datos que nos generan los sensores.
Con el sensor correcto se obtendrá una curva característica del eje de una maquina rotativa, donde se podrá analizar en qué posición se encuentra el eje al momento que gira.

Este proyecto investigativo ayudará a poder ver en que poción se encuentra el eje de una maquina rotativa, con esto contribuir al grupo de investigación Ciencia del Mantenimiento CIMANT, estos sensores tienen varias aplicaciones en las que se pueden aplicar a otros campos dando oportunidad a otros temas de investigación.

Objetivos

Objetivo general

Obtener la curva característica del movimiento de un eje de motor mediante los datos obtenidos por un sensor de tres dimensiones.

Objetivos específicos

- Analizar la disponibilidad de sensores XYZ, entender su funcionamiento y el porcentaje de error que se genera con el sensor seleccionado.
- Determinar el método de procesamiento de señales y el software a utilizar.
- Realizar el dispositivo de acople entre el sensor y PC.
- Obtener la curva que representa el posicionamiento del eje con respecto al origen.
- Analizar gráficamente la función de posicionamiento.

Hipótesis

El sensor de tres dimensiones permitirá determinar la posición del eje de un motor en el espacio de forma precisa.

- Variable dependiente

La posición del eje del motor.

- Variable independiente

Las señales del sensor de tres dimensiones.

CAPÍTULO I

1. MARCO TEÓRICO

1.1. Sensores

Es un dispositivo que detecta magnitudes tanto físicas como químicas y las convierte en variables medibles (Cuenca y León, 2017, p.15).

1.1.1. *Clasificación*

Existen varias formas de clasificar a los sensores, las más usuales son por el tipo de variable medida y por el principio de transducción. En la Tabla 1-1 se muestra la clasificación correspondiente:

Tabla 1-1: Clasificación de los sensores.

Principios de transducción	Variable física a medir
Químico	De posición, velocidad y aceleración
Magnético	De nivel y proximidad
Fotoeléctrico	De humedad y temperatura
Capacitivo	De fuerza y deformación
Piezo resistivo	De flujo y presión
Piezoeléctrico	De color, luz y visión
Termoeléctrico	De gas y pH
Ultrasónico	Biométricos
-	De corriente

Fuente: (Corona Ramírez et al., 2014, p.18).

Realizado por: Del Pino Valencia, Alan, 2020.

1.1.2. *Características*

Los sensores poseen características estáticas y dinámicas. Las características estáticas hacen mención a todos los parámetros que no cambian a través del tiempo o también cuando se presentan cambios demasiado lentos en la variable a medir, en cambio las características dinámicas especifican las propiedades del sensor en función del tiempo, y su comportamiento es transitorio (Cuenca y León, 2017, p.16).

1.1.2.1. *Características estáticas*

De acuerdo con (Cuenca y León, 2017, pp.16-17), establece que:

- **Sensibilidad:** hace referencia a la magnitud de entrada mínima que se necesita para emitir una señal de salida.
- **Rango:** es el intervalo que hay entre el valor mínimo y el valor máximo que puede leer el sensor.
- **Precisión:** hace referencia al grado de repetitividad, quiere decir, la capacidad que tiene el sensor para emitir la misma señal de salida por más de una vez.
- **Exactitud:** es la diferencia máxima que existe entre el valor de salida emitido por el sensor y el valor real medido.

1.1.2.2. *Características dinámicas*

Según (Cuenca y León, 2017, p.17), establece que:

- **Tiempo de respuesta:** hace referencia al tiempo que tarda el sensor en entregar el valor de la magnitud medida.
- **Histéresis:** es la facultad que tiene el sensor para seguir la curva de salida ideal.

1.2. Sistemas micro-electro-mecánicos (Mems)

MENS(Micro Electro Mechanical Systems), son sensores miniaturizados, de bajo costo y potencia, y compuestos principalmente de silicio, elementos mecánicos, sensores, actuadores y componentes electrónicos, formados en un solo sustrato(Aggarwal et al., 2010, p.1).

Una indicación importante en la construcción de los sensores es su miniaturización a través de la tecnología MEMS, lo que permite su integración en diversos dispositivos de pequeño tamaño como teléfonos, dispositivos GPS, cámaras digitales (Villaluenga Morán, 2015, p.24), teniendo muchas aplicaciones como la realidad virtual, robótica, estabilización de vehículos, drones, monitoreo médico de pacientes y seguimiento de objetos (David y Yanzapanta, 2016, p.6). En la Figura 1-1. Se muestra un acelerómetro ADXL de tres ejes que utiliza la tecnología MEMS, por eso su tamaño reducido.

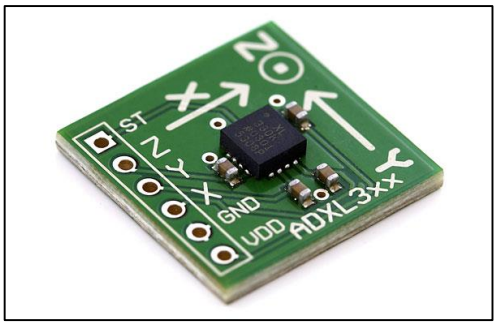


Figura 1-1. Tamaño de un acelerómetro con tecnología MEMS.
Fuente:(Sparkfun, 2011).

1.3. Sistema de medición inercial

Los sistemas de medición inercial o IMUs son dispositivos electrónicos que calculan e informan sobre la velocidad, orientación y fuerzas gravitacionales, usando una combinación de sensores: acelerómetros y giroscopios (Luengas Contreras, López Ávila y Jiménez Ezpinoza, 2017, p.95).

Hoy en día las IMUs incorporaran magnetómetros que ayudan a la orientación, el giroscopio sirve para la detección de cambios de rotación, acelerómetro para detectar la aceleración y el magnetómetro permite mejorar la precisión de los datos a través de una comparación con los puntos cardinales. El control y el procesamiento de estos sensores se realiza mediante microcontroladores (Cuenca y León, 2017, p.17). Podemos aclarar que, cada uno de los sensores que incorpora la IMU, nos da valores numéricos que son las magnitudes de medida en cado uno de los ejes en el espacio.

En otras palabras (Blascarr, 2017), decimos que:

- Acelerómetro: mide la aceleración de cada uno de los tres ejes en el espacio.
- Giroscopio: mide la velocidad angular respecto a los tres ejes en el espacio.
- Magnetómetro: mide la orientación angular respecto a los polos magnéticos de la tierra en sus ejes de coordenadas.

Los sistemas de medición inercial son muy útiles en la robótica, dispositivos que permitan realizar movimientos de objetos, teléfonos inteligentes, estudios de ergonomía, animación (Espinoza Páez, 2017, p.17).En si una gran aplicación que tienen estos sensores sobre todo para estudiar el movimiento de los cuerpos en el espacio.

En la Figura 2-1 se presenta una IMU que incluye un acelerómetro, giroscopio, y en algunos casos un magnetómetro, que se puede programar mediante un microcontrolador.

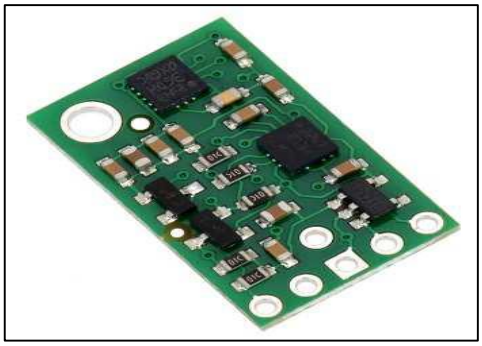


Figura 2-1. Unidad de medición inercial IMU.
Fuente: (Avila, 2015)

1.3.1. *Grados de libertad*

Los grados de libertad (D.O.F) se comprende como el número de dimensiones espaciales en los que un punto es libre de moverse (De La Cruz-Oré, 2013, p.5), quiere decir la libertad que tiene un cuerpo de moverse en el espacio.

Cuando se dice seis grados de libertad (6DOF) quiere decir la libertad de movimiento de un cuerpo en el espacio tridimensional, el cuerpo es libre de cambiar de posición como: avance, retroceso, arriba, abajo, izquierda, derecha, combinado con los cambios de orientación a través de la rotación de tres ejes perpendiculares: eje normal, eje lateral, eje longitudinal (Cuenca y León, 2017, p.18).

1.3.2. *Captura de movimiento*

Según (Luengas Contreras, López Ávila y Jiménez Ezpinoza, 2017) sirve para analizar, crear reportes y reproducir virtualmente los movimientos.

1.3.2.1. *Formas de captura de movimiento*

Hay varias formas de capturar y analizar el movimiento de un cuerpo, entre ellos está el seguimiento a través de imágenes de una serie de puntos de interés en el espacio y tiempo, para dar resultado a una representación tridimensional. La otra forma de capturar y analizar el movimiento es a través de sensores inerciales llamados IMUs.

Los IMUs permiten capturar y analizar el movimiento tomando datos de ángulos, posición y velocidad, para poder digitalizar los movimientos y crear una animación de una simulación en tres dimensiones (Luengas Contreras, López Ávila y Jiménez Ezpinoza, 2017, p.95).

1.4. Acelerómetros

Según el científico Isaac Newton se puede relacionar la aceleración con la masa y la fuerza, la aceleración es la cantidad de fuerza que se necesita para mover cada unidad de masa, estipulado en la Segunda Ley de Newton (Last Minute Engineers, 2019).

$$\boldsymbol{Aceleración\ (a)} = \frac{fuerza}{masa} \quad \textbf{(1)}$$

1.4.1. *Acelerómetro*

El acelerómetro mide la aceleración estática de la gravedad en las aplicaciones de detección de inclinación y la aceleración dinámica resultante del movimiento que son los golpes o la vibración (Last Minute Engineers, 2019). En otras palabras, la aceleración estática es la gravedad y la dinámica las vibraciones o cualquier otro movimiento. Según los grados de libertad de un acelerómetro, podrá detectar la aceleración y ser medida en uno, dos o en los tres ejes (Cuenca y León, 2017). En la Figura 3-1 se presenta un acelerómetro ADXL 335 de tres ejes, que mide la aceleración estática y dinámica.

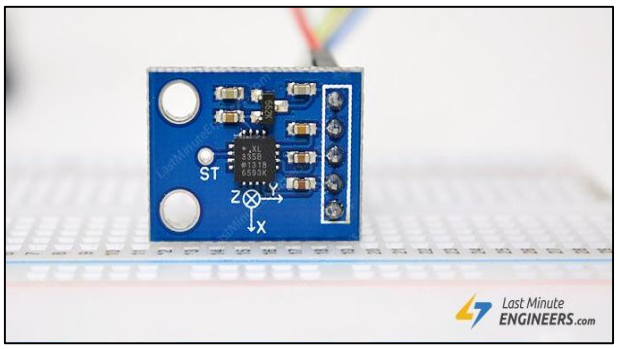


Figura 3-1. Acelerómetro ADXL 335.

Fuente: (Last Minute Engineers, 2019)

1.4.2. *Principio de funcionamiento*

Está basado en el funcionamiento de un sistema simple masa-resorte, cuando la aceleración se aplica a un cuerpo, este provoca un desplazamiento que es medido por el sensor. Como se indica en la Figura 4-1. Que consta de una masa $\boldsymbol{m}$, un resorte con una constante elástica $\boldsymbol{k}$, y un cambio de desplazamiento (Pujos Yanzapanta, 2016, p.10).

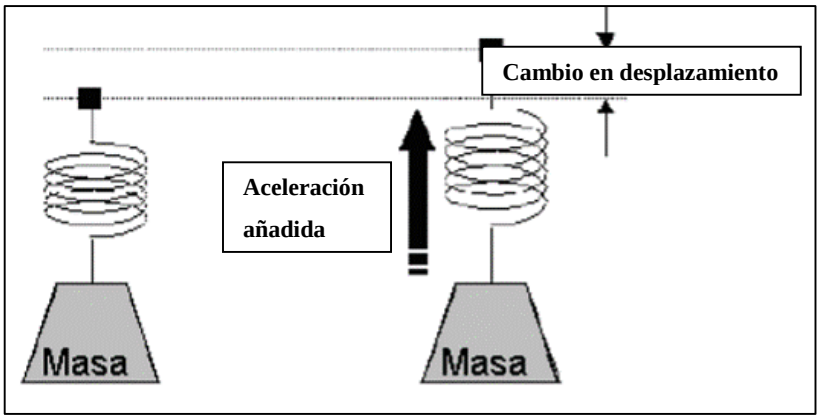


Figura 4-1. Sistema masa-resorte.

Fuente: (Sensor medidor de Aceleración, 2017, p.39)

Se especifican dos principios físicos que explican el funcionamiento del acelerómetro, la Ley de elasticidad de Hooke y la Segunda ley de Newton. La Ley de la elasticidad de Hooke establece que el alargamiento unitario que experimenta un material elástico es directamente proporcional a la fuerza aplicada sobre el mismo (Villaluenga Morán, 2015, p.19).

Considerando un sistema mecánico que consiste en una masa fija $\boldsymbol{m}$, un resorte con rigidez $\boldsymbol{k}$ (constante de elasticidad). Si la masa de desplaza a una distancia $\boldsymbol{x}$, la aceleración debido a la fuerza restauradora del resorte es (Sensor medidor de Aceleración, 2017, p.39):

$$\boldsymbol{F} = k * x \tag{2}$$

El otro principio físico sobre el que se sustentan las técnicas habituales que permiten detectar y medir las aceleraciones se base en el principio físico descubierto por Isaac Newton conocido como Segunda Ley de Newton, la cual establece que "la fuerza neta aplicada sobre un cuerpo es proporcional a la aceleración que adquiere dicho cuerpo"(Villaluenga Morán, 2015, p.19). La expresión matemática es:

$$\boldsymbol{F} = m * a \tag{3}$$

Reemplazando las dos ecuaciones matemáticas que miden la fuerza y despejando la aceleración es igual a:

$$\boldsymbol{a} = \frac{\text{k*x}}{\text{m}} \tag{4}$$

Esta manera de medir la aceleración tiene un limitante, es que solo podría ser medida en la dirección del eje del resorte. Para medir cualquier dirección en la que la masa sufriese una aceleración se debería repetir el sistema en todas las posibles direcciones. En la realidad eso sería

costoso e ineficiente, por lo que en realidad se hace para medir la aceleración, es calcular la vibración sufrida por la masa mediante la deformación soportada por el resorte (Villaluenga Morán, 2015, p.20). Por lo tanto, la segunda derivada de la masa, que será el objeto que reciba la vibración.

$$\boldsymbol{a(t)} = \ddot{x} \tag{5}$$

Se denomina a $\boldsymbol{y}$ a la posición de equilibrio inicial del resorte, la medición que se busca es determinar la deformación soportada por el resorte $\boldsymbol{y(t) - x(t)}$, a través de la conexión existente entre la aceleración y la deformación producida, se obtiene (Villaluenga Morán, 2015, p.20):

$$\boldsymbol{F} = k * (y - x) = m * \ddot{x} \tag{6}$$

Resolviendo la expresión anterior da como resultado la siguiente ecuación diferencial.

$$\boldsymbol{k * y} = m * x + \ddot{k} * x \tag{7}$$

Aplicando la transformada de Laplace en la ecuación anterior:

$$\boldsymbol{X(s)} = \frac{Y(s)}{1+\frac{m}{k}*s^2} \tag{8}$$

Empleando $\boldsymbol{\omega_0 = \sqrt{k/m}}$, frecuencia de resonancia del sistema, la expresión queda:

$$\boldsymbol{X(s)} = \frac{Y(s)}{1+\left(\frac{s}{\omega_0}\right)^2} \tag{9}$$

Mediante la expresión anterior es fácil obtener $\boldsymbol{Y(s)}$, y por la tanto hallar $\boldsymbol{Y(s) - X(s)}$

$$\boldsymbol{X(s)} = \frac{Y(s)*\left(\frac{s}{\omega_O}\right)^2}{1+\left(\frac{s}{\omega_0}\right)^2} \tag{10}$$

A partir de esta expresión se llega a concluir que, en la condición ideal del funcionamiento del acelerómetro, quiere decir que para una frecuencia de trabajo mucho menor que la frecuencia de resonancia $(\boldsymbol{\omega \ll \omega_0})$, el término $\left(\frac{\boldsymbol{s}}{\boldsymbol{\omega_o}}\right)^{\boldsymbol{2}}$ tiende a un valor próximo a cero, se establece que la deformación producida en el resorte depende directamente de la aceleración, por ende se podrá medir esta última conociendo la elongación soportada por el resorte (Villaluenga Morán, 2015, p.21).

Puntualizando ζ como factor de elongación se tiene:

$$\zeta = y(t) - x(t) \cong \frac{1}{\omega_0} * a(t) \quad \textbf{(11)}$$

De esta forma se ha transformado el problema inicial de medir la aceleración, en un problema de medir la elongación producida por el muelle. Según la tecnología de trabajo del acelerómetro esta medida tomada será convertida en una señal eléctrica (Villaluenga Morán, 2015, p.21).

Este principio fundamental, se utiliza en el más sofisticado y caro acelerómetro electromecánico, como también en los acelerómetros micro mecanizados (Sensor medidor de Aceleración, 2017).

1.4.3. *Clasificación*

Tipos de acelerómetros existentes y su funcionamiento.

1.4.3.1. *Acelerómetros mecánicos*

Los primeros acelerómetros fabricados son los mecánicos, emplean una masa inerte y resortes elásticos, midiendo las variaciones con galgas extensiométricas, incorporando sistemas de amortiguación que evitan la propia oscilación.

La aceleración produce una deformación de la galga que se convierte en una variación en la corriente detectada por un puente de Whetstone, de este modo la deformación es directamente proporcional a la aceleración aplicada al acelerómetro (Villaluenga Morán, 2015, p.13). Como se observa en la Figura 5-1. Un acelerómetro mecánico conformado por una galga extensométrica, masa inercial y resortes.

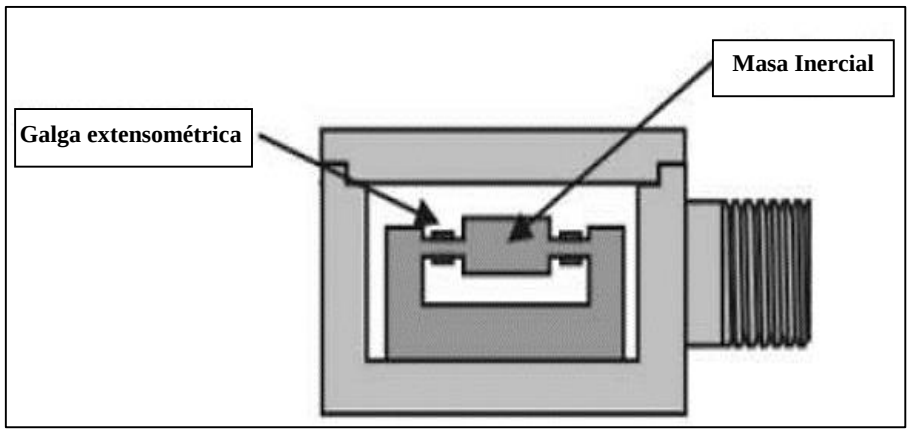


Figura 5-1. Acelerómetro mecánico.

Fuente:(Sensor medidor de Aceleración, 2017, p.41)

Dentro de este tipo de acelerómetros también se puede incluir a los acelerómetros térmicos que son capaces de detectar el desplazamiento de la masa a través de cambios en la transferencia de calor, y los servo acelerómetros (Villaluenga Morán, 2015, p.14).

Estos últimos tipos de sensores son habitualmente utilizados, su funcionamiento se basa en un sistema de equilibrio de fuerzas en lazo cerrado a través del uso de sistemas rotativos desequilibrados que originan movimientos oscilatorios cuando son expuestos a una aceleración.

Entre sus aplicaciones más usadas está en el control de dirección o el análisis de marcha en vehículos (Villaluenga Morán, 2015, p.14). Entre sus características principales están:

- El balance de fuerzas puede estar controlado electroestática o electromagnéticamente.
- Tienen una altísima precisión y sensibilidad.

En la Figura 6-1. Se muestra un esquema de cómo está compuesto un servo acelerómetro.

Miden el desplazamiento producido en las placas de un micro condensador cuando está sometido a una aceleración (Villaluenga Morán, 2015, p.15).

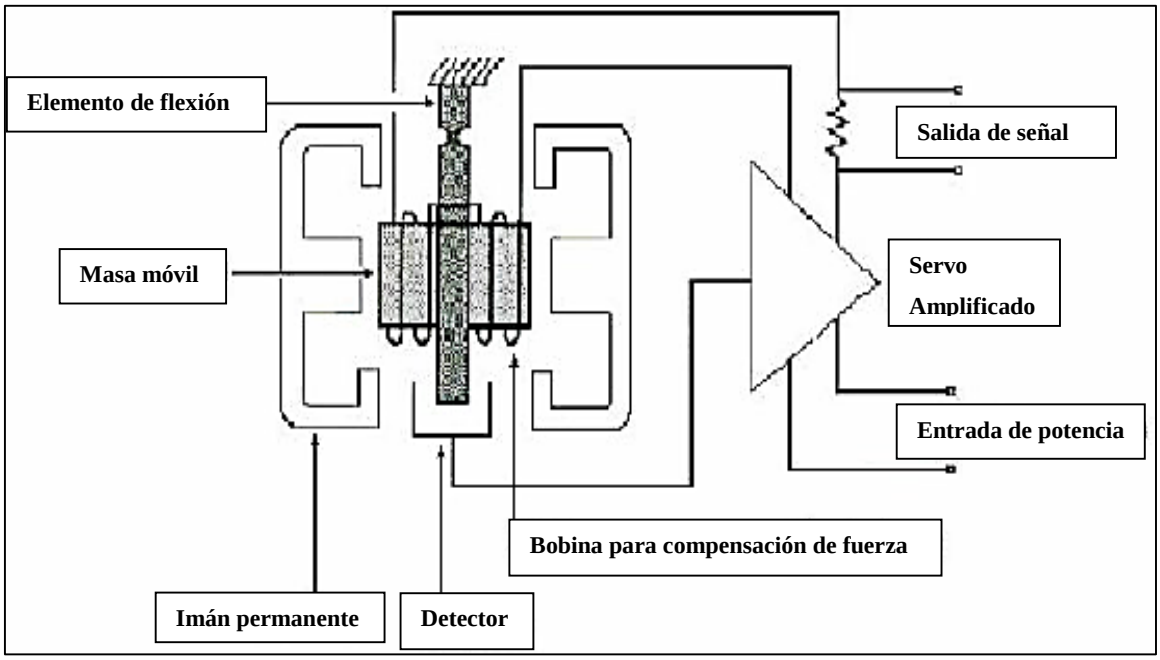


Figura 6-1. Esquema de un servo acelerómetro.

Fuente: (Villaluenga Morán, 2015, p.14)

1.4.3.2. *Acelerómetros capacitivos*

El micro condensador está formado por dos placas conductoras paralelas separadas por un material dieléctrico, disminuyendo el problema de medir la aceleración a medir la variación de capacitancia, representada en la siguiente relación (Villaluenga Morán, 2015, p.15):

$$\boldsymbol{C} = \varepsilon * \frac{A}{d} \quad \textbf{(12)}$$

Donde:

C, capacitancia

A, es el área de las placas en m^2

$\boldsymbol{\varepsilon}$, es la constante dieléctrica del material

d, es la distancia entre placas en metros

Al observar el sensor micro mecanizado se puede ver una estructura en forma de "H", en la cual está formada por una serie de anillos rectangulares de los cuales emergen filamentos con una masa que actúa como una placa central del condensador diferencial (Villaluenga Morán, 2015, p.15).

La aceleración o desaceleración en el eje central, desempeña una fuerza en la masa que provoca el desplazamiento de las placas del condensador. En cambio, cuando está en reposo ambas capacidades son iguales, y cuando se aplica una aceleración, se produce un aumento en una de las capacitancias y una disminución en la otra (Villaluenga Morán, 2015, p.15). Como se observa en la Figura 7-1.

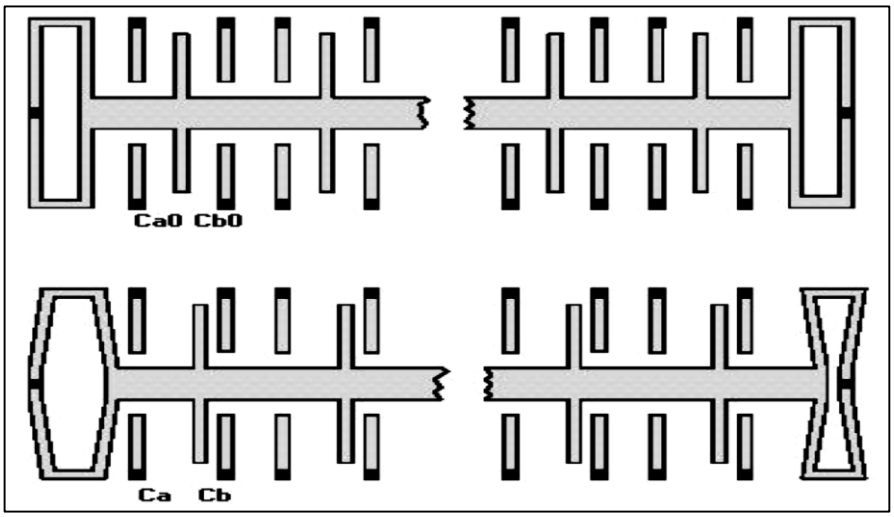


Figura 7-1. Sensor capacitivo, en reposo (arriba), sometido a una aceleración (abajo).

Fuente: (Villaluenga Morán, 2015, p.15)

Normalmente se venden como un circuito integrado en un chip de silicio, lo cual permite reducir diferentes problemáticas causadas por la temperatura, condiciones de humedad, capacidades parasitas y alta impedancia de entrada (Villaluenga Morán, 2015, p.16). Son ideales para medir frecuencias bajas a alta precisión y con un coste razonable, miden rangos entre $\pm 1G$ y $\pm 100G$, para rangos mayores se utiliza los acelerómetros piezoeléctricos (Villaluenga Morán, 2015, p.16).

1.4.3.3. *Acelerómetros piezoeléctricos*

El funcionamiento está basado en el efecto piezoeléctrico, fenómeno que aparece en algunos cristales cuando son sometidos a tensiones mecánicas (Villaluenga Morán, 2015, p.16). Esta propiedad puede mostrarse en los cristales naturales y en los sintéticos, los materiales más utilizados son el cuarzo o el tantalio de litio (Villaluenga Morán, 2015, p.16).

El fenómeno se puede explicar por la división de las cargas positivas y negativas fijas en la red cristalina que producen que aparezcan dipolos elementales. Cabe mencionar que se produce una proporcionalidad entre la fuerza aplicada y la carga eléctrica generada, esta propiedad es muy importante para calcular la aceleración de los acelerómetros piezoeléctricos (Villaluenga Morán, 2015, p.16).

Como podemos observar en la Figura 8-1, el acelerómetro piezoeléctrico está compuesto por una carcasa o base del transductor que está unida al objeto que se quiera medir, y la red cristalina (cristal piezoeléctrico) que está en el medio y la masa sísmica.

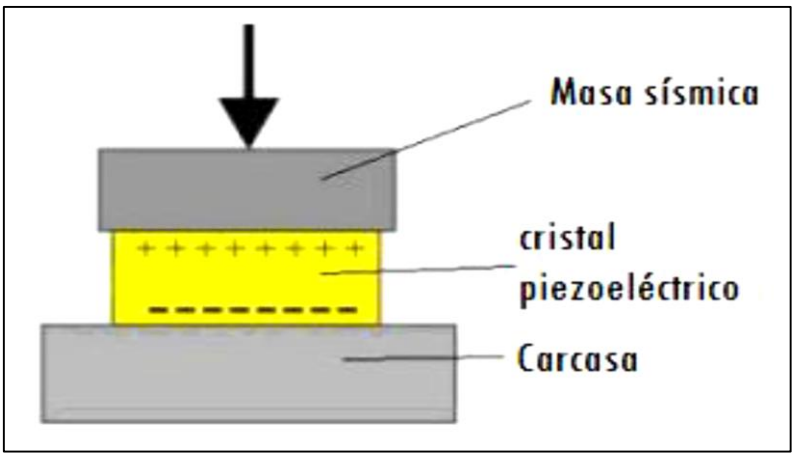


Figura 8-1. Componentes del acelerómetro piezoeléctrico.

Fuente:(Villaluenga Morán, 2015, p.17)

Cuando se produce una aceleración la masa sísmica presiona al cristal lo cual producirá una corriente eléctrica proporcional a la intensidad con la que ha sido presionado. De esta forma es posible medir la corriente directamente para poder encontrar la aceleración (Villaluenga Morán, 2015, p.17). Entre algunas de las características más importantes de estos sensores esta:

- Un amplio rango de frecuencias de trabajo, bordando las decenas de KHz.
- En bajas frecuencias y señales continuas su rendimiento no es satisfactorio.
- No necesitan de una fuente externa de alimentación.

1.4.3.4. *Acelerómetros piezo resistivos*

Son los primeros sensores de inercia micro maquinados y comercializados, utilizan el cambio de resistencia eléctrica que disponen algunos materiales cuando son sometidos a tenciones mecánicas, es el efecto piezo resistivo (Villaluenga Morán, 2015, p.17).

El efecto se debe al cambio de variación interatómica en el caso de los conductores, y también a la variación en el número de portadores en los semiconductores (Villaluenga Morán, 2015, p.17).
Estos sensores usualmente se fabrican en silicio, incluyendo impurezas (dopaje tipo p o tipo n) en un volumen pequeño del elemento.

Como se puede observar en la Figura 9-1, los componentes de este acelerómetro son: una masa inercial suspendida de un micro puente en cuyo borde se sitúa el material piezo resistivo. Cuando se produce una aceleración, la masa presiona la piezoresistencia provocando una tensión que modifica su resistividad eléctrica (Villaluenga Morán, 2015, p.18).

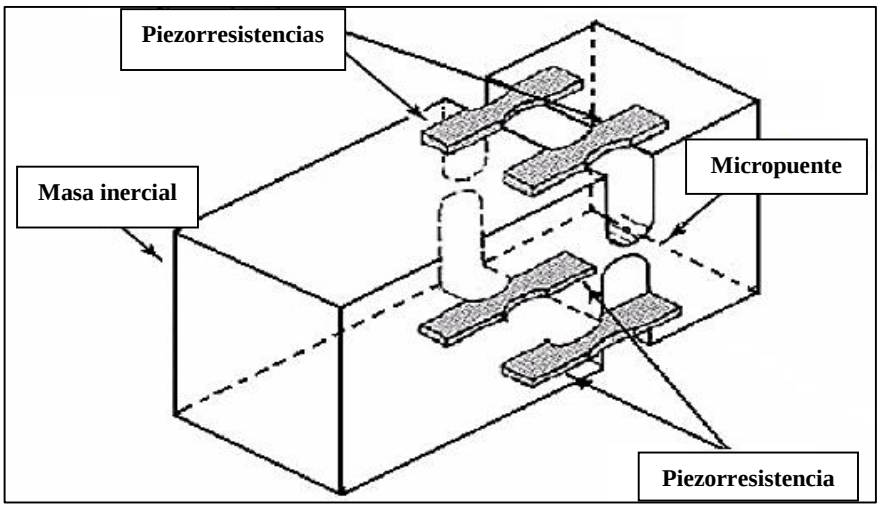


Figura 9-1. Componentes de un acelerómetro piezo resistivo.
Fuente: (Villaluenga Morán, 2015, p.18)

Entre algunas características del sensor podemos nombrar:

- Tienen una gran sensibilidad a la temperatura no deseada.
- Tienen un alto rango de media que puede llegar hasta 2000G.
- Funcionan en frecuencias de resonancia bajas.

1.5. Giroscopio

Es un sensor inercial que para describir completamente el movimiento de un cuerpo en el espacio tridimensional, mesura la velocidad angular en torno a uno de los ejes del cuerpo, su magnitud es en [°/seg], esta en el orden de 10^{-1} °/seg, y el rango puede extenderse hasta 10^{3} °/seg, estos sensores aprovechan los efectos de las fuerzas de Coriolis, presentes en el movimiento rotacional (Aggarwal et al., 2010; Pujos Yanzapanta, 2016; Corona Ramírez et al., 2014).

1.5.1. *Principio de funcionamiento*

Se basa en el efecto Coriolis lleva el nombre de un científico e ingeniero francés G.G. Delaware Coriolis, establece que la fuerza de Coriolis produce una aceleración aparente sobre un cuerpo respecto a un sistema de referencia en rotación sobre un eje (Pujos Yanzapanta, 2016, p.14).

Como se puede ver en la Figura 10-1, se comprende el efecto Coriolis, con el ejemplo de una pelota, si se la lanza desde el centro de un carrusel que está girando. Para una persona que esta fuera del carrusel la pelota sigue una trayectoria recta (línea azul), por otro lado, si la persona está dentro del carrusel, la pelota sigue una trayectoria curvilínea (línea roja), en otras palabras, percibe la aceleración en la pelota, aunque en realidad no existe, y es el resultado de la rotación a la cual es sometido (Pujos Yanzapanta, 2016, p.14).

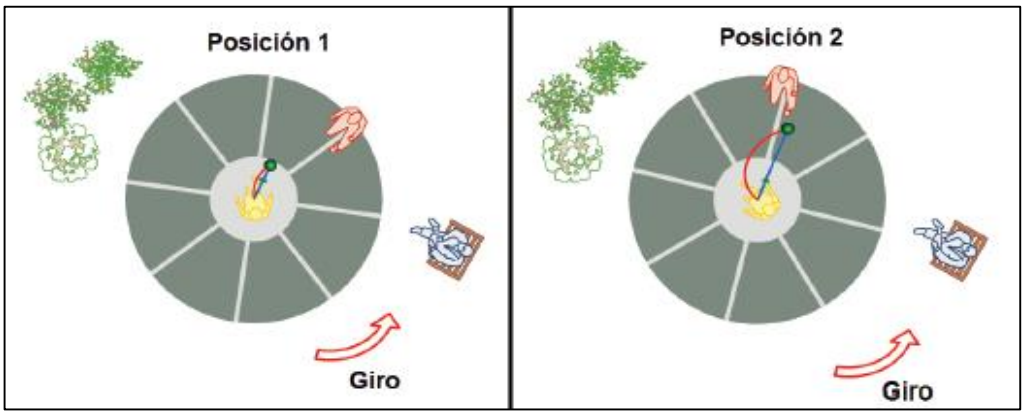


Figura 10-1. El efecto Coriolis.

Fuente:(Pujos Yanzapanta, 2016, p.14)

La fórmula de la aceleración de Coriolis viene dada por:

$$\overrightarrow{a_c} = \overrightarrow{2\omega} * \vec{v} \quad \textbf{(13)}$$

Donde:

$\overrightarrow{a_c}$, aceleración de Coriolis
$\vec{\omega}$, la rapidez angular
$\vec{v}$, la velocidad

Cuando una masa está en movimiento con una velocidad $\vec{v}$ se le aplica un movimiento angular $\vec{\omega}$, la masa percibe una fuerza en dirección perpendicular al plano formando por $\vec{\omega}$ y $\vec{v}$. Esta fuerza, y usando la segunda ley de Newton, es equivalente al producto de la masa del cuerpo por su aceleración, que en este caso es la aceleración de Coriolis (Pujos Yanzapanta, 2016, p.14).

En el giroscopio ocupa este fenómeno, pero mediante dos masas oscilando con velocidades opuestas, y cuando se provoca un movimiento angular, experimentan una fuerza de Coriolis opuesta respecto de la una a la otra, provocando un desplazamiento en dirección de esa fuerza. Lo cual sirve para determinar el valor de la velocidad angular (Pujos Yanzapanta, 2016, p.15). Como se puede observar en la Figura 11-1, se indica el sentido de la fuerza de Coriolis en las dos masas.

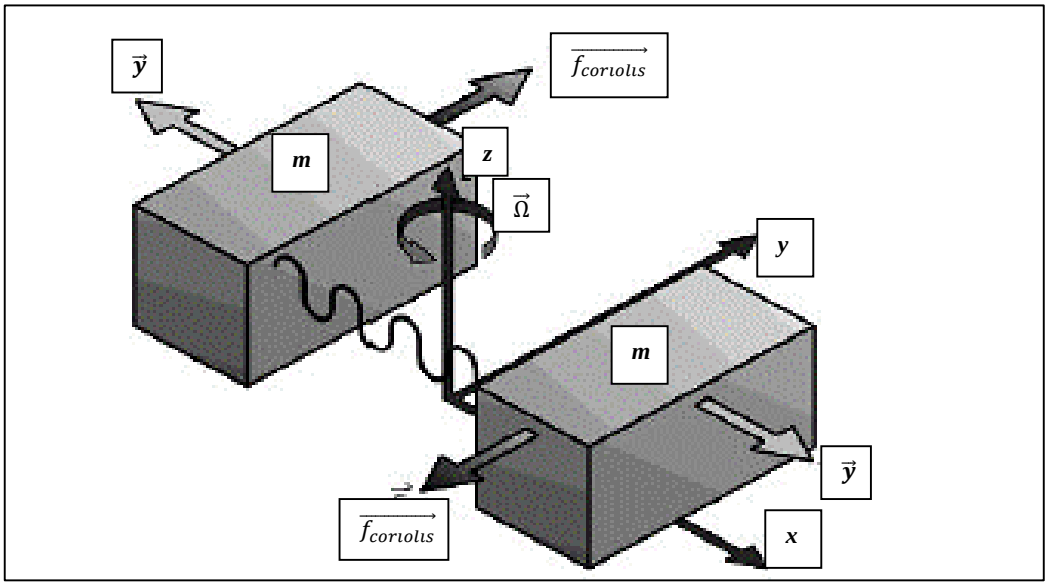


Figura 11-1. La fuerza de Coriolis en dos masas.

Fuente: (Corona Ramírez et al., 2014, p.104).

Si se la aplica a ese sistema una aceleración lineal, las masas se moverán en la misma dirección, quiere decir, que el sistema solo puede medir efectos producidos por el movimiento angular, mas no por algún tipo de movimiento lineal (Pujos Yanzapanta, 2016, p.15).

1.5.2. *Clasificación de los giroscopios*

1.5.2.1. *Diapasón*

Funciona con los extremos del diapasón en oscilación. Se los somete a una velocidad angular en torno al eje longitudinal, se ejerce un desplazamiento en direcciones opuestas de cada brazo de la "U", como se muestra en la Figura 12-1. Esta variación relativa de posición debida a la aceleración de Coriolis es localizada por el transductor que se utilice, usualmente capacitivo, después es medida por el giroscopio (Pujos Yanzapanta, 2016, p.16).

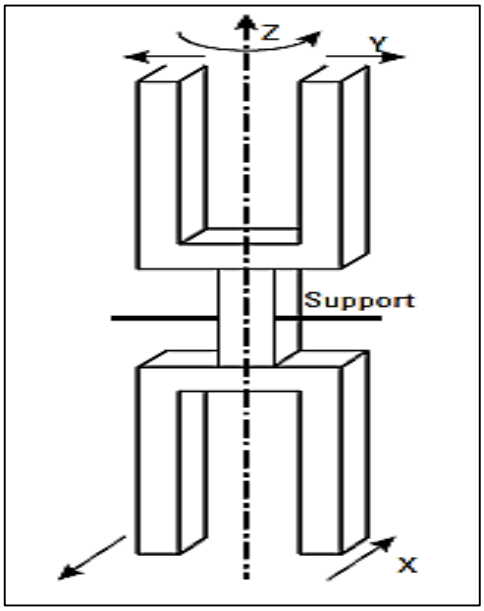


Figura 12-1. Esquema de un giroscopio diapasón.

Fuente: (Aggarwal et al., 2010, p.23).

1.5.2.2. *Rueda vibratoria*

Su funcionamiento es con una rueda en vibración alrededor de su eje axial. Cuando actúa una rotación en torno a su eje de simetría, se provoca una inclinación en la rueda la cual es medida mediante un traductor capacitivo, piezoeléctrico, piezo resistivo, etcétera (Aggarwal et al., 2010, p.23).

1.5.2.3. *Semiesférico resonante*

Giroscopios de copa de cristal o semiesférico resonante, son fabricados de sílice en vez de silicio, son precisos, y la resonancia de un anillo resonante es la variable de medida (Aggarwal et al., 2010, p.23).

1.6. Magnetómetro

Sensor que mide el campo magnético que lo bordea, como en dirección y magnitud. El campo magnético está compuesto por una mezcla entre el campo magnético terrestre y el campo generado por objetos cercanos. La unidad de media es en: μ Teslas(μT) o en Gauss (G) (Pujos Yanzapanta, 2016, p.21).

Nos permiten conocer la orientación de un objeto midiendo las fuerzas del campo magnético respecto a la tierra, de esta forma, se puede crear un vector que defina los ángulos de inclinación y declinación con respecto a los polos magnéticos de la tierra (Blascarr, 2017). Como se observa en la Figura 13-1 el campo magnético de la tierra, formado por el polo sur magnético y el polo norte magnético.

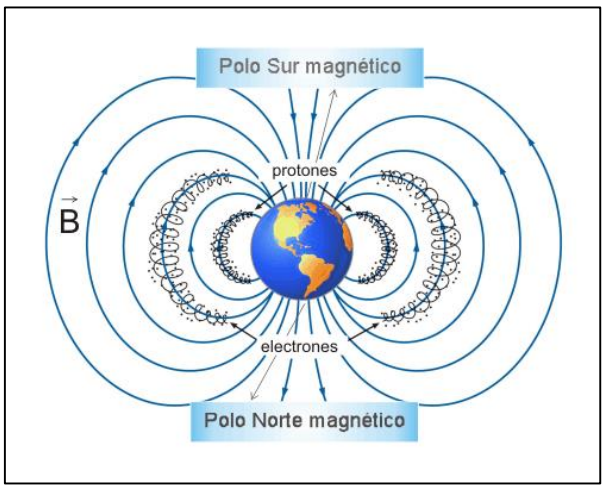


Figura 13-1. Campo magnético de la tierra.

Fuente: (Blas y Fernández).

Los magnetómetros que vienen en las IMU(unidad de medición inercial)permite tener una mejor exactitud en la detección de las variaciones de los ángulos, y en unión con el giroscopio, se obtiene resultados más precisos a la hora de obtener datos de un elemento que se mueve en el espacio (Cuenca y León, 2017, p.20).

1.6.1. *Principio de funcionamiento*

Dependiendo del fabricante el magnetómetro se puede basar en el efecto Hall, la fuerza de Lorenz o el principio piezoresistivo (Corona Ramírez et al., 2014). Sin embargo, la gran parte de los sensores magnéticos funcionan por medio el fenómeno físico efecto hall, en cual trata básicamente de un conductor por el que circula una corriente, en existencia de un campo magnético perpendicular al movimiento de las cargas, surge una separación de cargas que da lugar a un campo eléctrico en el interior del conductor, perpendicular a las cargas que están en movimiento y al campo magnético que se lo aplica.

Ese campo magnético es el que se le llama campo Hall (Electronica), como se observa en la Figura 14-1.

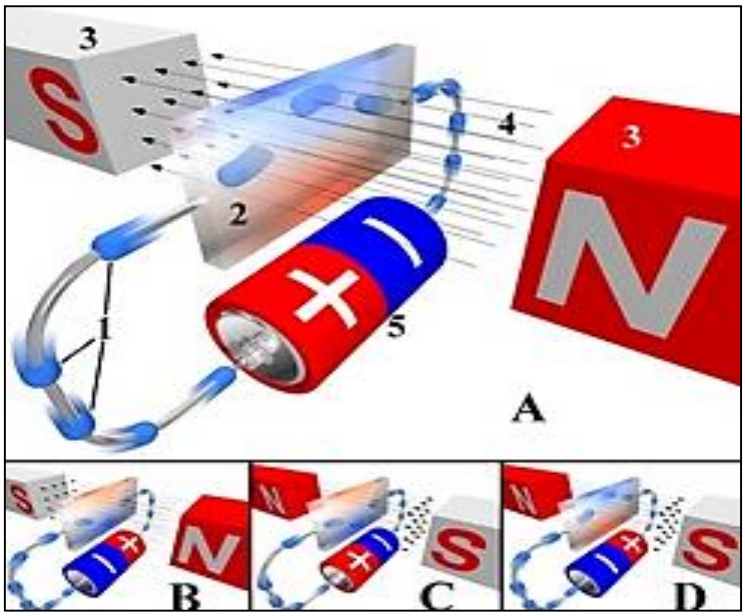


Figura 14-1. Efecto Hall.

Fuente:(Electrónica)

1.6.2. *Clasificación*

Los magnetómetros se clasifican en dos tipos:

1.6.2.1. *Magnetómetros escalares*

Miden la intensidad total del campo magnético resultante al cual están siendo sometidos en un punto, empero no aporta ningún dato sobre los componentes del campo vectorial (García, 2013, p.25). En la Figura 15-1 se muestra como es un magnetómetro escalar.

Figura 15-1. Magnetómetro escalar.

Fuente: (Electronica).

1.6.2.2. *Magnetómetros vectoriales*

Es la capacidad de mesurar la intensidad del campo magnético en una determinada dirección, tomando en cuenta la colocación que le demos al dispositivo (García, 2013, p.25), ofrecen una lectura más precisa y funcionan con un nivel muy bajo de ruido (Electronica), como se observa en la Figura 16-1.

Figura 16-1. Magnetómetro vectorial.

Fuente: (Electronica).

1.7. Placas de desarrollo

Es la que proporciona todo lo necesario para poder programar un microcontrolador, son plataformas de desarrollo basadas en lenguajes de programación, para poder realizar una serie de comandos con el fin de llevar a cabo una tarea específica. Las placas de desarrollo pueden ser de código abierto que facilita a la programación siendo capaz de enviar y recibir información hacia los dispositivos conectados en ella, también a través de internet o dispositivos inalámbricos, para poder controlar un dispositivo especifico (Rosero y Mazón, 2019, p.26).

Las placas de desarrollo incorporan una fuente de alimentación, soporte para conectar los sensores y actuadores, en algunos casos ya está incluida en la placa y el software de la compañía que ayuda a la creación y manejo del código para su ejecución (Lemus, 2019).

En el mercado existe una gran cantidad de placas de desarrollo, pero se mencionará las más importantes.

1.7.1. *Arduino*

Es una plataforma de código abierto, intuitiva y fácil de usar se basa en hardware y software libre, la placa de desarrollo Arduino lee las entadas y las convierte en salida para activar cualquier sensor o actuador dependiendo del modelo, todo eso lo hace enviando instrucciones al microcontrolador (dependiendo del tipo de microcontrolador) mediante líneas de código, su lenguaje de programación está basado en el cableado y el software de Arduino (IDE). Como se observa en la Figura 17-1 el Arduino Mega 2560 ofrece ventajas como su costo, que es multiplataforma para Windows, Macintosh OS X, Linux, y su programación es simple (Arduino, 2020).

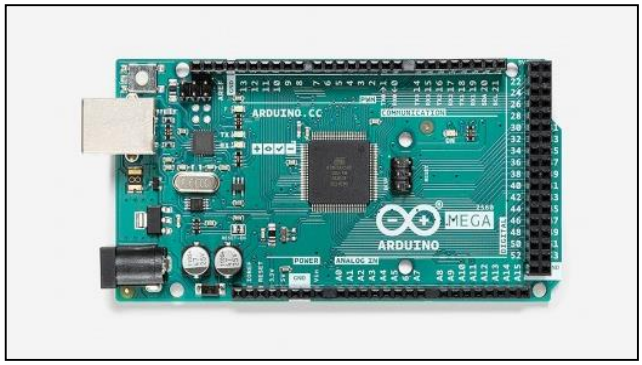


Figura 17-1. Arduino Mega 2560.
Fuente: (Arduino, 2020).

1.7.1.1. *Tipos de tarjetas de desarrollo*

Existen varias tarjetas Arduino que se diferencian unas de otras por el número de memorias y la utilidad que se le va a utilizar. Entre las más utilizadas están (Redrobán Aranda, 2018, p.16):

- Arduino uno R3
- Arduino Leonardo
- Arduino micro
- Arduino nano

- Arduino mega 2560
- Arduino Zero
- Arduino Due
- Arduino yún

1.7.2. *Raspberry PI*

Esta tarjeta de desarrollo es una computadora de tamaño reducido y de bajo costo que se puede conectar a una pantalla, teclado y un mouse, útil para proyectos electrónicos y cosas básicas que hace una computadora de escritorio como conectarse a internet, ver videos, juegos y procesamiento de texto. Posee un microcontrolador, memoria RAM y componentes esenciales para ejecutar un programa. Consta de una tarjeta SD donde se puede instalar sistemas operativos de código libre como Linux. Como se puede observar en la Figura 18-1 hay que saber de programación para usarla (Raspberry Pi).

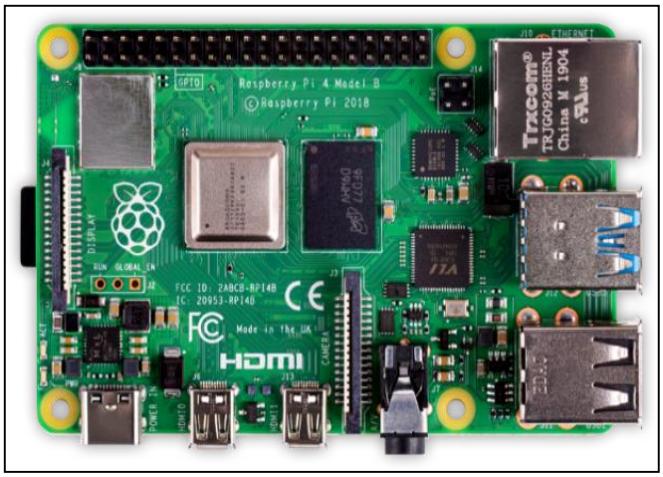


Figura 18-1. Raspberry Pi 4.

Fuente:(Raspberry Pi).

1.8. Filtro de Kalman

El filtro de Kalman fue formulado por Rudolf Kalman en 1960. Dispone de una solución recursiva al problema de filtrado lineal de datos discretos. El filtro de Kalman está constituido por un grupo de ecuaciones que se utilizan para estimar el estado del proceso, de forma que se reduzca el error cuadrático medio (Pujos Yanzapanta, 2016, p.48).

Las características primordiales del filtro de Kalman son el eficiencia y recursividad. Es recursivo porque no requiere de todos los datos para estimar el estado actual , y eficiente porque disminuye

el error en estimaciones a pesar de la presencia de variables estocásticas en el sistema (Pujos Yanzapanta, 2016, p.49).

Para implementar el filtro de Kalman se necesita de dos modelos uno del sistema y el otro de medición, los dos modelos involucran a las variables que serán objeto de estimación mediante una etapa de predicción y de corrección (Pujos Yanzapanta, 2016, p.49).

Entre las aplicaciones que tiene el filtro de Kalman están las de navegación y seguimiento, en la cual el filtro de Kalman ha tenido un buen desempeño, también resulta muy útil en sistemas de tiempo real. Siendo una poderosa herramienta para estimar estados pasados, presentes y futuros, e inclusive en circunstancias en las que la naturaleza precisa del modelo del sistema resulte desconocida (Pujos Yanzapanta, 2016, p.49; Control David Fernando Pozo Espín, 2010, p.23).

1.8.1. *Filtro del Kalman discreto*

Es aplicado a un sistema con comportamiento lineal y se lo puede expresar mediante matrices de estado (Pujos Yanzapanta, 2016, p.49).

1.8.1.1. *Modelo del sistema*

Especifica la evolución en el tiempo del proceso a ser estimado, a través de un vector de estado $\boldsymbol{x_k}$, se determina en función del estado anterior $\boldsymbol{x_{k-1}}$. Además, el proceso está sometido a una señal de ruido $\boldsymbol{w_{k-1}}$ que incurre en el valor de $\boldsymbol{x_k}$. Complementariamente, puede existir una señal de control $\boldsymbol{u_{k-1}}$, presente en procesos en los que se quiere estimar y controlar cierta magnitud física. Todo lo anterior se constituye en un sistema dinámico estocástico lineal (Pujos Yanzapanta, 2016, p.49), que se puede expresar de la siguiente manera:

$$\boldsymbol{x_k} = AX_{K-1} + BU_{K-1} + W_{K-1} \quad \textbf{(14)}$$

Las dimensiones en cada término son:

$\boldsymbol{x_k}, \boldsymbol{x_{k-1}}, \boldsymbol{w_{k-1}}$: n*1
$\boldsymbol{A}$: n*n
$\boldsymbol{B}$: n*l
$\boldsymbol{u_{k-1}}$: l*1

Siendo la matriz $\boldsymbol{A}$ de transición que relaciona el estado previo con el estado actual, y la matriz $\boldsymbol{B}$ relaciona la señal de control con $\boldsymbol{x_k}$ (Pujos Yanzapanta, 2016, p.49).

1.8.1.2. *Modelo de medición*

Relaciona las mediciones tomadas del proceso con las variables de estado del mismo. Las medidas $\boldsymbol{z_K}$, se relacionan con el vector de estados mediante un modelo lineal, mediante la matriz $\boldsymbol{H}$, y que incorpora a su vez un ruido $\boldsymbol{v_K}$ (Pujos Yanzapanta, 2016, p.50).

$$\boldsymbol{z_k} = Hx_k + v_k \quad \textbf{(15)}$$

Las dimensiones en cada termino es:

$\boldsymbol{z_k}, \boldsymbol{v_k}$: m*1
$\boldsymbol{H}$: m*n

El ruido del proceso $\boldsymbol{w_k}$ y ruido de la medición $\boldsymbol{v_k}$, son señales de ruido blanco, autónomos uno de los otros y con una distribución normal de probabilidad (Pujos Yanzapanta, 2016, p.50).

$$p(w) \sim N(0, Q) \quad \textbf{(16)}$$
$$p(v) \sim N(0, R) \quad \textbf{(17)}$$

$\boldsymbol{Q}$ y $\boldsymbol{R}$ son las covarianzas asociadas al proceso y medición, respectivamente, son las matrices diagonales a causa de la independencia de ruidos (Pujos Yanzapanta, 2016, p.50).

El filtro de Kalman maneja una estimación "a priori" del estado $\boldsymbol{\hat{x}_{\bar{k}}}$ que proyecta al estado justo antes de la estimación $\boldsymbol{\hat{x}_k}$, designado estimación "a posteori" y que emplea la medida $\boldsymbol{z_k}$. Congruentemente, debido a estas estimaciones, se definen errores "a priori" y "a posteriori" respecto al valor real del estado $\boldsymbol{x_k}$ (Pujos Yanzapanta, 2016, p.50).De esta forma:

$$\boldsymbol{e_{\bar{k}}} = x_k - \hat{x}_{\bar{k}} \quad \textbf{(18)}$$
$$\boldsymbol{e_k} = x_k - \hat{x}_k \quad \textbf{(19)}$$

y al mismo tiempo, la covarianza del error "a priori" $\boldsymbol{P_{\bar{K}}}$ y "a posteriori" $\boldsymbol{P_k}$ (Pujos Yanzapanta, 2016, p.50) equivalen a:

$$\boldsymbol{P_{\bar{K}}} = E[e_k^- \; e_k^{-T}] \quad \textbf{(20)}$$

$$\boldsymbol{P_k} = E[e_k \; e_k^T] \quad \textbf{(21)}$$

1.8.1.3. *Algoritmo*

Para el algoritmo del filtro de Kalman se divide en dos grupos de ecuaciones las primeras son las ecuaciones que se actualizan en el tiempo o llamadas de predicción y las segundas son las ecuaciones de actualización mediante observaciones o llamadas de corrección (POZO ESPÍN, 2010, p.26; Welch y Bishop, 2006).

Las ecuaciones de predicción son las encomendadas de obtener la estimación a priori de la matriz de covarianza del error, al igual que el estado actual en el tiempo $\boldsymbol{k}$, en base al estado anterior en el tiempo $\boldsymbol{k-1}$.

Las ecuaciones de corrección realizan una retroalimentación, en otras palabras, la incorporación de nuevas mediciones en la estimación a priori del estado, con el objetivo de una estimación a posteriori mejorada (POZO ESPÍN, 2010, p.26; Welch y Bishop, 2006).

Para conseguir la estimación final del estado, es esencial la etapa de predicción y la de corrección en el algoritmo del filtro de Kalman (Pozo Espín 2010, p.26). El Ciclo discreto del filtro de Kalman se muestra en la Figura 19-1:

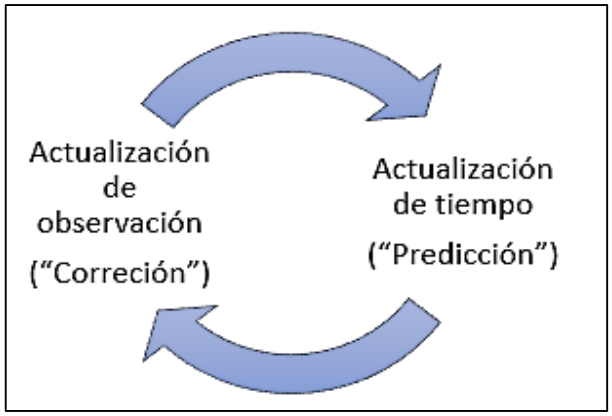


Figura 19-1. Ciclo discreto del filtro de Kalman.

Fuente: (Welch y Bishop, 2006).

1.8.2. *Etapas*

Las etapas que componen el filtro de Kalman son la de predicción y corrección (Pujos Yanzapanta, 2016, p.50).

1.8.2.1. *Etapa de predicción*

Consiste en ecuaciones que proyectan el estado al instante k, emplean los valores de la estimación del estado y covarianza del error provenientes del instante anterior $\boldsymbol{k-1}$. Esta es la etapa en donde se calcula la estimación "a priori" del estado y de la covarianza del error (Pujos Yanzapanta, 2016, p.51), mediante las siguientes ecuaciones:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = A\hat{x}_{k-1} + Bu_{k-1} \quad \textbf{(22)}$$

$$\boldsymbol{P}_{\boldsymbol{k}}^{-} = AP_{k-1}\,A^{T} + Q \quad \textbf{(23)}$$

1.8.2.2. *Etapas de corrección*

Usa ecuaciones para la actualización de la medición (Pujos Yanzapanta, 2016, p.51), siendo estas:

$$\boldsymbol{K}_{\boldsymbol{k}} = P_k^{-}H^{T}(HP_k^{-}H^{T} + R)^{-1} \quad \textbf{(24)}$$

$$\widehat{\boldsymbol{x}}_{\boldsymbol{k}} = \hat{x}_k^{-} + K_k(z_k - \widehat{Hx}_k^{-}) \quad \textbf{(25)}$$

$$\boldsymbol{P}_{\boldsymbol{k}} = (I - K_kH)P_k^{-} \quad \textbf{(26)}$$

$\boldsymbol{K}_{\boldsymbol{k}}$ es la ganancia del filtro de Kalman, siendo una matriz de n*m cuya función es reducir la covarianza del error "a posteori" y luego es utilizada en la estimación del estado $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$. Siendo $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$ la estimación del estado es una mezcla lineal entre el estado "a priori" y una diferencia compensada entre la medida real tomada $\boldsymbol{z}_{\boldsymbol{k}}$ y su estimación $\boldsymbol{H}\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$. A la diferencia $(\boldsymbol{z}_{\boldsymbol{k}} - \boldsymbol{H}\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-})$ se entiende como innovación o residuo, figura la diferencia entre la medida prevista y la real. $\boldsymbol{P}_{\boldsymbol{k}}$ viene a ser la matriz de covarianza del error en el instante $\boldsymbol{k}$ (Pujos Yanzapanta, 2016, p.51).

Posterior de cada par de actualizaciones (predicción y corrección), se puede considerar como una variable estimada con anterioridad puede ser mejorada por medio de una observación, y a la vez este nuevo estado estimado es usado para dar inicio una vez más a la etapa de predicción, y as í es como el filtro de Kalman cumple con su característica de recursividad (Pozo, 2010, p.28). En la Figura 20-1 se observa el algoritmo completo del filtro de Kalman.

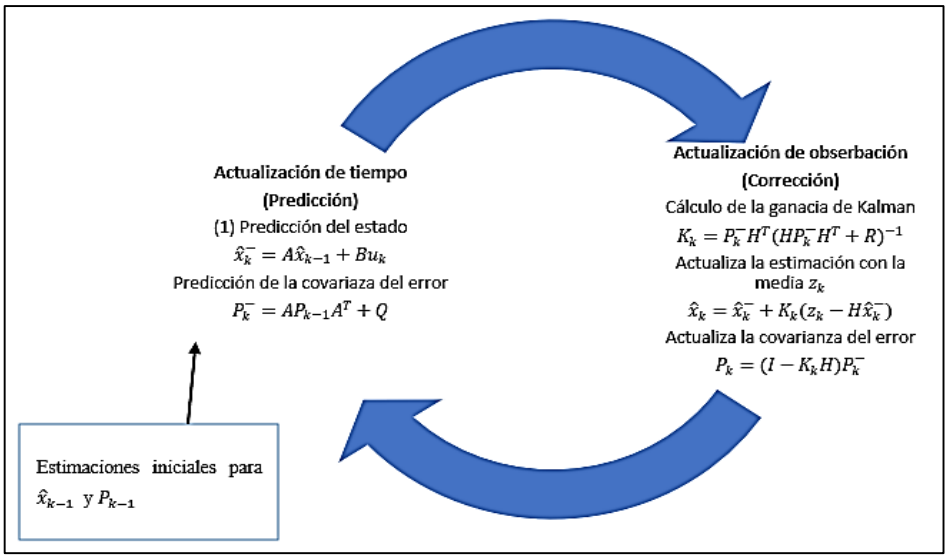


Figura 20-1. Funcionamiento completo del filtro de Kalman.

Fuente: (Welch y Bishop, 2006).

1.8.3. *Utilización del filtro de Kalman en los sensores*

Una de las mayores aplicaciones que se le da al filtro de Kalman es en sistemas de navegación y posición, donde se tiene sensores o sistemas de medición inercial como los acelerómetros, giroscopios, magnetómetros, etc., cada uno de esos sensores proporciona información y puede que no se tan confiable en ciertos aspectos (Pozo Espín, 2010, p.30).

Teniendo en cuanta lo anterior el filtro de Kalman lo que hace es aprovechar las mejores características de cada uno de los sensores para poder obtener una información más precisa y fiable (Pozo Espín, 2010, p.31).

1.9. Motores eléctricos

Los motores eléctricos son máquinas rotativas que transforman la energía eléctrica en energía mecánica, mediante las espiras rotativas del conductor que son guiadas mediante la fuerza magnética ejercida por el campo magnético y la corriente eléctrica (Chacha, 2016, p.3). Como en las industrias se utilizan mayormente motores de corriente alterna se detalla sus partes y componentes.

Básicamente un motor eléctrico de corriente alterna este compuesto por los siguientes elementos:

- Carcasa
- rotor

- estator
- eje del motor
- ventilador
- bornera
- aletas de refrigeración
- rodamientos

En la Figura 21-1. Se puede apreciar las partes y componentes de un motor eléctrico trifásico asíncrono jaula de ardilla (Chacha, 2016, p.4).

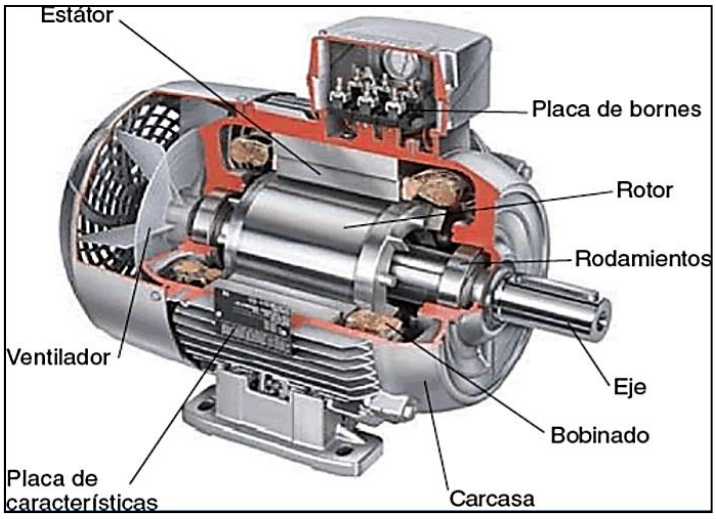


Figura 21-1. Elementos de un motor trifásico asíncrono jaula de ardilla.
Fuente: (José, 2012).

Los motores de corriente alterna son de más uso en la sociedad industrial, por eso son de fácil manejo de transmisión, distribución y transformación de la corriente alterna (José, 2012).

1.9.1. *Importancia del buen montaje*

El correcto montaje de un motor eléctrico es de alta importancia porque el desalineamiento de los ejes del motor eléctrico puede afectar mucho en la vida útil del motor, los rodamientos y el consumo energético (Kosow, 1993).

1.10. Desalineación de ejes

Se dice que la desalineación es cuando la posición de los ejes no son colineales de las máquinas acopladas y existen varios tipos de desalineación (Pedro Nelson , p.1).

1.10.1. *Tipos de desalineación*

1.10.1.1. *Desalineación en paralelo*

Es la distancia perpendicular que existe entre la línea central de un eje con respecto al otro eje, puede existir desalineación paralela vertical y paralela horizontal (Byron y Pablo, 2016, p.11). Como se muestra en la Figura 22-1 un desalineamiento paralelo de dos ejes.

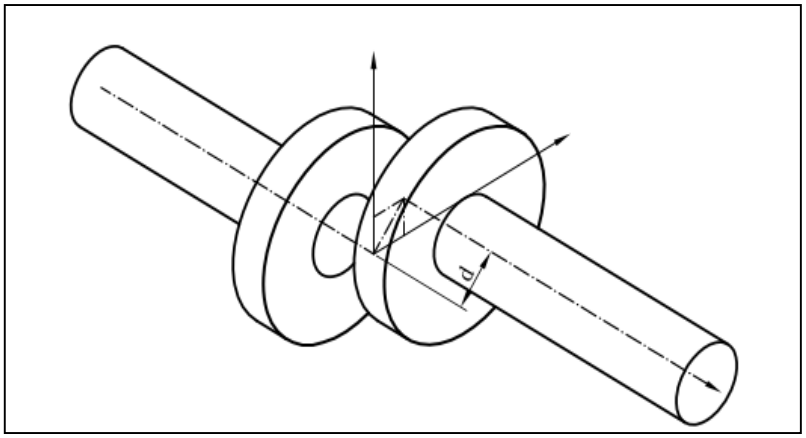


Figura 22-1. Desalineamiento paralelo.

Fuente:(Sanchez Marin et al., 2007).

1.10.1.2. *Desalineación angular*

Ocurre cuando la línea central de los ejes acoplados forma un ángulo entre sí. Puede a ver desalineación angular vertical u horizontal (Byron y Pablo, 2016, p.11). Como se muestra en la Figura 23-1 un desalineamiento angular.

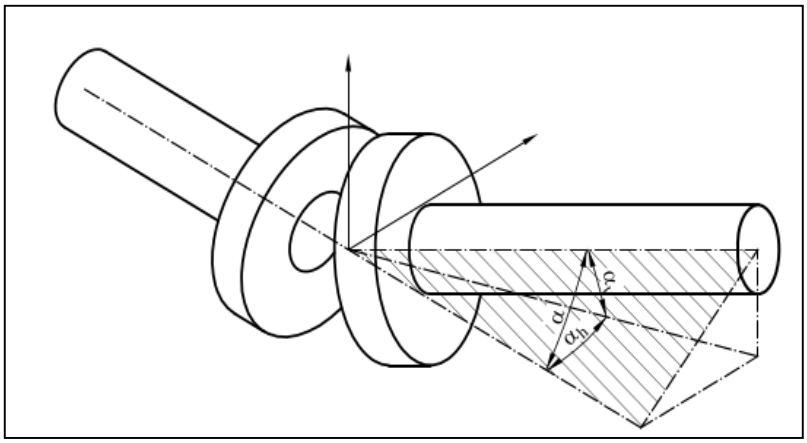


Figura 23-1. Desalineamiento angular.

Fuente: (Sanchez Marin et al., 2007).

1.10.1.3. *Desalineación combinada*

Este tipo de caso es el más común que se presentan en las industrias porque es la combinación de la desalineación paralela y angular (Byron y Pablo, 2016, p.11). Como se muestra en la Figura 24-1 un desalineamiento combinado de dos ejes.

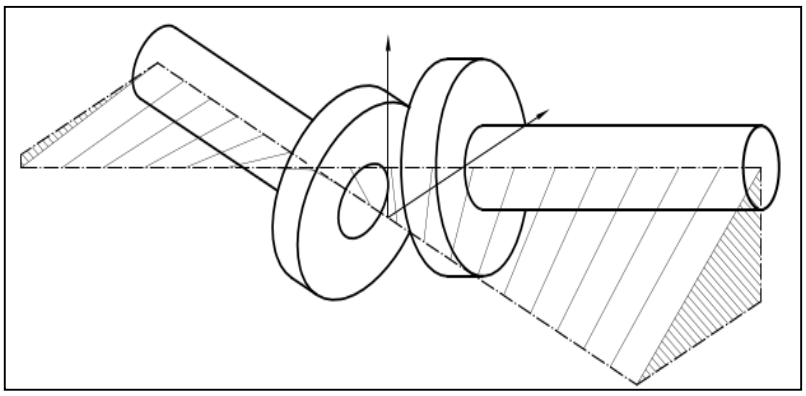


Figura 24-1. Desalineamiento combinado.

Fuente: (Sanchez Marin et al., 2007).

1.10.2. *Causas de desalineación*

Algunas de las causas para que se produzca la desalineación de máquinas acopladas según (Chacha, 2016, p.24) son:

- El ensamblado impreciso de los componentes.
- La posición se altera después de a ver montado la maquina.
- El frente del acople no está perpendicular al eje de la flecha.
- Distorsiones debido a fuerzas en tuberías.

1.10.3. *Síntomas del desalineamiento de ejes*

Según (CENTRO NACIONAL DE CAPACITACIÓN DE CELAYA (CENAC)):

- Falla temprana de los rodamientos, sello mecánico y acople
- Altas temperaturas de operación
- Un consumo eminente de energía
- Calentamiento del acople
- Aumento de vibración
- Los tornillos flojos en el acople o en la base de la maquina

- Calzas flojas.

1.10.4. *Alineación de ejes*

Es el proceso de conseguir la colinealidad entre las líneas centrales de los ejes de las maquinas acopladas, tanto en paralelismo y perpendicularidad y que estén dentro de las tolerancias admisibles (Centro Nacional de Capacitación de Celaya (CENAC)).

1.10.5. *Métodos de alineación*

De acuerdo con (Sanchez Marin et al., 2007), existen dos métodos de alienación

1.10.5.1. *Sensoriales (Rudimnetarios)*

Utilizan los sentidos de la vista y tacto. Este método es rudimentario y de menor precisión, para la desalineación paralela puede medir utilizado una regla, colocándola en la cara extrema del acoplamiento y paralela a uno de los ejes del acoplamiento (Sanchez Marin et al., 2007) como se puede observar en la Figura 25-1:

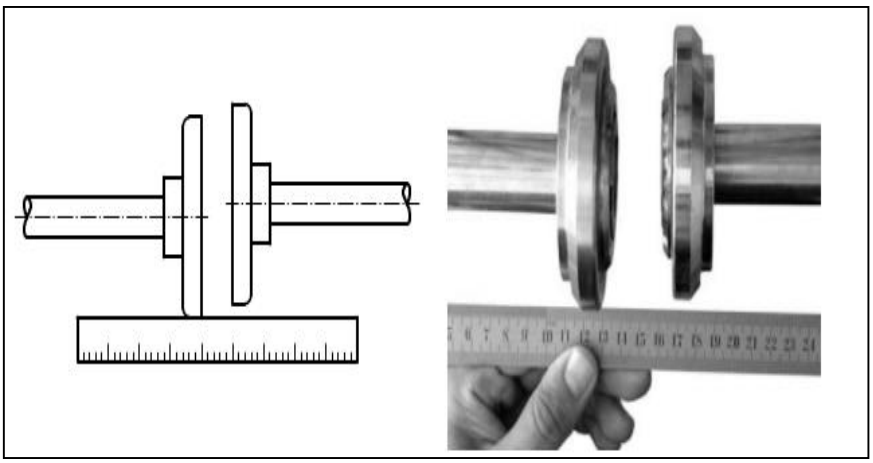


Figura 25-1. Método utilizando una regla para medir la desalineación paralela.
Fuente: (Sanchez Marin et al., 2007).

Para la desalineación angular, se puede medir con un micrómetro la división de las caras frontales externas de ambos platos en ambos lados (Sanchez Marin et al., 2007) como se muestra en la Figura 26-1.

Estos procedimientos para medir la desalineación sirven como comprobación porque son imprecisos (Sanchez Marin et al., 2007).

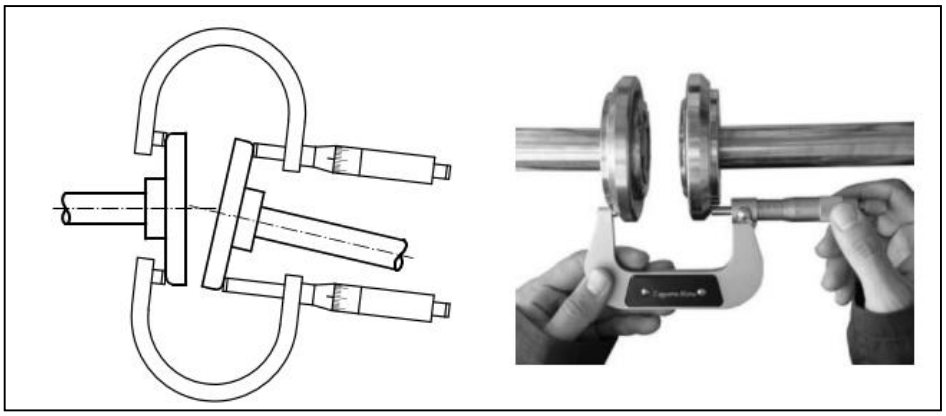


Figura 26-1. Método utilizando un micrómetro para medir la desalineación angular.
Fuente: (Sanchez Marin et al., 2007).

1.10.6. *Instrumentales*

Este método utiliza instrumentos de medida se distribuye en dos, alineación mediante relojes palpadores y alineación con equipo laser (Hernández Dávila et al., 2020).

1.10.6.1. *Método de los relojes comparadores*

Se utiliza un instrumento de medida mecánico, conformado de un reloj comparador que se sujeta fuertemente en la manzana de un acople, luego se lo desliza sobre el acople opuesto, se realiza una vuelta completa de 360 grados y cado 90 grados se hacen registros. Existen relojes comparadores que se mueven a lo largo de una dirección radial (relación al comparador) y otros en los que la dirección es axial. La división de la aguja es de 100 divisiones, siendo una resolución de 0.01 mm (10μm) (Lagla y Lanche, 2016, p.13; Sanchez Marin et al., 2007). Como se puede observar en la Figura 27-1 el método de alineación mediante relojes comparadores:

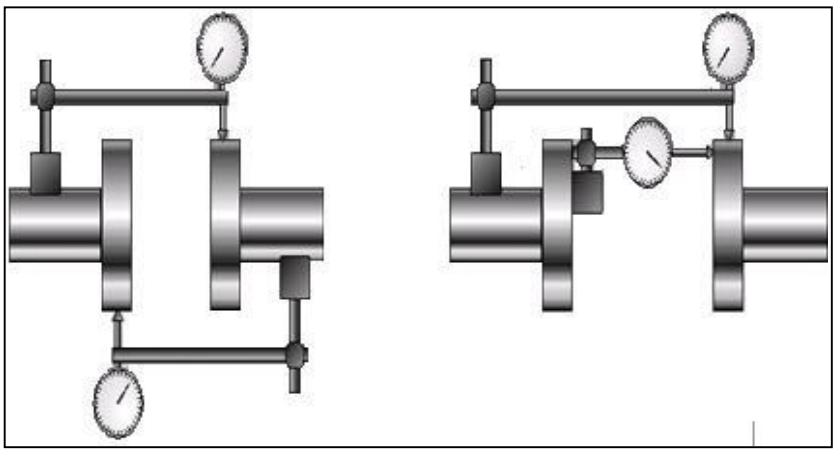


Figura 27-1. Método de alineación mediante relojes comparadores.
Fuente: (Lagla y Lanche, 2016, p.13).

1.10.6.2. *Método de alineación con equipo laser*

Es el método más rápido y preciso, aunque el equipo es costoso, el método de medida de la desalineación depende del analizador y del programa informático que se utiliza. El programa se basa en cuatro medidas dos horizontales y dos verticales de forma parecida al método de los relojes comparadores, y la diferencia radica en la utilización de equipo laser que poseen inclinómetros que ofrecen información más exacta de la posición de los ejes (Hernández Dávila et al., 2020; Sanchez Marin et al., 2007). Como se muestra en la Figura 28-1 se muestra una imagen del método de alineación láser.

Figura 28-1. Método de alineación láser.

Fuente: (Lagla y Lanche, 2016, p.13)

CAPÍTULO II

2. MARCO METODOLÓGICO

Para el desarrollo de este trabajo experimental se trabajó con un sensor de tres ejes que es un acelerómetro de tipo ADXL 335 y una IMU (sistema de medición inercial) BWT901CL de 9 ejes con conexión bluetooth, con el fin de saber la posición del eje de un motor eléctrico. La obtención de la curva característica se obtuvo a través de los datos que proporciona la IMU, el software WitMotion Shenzhen del propio sensor y con el software Matlab. A continuación, se detalla de manera más específica cada uno de los sensores, microcontroladores y software que se utilizaron para este trabajo experimental.

2.1. Selección del sensor

Para la selección del sensor se tomó varios criterios como datos que muestra el sensor, tamaño, capacidad para medir parámetros eléctricos, frecuencia de trabajo máxima, necesidad de alimentación externa, rango de temperaturas soportadas, sensibilidad y precisión, y se seleccionó como primer sensor un acelerómetro de tres ejes ADXL 335, por su tamaño y fácil programación con Arduino, se notó que este sensor solo da información de la aceleración y ángulos de inclinación y se necesita de más información para poder hacer una gráfica de la posición de un eje, así que se tomó otro sensor un IMU (sistema de medición inercial) de 9 ejes que es más completo porque consta de tres sensores un acelerómetro, giroscopio y un magnetómetro, incluido su propio microcontrolador, además de contar con su propio software para su manejo, con este sensor ya se pudo hacer la gráfica de la posición de un eje.

A continuación, se describe de manera más detallada cada sensor.

2.1.1. *Acelerómetro ADXL335*

El ADXL335 es un micrómetro de aceleración de 3 ejes pequeño, delgado y de baja potencia, con salidas de voltaje condicionado por señales. Mide la aceleración con un rango mínimo de escala completa de ± 3G.

Puede medir la aceleración estática de la gravedad en aplicaciones de detección de inclinación, así como la aceleración dinámica resultante del movimiento, choque o vibración (Analog Devices ADXL335, 2017, p. 1).

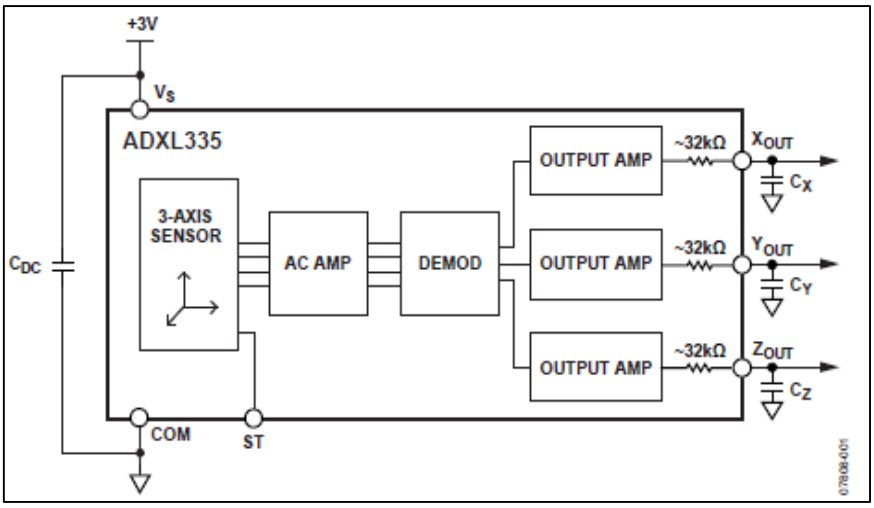


Figura 1-2. Diagrama de bloques funcional.

Fuente: (Analog Devices ADXL335, 2017).

En la Figura 1-2 se muestra el diagrama de bloques del acelerómetro ADXL335.

2.1.1.1. *Características técnicas*

Tabla 1-2: Características técnicas del ADXL335.

Descripción	Unidad / Condición
Dimensiones	Base: 4mm
	Ancho: 4 mm
	Altura: 1.45 mm
Intensidad	350 µA
Voltaje de alimentación	1.8V a 3.6V
Resistencia al choque	10 Kg
Estabilidad de temperatura	Excelente

Fuente: (Analog Devices ADXL335, 2017).

Realizado por: (Del Pino, A. 2020).

2.1.1.2. *Aplicaciones*

- Para detección de movimiento e inclinación en circuitos de baja potencia.
- Dispositivos móviles.
- Protección de unidades de disco.
- Determinación de la orientación a través de la gravedad.
- Dispositivos deportivos y de salud (Analog Devices ADXL335, 2017).

2.1.1.3. *Especificaciones técnicas*

Tabla 2-2: Especificaciones técnicas.

Parámetro	Condiciones	Mín	Prom	Máx	Unidad
Entrada del sensor	Cada eje				
Rango de medición		±3	±3.6		g
No linealidad			±0.3		%
Error de alineación de paquete			±1		Grados
Error de alineación de interacción	% de escala completa		±0.1		Grados
Eje cruzado Sensibilidad [1]			±1		%
SENSIBILIDAD (RATIOMÉTRICA) [1]	Cada eje				
Sensibilidad en XOUT, YOUT, ZOUT	VS = 3 V	270	300	300	mV/g
Cambio de sensibilidad debido a la temperatura [3]	VS = 3 V		±0.01		%/°C
CERO g NIVEL DE BIAS (RATIOMÉTRICO)					
0 g Voltaje en XOUT, YOUT	VS = 3 V	1.35	1.5		v
0 g Voltaje en ZOUT	VS = 3 V	1.2	1.5	1.65	v
0 g Compensación vs. Temperatura			±1	1.8	mg/ºc
RENDIMIENTO DE RUIDO					
Densidad de ruido XOUT, YOUT			150		µg/√Hz rms
Densidad de ruido ZOUT			300		µg/√Hz rms
RESPUESTA DE FRECUENCIA [4]					
Bandwidth XOUT, YOUT[5]	Sin filtro externo		1600		Hz
Ancho de banda ZOUT [5]	Sin filtro externo		550		Hz
Tolerancia RFILT			32 ± 15%		kΩ
Frecuencia de resonancia del sensor			5.5		kHz
PRUEBA AUTOMÁTICA					
Entrada lógica baja			0.6		v
Entrada lógica alta			2.4		v
Corriente de actuación ST			60		µA
Cambio de salida en XOUT	Autocomprobación 0 a Autocomprobación 1	−150	-325	-600	mV
Cambio de salida en YOUT	Autocomprobación 0 a Autocomprobación 1	150	325	600	mV
Cambio de salida en ZOUT	Autocomprobación 0 a Autocomprobación 1	150	550	1000	mV
AMPLIFICADOR DE SALIDA					
Oscilación de salida baja	Sin carga		0.1		v
Oscilación de salida alta	Sin carga		2.8		v
FUENTE DE ALIMENTACIÓN					
Rango de voltaje operativo		1.8		3.6	v
corriente de suministro	VS = 3 V		350		µA
Tiempo de encendido [7]	Sin filtro externo		1		ms
TEMPERATURA					
Rango de temperatura de funcionamiento		-40		85	°C

Fuente: (Analog Devices ADXL335, 2017)

Realizado por: (Del Pino, A. 2020)

2.1.1.4. *Configuración y descripción de las funciones de los PINES*

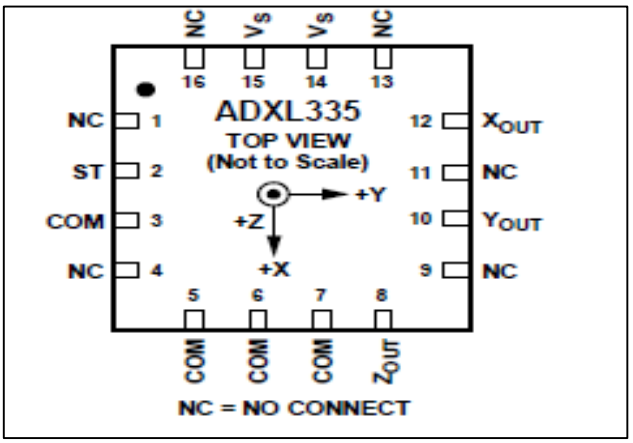


Figura 2-2. Pines de conexión del acelerómetro ADXL335.

Fuente: (Analog Devices ADXL335, 2017).

La nomenclatura de las funciones se describe en la siguiente Tabla 3-2:

Tabla 3-2: Descripción de funciones.

No. Pin	Función	Descripción
1	NC	Sin conexión
2	ST	Autotest.
3	COM	Común.
4	NC	Sin conexión
5	COM	Común.
6	COM	Común.
7	COM	Común.
8	Zout	Salida de canal Z
9	NC	Sin conexión
10	Yout	Salida de canal Y
11	NC	Sin conexión
12	Xout	X canal de salida.
13	NC	Sin conexión
14	Vs	Tensión de alimentación (1.8 V to 3.6 V).
15	Vs	Tensión de alimentación (1.8 V to 3.6 V).
16	NC	Sin conexión
EP	almohadilla expuesta	No está conectado internamente.

Fuente: (Analog Devices ADXL335, 2017)

Realizado por: (Del Pino, A. 2020)

Los pines sin conexión (NC) no están conectados internamente y pueden conectarse a los pines común.

2.1.1.5. *Principio de funcionamiento*

El acelerómetro ADXL335 utiliza una estructura única para detectar los ejes X, Y y Z, como resultado, las direcciones de detección de estos tres ejes son altamente ortogonales y tienen poca sensibilidad en el eje transversal. La desalineación mecánica de la matriz del sensor es la principal fuente de sensibilidad del eje transversal (Analog Devices ADXL335, 2017).

El voltaje de alimentación (Vs) para el acelerómetro es de 3V, aunque también puede funcionar con un Vs=1.8V, hasta Vs=3.6V. Sin embargo, se debe tener en consideración que algunos parámetros de rendimiento cambian a medida que varía el Vs. La salida de este acelerómetro es radiométrica, por lo que su sensibilidad de salida varía proporcionalmente al Vs. Con un Vs=3.6V la sensibilidad de salida es de 360mV/g, mientras que cuando el Vs=2V, la sensibilidad de salida es típicamente de 195mV/g. La salida de polarización es igual a Vs/2 para todos los Vs. La corriente del suministro Is disminuye si el voltaje de suministro disminuye o aumenta a medida que el Vs aumenta, es decir si se tiene un Vs= 3.6V, la Is= 375 μA, y si se tiene un Vs=2V, la Is= 200 μA (Analog Devices ADXL335, 2017).

2.1.2. *Arduino Mega 2560*

La Arduino Mega 2560 es una placa electrónica basada en el microcontrolador AT mega 2560, está conformada por 54 E/S (entradas y salidas) digitales, dentro de las cuales 15 salidas se pueden utilizar para la modulación de ancho de pulso (PWM). Además, cuenta con 4 puertos seriales por hadware, un cristal de 16 MHz, una conexión USB, un conector de alimentación un conector ICSP para la programación y un botón de reinicio. El Mega 2560 es una actualización que sustituye al Arduino mega. Las partes principales se representan en la siguiente figura:

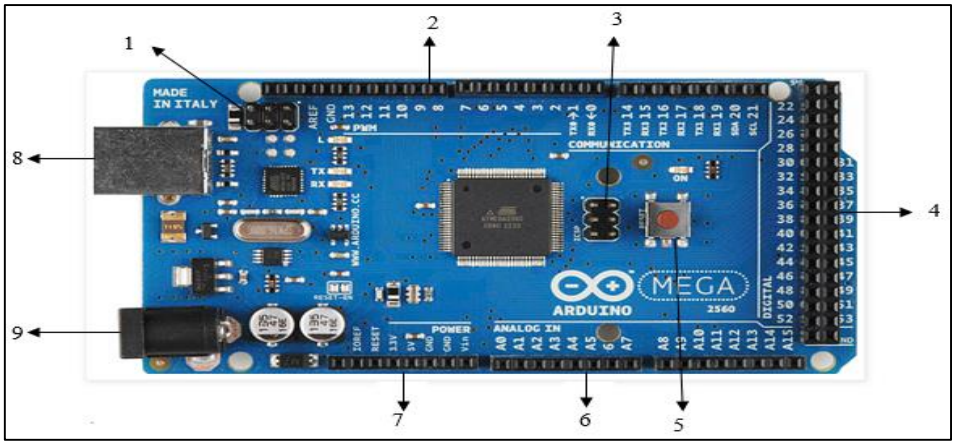


Figura 3-2. Partes principales de la placa Arduino 2560.

Fuente: (Manuel Delgado, 2016).

1) ICSP para 16U2

2) Pines digitales

3) ICSP

4) Pines digitales

5) Botón de reseteo

6) Pines de potencia

7) Jack de alimentación

8) Puerto USB

2.1.3. *Características técnicas*

Tabla 4-2: Características técnicas de la placa Arduino 2560.

Parámetros	Unidades
Tensión de trabajo	5V
Tensión de alimentación (recomendado)	7-12V
Tensión de alimentación (mínimo y máximo)	6-20V
Pines Digitales E/S	54 (15 para salidas PWM)
Pines de entradas analógicas	16
DC Corriente por Pin I/O	20 mA
DC Corriente por Pin 3.3V	50 mA
Memoria	256 kb
Velocidad del reloj	16 MHz
Largo	101.52 mm
Ancho	53.3 mm
Peso	37 g

Fuente: (Manuel Delgado, 2016).
Realizado por: (Del Pino, A. 2020).

2.1.4. *Programación*

La placa Arduino Mega 2560 se programa con el software libre de Arduino versión 1.8.13, cuyo lenguaje de programación está basado en C/C++ es textual y de fácil interpretación; este software es compatible con diferentes sistemas operativos como Windows, Linux o Mac. Esta placa viene precargada con un gestor de arranque (bootloader) el cual permite insertar un nuevo código sin la necesidad de un hadware externo.

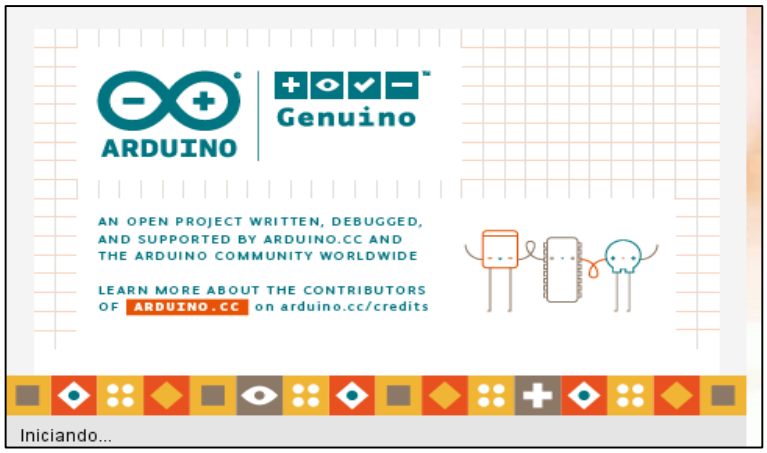


Figura 4-2. Interfaz del software Arduino.

Fuente: (Del Pino, A. 2020).

La placa Arduino 2560 vienen preprogramadas con un cargador para el arranque que permite cargar un nuevo código sin la necesidad de instalar un hardware externo. Además, poseen un poli fusible para proteger a los puertos USB de los ordenadores como cortocircuitos y sobreintensidades.

2.1.4.1. *Pines de entrada*

- Vin: Es el voltaje de entrada a la placa cuando se utiliza una fuente de alimentación externa.
- 5V: Es un pin de salida regulado.
- 3,3V: Es un voltaje de entrada generado por el regulador de placa.
- GND: Pin de descarga a tierra.
- IOREF: Pin que proporciona la referencia de tensión con la que funciona el microcontrolador.

2.2. Sensor giroscópico WT901BLECL

Un sensor giroscopio es un dispositivo que sirve para medir, mantener o cambiar la orientación de un sistema de referencia. Este sensor consta con un giroscopio de alta presión, un acelerómetro, un sensor geomagnético MPU9250 y microprocesadores de alto rendimiento. Este sensor es de 9 ejes (ángulo de 3 ejes, velocidad y aceleración angular y campo magnético).

El procesador es de tipo Cortex-M0 de alto rendimiento y funciona con una frecuencia de hasta 50MHz, además del bajo consumo de energía y el alto rendimiento, la transmisión es de tipo inalámbrica, para lo cual cuenta con bluetooth, con una estabilidad de transmisión y una distancia superior a 50 metros (Model y Imu, 2019).

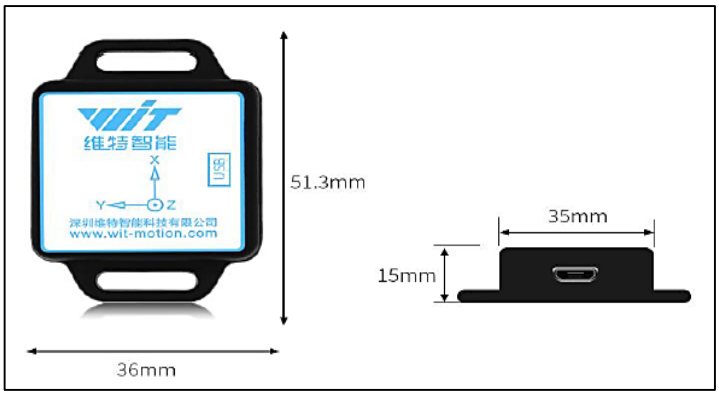


Figura 5-2. Dimensiones del sensor WT901BLECL.

Fuente: (Model y Imu, 2019).

2.2.1. *Características técnicas*

Tabla 5-2: Características técnicas.

Parámetro	Unidad
Voltaje de alimentación	3.3V – 5 V
Intensidad de trabajo	15mA
Intensidad de transmisión	150 uA
Corriente de espera	10uA-50uA
Dimensiones:	Base= 36mm
	Ancho= 15mm
	Altura= 51.3mm
Aceleración	± 16g
Velocidad angular	± 2000 ° / s
Ángulo	X Z ± 180 °
	Y ± 90 °
Frecuencia de salida	0.1Hz - 50Hz
Distancia de transmisión de Bluetooth	> 50m

Fuente: (Model y Imu, 2019).

Realizado por: (Del Pino, A. 2020).

2.3. Selección del software

2.3.1. *Arduino IDE*

Se utilizó el software Arduino (IDE) que es de código abierto en la versión 1.8.13, para poder programar el acelerómetro ADXL 335, a fin de poder ver los ángulos de inclinación de los dos acelerómetros. En la Figura 6-2 se visualiza el Software Arduino (IDE).

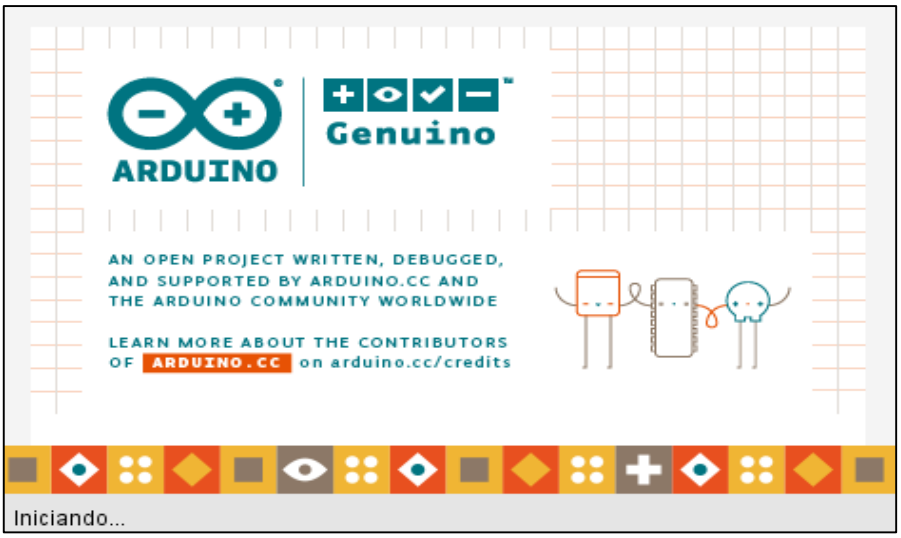


Figura 6-2. Software Arduino (IDE) versión 1.8.13.

Fuente: (Del Pino, 2020).

2.3.2. *MiniIMU*

El software que se utilizó para manejar el sensor WT901BLECL es el MiniIMU versión 5.0 de la empresa WitMotion propio del sensor, en el cual podemos calibrar el sensor y obtener los datos que genera el sensor como el tiempo, la aceleración, la velocidad angular, y los ángulos de inclinación, como se observa en la Figura 7.2:

Figura 7-2. Software MiniIMU versión 5.0.

Fuente: (Del Pino, 2020).

2.3.3. *Matlab*

El software de Matlab, cuyo imagotipo se muestra en la Figura 8-2, tiene un lenguaje de programación basado en matrices, en la cual se puede representar los datos y funciones, está

disponible para las plataformas de Unix, Windows, macOS y GNU/Linux. Se utilizó este software Matlab versión 2019 para poder recolectar los datos que generan los sensores, graficar los datos obtenidos del acelerómetro, del IMU, encontrar la ecuación de la curva característica del eje y poder también hacer una simulación del movimiento del eje.

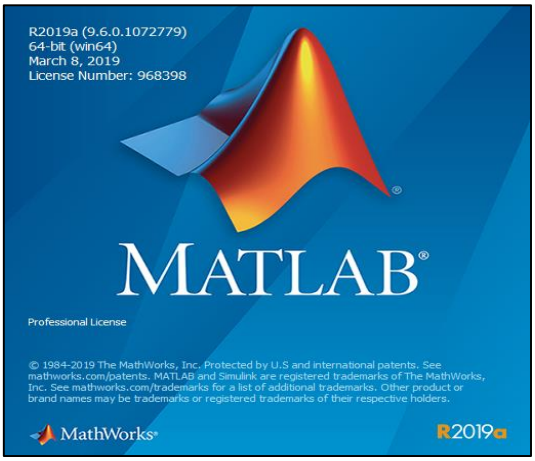


Figura 8-2. Software Matlab versión R2019a.

Fuente: (Del Pino, 2020).

2.4. Conexión entre microcontrolador y sensores

Los acelerómetros permiten medir la aceleración estática de la gravedad como la detección de la inclinación y la aceleración dinámica resultante del movimiento, los golpes o la vibración. Para el desarrollo de este trabajo se va a medir la aceleración estática, porque se quiere saber cuántos grados se está moviendo el eje para posteriormente determinar un método que identifique en qué posición se encuentra el eje en el espacio. Para realizar la conexión entre el microcontrolador y los sensores se utilizó la Arduino Mega 2560, y se realizó de la forma mostrada en la Figura 9-2:

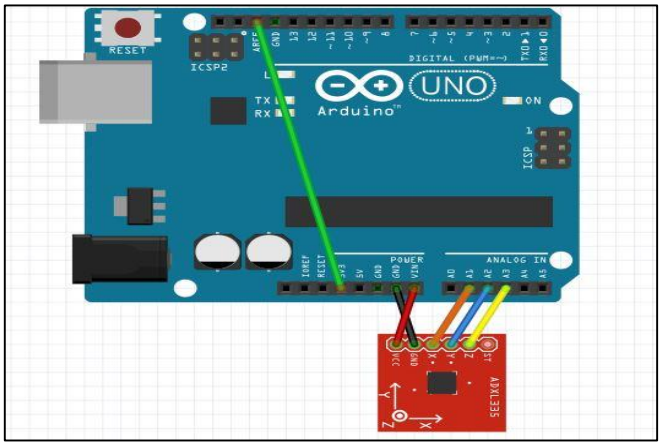


Figura 9-2. Conexión microcontrolador y sensor.

Fuente: (Analog Devices ADXL335, 2017).

2.4.1. *Programación del acelerómetro ADXL 335*

Para realizar la programación del acelerómetro ADXL 335 se utilizó el software Arduino versión 1.8.13, la cual sirve para observar la inclinación en ángulos de 0º a 360º en los ejes ***X, Y, Z***. Luego de la programación se realiza la conexión entre el ordenador y el microcontrolador, para luego recolectar los datos a través del software Matlab versión R2019a.

En las Figuras 10-2 a 15-2 se muestra la programación para el acelerómetro ADXL 335.

- Primero se declara las variables de todos los pines como entradas que van a tomar los valores de los acelerómetros.

Figura 10-2. Declaración de variables.

Fuente: (Del Pino, 2020).

Se coloca los valores máximos y mínimos que tiene el acelerómetro mientras está parado.

Figura 11-2. Valores máximos y mínimos que tiene el acelerómetro ADXL335.

Fuente: (Del Pino, 2020).

Los nombres de los valores calculados en x, y, z, x1, y1, z1.

Figura 12-2. Constantes de los acelerómetros.

Fuente: (Del Pino, 2020).

Después se inicia el puerto serial para enviar la información, el *void loop* solo empieza a tomar lecturas y leer los valores analógicos del acelerómetro.

```
void setup(){
Serial.begin(9600);
int xRead = analogRead(xPin);
int yRead = analogRead(yPin);
int zRead = analogRead(zPin);


}

void loop(){

//read the analog values from the accelerometer
int xRead = analogRead(xPin);
int yRead = analogRead(yPin);
int zRead = analogRead(zPin);
int x1Read = analogRead(x1Pin);
int y1Read = analogRead(y1Pin);
int z1Read = analogRead(z1Pin);
```

Figura 13-2. Inicio de la comunicación serial.

Fuente: (Del Pino, 2020).

Luego se convierte los valores leídos a grados de 90, -90 grados, se calcula los valores de 360 grados, se usa la función de mapeo. La función *map* funciona como una regla de 3 en donde se introduce un rango de valores, para generar un rango de valores que se necesita.

```
//convert read values to degrees -90 to 90 - Needed for atan2
int xAng = map(xRead, minVal, maxVal, -90, 90);
int yAng = map(yRead, minVal, maxVal, -90, 90);
int zAng = map(zRead, minVal, maxVal, -90, 90);

//Caculate 360deg values like so: atan2(-yAng, -zAng)
//atan2 outputs the value of -π to π (radians)
//We are then converting the radians to degrees
x = RAD_TO_DEG * (atan2(-yAng, -zAng) + PI);
y = RAD_TO_DEG * (atan2(-xAng, -zAng) + PI);
z = RAD_TO_DEG * (atan2(-yAng, -xAng) + PI);



int x1Ang = map(x1Read, minVal, maxVal, -90, 90);
int y1Ang = map(y1Read, minVal, maxVal, -90, 90);
int z1Ang = map(z1Read, minVal, maxVal, -90, 90);

//Caculate 360deg values like so: atan2(-yAng, -zAng)
//atan2 outputs the value of -π to π (radians)
//We are then converting the radians to degrees
x1 = RAD_TO_DEG * (atan2(-y1Ang, -z1Ang) + PI);
y1 = RAD_TO_DEG * (atan2(-x1Ang, -z1Ang) + PI);
z1 = RAD_TO_DEG * (atan2(-y1Ang, -x1Ang) + PI);
```

Figura 14-2. Comienzo de la función *map*.

Fuente: (Del Pino, 2020).

Finalmente se hace la salida de los datos en el *serial print*

```
//Output the caculations
Serial.print(x);
Serial.print(",");
Serial.print(y);
Serial.print(",");
Serial.print(z);
Serial.print(",");
Serial.print(x1);
Serial.print(",");
Serial.print(y1);
Serial.print(",");
Serial.println(z1);
delay(1000);//just here to slow down the serial output - Easier to read
}
```

Figura 15-2. Envió de datos con la función *serial print*.

Fuente: (Del Pino, 2020).

2.4.2. *Programación del IMU*

Por defecto ya viene programado el sensor WT901BLECL, que además se utiliza con el propio software, por este motivo solamente se programó la parte de recopilación de datos, la simulación y las gráficas en el software Matlab versión R2019a.

2.4.3. *Calibración del acelerómetro*

Antes de poner en funcionamiento al acelerómetro, primero hay que calibrar manualmente. Esto se realiza de la siguiente manera:

- Se coloca al sensor en una superficie plana y estable.
- Luego se abre el programa Arduino y se lee los valores del mapeo
- Después se coloca al acelerómetro en posición vertical con respecto al eje "X" en 90º, para ver los valores que se genera, se tiene que esperar unos 5 minutos hasta que se estabilice y se anote ese dato, luego se coloca en posición contraria para tener el valor del acelerómetro en esa posición.
- Se coloca luego en posición vertical con respecto al eje "Y" en 90º, para ver los valores que genera, luego se coloca en posición contraria, y se anota los valores que genera en esas dos posiciones
- Para el eje "Z" se coloca en posición horizontal en 180º, y luego en sentido contrario a 360º; se anota los valores que genera el acelerómetro en esas posiciones.
- Los valores anteriores se colocan en el código de mapeo
- En el serial print del código se lee los valores de los ejes X, Y, Z en ángulos ya calibrados con los valores que se generó del acelerómetro en las posiciones indicadas anteriormente.

En la Tabla 6-2 se presentan los valores de los ejes del acelerómetro:

Tabla 6-2. Valores de los ejes ***X, Y, Z*** del acelerómetro.

EJES	VALORES DEL ACELERÓMETRO	GRADOS (°)
X	412	-90
	614	90
Y	403	-90
	609	90
Z	398	180
	599	0
	398	180
	599	360

Realizado por: (Del Pino, A. 2020).

2.4.4. *Calibración del IMU*

2.4.4.1. *Calibración de la aceleración*

La calibración de la aceleración se realiza para eliminar el desplazamiento cero del acelerómetro. El sensor tendrá diferentes grados de error de fábrica, la medición será precisa después de que se realice la calibración manual.

El procedimiento para realizar esta calibración será de la siguiente manera:

- Mantener el módulo estacionario horizontalmente.
- Hacer clic en el botón de la aceleración, luego en la "Configuración" y esperar 5 segundos.
- La calibración se realiza de manera correcta si se muestra el OK.

El proceso anterior se muestra en la Figura 16-2:

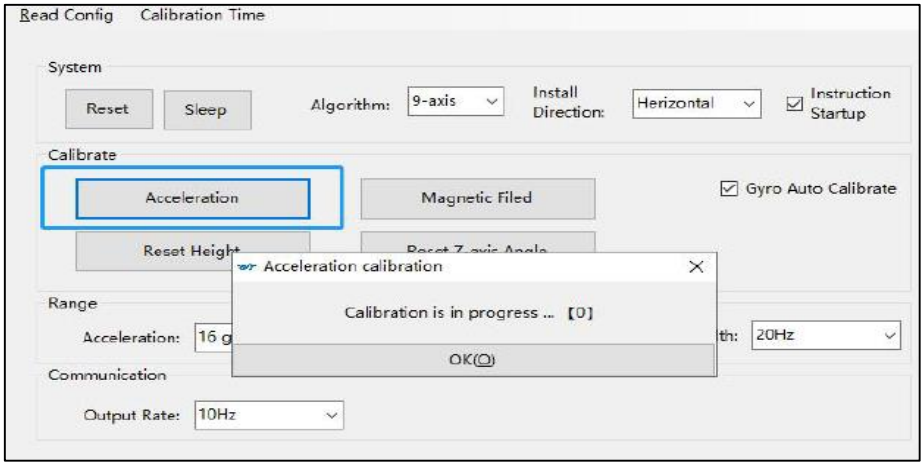


Figura 16-2. Calibración del acelerómetro.

Fuente: (Ble, 2019).

2.4.4.2. *Calibración del campo magnético*

Este tipo de calibración se utiliza para eliminar la polarización cero del sensor de campo magnético.

El sensor de campo magnético por lo general tiene un error de fabricación de cero. Si esto no se calibra, se generará un gran error de medición y afectará la precisión de la medición del ángulo del eje ***Z***.

La calibración se prepara de la siguiente manera:

- Primero se debe conectar los sensores a 20cm de distancia de materiales magnéticos, como el hierro, imanes u otros metales.
- El valor de ***H*** en el campo magnético debe ser inferior a 350º. Los parámetros de comprobación se muestran en la Figura 17-2:

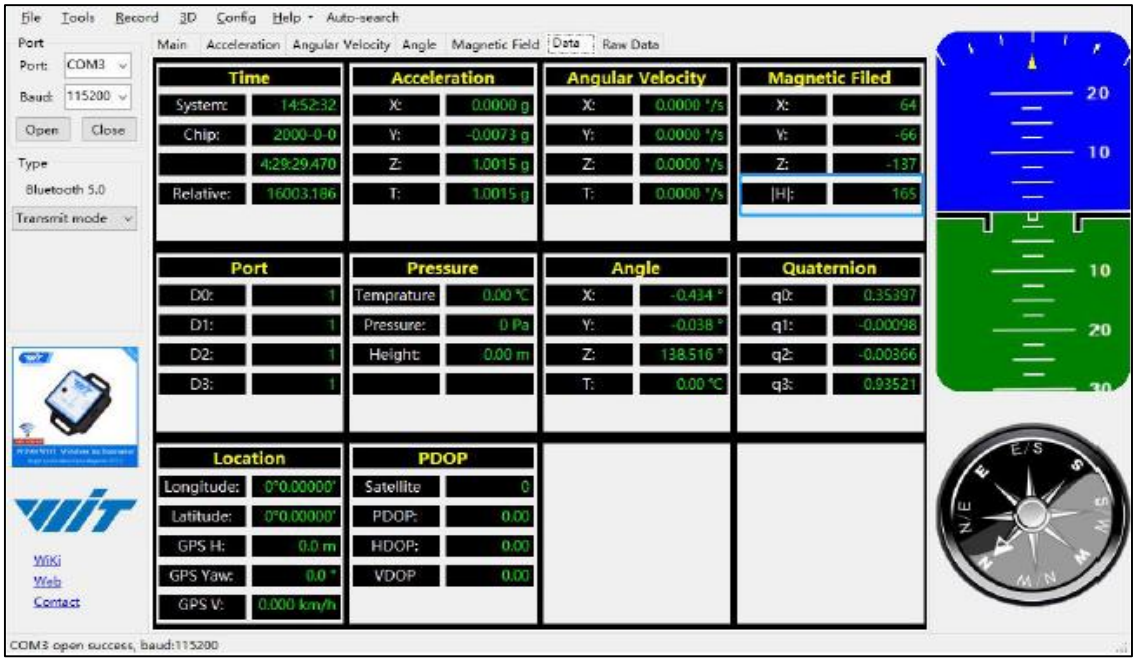


Figura 17-2. Verificación de parámetros para la calibración.

Fuente: (Ble, 2019).

La calibración se realiza de la siguiente manera:

- Se abre el menú de configuración.
- Luego se hace clic en el botón de "campo magnético" y se gira lentamente el sensor 360° alrededor de los ejes X, Y, Z, tal como se muestra en la Figura 18-2:

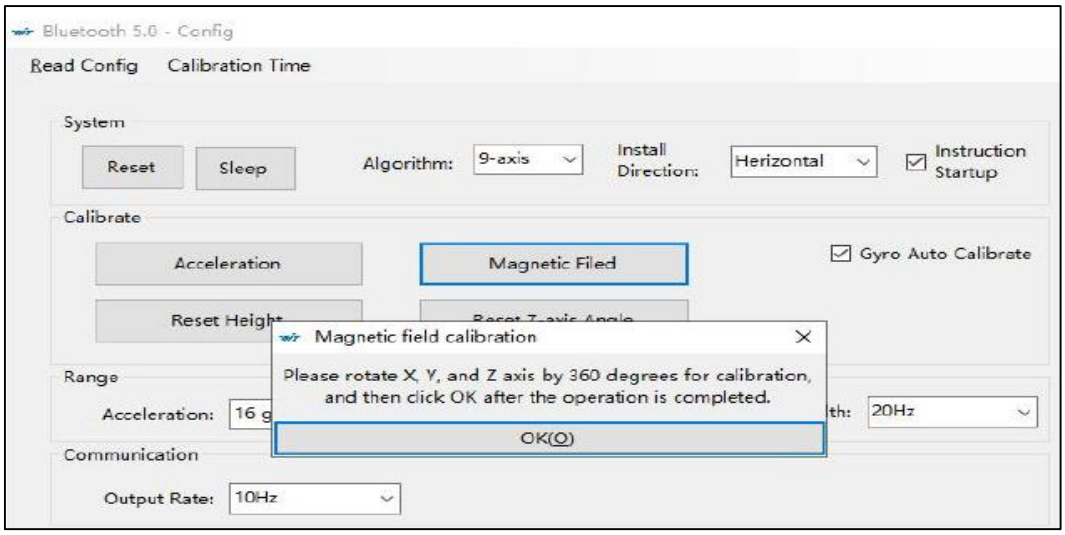


Figura 18-2. Calibración manual del acelerómetro.

Fuente: (Ble, 2019).

- Después se da clic en Aceptar una vez realizada la calibración.
- Luego se coloca el sensor horizontalmente estacionario y se coloca que el eje Y apunte hacia el norte. Esto se muestra en la Figura 19-2:

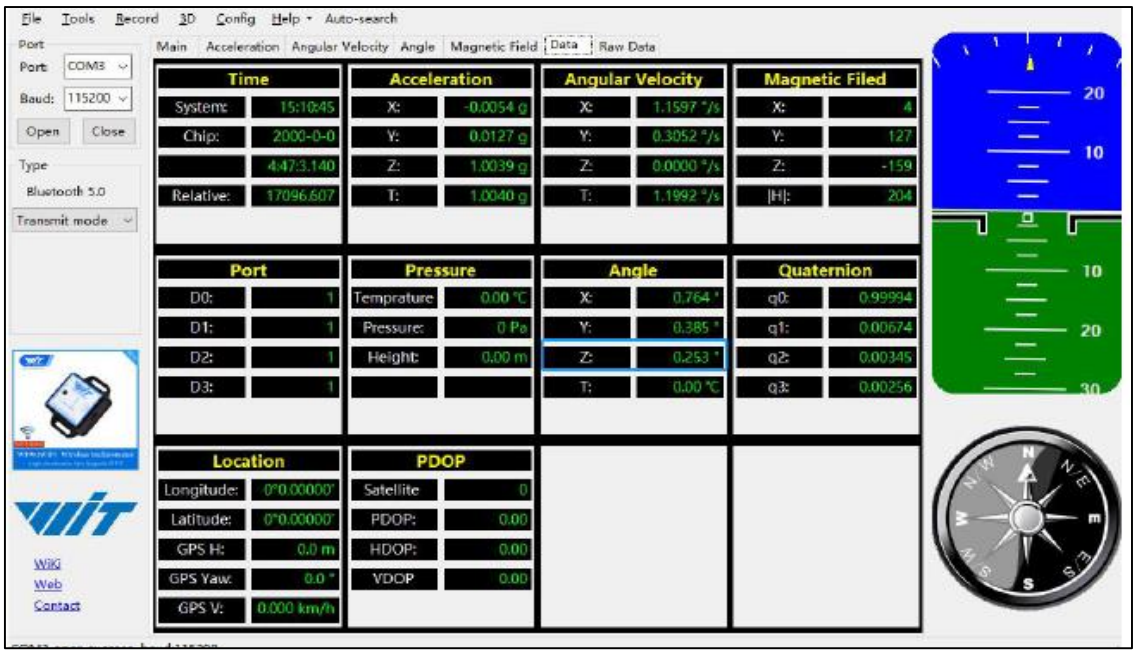


Figura 19-2. Verificación de los parámetros para la calibración.

Fuente: (Ble, 2019).

- Para finalizar se debe dar algunas vueltas al sensor, luego se debe girar 360º alrededor del eje ***Y***, después se debe girar 360° alrededor del eje ***Z***, luego se debe dar algunas vueltas al azar, y por último se debe dar clic en la opción "Finalizar" para terminar la calibración.

2.5. Cálculo de la aceleración en los ejes X, Y, Z

Las fórmulas para el cálculo de la aceleración en cada uno de los ejes son las siguientes:

Para el eje ***X***:

$$\boldsymbol{ax} = \frac{((\text{axH}<<8))|\text{axL}}{32768} * \ 16g \qquad \textbf{(27)}$$

Donde:

$\boldsymbol{ax}$**=** aceleración en ***x***

axH = X aceleración alta 8 bytes

axL = X aceleración baja 8 bytes

$\boldsymbol{g}$ = aceleración por la gravedad $9.8m/s^2$

Para el eje ***Y***:

$$\boldsymbol{ay} = \frac{((\text{ayH}<<8)|\text{ayL})}{32768} * \ 16g \qquad \textbf{(28)}$$

Donde:

$\boldsymbol{ay}$ = aceleración en ***y***

ayH = Y aceleración alta 8 bytes

ayL = Y aceleración baja 8 bytes

$\boldsymbol{g}$ = aceleración por la gravedad $9.8m/s^2$

Para el eje ***Z***:

$$\boldsymbol{az} = \frac{((\text{azH}<<8)|\text{azL})}{32768} * \ 16g \qquad \textbf{(29)}$$

Donde:

$\boldsymbol{az}$ **=** aceleración en ***z***

azH = Z aceleración alta 8 bytes

azL = Z aceleración baja 8 bytes

$\boldsymbol{g}$ = aceleración por la gravedad $9.8m/s^2$

2.6. Cálculo de la velocidad angular en los ejes X, Y, Z

Para el eje ***X***:

$$\boldsymbol{Wx} = \frac{((\text{wxH}<<8)|\text{wxL})}{32768} * \ 2000(°/\text{s}) \qquad \textbf{(30)}$$

Donde:

$\boldsymbol{Wx}$**=** velocidad angular en el eje $\boldsymbol{x}$

wxH= X velocidad angular alta 8 bytes

wxL = X velocidad angular baja 8 bytes

Para el eje $\boldsymbol{Y}$:

$$\boldsymbol{Wy} = \frac{((\text{wyH}<<8)|\text{wyL})}{32768} * \ 2000(°/\text{s}) \quad \textbf{(31)}$$

Donde:

$\boldsymbol{Wy}$**=** velocidad angular en el eje $\boldsymbol{y}$

wyH= Y velocidad angular alta 8 bytes

wyL = Y velocidad angular baja 8 bytes

Para el eje $\boldsymbol{Z}$:

$$\boldsymbol{Wz} = \frac{((\text{wzH}<<8)|\text{wzL})}{32768} * \ 2000(°/\text{s}) \quad \textbf{(32)}$$

Donde:

$\boldsymbol{Wz}$**=** velocidad angular en el eje $\boldsymbol{z}$

wzH= Z velocidad angular alta 8 bytes

wzL = Z velocidad angular baja 8 bytes

2.6.1. *Cálculo de posición*

Movimiento (eje $\boldsymbol{X}$):

$$\boldsymbol{movimiento} = \frac{((\text{giroH}<<8)|\text{giroL})}{32768} * \ 180(°) \quad \textbf{(33)}$$

Donde:

giroH= X ángulo alto 8 bytes

giroL= X ángulo bajo 8 bytes

Inclinación (eje $\boldsymbol{Y}$):

$$\boldsymbol{Inclinación} = \frac{((\text{incl H}<<8)|\text{incl L})}{32768} * \ 180(°) \quad \textbf{(34)}$$

Donde:

incl H= Y ángulo alto 8 bytes

incl L= Y angulo bajo 8 bytes

Angulo de desvío（eje $\boldsymbol{Z}$):

$$\boldsymbol{Desvío} = \frac{((\text{desvíoH}<<8)|\text{desvíoL})}{32768} * \ 180(°) \quad \textbf{(35)}$$

Donde:

desvíoH = Z ángulo alto 8 bytes

desvíoL = Z ángulo bajo 8 bytes

El orden de la secuencia de rotación de los ejes es: ***Z***, ***Y***, ***X***, mientras que el rango del ángulo de inclinación (eje ***Y***) igual a ± 90º. Cuando el ángulo de inclinación es > 90º y <90º al mismo tiempo, el ángulo de movimiento (eje ***X***) será mayor de 180 grados. Los datos se envían en formato hexadecimal, de orden descendente a ascendente, y los dos se combinan en un tipo corto de datos con signo.

2.7. Diseño

2.7.1. *Diseño de los soportes para el acelerómetro ADXL 335*

El diseño de los soportes se lo realizo mediante el software Solidworks 2020. Se realizaron 2 tipos de soportes, uno que es la base y el otro que es el soporte para los sensores y luego se los imprimió en una impresora 3D con material PLA (poliácido láctico). El proceso para los soportes se describe a continuación:

2.7.2. *Soporte para los acelerómetros*

Para realizar este tipo de soporte primero se establecen las medidas en cada una de las vistas, para observar la perspectiva isométrica del soporte del sensor ADXL335 ver ANEXO B.

Después de insertar las medidas, se realiza la proyección para obtener la isometría del soporte deseado, luego se inserta el tipo de material antes de realizar la impresión en 3D. En la Figura 20-2 se observa la isometría del soporte para los acelerómetros.

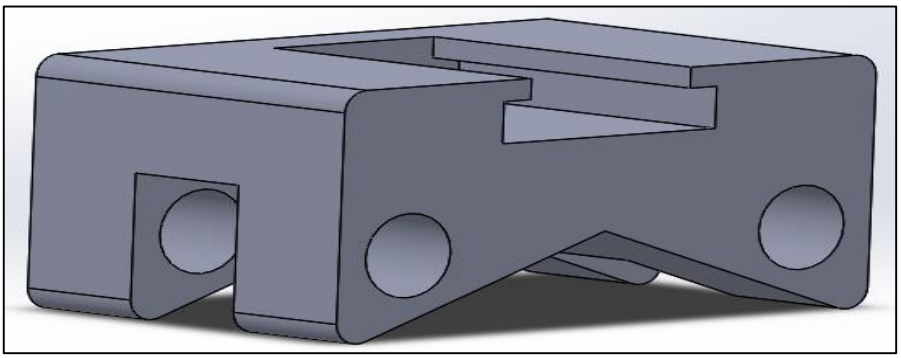


Figura 20-2. Isometría del soporte para los acelerómetros.

Realizado por: (Del Pino, A. 2020).

2.7.3. *Diseño de los soportes para el sensor WT901BLECL*

El diseño de la estructura del IMU se lo realiza mediante el software Solidworks 2020 y luego se imprimió en una impresora 3D con material PLA (poliácido láctico). Cada pieza se detalla a continuación, en los cuales se especifican el diseño de cada una.

2.7.3.1. *Argolla parte superior*

Para el diseño de la argolla se realiza en 2 partes, la primera es la parte superior, el material es de tipo PLA (Polímero de ácido pililactico), al igual que el diseño de los soportes se establecen las medidas para realizar el diseño, para ver las isometrías del ensamble ver ANEXO C.
Con las medidas establecidas se insertan en el solidworks indicando cada uno de los detalles y luego se extruye el sólido y se obtiene la isometría deseada, como se visualiza en la Figura 21-2.

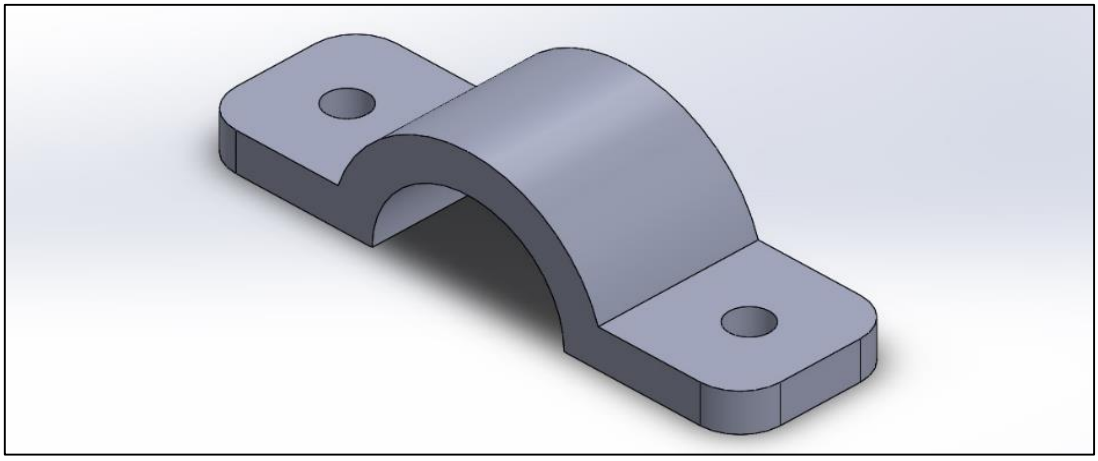


Figura 21-2. Isometría de la parte superior de la argolla.
Realizado por: (Del Pino, A. 2020).

2.7.3.2. *Argolla parte inferior*

El procedimiento para realizar la parte inferior de la argolla es el mismo que el de la parte superior, esto se simplifica realizando una simetría de cada una de las medidas anteriores, esto es porque son iguales todas las medidas en cada una de las vistas, ver ANEXO C.

Con las medidas establecidas se procede a realizar el diseño en 3D en solidworks con las operaciones de extrusión y corte, para la perforación de los agujeros y se realiza un redondeo, como se observa en la Figura 22-2.

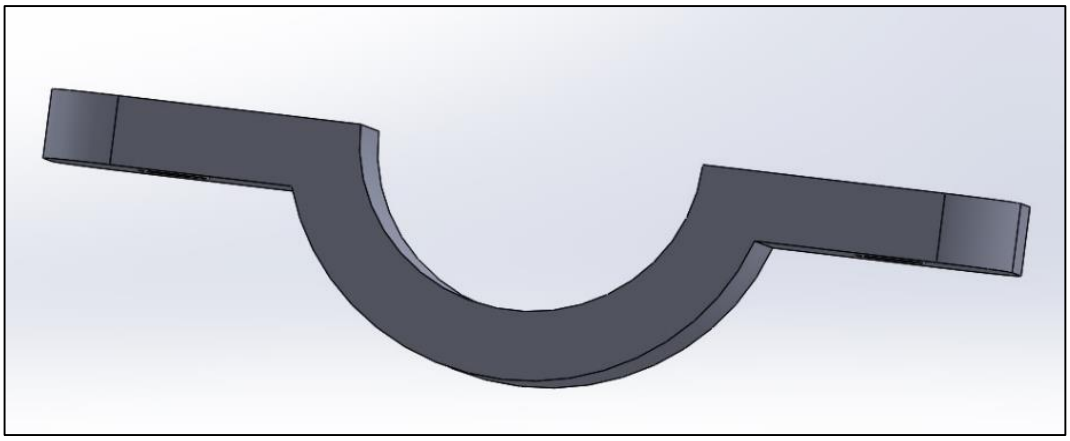


Figura 22-2. Isometría de la parte inferior de la argolla.

Realizado por: (Del Pino, A. 2020).

2.7.3.3. *Base medio*

Para realizar la base de la estructura del IMU se debe realizar de manera exacta y detallada cada una de las medidas en las 3 vistas del dibujo, ver ANEXO C.

Se procede a realizar la isometría de la base con las operaciones de extrusión, corte y redondeo en Solidworks, se puede ver en la Figura 23-2.

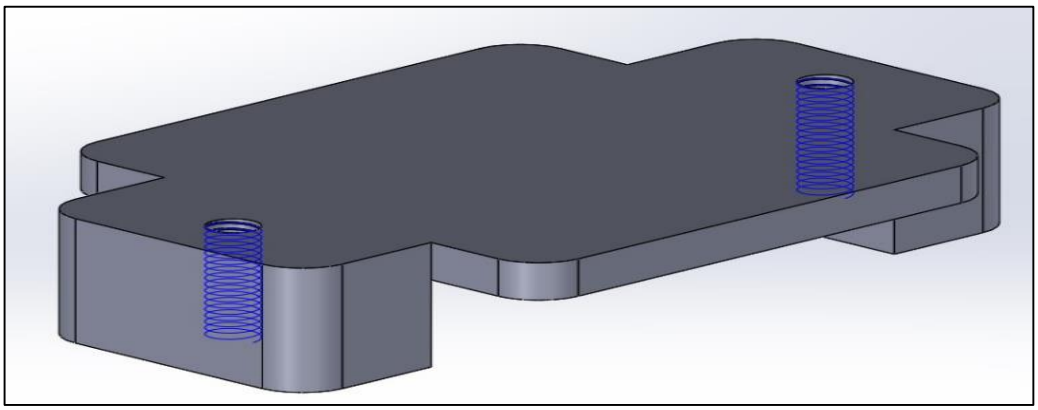


Figura 23-2. Isometría de la base para la caja.

Realizado por: (Del Pino, A. 2020).

2.7.3.4. *Caja*

El diseño de la caja se realiza de la misma manera que los elementos anteriores, detallando las medidas en cada una de las vistas para luego realizar el diseño en 3D, ver ANEXO C.

Se realiza el diseño de la isometría en Solidworks, el diseño de la caja se presenta en la Figura 24-2.

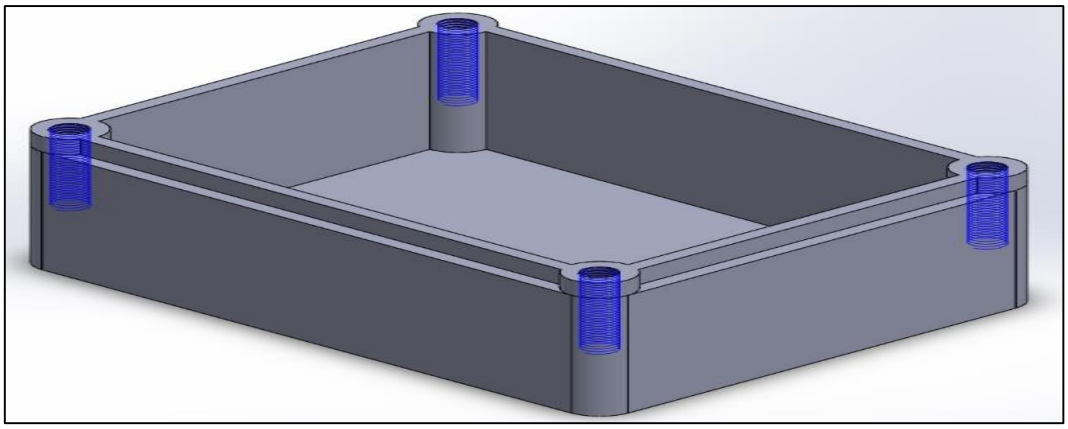


Figura 24-2: Isometría de la caja.

Realizado por: (Del Pino, A. 2020).

2.7.3.5. *Tapa de la caja*

La tapa de la caja se realiza de igual manera que los elementos anteriores, en donde se establecen las medidas en cada una de las vistas para realizar después el diseño en 3D, para ver las isometrías ver ANEXO D.

Se realiza la isometría de la tapa en el software solidworks, como se muentra en la Figura 25-2.

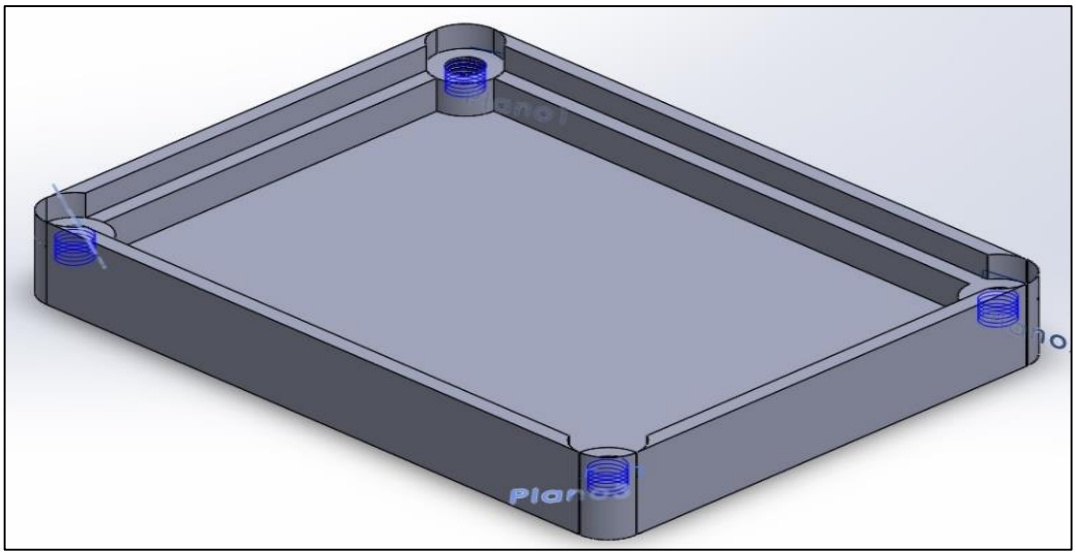


Figura 25-2. Isometría de la tapa de la caja.

Realizado por: (Del Pino, A. 2020).

Con los elementos descritos anteriormente se realiza el ensamble de cada una de las piezas para la obtención de la estructura del sensor WT901BLECL, el ensamble completo se visualiza en el ANEXO C.

2.8. Recopilación de datos

2.8.1. *Proceso para la obtención de datos del acelerómetro ADXL 335*

Para el proceso de obtención de datos para nuestra experimentación con el acelerómetro ADXL 335, se lo realizo con el software Arduino, y para la toma de datos el software Matlab versión R2019a, para poderlos visualizar en tablas ".xlsx" y poder graficar los valores.

A continuación, se detalla el proceso:

1. Conectar el sensor al Arduino
2. Conectar el Arduino al pc, a través de un cable
3. Calibrar el sensor
4. Cargar la programación al Arduino
5. Colocar el puerto COM a la programación del Matlab para que se conecte con el Arduino
6. Abrir el programa en Matlab y dar click en sensor ADXL335
7. Se nos abre una ventana donde dice toma de datos ADXL 335, damos click en puertos COM, y seleccionamos el puerto COM del Arduino
8. Ponemos el número de muestras que queremos obtener de la prueba y damos click en iniciar
9. Comenzamos a girar el sensor hasta dar una vuelta completa
10. Luego esperamos a que se carguen las muestran en el panel de datos y que se generan las graficas
11. Ponemos un nombre al archivo y damos click en guradar archivo
12. Ese archivo se guardará en una carpeta llamada datosAdxl, en formato .xlsx, donde se encuentran las muestras de las pruebas en tablas
13. Tomando en cuenta que se tomó como ángulo de referencia al eje ***x*** para el acelerómetro ADXL 335, porque en ese eje se va a dar la rotación con respecto a los ejes ***y*** y ***z***.

2.8.2. *Proceso de obtención y toma de datos del sensor WT901BLECL*

Para el proceso de obtención de datos para nuestra experimentación se utilizó el software MiniIMU versión 6.1.2, y para la toma de datos el software Matlab R2019a para almacenar los datos y poderlos graficar.
A continuación, se detalla el proceso.

1. Conectar el sensor WT901BLECL a la computadora mediante el adaptador bluetooth.

2. Encender el sensor poner en ON, y esperar a que se empareje con el adaptador hasta que esté una luz azul en el adaptador y una luz azul intermitente en el sensor
3. Abrir el programa miniIMU versión 6.1.2
4. Calibramos el sensor
5. Damos click en config, se nos abre una ventana
6. Damos click en Acceleration para calibrar la aceleración, esperamos hasta que se calibre y este en cero, tiene que estar el sensor en una superficie nivelada
7. Damos clik en Reset Z- axis Angle para resetear el eje ***z*** según la posición que queramos
8. Luego damos click en save config y damos en aceptar
9. Ya está listo el sensor para la obtención de datos
10. Colocamos el sensor en el eje
11. Damos click en record y ponemos en begin para comenzar a tomar los datos, giramos el sensor y luego damos en stop
12. Los datos se guardan en block de notas
13. Luego el block de notas tenemos que guardarlo en una carpeta llamada datos IMU, ahí se almacenan todos los archivos .txt de todas las mediciones que se hacen
14. Abrimos el Matlab con la programación ya hecha, damos clik en toma de datos
15. Se nos abre otra ventana que dice toma de datos, en la parte que dice archivos van a estar todos los archivos .txt que guardamos de la carpeta datos IMU
16. Damos click en cargar datos y esperamos a que se carguen los datos y poder visualizarlos en una tabla en formato .xls
17. En la parte que dice nombre del archivo, tenemos que colocar un nombre y luego datos click en guardar datos
18. Esos datos se van a guardar en una carpeta llamada datos, y estarán listos para poder ser graficados y hacer una animación

2.8.3. *Proceso para la obtención de las gráficas, animación y resultados en Matlab para el sensor WT901BLECL*

1. Abrimos la programación en Matlab, y damos click en graficar y animar
2. Se nos abre otra ventana de gráfica y animación, y en el panel de configuración elegimos en archivos el documento .xlsx que quremos ver
3. Esperamos a que se carguen las gráficas del acelerómetro y giroscopio en función del tiempo del sensor WT901BLECL
4. Nos aparece una ventana diciendo si queremos realizar la simulación de los datos del archivo .xlsx, damos click en yes para realizar la simulación

5. Podemos observar cómo se realiza la simulación de los datos, hay una barra de estado donde se ve el porcentaje de simulación cuando llaga al 100% nos aparece una ventana que nos dice si queremos ver los resultados, de damos click en yes para ver los resultados de los datos del archivo ".xlsx".
6. Se nos abre otra ventana de resultados, donde podemos observar la tabla de los datos, la gráfica de ángulo con respecto al tiempo, la gráfica con respecto al eje de referencia que es el eje ***x***, se carga automáticamente
7. Para sacar la ecuación característica de la gráfica con respecto al eje de referencia, no dirigimos a la parte que dice calcular la ecuación
8. Elegimos un grado de polinomio, vemos el q más se ajuste a la curva y le damos click en calcular la ecuación.
9. Nos aparece una ventana diciendo que se calculó las aproximaciones polinómicas de damos en ok, y me da a conocer el polinomio tanto en el eje ***y*** como el eje ***z*** con respecto al ángulo de inclinación que es mi eje ***x*.**
10. En la parte de configuración nos da en una tabla los ángulos en **x, y, z, Fy, Fz** que son los valores que me da a través del polinomio calculado y el error en el eje ***y*** y eje ***z***
11. Vemos también que me da el resultado del error relativo de la ecuación polinómica tanto en el eje *y* como el eje ***z***
12. Listo para hacer los análisis y descripciones de cada gráfica.

Se tomó como ángulo de referencia al eje ***x*** porque es donde se va a dar la rotación, se va a calcular la variación de ángulo con respecto al eje ***y*** y al eje ***z***.

CAPÍTULO III

3. MARCO DE ANÁLISIS E INTERPRETACIÓN DE RESULTADOS

En el trabajo experimental para poder calcular la ecuación de la curva característica de los datos de las pruebas se vio varios métodos de regresión polinómica: regresión de polinomio, regresión lineal, regresión Lagrange, exponencial, potencial y la logarítmica, siendo la regresión Lagrange la que más se ajustó a la curva característica porque tenemos datos positivos y negativos, esta regresión polinómica se la hizo en Matlab para poder ser graficada , y el polinomio que se utilizó es el siguiente:

$$p(x) = \sum_{k=0}^{n} \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i} yk \qquad \textbf{(36)}$$

Siendo:

$\boldsymbol{p(x)}$= Representa el polinomio interpolante de Lagrange

$\sum_{\boldsymbol{k=0}}^{\boldsymbol{n}}$ = Denota sumatoria de los valores de $\mathbf{0}$ hasta $\boldsymbol{n}$

$\prod_{\substack{\boldsymbol{i=0} \\ \boldsymbol{i \neq k}}}^{\boldsymbol{n}}$ =Denota producto de los valores, donde $\boldsymbol{i = 0}$ y $\boldsymbol{i \neq k}$

$\boldsymbol{x}$ = son las variables de los datos, donde $\boldsymbol{i \neq k}$ e $\boldsymbol{i = 0}$

$\boldsymbol{x_i}$ = son las variables de los datos (n+1), donde $\boldsymbol{i \neq k}$ e $\boldsymbol{i} = 0$

$\boldsymbol{x_k}$ = son las variables de los datos donde $\boldsymbol{i \neq k}$ e $\boldsymbol{i = 0}$

$\boldsymbol{yk}$ = ordenada en $\boldsymbol{y}$ multiplicada por cada polinomio

3.1. Resultados de las pruebas de los sensores

En la experimentación se va a poner a prueba el sensor ADXL 335 y al sensor WT901BLECL y ver los resultados que arroja cada sensor, viendo sus limitaciones, comparaciones, grados de error, y escoger el más adecuado para poder obtener la curva característica del eje de un motor eléctrico.

3.1.1. *Resultados del sensor ADXL 335*

A continuación, se realizará diferentes pruebas al sensor ADXL335, montado sobre un eje en posición horizontal, inclinado hacia arriba 5 grados, abajo 5 grados, a lado izquierdo 5 grados, a lado derecho 5 grados, para ver el comportamiento de este sensor, analizar gráficamente, y las limitaciones que tiene este sensor, se podrá observar los datos y las gráficas en las que se tomó como eje de referencia al eje ***Y*** ya que este eje no se mueve y los otros ejes el ***X*** y ***Z*** giran con respecto al eje ***Y***, a través del programa que se realizó en Matlab versión R2019a.

3.1.1.1. *Prueba número 1 con el eje en una posicion horizontal*

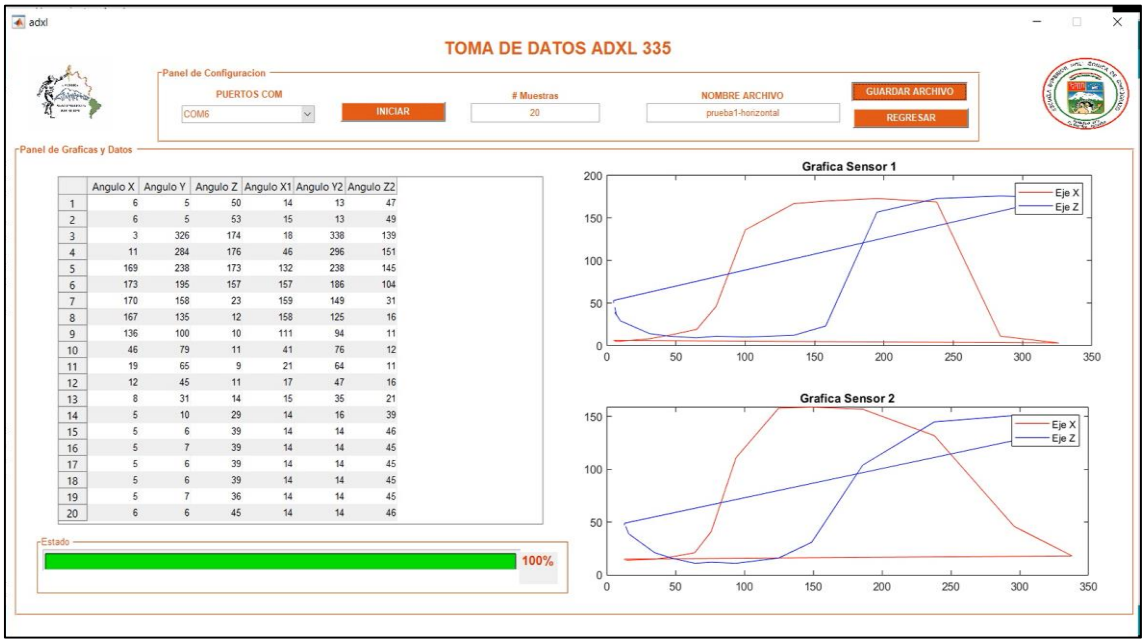


Gráfico 1-3. Sensores ADXL335 y datos de las muestras de la prueba uno**.**

Realizado por: (Del Pino, 2020)

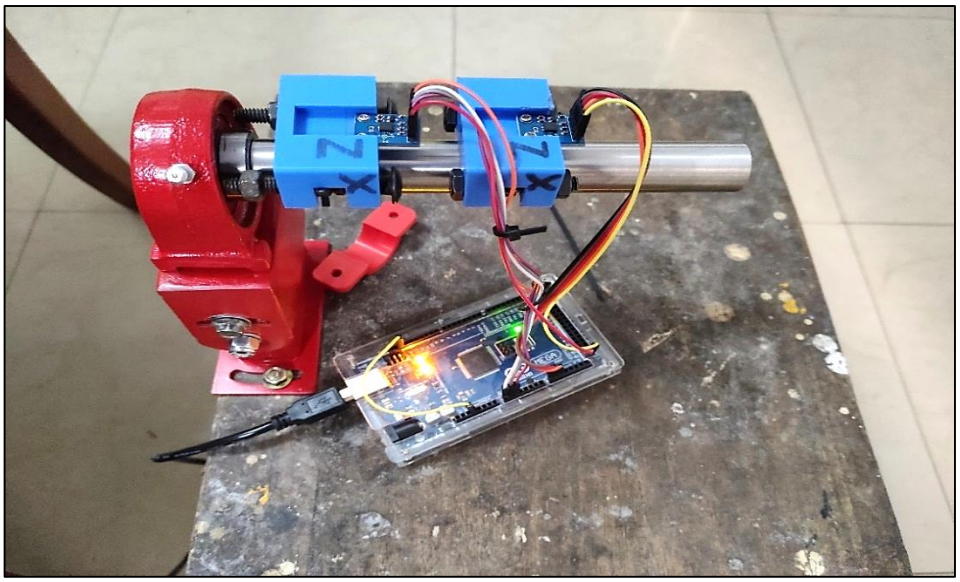
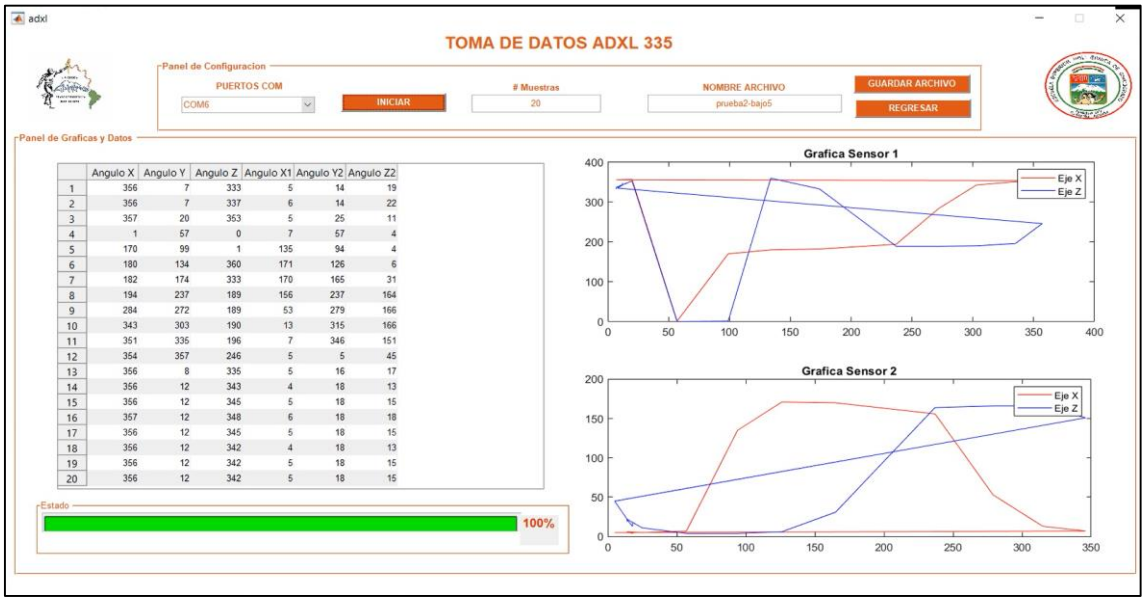


Figura 1-3. Comportamiento del sensor ADXL335 sobre un eje de pruebas.

Realizado por: (Del Pino, 2020).

En la Gráfico 1-3 de la prueba número 1 con el eje en una posición horizontal, los resultados que se obtienen al dar una vuelta completa con 20 muestras de datos, se puede observar que el eje ***Y*** es en el cual gira el motor, se toma datos de los ángulos ***X*** y ***Z***, donde se puede ver que existe una gran diferencia de medición, por ejemplo, en el ángulo ***Y*** igual a 45 grados, ***X*** y ***Z*** 12 y 11 grados respectivamente vemos que están prácticamente paralelos los ángulos, existiendo un gran error en el sensor. En la Figura 1-3 se puede observar cómo se realizó la prueba al sensor ADXL335.

3.1.1.2. *Prueba número 2 con el eje inclinado cinco grados hacia arriba*



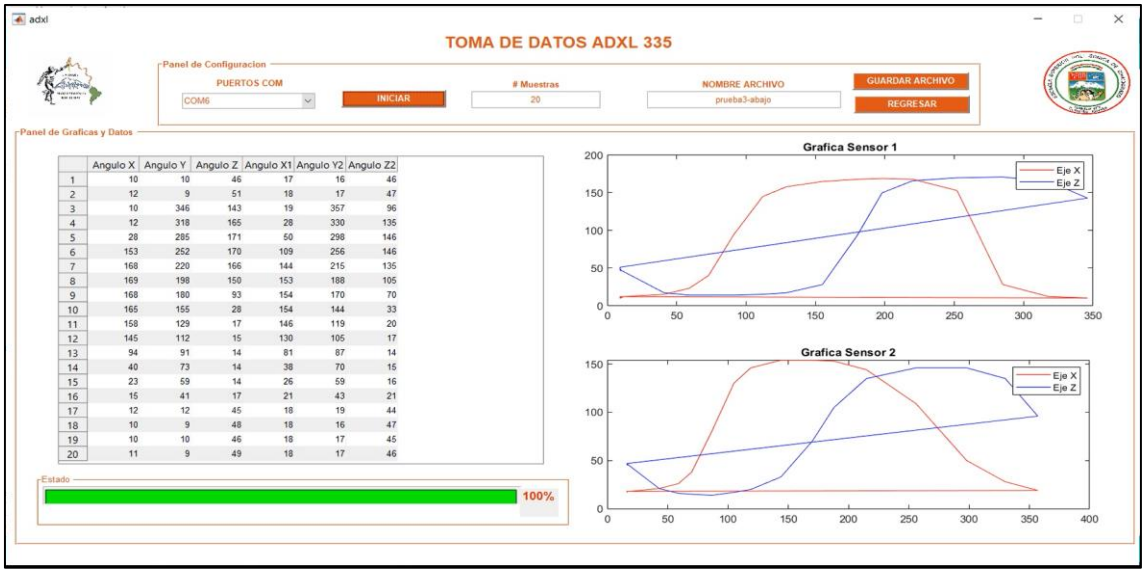
	Angulo X	Angulo Y	Angulo Z	Angulo X1	Angulo Y2	Angulo Z2
1	356	7	333	5	14	19
2	356	7	337	6	14	22
3	357	20	353	5	25	11
4	1	57	0	7	57	4
5	170	99	1	135	94	4
6	180	134	360	171	126	6
7	182	174	333	170	165	31
8	194	237	189	156	237	164
9	284	272	189	53	279	166
10	343	303	190	13	315	166
11	351	335	196	7	346	151
12	354	357	246	5	5	45
13	356	8	335	5	16	17
14	356	12	343	4	18	13
15	356	12	345	5	18	15
16	357	12	348	6	18	18
17	356	12	345	5	18	15
18	356	12	342	4	18	13
19	356	12	342	5	18	15
20	356	12	342	5	18	15

Gráfico 2-3. Sensores ADXL335 y datos de las muestras de la prueba dos.

Realizado por: (Del Pino, 2020)

En el Gráfico 2-3 de la prueba 2 con el eje inclinado cinco grados hacia arriba, los resultados que se obtienen al igual que en la anterior prueba, se hizo una vuelta completa con 20 muestras, donde se observa que hay una gran diferencia de medición en los ángulos, como vemos a partir de la muestra numero 12 el ángulo ***X1*** detecta los 5 grados provocados con un pequeño error, en cambio el ángulo ***X*** los datos ya son erróneos nos da 354 grados y el ángulo ***Y*** 357 grados, prácticamente nos dice que están paralelos esos dos ángulos, dando así datos no reales.

3.1.1.3. *Prueba número 3 con el eje inclinado cinco grados hacia abajo*



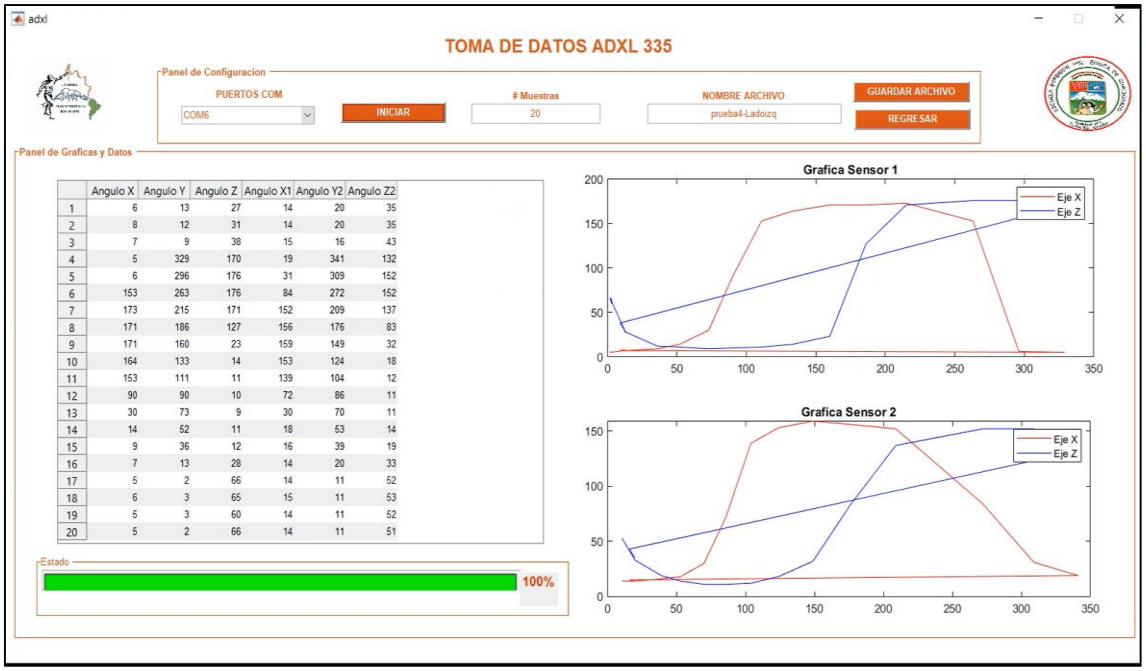
	Angulo X	Angulo Y	Angulo Z	Angulo X1	Angulo Y2	Angulo Z2
1	10	10	46	17	16	46
2	12	9	51	18	17	47
3	10	346	143	19	357	96
4	12	318	165	28	330	135
5	28	285	171	50	298	146
6	153	252	170	109	256	146
7	168	220	166	144	215	135
8	169	198	150	153	188	105
9	168	180	93	154	170	70
10	165	155	28	154	144	33
11	158	129	17	146	119	20
12	145	112	15	130	105	17
13	94	91	14	81	87	14
14	40	73	14	38	70	15
15	23	59	14	26	59	16
16	15	41	17	21	43	21
17	12	12	45	18	19	44
18	10	9	48	18	16	47
19	10	10	46	18	17	45
20	11	9	49	18	17	46

Gráfico 3-3. Sensores ADXL335 y datos de las muestras de la prueba tres.

Realizado por: (Del Pino, 2020)

En el Gráfico 3-3 de la prueba 3 con el eje inclinado cinco grados hacia abajo, los resultados que se tiene al igual que la anterior prueba se hizo una vuelta completa con 20 muestras, donde se observa que existe una gran diferencia en las mediciones, en este caso no detecta en ningún punto los cinco grados provocados, de igual forma los datos que se muestran no son reales, en la muestra 9 vemos que el ángulo ***X*** y ***Y*** están casi paralelos con 168 y 180 grados respectivamente de igual forma en el ángulo ***X1*** y ***Y1***.

3.1.1.4. *Prueba número 4 con el eje horizontal, pero girado hacia el lado izquierdo*



	Angulo X	Angulo Y	Angulo Z	Angulo X1	Angulo Y2	Angulo Z2
1	6	13	27	14	20	35
2	8	12	31	14	20	35
3	7	9	38	15	16	43
4	5	329	170	19	341	132
5	6	296	176	31	309	152
6	153	263	176	84	272	152
7	173	215	171	152	209	137
8	171	186	127	156	176	83
9	171	160	23	159	149	32
10	164	133	14	153	124	18
11	153	111	11	139	104	12
12	90	90	10	72	86	11
13	30	73	9	30	70	11
14	14	52	11	18	53	14
15	9	36	12	16	39	19
16	7	13	28	14	20	33
17	5	2	66	14	11	52
18	6	3	65	15	11	53
19	5	3	60	14	11	52
20	5	2	66	14	11	51

Gráfico 4-3. Sensores ADXL335 y datos de las muestras de la prueba cuatro.

Realizado por: (Del Pino, 2020)

En el Gráfico 4-3 de la prueba 4 con el eje horizontal girado cinco grados hacia el lado izquierdo, los resultados que se tiene de igual forma que la anterior prueba se hizo una vuelta completa con 20 muestras, donde se observa que hay una enorme diferencia en las mediciones, se provocó cinco grados hacia la derecha que viene a ser el eje ***Z*** el cual vemos que no ha detectado el sensor ese movimiento, como podemos observar en la muestra 7 donde el ángulo ***Z*** tiene 171 grados y el ángulo ***X*** 173 grados , siendo casi iguales, y nos damos cuenta que no son reales los datos y en todas las muestras en el ángulo ***Z*** nos da valores bien diferentes a lo esperado.

3.1.1.5. *Prueba número 5 con el eje horizontal, pero girado hacia el lado derecho*

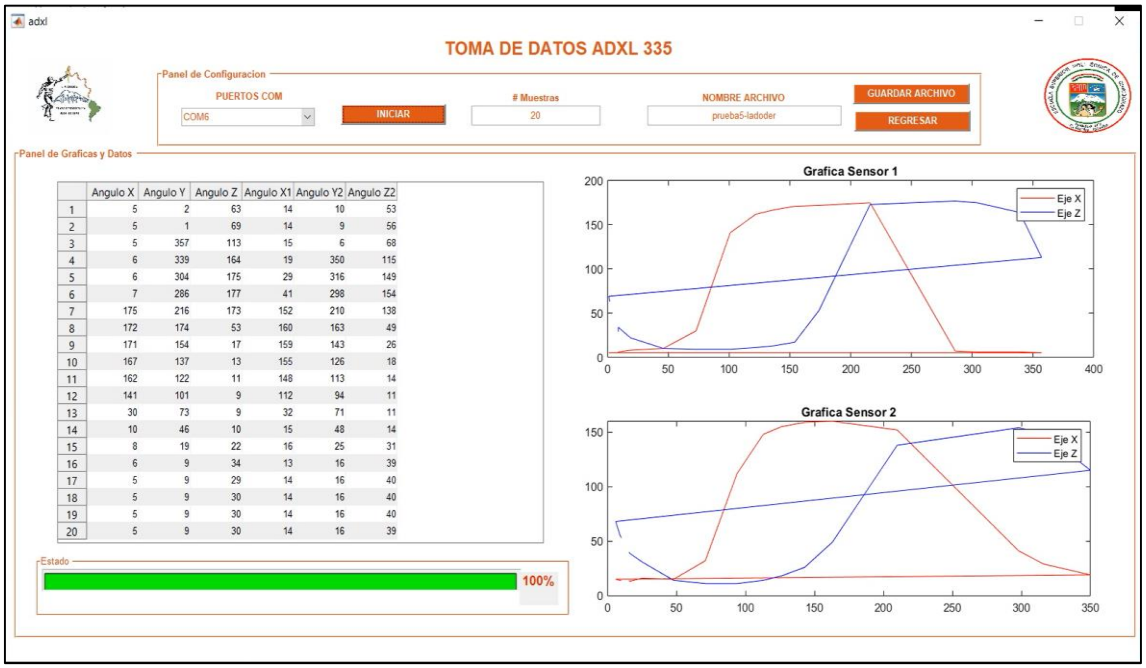


Gráfico 5-3. Sensores ADXL335 y datos de las muestras de la prueba cinco.

Realizado por: (Del Pino, 2020)

En el Gráfico 5-3 de la última prueba con el eje horizontal girado cinco grados hacia la derecha, se hizo una vuelta completa con 20 muestras, donde observamos al igual que la anterior prueba, pero con lado contrario, nos arroja valores no reales, como podemos ver en un punto el 14 el ángulo ***X*** es igual al ángulo ***Z*** con 10 grados respectivamente, dándonos a entender que los ángulos se encuentran paralelamente, viendo así en la prueba experimental que el sensor tiene muchas fallas a la hora de las lecturas.

3.1.2. *Resultados del sensor WT901BLECL*

En las siguientes gráficas y tablas se puede observar las pruebas realizadas sobre el sensor WT901BLECL en un eje de acero sin desviaciones y a nivel, así como desviaciones provocadas, los datos obtenidos se analizaron gracias a la aplicación desarrollada en la plataforma Matlab la cual nos da a conocer las aceleraciones por cada eje y las velocidades angulares, además de proporcionarnos una simulación en la cual se puede observar el comportamiento del eje en el espacio.

3.1.2.1. *Prueba 1 del sensor WT901BLECL con el eje en posición hoirzontal*

Gráfico 6-3. Giroscopio con respecto al tiempo, acelerómetro con respecto al tiempo y simulación de la primera prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

Gráfico 7-3. Gráfica con respecto al eje de referencia y de ángulo vs. tiempo de la primera prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

Figura 2-3. Comportamiento del sensor WT901BLECL sobre el eje de pruebas.

Realizado por: (Del Pino, 2020)

La presente prueba se realizó con un eje a nivel y la prueba consistió en analizar el giro del eje como se observa en la prueba que duro 17,12 segundos, aparentemente la velocidad angular en los ejes **Y** y **Z** son cero como se muestra en Grafico 7-3. En la gráfica de ángulo, por lo cual no existe desviación del eje, pero en la parte del proceso como se puede observar en la Grafica 7-3. En la gráfica con respecto al eje de referencia, existe un pequeño desvió tanto en el eje ***Y*** y ***Z*** cuyo valor máximo en ángulo es de 1,7 grados y 2 grados respectivamente cabe recalcar que la desviación es mínima y las causas que lo generan puede ser varias como la falta de ajuste de tuercas, la base del motor etc. En la Figura 2-3 se puede observar como se realizo la prueba del sensor.

3.1.2.2. *Prueba 2 del sensor WT901BLECL con eje inclinado cinco grados hacia abajo*

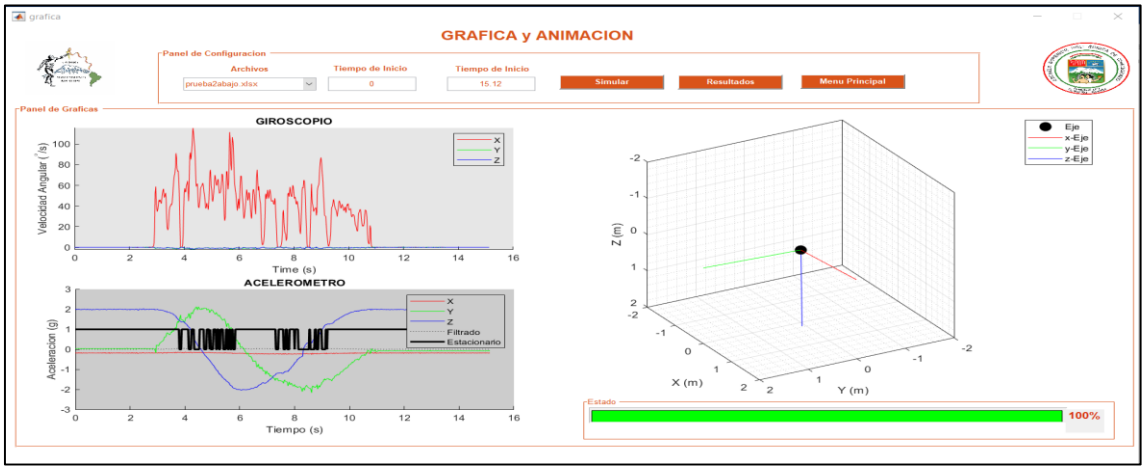


Gráfico 8-3. Gráfica del giroscopio con respecto al tiempo, acelerómetro con respecto al tiempo y simulación de la segunda prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

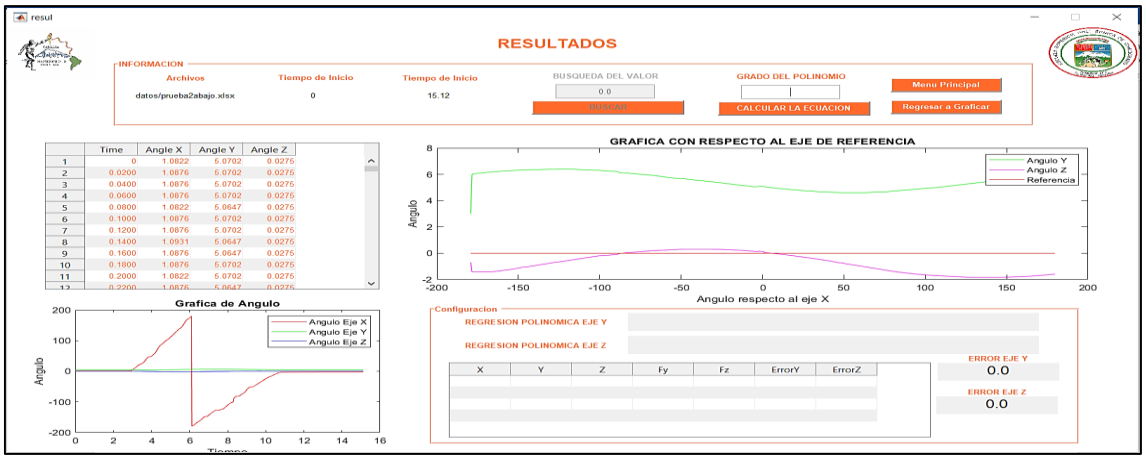


Gráfico 9-3. Gráfica con respecto al eje de referencia y de ángulo vs. tiempo de la segunda prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

La presente prueba se realizó con una desviación en el eje ***Y*** de cinco grados, la prueba consistió en analizar el giro del eje como se observa la prueba duro 15,12 segundos, además se puede observar que aparentemente la velocidad en los ejes ***Y*** y ***Z*** son cero como se observa en la Gráfica 9-3. Gráfica de ángulo con respecto al tiempo, como en la anterior prueba por lo cual no existe desviación del eje, pero existe una desviación permanente del eje ***Y*** que empieza u oscila entre los 5 grados donde se provocó el desvío, en la segunda parte del proceso se puede observar un desvío notable en ***Y*** como se observa en la Grafica 9-3. Grafica con respecto al eje de referencia, con un valor máximo de 6,5 grados. Dicho valor oscila como se dijo anteriormente sobre los 5 grados de desviación provocado para dicha prueba, por otra parten el valor del eje ***Z*** oscila sobre cero grados con un valor máximo de 2 grados aproximadamente.

3.1.2.3. *Prueba 3 del sensor WT901BLECL con eje inclinado cinco grados hacia arriba*

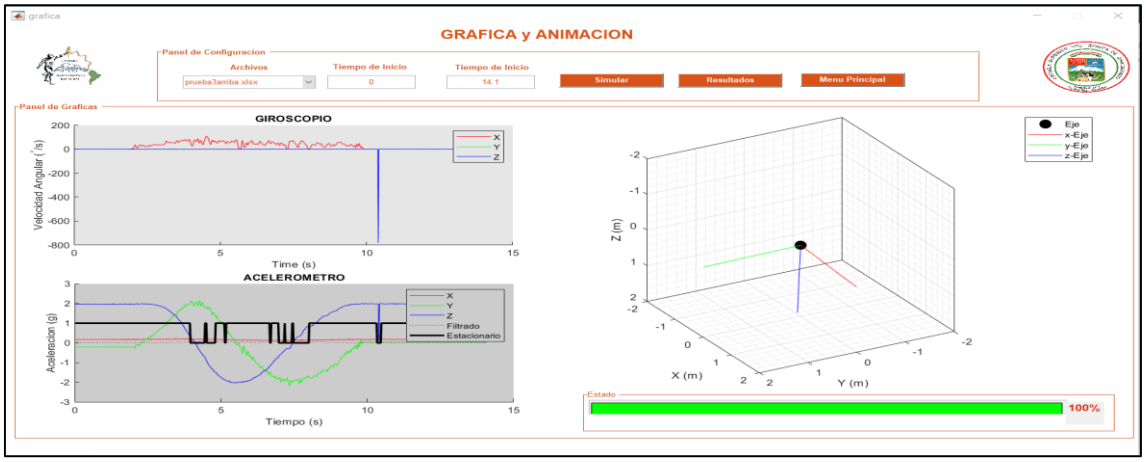


Gráfico 10-3. Gráfica del giroscopio con respecto al tiempo, acelerómetro con respecto al tiempo y simulación de la tercera prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

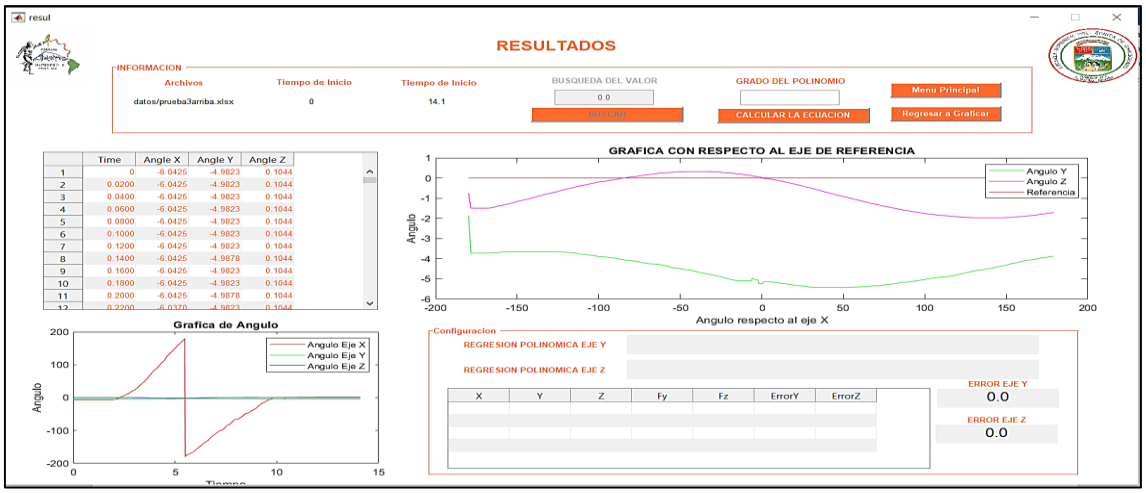


Gráfico 11-3. Gráfica con respecto al eje de referencia y de ángulo vs. tiempo de la tercera prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

La presente prueba se realizó con una deviación en el eje ***Y*** de cinco grados, pero con diferente dirección, la prueba consistió en nuevamente analizar el giro del eje, como se observa la prueba duro 14,1 segundos, además se puede observar que la velocidad angular en los ejes ***Y*** y ***Z*** *son* un poco mayores a cero, por lo cual no existe desviación del eje pero existe una desviación permanente del eje ***Y*** pero con signo cambiado comparado con la prueba anterior, la cual empieza u oscila entre los 5 grados donde se provocó el desvío, en la segunda parte del proceso se puede observar en la Grafica 11-3. Existe un desvío notable en ***Y*** con un valor máximo de -5,6 grados. Dicho valor oscila como se dijo anteriormente sobre los 5 grados de desviación provocado para dicha prueba, por otra parte, el valor de ***Z*** oscila sobre cero grados con valor máximo de 2 grados aproximadamente.

3.1.2.4. *Prueba 4 del sensor WT901BLECL con el eje a lado izquiedo cinco grados*

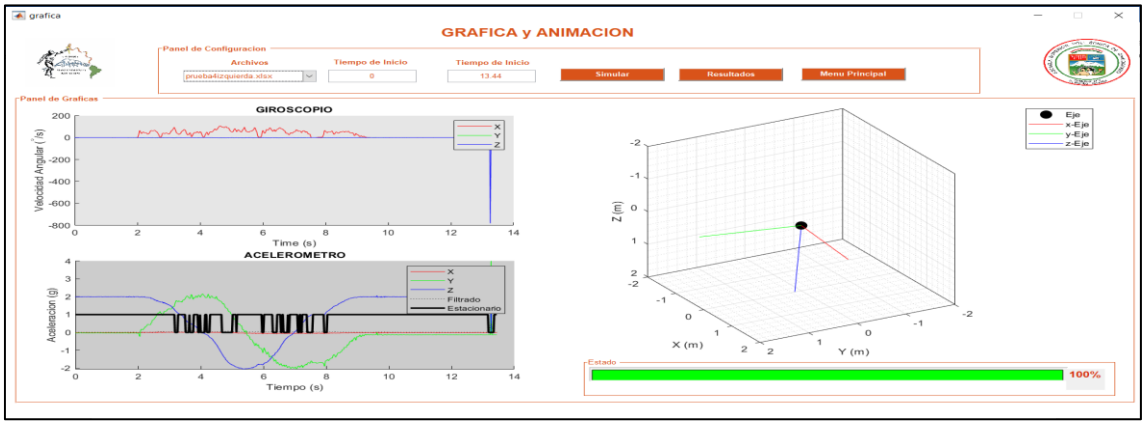


Gráfico 12-3. Giroscopio con respecto al tiempo, acelerómetro con respecto al tiempo y simulación de la cuarta prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

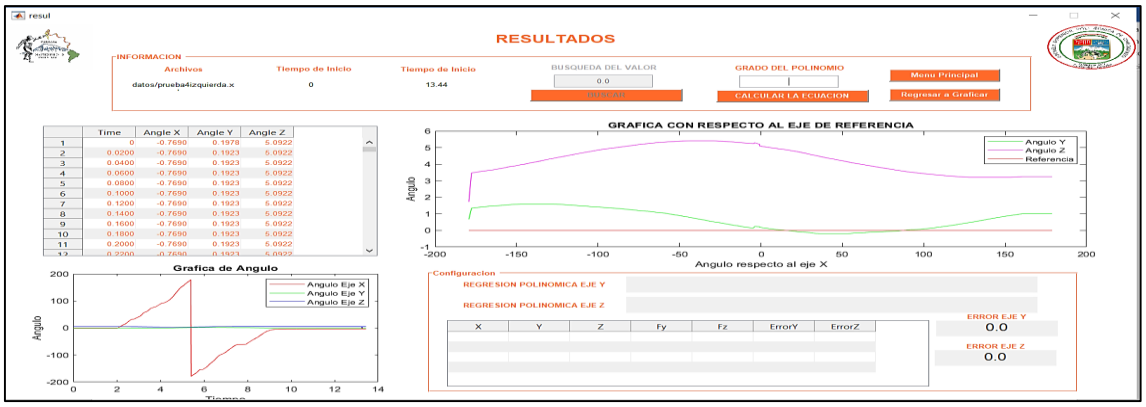


Gráfico 13-3. Gráfica con respecto al eje de referencia y de ángulo vs. tiempo de la cuarta prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

La presente prueba se realizó con una desviación en el eje ***Z*** de cinco grados es decir se desvío hacia la parte izquierda comparando con el eje del motor, la prueba consistió en nuevamente en analizar el giro del eje, como se observa la prueba duro 13,44 segundos, además se puede observar que la velocidad angular en los ejes ***Y*** y ***Z*** como las anteriores pruebas no son tan diferentes a cero, por lo cual no existe desviación del eje, pero existe una desviación permante del eje ***Z,*** la cual empieza y oscila entre los 5 grados donde se provocó el desvío, en la segunda parte del proceso se puede observar que existe un desvío notable en el eje ***Z*** con un valor máximo de 5,5 grados aproximadamente. Dicho valor oscila como se dijo anteriormente sobre los 5 grados de desviación provocado para dicha prueba, por otra parte, el valor del eje ***Y*** oscila sobre cero grados con valor máximo de 1,65 grados aproximadamente como se observa en la Grafica 13-3. Grafica con respecto al eje de referencia.

3.1.2.5. *Prueba 5 del sensor WT901BLECL con el eje a lado derecho cinco grados*

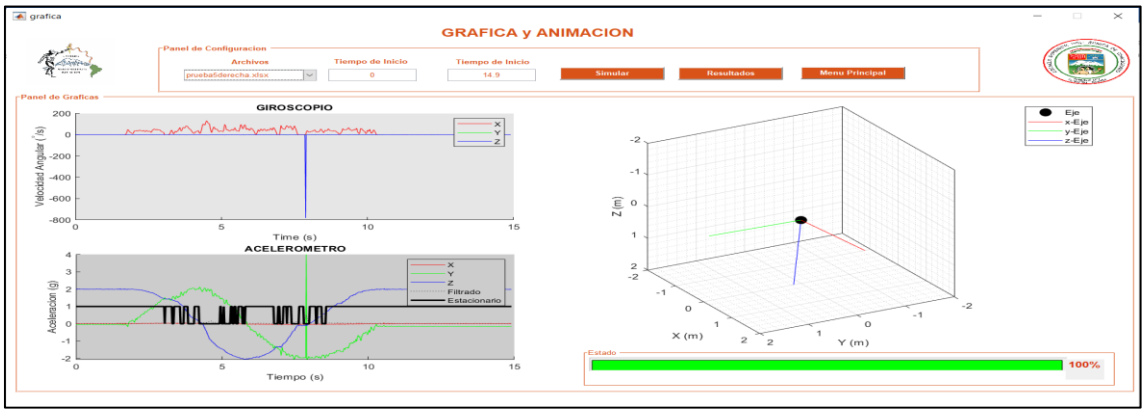


Gráfico 14-3. Giroscopio con respecto al tiempo, acelerómetro con respecto al tiempo y simulación de la quinta prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

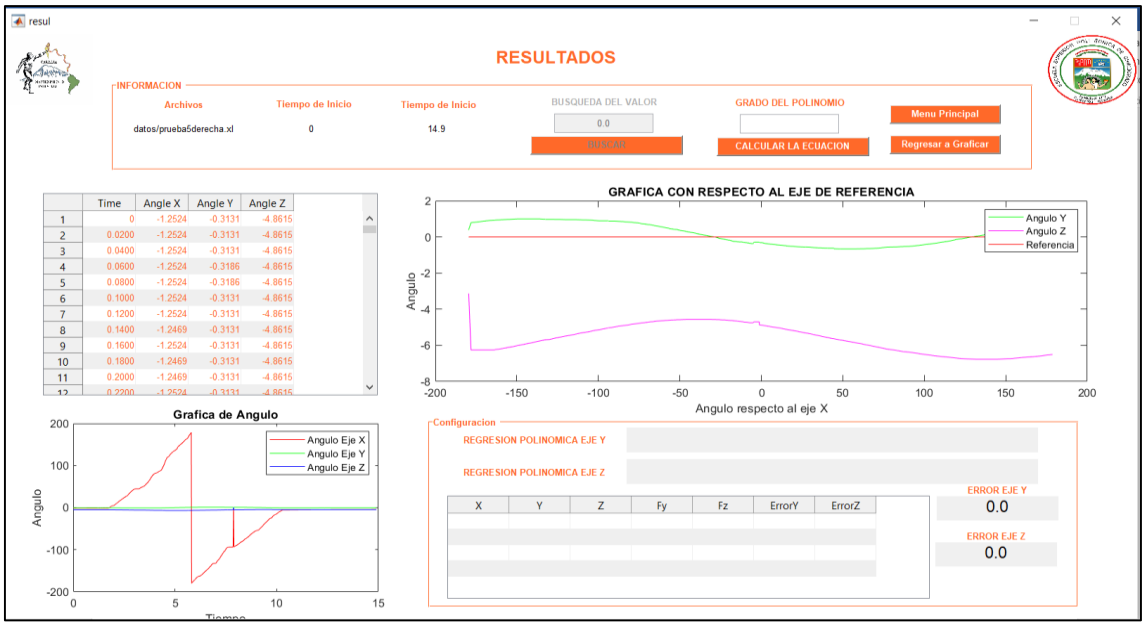


Gráfico 15-3. Gráfica con respecto al eje de referencia, grafica de ángulo vs. tiempo de la quinta prueba del sensor WT901BLECL.

Realizado por: (Del Pino, 2020)

La presente prueba se realizó con una desviación en el eje ***Z*** de 5 grados pero con dirección diferente es decir se desvío hacia la parte derecha, comparado con la prueba anterior, la prueba consistió en nuevamente analizar el giro del eje, como se observa duro 14,9 segundos, además se puede observar que la velocidad angular en los ejes ***Y*** y ***Z*** como en las anteriores pruebas no son tan diferentes a cero, por lo cual no existe desviación del eje pero existe una desviación permanente del eje ***Z***, la cual empieza u oscila entre los 5 grados donde se provocó el desvío, en la segunda parte del proceso como se puede observar en la Grafica 15-3. Grafica con respecto al eje de referencia, existe un desvío notable en el ***Z*** con un valor máximo de 6,5 grados aproximadamente. Dicho valor oscila como se dijo anteriormente sobre los 5 grados de desviación provocado para dicha prueba, por otra parte, el valor del eje ***Y*** oscila sobre cero grados con un valor máximo de 1 grado aproximadamente.

3.1.3. *Comparación de los errores del sensor ADXL335 vs WT901BLECL*

Tabla 1-3: Versus entre el sensor WT901BLECL y el sensor ADXL335.

SENSOR WT901BLECL										SENSOR ADXL335						
	VELOCIDAD ANGULAR			ACELERACION			ÁNGULOS			SENSOR 1				SENSOR 2		
Tiempo	wx (deg/s)	wy (deg/s)	wz (deg/s)	ax(g)	ay(g)	az(g)	Angulo x (deg)	Angulo y (deg)	Angulo z (deg)	MUESTRAS	ANGULO X (°)	ANGULO Y (°)	ANGULO Z (°)	ANGULO X1 (°)	ANGULO Y2 (°)	ANGULO Z2 (°)
0	36,2549	-0,4272	-0,6104	0,0103	2,0557	0,1611	85,3033	-0,2747	-1,1316	1	5	360	90	13	12	47
0,02	16,9678	-0,4883	-0,4272	-0,0156	1,9497	-0,3994	100,9314	-0,2472	-1,3293	2	4	348	157	14	2	78
0,04	27,1606	-0,3662	-0,1221	-0,0059	1,1455	-1,729	148,3374	0,368	-1,6644	3	3	314	176	20	332	144
0,06	23,9258	-0,7935	-0,1221	-0,0229	0,333	-2	171,9031	0,6482	-1,5381	4	8	284	177	37	303	153
0,08	34,1187	-0,5493	-0,7324	-0,0317	-0,4136	-1,957	-167,1295	0,8734	-1,3458	5	165	257	176	82	273	153
0,1	20,5078	-0,3662	-0,6714	-0,041	-1,2319	-1,582	-142,7124	1,1041	-0,9778	6	174	212	170	150	212	138
0,12	19,6533	-0,1831	0	-0,0361	-1,6411	-0,9917	-121,0144	1,17	-0,6427	7	174	180	97	160	177	81
0,14	24,353	-0,2441	-1,3428	-0,0044	-1,7681	-0,8369	-114,5544	1,2579	-0,5383	8	172	159	19	161	153	33
0,16	15,564	0	-0,1221	-0,062	-1,9761	-0,0781	-93,0872	1,2854	-0,2142	9	166	135	13	158	130	18
0,18	14,5264	-0,3052	-0,3662	-0,0283	-1,9751	0,4487	-77,1954	1,1206	0	10	157	114	10	148	111	13
0,2	45,166	0	0	-0,0103	-1,5059	1,2271	-51,9269	0,769	0,2307	11	122	96	10	131	100	12
0,22	45,2881	-0,7324	-0,3662	-0,0195	-0,1729	1,9819	-6,3336	0,1208	0,1538	12	45	80	9	82	88	10
0,24	0	0,3052	-0,061	-0,0073	0,2036	1,9932	5,4327	-0,033	0,0824	13	24	68	10	43	78	11
0,26	-0,1221	0,1831	0,2441	-0,0205	0,2075	2,0122	5,6689	0,0714	0,0989	14	12	52	9	22	64	11
0,28	0,2441	-0,1221	0,1221	-0,0083	0,2144	1,9727	5,7733	0,0385	0,1208	15	5	13	23	14	33	21
0,3	0,1221	-0,1831	0,061	-0,0269	0,2109	2,0059	5,8502	0,011	0,1483	16	4	347	158	14	10	53
0,32	0	0	0	0,0029	0,2095	1,9917	5,9216	0,0385	0,1538	17	4	345	162	14	8	60
										18	4	344	162	14	8	59
										19	4	344	162	14	8	59
										20	3	343	167	14	7	62

Realizado por: (Del Pino, 2020)

Figura 3-3. Sensor ADXL335 vs WT901BLECL sobre el eje de pruebas.

Realizado por: (Del Pino, 2020)

Si se compara el sensor seleccionado WT901BLECL versus el sensor ADXL335 en un punto por ejemplo el valor de desvío en el ángulo 150 en el sensor WT901BLECL es de 0,33 grados aproximadamente y del sensor ADXL335 es 166 grados como se observa en la Tabla 1-3, si se calcula el error y mediante los datos obtenidos se tiene un error aproximado de 500% un valor extremadamente ilógico por ello las pruebas realizadas sobe el sensor ADXL335 son para comprobar el error del sensor y la selección del mejor sensor para la ejecución de este trabajo experimental. En la Figura 3-3 se observa como se hizo este versus entre los dos sensores.

3.2. Curva característica de las pruebas del sensor WT901BLECL

El sensor seleccionado y q nos proporciona información suficiente para poder obtener la curva característica es el sensor WT901BLECL, el cual mediante las pruebas se pudo obtener una curva característica y calcular la ecuación característica que se ajuste a la curva generada con un error mínimo aceptable.

3.2.1.1. *Curva caracteritica de la primera prueba con el sensor WT901BLECL*

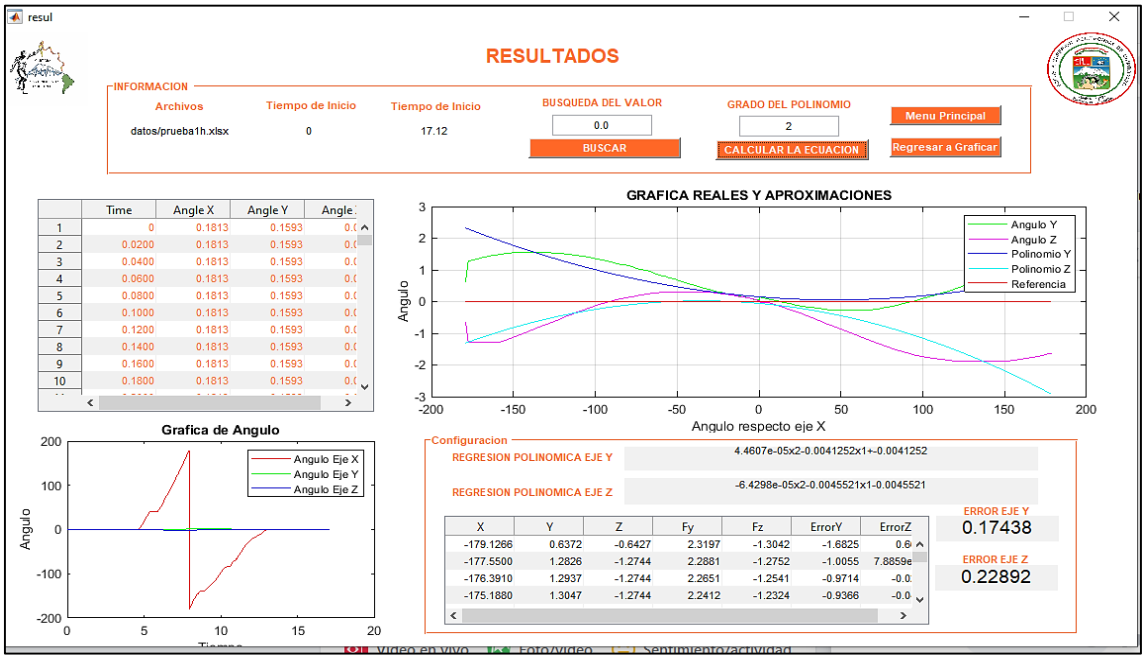


Gráfico 16-3. Curva característica de la primera prueba con grado de polinomio 2.

Realizado por: (Del Pino, 2020)

Ya obteniendo los datos del sensor se procede a calcular la ecuación en este caso mediante una regresión polinómica ya que es la que mejor se acopla, en la Gráfica 16-3. Se puede observar que con un polinomio de grado 2 se obtiene una curva característica muy parecida a la real, pero procediendo hacer el análisis de la ecuación obtenida da como resultado un error superior al 17% y 22% con respecto al eje ***Y*** y ***Z***, dichos valores de error son superiores a lo esperado por lo consiguiente se procede a realizar una regresión con más términos para conocer qué resultados se obtendrán.

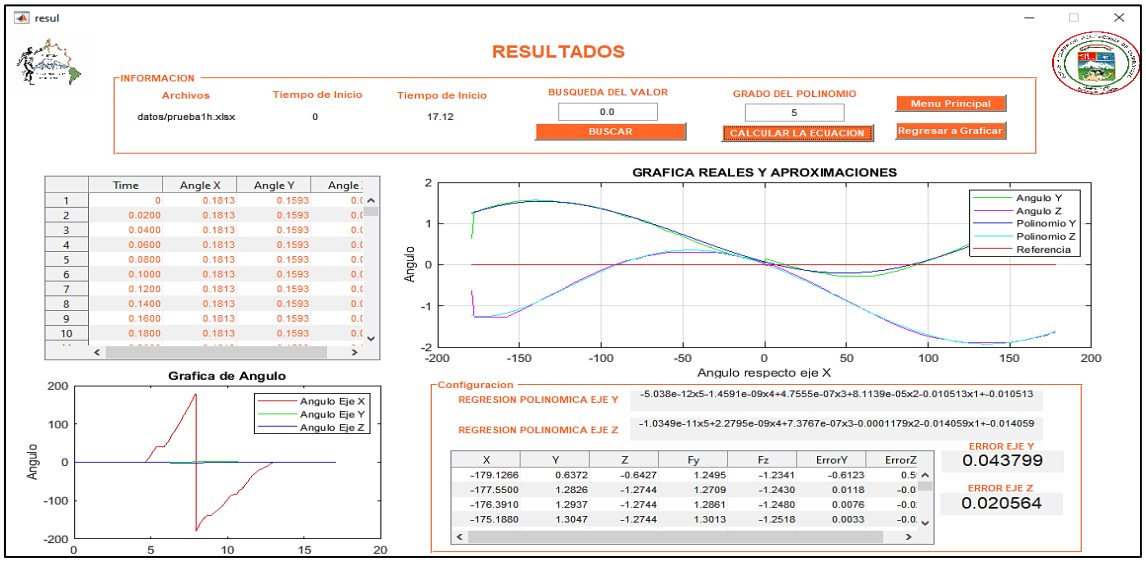


Gráfico 17-3. Curva característica de la primera prueba con grado de polinomio 5.

Fuente: (Del Pino, 2020)

Realizando un nuevo cálculo de regresión se obtiene sin duda mejores resultados con un error en el eje ***Y*** del 4.23% y del eje ***Z*** del 2%, como se observa en la Gráfica 17-3. Lo cual vuelve a la ecuación una aproximación adecuada ya que no supera el 5% de error relativo esperado además gráficamente o visualmente se observa que los 2 graficas de regresión se acopla perfectamente comparado con el cálculo anterior.

3.2.1.2. *Curva caracteritica de la segunda prueba con el sensor WT901BLECL*

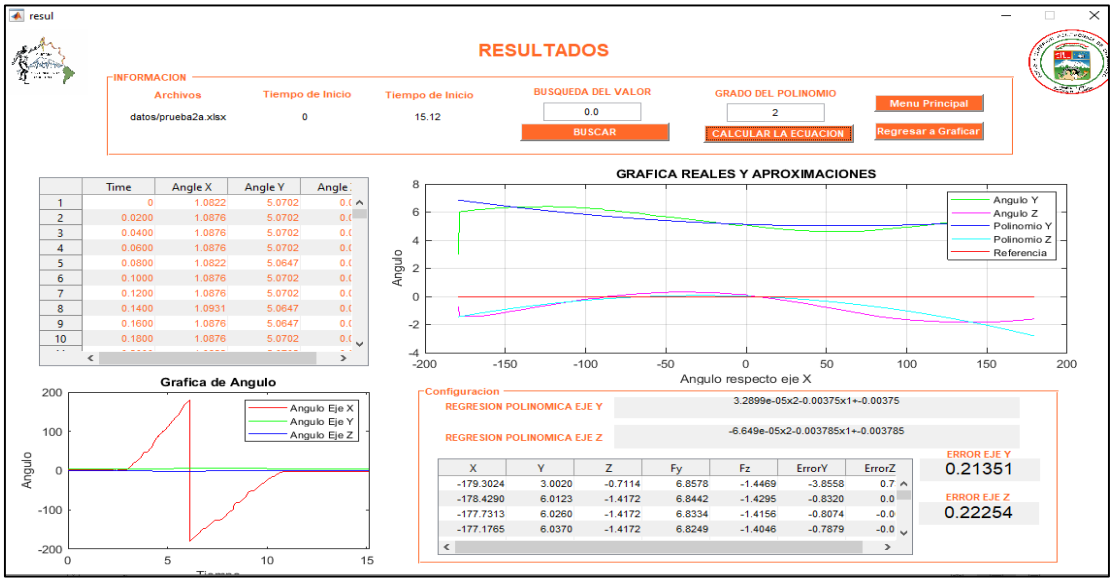


Gráfico 18-3. Gráfico de la curva característica de la segunda prueba con grado de polinomio 2.

Fuente: (Del Pino, 2020)

La siguiente prueba o cálculo de la curva dieron resultados parecidos a la anterior con la diferencia que se observa un Angulo notable en el eje ***Y*** oscilante entre el valor causado para la prueba, de igual manera el calculó de la curva se realizó mediante una regresión polinómica la cual arrojo resultados malos se podría decir ya que en el análisis se obtuvo un error superior al 20% un error considerable y no aceptable como se observa en la Grafica 18-3.

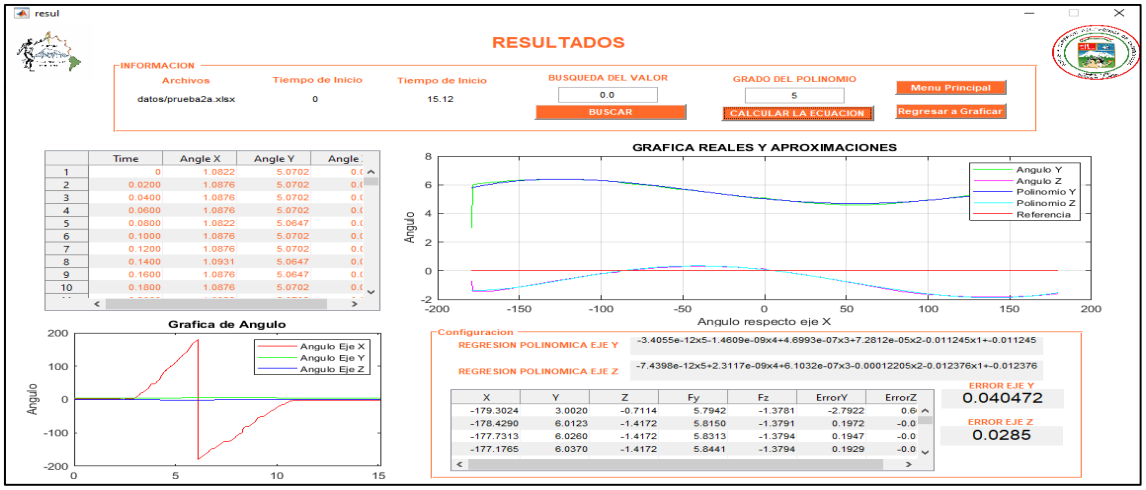


Gráfico 19-3. Curva característica de la segunda prueba con grado de polinomio 5.

Fuente: (Del Pino, 2020)

Realizando un nuevo cálculo de regresión polinómica se obtiene mejores resultados con un error en el eje ***Y*** del 4% y del eje ***Z*** del 2,8% como se observa en la Grafica 19-3. Estos valores no superan el 5% planteado por ello con un polinomio calculado de grado 5 cumple con los requisitos para ser aceptada como ecuación característica tanto para el eje ***Y*** y ***Z***.

3.2.1.3. *Curva caracteritica de la tercera prueba con el sensor WT901BLECL*

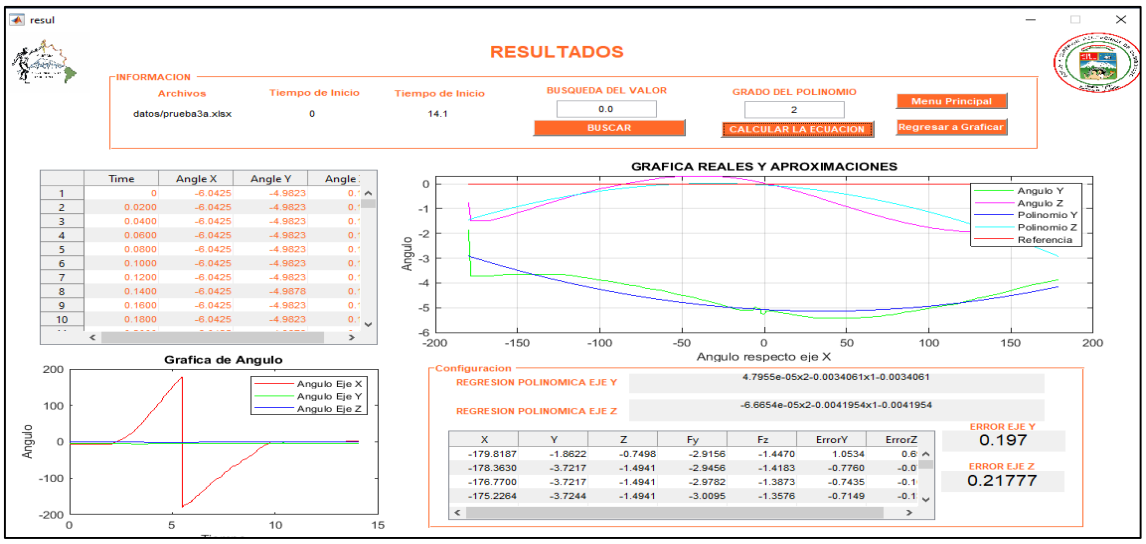


Gráfico 20-3. Curva característica de la tercera prueba con grado de polinomio 2.

Fuente: (Del Pino, 2020)

La siguiente prueba o cálculo de la curva dieron resultados parecidos a los anteriores, como la prueba se realizó variando el Angulo ***Y*** aproximadamente 5 grados la ecuación se rige a esos valores, utilizando como las anteriores pruebas un polinomio de segundo orden se tienen valor que superan al 20 % de error por lo consiguiente se descartan como posibles ecuaciones características, como se observa en la Gráfica 20-3.

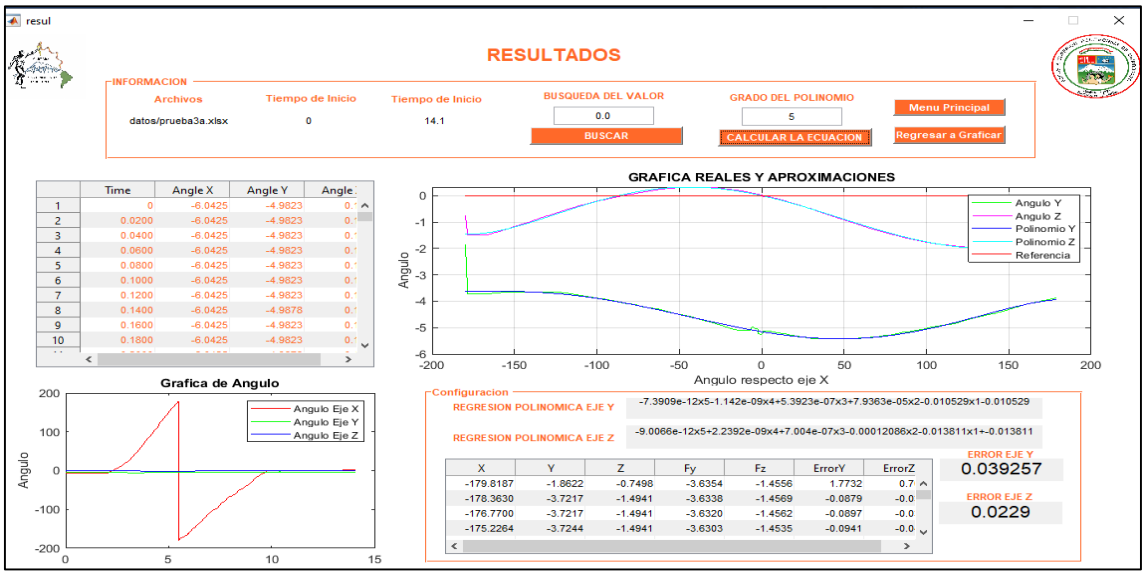


Gráfico 21-3. Curva característica de la tercera prueba con grado de polinomio 5.

Fuente: (Del Pino, 2020)

Para obtener mejores resultados se realizó otra regresión polinómica aumentando el grado del polinomio, así se logra mejorar el grado de error al 3,9% y 2,2% de error en los ejes ***Y*** y ***Z*** respectivamente, con estos valores se puede decir que el polinomio calculado es aceptable para la ecuación característica de nuestra curva, como se observa en la Gráfica 21-3.

3.2.1.4. *Curva caracteritica de la cuarta prueba con el sensor WT901BLECL*

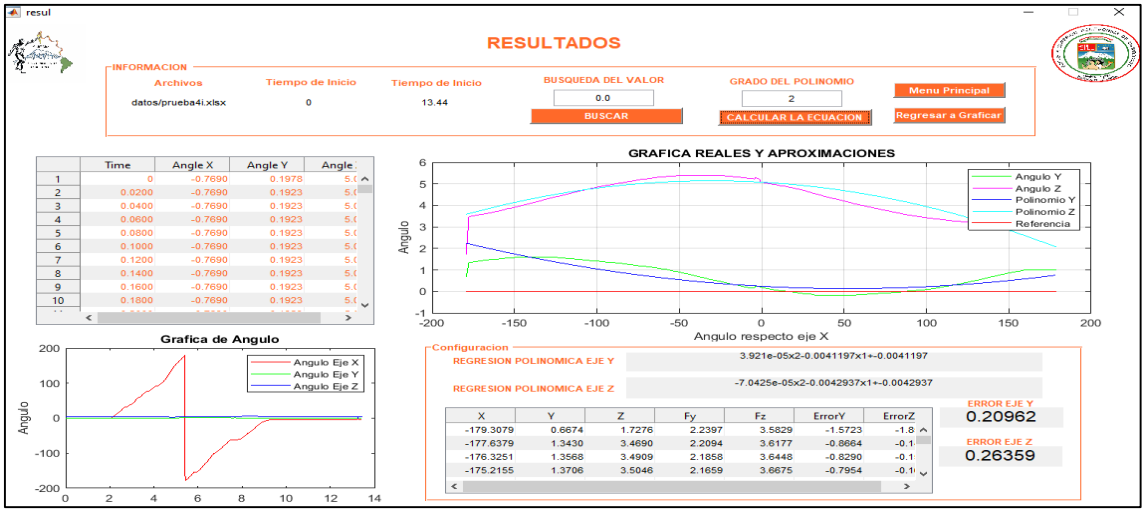


Gráfico 22-3. Curva característica de la cuarta prueba con grado de polinomio 2.

Fuente: (Del Pino, 2020)

Los datos de esta prueba como se mostró anteriormente se realizaron variando el Ángulo en el eje ***Z*** por lo cual existe una variación u oscilación en el Angulo de inclinación en este caso en ***Z***, para el cálculo de la ecuación se realizó la regresión polinómica de grado 2 dando como resultado un error del 20% en el eje ***Y*** y 26,3% en el eje ***Z***, estos valores son considerablemente grandes por ello se descartan y se procede a realizar otra regresión, como se observa en la Gráfica 22-3.

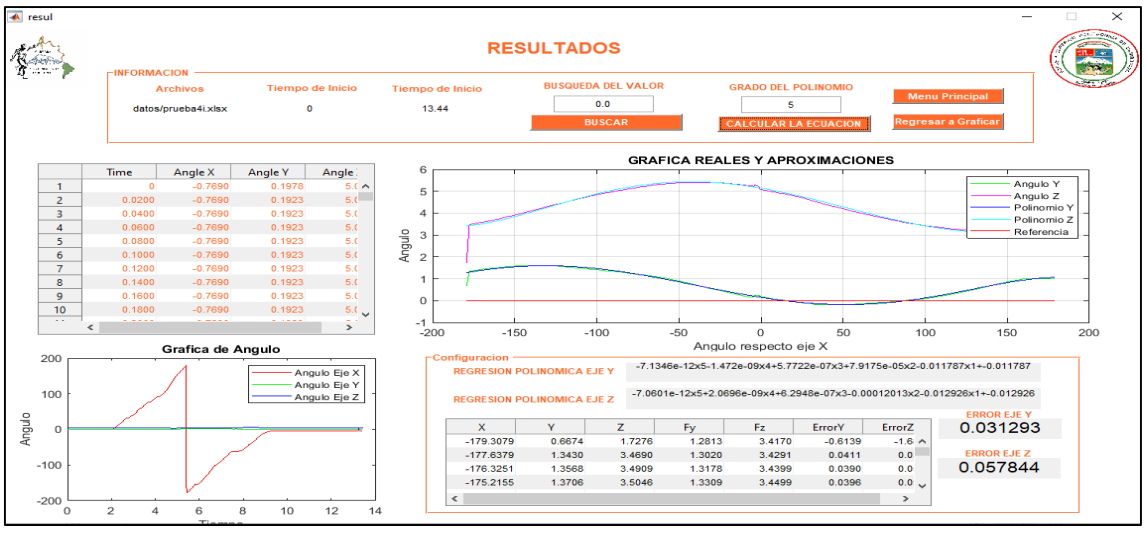


Gráfico 23-3. Curva característica de la cuarta prueba con grado de polinomio 5.

Fuente: (Del Pino, 2020)

La nueva regresión tiene resultados alentadores ya que solo con la gráfica de la ecuación y comprada con la gráfica real los datos se parecen mucho, llegando a tener un grado de error del 3,1% en el eje ***Y*** y 5,7% grado de error en el eje ***Z*** valores aceptables dentro de los parámetros planteados por ellos la ecuación de grado 5 se considera como ya la ecuación característica del eje, como se observa en la Grafica 23-3.

3.2.1.5. *Curva caracteritica de la quinta prueba con el sensor WT901BLECL*

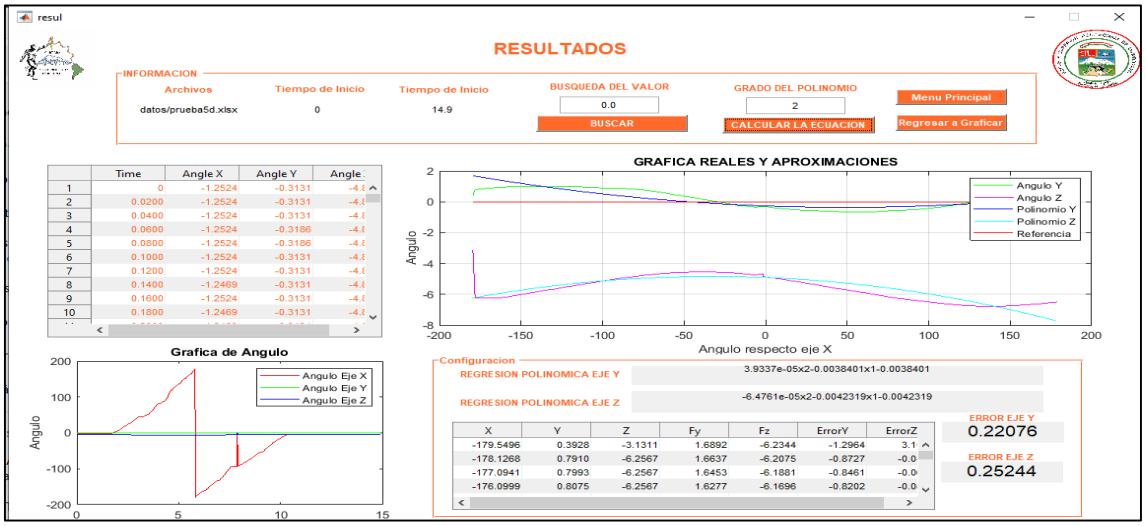


Gráfico 24-3. Curva característica de la cuarta prueba con grado de polinomio 2.

Fuente: (Del Pino, 2020)

Para la última prueba y cálculo de la ecuación de igual manera se aplica una regresión polinómica con un grado bajo y también se obtuvo valores parecidos ya que se aproxima a un error superior al 20% en casi todas las pruebas realizadas, por ello esta ecuación también es descartada como ecuación característica del eje, como se observa en la Gráfica 24-3.

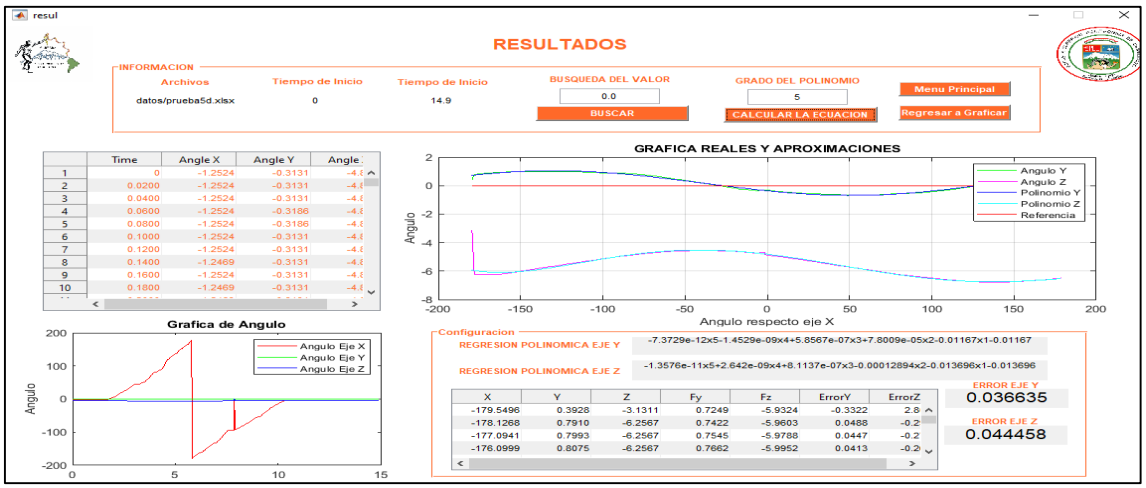


Gráfico 25-3. Curva característica de la cuarta prueba con grado de polinomio 5.

Fuente: (Del Pino, 2020)

La ecuación característica de esta prueba de igual manera resulto ser un polinomio de 5to grado ya que se obtuvo un grado de error de 3,6% en el eje Y y un error de 4,4% en el eje Z, siendo un error menor al 5% por lo cual se acepta como ecuación característica al polinomio obtenido mediante la regresión, como se observa en la Gráfica 25-3.

Todas la pruebas realizadas tienen un parecido tanto en el resultado como en el comportamiento del eje ya que en todas las desviaciones provocadas los valores oscilaban entre dichos valores causados y el valor era relativamente bajo, como se dijo anteriormente las causas de estas oscilaciones se debe a varios factores, ya sean mecánicos o causado por otros motivos, además se puede desatacar la precisión y la facilidad del uso del sensor, además un aspecto importante es la simulación en la cual nos muestra paso a paso el comportamiento en cada Ángulo de giro del eje del motor.

CONCLUSIONES

En este trabajo experimental se obtuvo la curva característica del movimiento de un eje de motor mediante los datos obtenidos por el sensor WT901BLECL, se pudo observar gráficamente cómo se comporta el eje en el espacio al dar una vuelta completa porque podemos ver y analizar en qué ángulo existe una falla o cabeceo, el sensor WT901BLECL fue el que ayudo a obtener la curva característica porque es un sensor que puede medir los ángulos, aceleración, velocidad angular, campo magnético, todos en los 3 ejes, siendo de gran precisión, de fácil uso, y de un costo bajo en comparación a sensores de alta precisión , lo complicado fue en que tiene una codificación diferente al catálogo y no puede leerse directamente con Matlab, por el momento, porque es un sensor bien actual y todavía no existe una librería para que se comunique con Matlab, otra complicación fue en calcular la ecuación características por la cantidad de datos siendo estos positivos y negativos.

El sensor ADXL 335 al ser un sensor analógico y según el Datasheet posee una sensibilidad del ±1% y en pruebas se vio que logra errores del 100%, en cambio el sensor WT901BLECL, y según el Datasheet tiene una sensibilidad 0.05 grados en los 3 ejes, y en pruebas se vio que tiene una precisión de más del 98% y con un error demasiado bajo del 2%.

El voltaje de trabajo que se utilizó para el sensor ADXL335 fue de 3,3 v, y según el voltaje de trabajo varia la sensibilidad del sensor si se utiliza menor a 3 voltios la sensibilidad es menor, se utilizó el software MATLAB como plataforma para la obtención de datos y la realización de las gráficas, y en las pruebas que se realizó experimentalmente se vio que con este sensor no se puede obtener la curva característica porque solo genera datos de aceleración, y se necesita también

datos de aceleración angular para poder generar las curvas, cabe recalcar que dicho sensor se usó solo para experimentos y selección del sensor adecuado, además los datos obtenidos experimentalmente los cuales no son alentadores como se pudo observar en las pruebas el error de medición pese a una calibración previa y a la aplicación de fórmulas del Datasheet es demasiado grande.

Cuando se realizó las pruebas con el sensor ADXL335 era un poco complicado debido a que el acople entre el sensor - microcontrolador eran a través de cables y el ordenador era a través de un cable usb que dificultaba girar completamente el sensor, se debería tener en cuenta usar un módulo bluetooth entre el Arduino y el sensor para que se facilite la toma de datos.

Cuando se realizó las pruebas con el sensor WT901BLECL, se hizo a través de conexión bluetooth entre el sensor y la computadora facilitando en gran manera a la hora de tomar los datos, siendo más fácil mover el sensor sin interrupciones de cables como en el caso del sensor ADXL335.

En las pruebas realizadas se provocó una desviación de 5 grados en dirección horizontal y vertical en el eje de pruebas, en el sensor ADXL335 se vio que no detecta esas desviaciones provocadas, en cambio en el sensor WT901BLECL si las detecta con un valor mínimo de diferencia que oscilaba máximo en 2 grados en todas las pruebas que se le hizo al sensor, ya sea en el eje ***Y*** o en el eje ***Z***.

En las gráficas de las curvas características que se hizo con el sensor WT901BLECL, analizando gráficamente se pudo observar que en algunos casos había oscilaciones provocadas, y en un motor eléctrico que tiene esas desviaciones quiere decir que se encuentra desalineado o hay alguna falla en esa desviación, siendo útil el sensor para el mantenimiento predictivo.

La precisión de la regresión polinómica es muy buena además de adaptiva ya que mediante la aplicación de Matlab se puede experimentar con la regresión para acoplarla a la mejor curva que se desea replicar.

Los errores obtenidos de la regresión polinómica de la señal del sensor WT901BLECL son bajos, se mantiene por debajo del 5% lo cual lo hace una ecuación con un grado de aceptación alto, además se utilizó el método de regresión polinómica por el motivo que en los datos se maneja con valores positivos y negativos, estos valores si son negativos los otros métodos existentes el logarítmico, exponencial u otros se tiene dificultad al calcular, para tener datos de muy alta

precisión se debería tomar en cuanta un sensor de más alta sensibilidad, y de más costo pues lo que se toma aquí son datos experimentales.

RECOMENDACIONES

Para la realización de este trabajo y como se pudo observar la utilización de un sensor IMU es esencial por ellos en el mercado existe sensores con mayor sensibilidad y elementos adicionales los cuales pueden adicionar nuevas herramientas a la aplicación desarrollada.

No se debería usar sensores de baja sensibilidad ya que estos tienen un grado de medición muy deficiente como se demostró en las pruebas realizadas.

Para obtener una regresión polinómica aceptable se recomienda usar un polinomio de grado superior al 4to grado para obtener resultados favorables.

BIBLIOGRAFÍA

AGGARWAL, P., SYED, Z., NOURELDIN, A., & EL-SHEIMY, N. (*MEMS-Based Integrated Navigation* [en línea], 2010. Disponible en: https://books.google.com.ec/books?hl=es&lr=&id=IfLdaZ_evpgC&oi=fnd&pg=PR5&dq=Aggarwal,+P.,+Syed,+Z.,+Noureldin,+A.,+%26+El-Sheimy,+N.+(2010).+MEMS-Based+Integrated+Navigation.&ots=KTG0OihExi&sig=YnoQGs_6WIC9o-rc3FAdUoenHZs#v=onepage&q&f=false

ANALOG DEVICES ADXL335, 2017. Small, Low Power, 3-Axis Accelerometer (ADXL335). Analog Device, Inc. Disponible en: https://benjad.github.io/2017/04/30/conexion-arduino-adxl/

ARDUINO – INTRODUCCIÓN, 2020.
Disponible en: https://www.arduino.cc/en/Guide/Introduction.

AVILA, J. IMU & AHRS ALGORITHMS, 2015.

MARTÍN, Teresa & SERRANO, Ana. *Magnetismo. Introducción* [blog]. [Consulta: 1 junio 2010]. Disponible en: http://www2.montes.upm.es/dptos/digfa/cfisica/magnet/tierraiman.html

BLASCARR. *Introducción al IMU - Sistemas de navegación Inercial* [blog], 2017. [Consulta: 18 julio 2010]. Disponible en: http://blascarr.com/lessons/introduccion-al-imu-sistemas-de-navegacion-inercial/

BLE, W. *Manual wt901blecl ble5.0. 1–30,* 2019.

CENTRO NACIONAL DE CAPACITACIÓN DE CELAYA (CENAC). (n.d.). *Alineación en ejes de maquinaria*.

CHACHA, Darwin. Determinación de perdidas energéticas que se producen en motores trifásicos asincrónicos tipo jaula de ardilla (maja3f) por la evolución de modos [En línea] (Trabajo de titulación). (Pregrado) Escuela Superior Politécnica de Chimborazo, Riobamba - Ecuador. 2016. [Consulta: 2020-07-15]. Disponible en: http://dspace.espoch.edu.ec/bitstream/123456789/6459/1/25T00284.pdf

CORONA RAMÍREZ, L. G., ABARCA JIMÉNEZ, G. S., & MARES CARREÑO, J. *Sensores y actuadores. Aplicaciones con Arduino* [En línea]. In Publicación En Internet (Issue October). 2014. Disponible en: https://doi.org/10.1007/s40722-017-0084-8

CUENCA, L., & LEÓN, J. Implementación de un sistema de navegación inercial, Para mejorar la precisión de posicionamiento de un prototipo GPS en una trayectoria dentro de la ESPOCH [En línea] (Trabajo de titulación). (Pregrado) Escuela Superior Politécnica de Chimborazo, Riobamba, Ecuador. 2017. [Consulta: 2020-06-20]. Disponible en: http://dspace.espoch.edu.ec/bitstream/123456789/7959/1/98T00176.pdf

DE LA CRUZ-ORÉ, J. L. "¿Qué significan los grados de libertad?". Revista Peruana de Epidemiología [en línea], 2013, (Perú) 17(1), pp. 1–6. [Consulta: 2020-06-08]. E-ISSN: 1609-7211. Disponible en: https://www.redalyc.org/pdf/2031/203129458002.pdf

5 HERTZ ELECTRONICA. (n.d.). *Sensores magnéticos* [blog]. [Consulta: 25 julio 2020]. Disponible en: https://www.5hertz.com/index.php?route=tutoriales/tutorial&tutorial_id=1

ESPINOZA PÁEZ, L. H. Sistema para monitorear el grado de curvatura de la columna vertebral, mediante la utilización de sensores IMU [En línea] (Trabajo de titulación). (Pregrado) Escuela Superior Politécnica de Chimborazo, Riobamba, Ecuador. 2017. [Consulta: 2020-07-07]. Disponible en: http://dspace.espoch.edu.ec/bitstream/123456789/9002/1/108T0236.pdf

GARCÍA, Sergio. Diseño y construcción de magnetómetro triaxial para análisis y experimentación de aislamientos magnéticos [En línea] (Trabajo de titulación). (Pregrado) Universidad Carlos III de Madrid, Madrid, España. 2013. [Consulta: 2020-09-13].
Disponible en: https://e-archivo.uc3m.es/bitstream/handle/10016/17038/pfc_sergio_garcia_garcia_2013.pdf?sequence=1&isAllowed=y

HERNÁNDEZ DÁVILA, E. S., MOYANO ARÉVALO, J. R., & BARAHONA ALVEAR, N. S. "Método para la alineación de ejes de máquinas rotacionales conformado por componentes de poliácido láctico". 2020, 6, pp. 975–999.

ROSALES, José. *Motores eléctricos para la industria. Motores eléctricos para la industria*, 45. 2012. [Consulta: 11 septiembre 2020]. Disponible en: https://doi.org/10.1108/13552550010362750

KOSOW, I. *Máquinas eléctricas y transformadores* [en línea]. Segunda edición. México D.F.: Prentice-Hall Hispanoamericana, S.A., 1993 [Consulta: 23 julio 2020]. Disponible en: https://books.google.com.ec/books?id=5hJzpimPyXQC&pg=PA538&dq=importancia+de+la+posicion+del+eje+en+los+motores+electricos&hl=es&sa=X&ved=2ahUKEwiynZjBla_qAhVth-AKHWq-BF8Q6AEwAHoECAMQAg#v=onepage&q=importancia de la posición del eje en los motores eléctricos

LAGLA, Byron, & LANCHE, Pablo. "Desarrollo de estándares de tolerancia de desalineamiento y su consumo energético de motores eléctricos en el laboratorio de análisis vibracional de la escuela de ingeniería de mantenimiento." [en línea] (Trabajo de titulación). (Pregrado) Escuela Superior Politécnica de Chimborazo, Riobamba - Ecuador. 2016. [Consulta: 2020-07-20]. Disponible en: http://dspace.espoch.edu.ec/bitstream/123456789/5179/1/25T00269.pdf

LAST MINUTE ENGINEERS. *How Accelerometer works? Interface ADXL335 with Arduino*, 2019 [blog]. Disponible en: https://lastminuteengineers.com/adxl335-accelerometer-arduino-tutorial/

LEMUS, I. *¿Qué es una tarjeta de desarrollo microcontrolador para IOT? - Conocimiento Libre*, 2019. Disponible en: https://conocimientolibre.mx/microcontrolador/

LUENGAS CONTRERAS, L., LÓPEZ ÁVILA, B., & JIMÉNEZ EZPINOZA, J. "Caracterización de unidades de medición inercial (IMUs) en estática y dinámica". Ingenio Magno [en línea], 2017, (Colombia) 8(1), pp. 92–102. [Consulta: 2020-06-24]. E-ISSN: 2422-2399. Disponible en: http://revistas.ustatunja.edu.co/index.php/ingeniomagno/article/view/1391/1287

DELGADO, Manuel. Arduino Mega 2560. Arduino En Español. 2016.

POZO ESPÍN, D. Diseño y construcción de una plataforma didáctica para medir ángulos de inclinación usando sensores inerciales como acelerómetro y giroscopio. 2010. Disponible en: http://bibdigital.epn.edu.ec/bitstream/15000/1794/1/CD-2772.pdf

PUJOS YANZAPANTA, M. Diseño e implementación de un prototipo para el control y tarifación del servicio de taxi a través de sensores inerciales [en línea] (Trabajo de titulación). (Pregrado) Escuela Superior Politécnica de Chimborazo, Riobamba - Ecuador. 2016. [Consulta: 2020-07-07]. Disponible en:

http://dspace.espoch.edu.ec/bitstream/123456789/9207/1/108T0251.pdf

PUJOS YANZAPANTA, M. 2016b. Diseño e implementación de una unidad de medición inercial "IMU" embebida en base a un sistema microcontrolado [en línea] (Trabajo de titulación). (Pregrado) Escuela Superior Politécnica de Chimborazo, Riobamba - Ecuador. 2016. [Consulta: 2020-07-21]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/15102

RASPBERRY PI. (n.d.). FAQs - Raspberry Pi Documentation. [Consulta: 2020-06-30]. Disponible en: https://www.raspberrypi.org/documentation/faqs/#pi-software

RASPBERRYPI. (N.D.). Buy a Raspberry Pi 4 Model B – Raspberry Pi. [Consulta: 2020-06-30]. Disponible en: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

REDROBÁN ARANDA, L. "Diseño e implementación de un prototipo para el control y tarifación del servicio de taxi a través de sensores inerciales.". 2018.

ROSERO, B., & MAZÓN, T. Diseño e implementación de un prototipo de robot subacuático autónomo dotado de un sistema de navegación inercial [en línea] (Trabajo de titulación). (Pregrado). Escuela Superior Politécnica de Chimborazo, Riobamba - Ecuador. 2019. [Consulta: 2020-08-07]. Disponible en: https://1library.co/document/zx5n4vdq-diseno-implementacion-prototipo-subacuatico-autonomo-sistema-navegacion-inercial.html

SAAVEDRA, P. N. (n.d.). Tutorial: Diagnóstico del desalineamiento de ejes en máquinas acopladas a través del análisis. 1–34. [Consulta: 22 agosto 2020]. Disponible en: https://docplayer.es/24732260-Diagnostico-del-desalineamiento-de-ejes-en-maquinas-acopladas-a-traves-del-analisis-de-vibraciones-p-n-saavedra.html

SALAZAR ABALCO, Steven, & SERRANO CRIOLLO, Karina. Implementación de un prototipo para el monitoreo sísmico utilizando acelerómetros de bajo costo [en línea] (Trabajo de titulación) (Pregrado). Escuela Superior Politécnica de Chimborazo, Riobamba - Ecuador. 2018. [Consulta: 2021-01-15]. Disponible en: https://bibdigital.epn.edu.ec/bitstream/15000/19626/1/CD-9029.pdf

SANCHEZ MARIN, F. T., PÉREZ GONZÁLEZ, A., SANCHO BRU, J. L., & RODRIGUEZ CERVANTES, P. J. *Mantenimiento mecánico de máquinas*. [en línea]. Segunda edición. 2007. [Consulta: 28 julio 2020]. Disponible en: https://elibro.net/es/lc/espoch/titulos/42317

UNIVERSIDAD DE SEVILLA. Sensor medidor de Aceleración. 2017. Disponible en: http://bibing.us.es/proyectos/abreproy/11638/fichero/Capitulo+4.pdf

SPARKFUN. Triple Axis Accelerometer Breakout - ADXL330 - SEN-00692 - SparkFun Electronics. 2011. Disponible en: https://www.sparkfun.com/products/retired/692

VILLALUENGA MORÁN, José. Uso de acelerómetros para el control de dispositivos mediante captura de movimiento [en línea] (Trabajo de titulación) (Maestría). Universitat Oberta de Catalunya y Universitat Ramon Llull, Barcelona – España. 2015. [Consulta: 2021-02-05]. Disponible en: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/42670/7/jvillaluengaTFM0615memoria.pdf

WELCH, G., & BISHOP, G. 2006. An Introduction to the Kalman Filter. 1–16.

WITMOTION SHENZHEN CO. Bluetooth AHRS IMU sensor | WT901BLECL. 2019. [Consulta: 18 junio 2020]. Disponible en: http://wiki.wit-motion.com/english

Firmado electrónicamente por:
JHONATAN RODRIGO
PARREÑO UQUILLAS

ANEXOS

ANEXO A: PROGRAMACIÓN DE LA APLICACIÓN DE MATLAB VERSIÓN R2019A.

menuP.m

```
function varargout = menuP(varargin)
% MENUP MATLAB code for menuP.fig
%      MENUP, by itself, creates a new MENUP or raises
the existing
%      singleton*.
%
%      H = MENUP returns the handle to a new MENUP or
the handle to
%      the existing singleton*.
%
%      MENUP('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in MENUP.M with the given
input arguments.
%
%      MENUP('Property','Value',...) creates a new MENUP
or raises the
%      existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before menuP_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value makes
property application
%      stop.  All inputs are passed to menuP_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
menuP

% Last Modified by GUIDE v2.5 22-Nov-2020 23:07:16

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn',
@menuP_OpeningFcn, ...
                   'gui_OutputFcn',   @menuP_OutputFcn,
...
                   'gui_LayoutFcn',   [] , ...
                   'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]   =   gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before menuP is made visible.
function menuP_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin     command line arguments to menuP (see
VARARGIN)

% Choose default command line output for menuP
handles.output = hObject;
axes(handles.ima1)
handles.imagen=imread('imagenes/espoch.jpg');
imagesc(handles.imagen)
axis off
axes(handles.ima2)
handles.imagen=imread('imagenes/escuela.png');
imagesc(handles.imagen)
axis off
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes menuP wait for user response (see
UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the
command line.
function    varargout    =    menuP_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in datos.
function datos_Callback(hObject, eventdata, handles)
% hObject    handle to datos (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
tomadatos

% --- Executes on button press in graficar.
function graficar_Callback(hObject, eventdata, handles)
% hObject    handle to graficar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
grafica


% --- Executes on button press in adx.
function adx_Callback(hObject, eventdata, handles)
% hObject    handle to adx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
```

```
delete(guide);
adxl
```

tomadatos.m

```
function varargout = tomadatos(varargin)
% TOMADATOS MATLAB code for tomadatos.fig
%      TOMADATOS, by itself, creates a new TOMADATOS or
raises the existing
%      singleton*.
%
%         H = TOMADATOS returns the handle to a new
TOMADATOS or the handle to
%      the existing singleton*.
%
%
TOMADATOS('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in TOMADATOS.M with the
given input arguments.
%
%      TOMADATOS('Property','Value',...) creates a new
TOMADATOS or raises the
%       existing singleton*.  Starting from the left,
property value pairs are
%       applied to the GUI before tomadatos_OpeningFcn
gets called.  An
%     unrecognized property name or invalid value makes
property application
%               stop.     All   inputs   are   passed   to
tomadatos_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
tomadatos

% Last Modified by GUIDE v2.5 25-Jul-2020 01:01:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
```

```
                   'gui_OpeningFcn',
@tomadatos_OpeningFcn, ...
                   'gui_OutputFcn',
@tomadatos_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]   =   gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tomadatos is made visible.
function    tomadatos_OpeningFcn(hObject,    eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to tomadatos (see
VARARGIN)

% Choose default command line output for tomadatos
handles.output = hObject;
axes(handles.ima1)
handles.imagen=imread('imagenes/espoch.jpg');
imagesc(handles.imagen)
axis off
axes(handles.ima2)
handles.imagen=imread('imagenes/escuela.png');
imagesc(handles.imagen)
axis off
arch=dir('datosIMU')
[m n]=size(arch)
d=[]
for i=3:m
    d=[d strcat(arch(i).name,"")]
end
```

```
set(handles.archivos,'String',d)
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tomadatos wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
function varargout = tomadatos_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
menuP


% --- Executes on mouse press over figure background.
function figure1_ButtonDownFcn(hObject, eventdata,
handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata,
handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String'))
returns popupmenu1 contents as cell array
%       contents{get(hObject,'Value')} returns selected
item from popupmenu1


% --- Executes during object creation, after setting
all properties.
function popupmenu1_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: popupmenu controls usually have a white
background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in archivos.
function archivos_Callback(hObject, eventdata, handles)
% hObject    handle to archivos (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String'))
returns archivos contents as cell array
%       contents{get(hObject,'Value')} returns selected
item from archivos
```

```
% --- Executes during object creation, after setting
all properties.
function     archivos_CreateFcn(hObject,     eventdata,
handles)
% hObject    handle to archivos (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles     empty - handles not created until after
all CreateFcns called

% Hint: popupmenu controls usually have a white
background on Windows.
%       See ISPC and COMPUTER.
if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
edit1 as text
%            str2double(get(hObject,'String')) returns
contents of edit1 as a double


% --- Executes during object creation, after setting
all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles     empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
edit2 as text
%        str2double(get(hObject,'String')) returns
contents of edit2 as a double


% --- Executes during object creation, after setting
all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global datos
nom=get(handles.nombre,'String');
if not(strcmp(nom,''))
    nom=strcat(nom,'.xlsx');
    nom=strcat('datos/',nom);
```

```
    xlswrite(nom,datos)
    sms = msgbox('Datos Guardados','Exito');
else
 sms = msgbox('Ingrese un Nombre correcto','ERROR');
end


% --- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
% hObject    handle to menu (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
menuP



% --- Executes on button press in cargar.
function cargar_Callback(hObject, eventdata, handles)
% hObject    handle to cargar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global datos
archiv=get(handles.archivos,'String');
archiv=archiv{get(handles.archivos,'Value')}
archiv=strcat('datosIMU/',archiv);
A=readtable(archiv);
Tiempo= A(:,2);
gyrX = A(:,6);
gyrY = A(:,7);
gyrZ = A(:,8);
accX = A(:,3);
accY = A(:,4);
accZ = A(:,5);
angleX= A(:,9);
angleY= A(:,10);
angleZ= A(:,11);
[m n]=size(Tiempo);
T=array2table([0:1/50:(m-1)*(1/50)]');
AN=[T  gyrX  gyrY  gyrZ  accX  accY  accZ  angleX  angleY
angleZ];
An=table2cell(AN);
```

```
datos=An;
set( handles.uitable1,'Data',An)
msgbox('Datos Cargados exitosamente ','AVISO');



function nombre_Callback(hObject, eventdata, handles)
% hObject    handle to nombre (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
nombre as text
%        str2double(get(hObject,'String')) returns
contents of nombre as a double


% --- Executes during object creation, after setting
all properties.
function nombre_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nombre (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

grafica.m

```
function varargout = grafica(varargin)
% GRAFICA MATLAB code for grafica.fig
%       GRAFICA, by itself, creates a new GRAFICA or
raises the existing
%      singleton*.
%
%      H = GRAFICA returns the handle to a new GRAFICA
or the handle to
%      the existing singleton*.
%
```

```
%
GRAFICA('CALLBACK',hObject,eventData,handles,...)
calls the local
%       function named CALLBACK in GRAFICA.M with the
given input arguments.
%
%       GRAFICA('Property','Value',...) creates a new
GRAFICA or raises the
%       existing singleton*.  Starting from the left,
property value pairs are
%     applied to the GUI before grafica_OpeningFcn gets
called.  An
%     unrecognized property name or invalid value makes
property application
%     stop.  All inputs are passed to grafica_OpeningFcn
via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
grafica

% Last Modified by GUIDE v2.5 06-Apr-2020 16:19:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@grafica_OpeningFcn, ...
                   'gui_OutputFcn',
@grafica_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]   =   gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
```

```
end
% End initialization code - DO NOT EDIT


% --- Executes just before grafica is made visible.
function grafica_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to grafica (see
VARARGIN)

% Choose default command line output for grafica
handles.output = hObject;
axes(handles.ima1)
handles.imagen=imread('imagenes/espoch.jpg');
imagesc(handles.imagen)
axis off
axes(handles.ima2)
handles.imagen=imread('imagenes/escuela.png');
imagesc(handles.imagen)
axis off
arch=dir('datos')
[m n]=size(arch)
d=[]

for i=3:m
    d=[d strcat(arch(i).name,"")]
end
set(handles.archivos,'String',d)
    h=handles.barra;
    set(h,'Visible','on','color',[0.7 0.7 0.7]);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes grafica wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
```

```
function varargout = grafica_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in archivos.
function archivos_Callback(hObject, eventdata, handles)
% hObject    handle to archivos (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global angleX angleY angleZ
archiv=get(handles.archivos,'String');
archiv=archiv{get(handles.archivos,'Value')}
archiv=strcat('datos/',archiv)
cla(handles.axesT1)
cla(handles.axesT2)
cla(handles.axesP)
cla(handles.barra)

if(1)
try
samplePeriod = 1/50;
[datos,txt,cell] = xlsread(archiv);
time = datos(:,1);
gyrX = datos(:,2);
gyrY = datos(:,3);
gyrZ = datos(:,4);
accX = datos(:,5);
accY = datos(:,6);
accZ = datos(:,7);
angleX= datos(:,8);
angleY= datos(:,9);
angleZ= datos(:,10);
set(handles.tfinal,'String',num2str(max(time)));
startTime=str2num(get(handles.tinicio,'String'));
stopTime=str2num(get(handles.tfinal,'String'));
```

```
indexSel    =    find(sign(time-startTime)+1,    1)    :
find(sign(time-stopTime)+1, 1);
time = time(indexSel);
gyrX = gyrX(indexSel);
gyrY = gyrY(indexSel);
gyrZ = gyrZ(indexSel);
accX = accX(indexSel);
accY = accY(indexSel);
accZ = accZ(indexSel);
angleX= angleX(indexSel);
angleY = angleY(indexSel);
angleZ = angleZ(indexSel);
acc_mag = sqrt(accX.*accX + accY.*accY + accZ.*accZ);
% HP filter accelerometer data
filtCutOff = 0.0001;
[b,  a]  =  butter(1,  (2*filtCutOff)/(1/samplePeriod),
'high');
acc_magFilt = filtfilt(b, a, acc_mag);
% Compute absolute value
acc_magFilt = abs(acc_magFilt);
% LP filter accelerometer data
filtCutOff = 5;
[b,  a]  =  butter(1,  (2*filtCutOff)/(1/samplePeriod),
'low');
acc_magFilt = filtfilt(b, a, acc_magFilt);
% Threshold detection
stationary = acc_magFilt < 0.055;
% ------------------------------------------------
--------------------
% Plot data raw sensor data and stationary periods
    axes(handles.axesT1)
    hold on;
    plot(time, gyrX, 'r');
    plot(time, gyrY, 'g');
    plot(time, gyrZ, 'b');
    title('GIROSCOPIO');
    xlabel('Time (s)');
    ylabel('Velocidad Angular (^\circ/s)');
    legend('X', 'Y', 'Z');
    hold off;
    axes(handles.axesT2)
    hold on;
    plot(time, accX, 'r');
    plot(time, accY, 'g');
    plot(time, accZ, 'b');
    plot(time, acc_magFilt, ':k');
    plot(time, stationary, 'k', 'LineWidth', 2);
```

```
    title('ACELEROMETRO');
    xlabel('Tiempo (s)');
    ylabel('Aceleracion (g)');
    legend('X', 'Y', 'Z', 'Filtrado', 'Estacionario');
    hold off;
    answer     =     questdlg('Desea     iniciar     la
Simulacion?','Finalizar','yes');
if(strcmp(answer,'Yes'))
   graficar_Callback(hObject, eventdata, handles)
end
catch
   msgbox('Error al cargar el archivo o definir los
limite del tiempo.','Error');
end
end

% Hints: contents = cellstr(get(hObject,'String'))
returns archivos contents as cell array
%       contents{get(hObject,'Value')} returns selected
item from archivos


% --- Executes during object creation, after setting
all properties.
function     archivos_CreateFcn(hObject,     eventdata,
handles)
% hObject    handle to archivos (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: popupmenu controls usually have a white
background on Windows.
%       See ISPC and COMPUTER.
if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function tinicio_Callback(hObject, eventdata, handles)
% hObject    handle to tinicio (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of
tinicio as text
%        str2double(get(hObject,'String')) returns
contents of tinicio as a double


% --- Executes during object creation, after setting
all properties.
function tinicio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tinicio (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function tfinal_Callback(hObject, eventdata, handles)
% hObject    handle to tfinal (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
tfinal as text
%        str2double(get(hObject,'String')) returns
contents of tfinal as a double


% --- Executes during object creation, after setting
all properties.
function tfinal_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tfinal (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called
```

```
% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in graficar.
function graficar_Callback(hObject, eventdata, handles)
% hObject    handle to graficar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global angleX angleY angleZ
    h=handles.barra;
    set(h,'Visible','on','color',[0.7 0.7 0.7]);
    axes(h);
    axis([0,1,0,1]);
    tp   =   theaterPlot('XLimit',[-2   2],'YLimit',[-2
2],'ZLimit',[-2 2],'Parent',handles.axesP);
    op                                                  =
orientationPlotter(tp,'DisplayName','Eje','LocalAxesLe
ngth',2);
for i=1:numel(angleZ)
    plotOrientation(op,angleX(i),angleY(i),angleZ(i))
    value=i/numel(angleZ);

patch([0,value,value,0],[0,0,1,1],'g','Parent',handles
.barra);

set(handles.porciento,'String',strcat(num2str(floor(va
lue*100)),'%'));
    drawnow
end

set(handles.resultado,'Enable','on')
answer       =       questdlg('Desea       ver       los
Resultados?','Finalizo','yes');
if(strcmp(answer,'Yes'))
   resultado_Callback(hObject, eventdata, handles)
end


% --- Executes on button press in resultado.
```

```
function     resultado_Callback(hObject,     eventdata,
handles)
% hObject    handle to resultado (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
archiv=get(handles.archivos,'String');
archiv=archiv{get(handles.archivos,'Value')}
archiv=strcat('datos/',archiv)
startTime = str2num(get(handles.tinicio,'String'));
stopTime = str2num(get(handles.tfinal,'String'));
guide=get(0,'CurrentFigure');
delete(guide);
resul(archiv,startTime,stopTime)



% --- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
% hObject    handle to menu (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
menuP
```

resul.m

```
function varargout = resul(varargin)
% RESUL MATLAB code for resul.fig
%      RESUL, by itself, creates a new RESUL or raises
the existing
%      singleton*.
%
%      H = RESUL returns the handle to a new RESUL or
the handle to
%      the existing singleton*.
%
%      RESUL('CALLBACK',hObject,eventData,handles,...)
calls the local
%     function named CALLBACK in RESUL.M with the given
input arguments.
%
%     RESUL('Property','Value',...) creates a new RESUL
or raises the
```

```
%       existing singleton*.  Starting from the left,
property value pairs are
%       applied to the GUI before resul_OpeningFcn gets
called.  An
%     unrecognized property name or invalid value makes
property application
%       stop.  All inputs are passed to resul_OpeningFcn
via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%       instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
resul

% Last Modified by GUIDE v2.5 30-Oct-2020 23:51:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@resul_OpeningFcn, ...
                   'gui_OutputFcn',   @resul_OutputFcn,
...
                   'gui_LayoutFcn',   [] , ...
                   'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]   =   gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before resul is made visible.
function resul_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin    command line arguments to resul (see
VARARGIN)

% Choose default command line output for resul
global time angleX angleY angleZ
handles.output = hObject;
axes(handles.ima1)
handles.imagen=imread('imagenes/espoch.jpg');
imagesc(handles.imagen)
axis off
axes(handles.ima2)
handles.imagen=imread('imagenes/escuela.png');
imagesc(handles.imagen)
axis off
cla(handles.axes1)
set(handles.archivo,'String',varargin{1})
set(handles.tiempoI,'String',num2str(varargin{2}))
set(handles.tiempoF,'String',num2str(varargin{3}))
archiv=get(handles.archivo,'String');
startTime = str2num(get(handles.tiempoI,'String'));
stopTime = str2num(get(handles.tiempoF,'String'));
[datos,txt,cell] = xlsread(archiv);
time = datos(:,1);
angleX= datos(:,8);
angleY= datos(:,9);
angleZ= datos(:,10);
indexSel    =    find(sign(time-startTime)+1,    1)    :
find(sign(time-stopTime)+1, 1);
time = time(indexSel);
angleX= angleX(indexSel);
angleY = angleY(indexSel);
angleZ = angleZ(indexSel);
dat=[time angleX angleY angleZ];
set(handles.uitable1,'data',dat)
axes(handles.axes1)
plot(time,angleX,'r')
hold on
plot(time,angleY,'g')
plot(time,angleZ,'b')
title('Grafica de Angulo')
legend('Angulo Eje X','Angulo Eje Y','Angulo Eje Z')
xlabel('Tiempo')
ylabel('Angulo')
```

```
k=1;
n=length(angleX);
while k<=n
j=1;
while j<=n
if k~=j
if angleX(k)==angleX(j)
angleX(j)=[];
angleY(j)=[];
angleZ(j)=[];
n=length(angleX);
end
end
j=j+1;
end
k=k+1;
end
data=[angleX angleY angleZ];
[~, s] = sort(data(:, 1));
data=data(s, :);
angleX=data(:, 1);
angleY=data(:, 2);
angleZ=data(:, 3);
angleY = medfilt1(angleY,30);
angleZ = medfilt1(angleZ,30);

axes(handles.axes5)
plot(angleX,angleY,'g')
hold on
plot(angleX,angleZ,'m')
hold on
XR=zeros(length(angleX));
plot(angleX,XR,'r')
title('GRAFICA CON RESPECTO AL EJE DE REFERENCIA')
legend('Angulo Y ','Angulo Z ','Referencia')
xlabel('Angulo respecto al eje X')
ylabel('Angulo')


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes resul wait for user response (see
UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the
command line.
function    varargout    =    resul_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in start.
function start_Callback(hObject, eventdata, handles)
% hObject    handle to start (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global angleY angleX angleZ T

x=angleX;
y=angleY;
yp=angleZ;
cla(handles.axes5)
axes(handles.axes5)
plot(x,y,'g')
hold on
plot(angleX,angleZ,'m')

grado=str2num(get(handles.grado,'String'));
p=polyfit(x,y,grado);
z=@(x) polyval(p,x);
fplot(z,[x(1),x(end)],'b')


p1=polyfit(x,yp,grado);
z1=@(x) polyval(p1,x);
fplot(z1,[x(1),x(end)],'c')

XR=zeros(length(angleX));
plot(angleX,XR,'r')

f = polyval(p,x);
```

```
fp = polyval(p1,x);

T = [x y  yp f fp y-f yp-fp];

error=abs(y-f);
error= sum(error)/length(error);

error1=abs(yp-fp);
error1= sum(error1)/length(error1);

set(handles.uitable2,'data',T)

set(handles.error,'String',num2str(error))
set(handles.error1,'String',num2str(error1))

text=strcat(num2str(p(1)),'x',num2str((length(p)-1)))
for c = 2:length(p)-1
    if(p(c)<0)

text=strcat(text,num2str(p(c)),'x',num2str(length(p)-
c))
    else

text=strcat(text,'+',num2str(p(c)),'x',num2str(length(
p)-c))
    end
end
if(p(length(p))<0)
   text=strcat(text,num2str(p(c)))
else
    text=strcat(text,'+',num2str(p(c)))
end
set(handles.pol,'String',text)

p=p1;
text=strcat(num2str(p(1)),'x',num2str((length(p)-1)))
for c = 2:length(p)-1
    if(p(c)<0)

text=strcat(text,num2str(p(c)),'x',num2str(length(p)-
c))
    else

text=strcat(text,'+',num2str(p(c)),'x',num2str(length(
p)-c))
    end
end
if(p(length(p))<0)
```

```
   text=strcat(text,num2str(p(c)))
else
    text=strcat(text,'+',num2str(p(c)))
end
set(handles.pol1,'String',text)

title('GRAFICA REALES Y APROXIMACIONES')
legend('Angulo Y ','Angulo Z ','Polinomio Y','Polinomio
Z','Referencia')
grid on
xlabel('Angulo respecto eje X')
ylabel('Angulo')
msgbox('Solo  se  calculo  Aproximaciones  Polinomicas
','AVISO');
set(handles.valor,'Enable','on')
set(handles.buscar,'Enable','on')
set(handles.Tbuscar,'Enable','on')
```

```
% --- Executes on button press in graficaRegresar.
function  graficaRegresar_Callback(hObject,  eventdata,
handles)
% hObject    handle to graficaRegresar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
grafica
```

```
% --- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
% hObject    handle to menu (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
```

menuP

```
function nombre_Callback(hObject, eventdata, handles)
% hObject    handle to nombre (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
nombre as text
%        str2double(get(hObject,'String')) returns
contents of nombre as a double


% --- Executes during object creation, after setting
all properties.
function nombre_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nombre (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in guardar.
function guardar_Callback(hObject, eventdata, handles)
% hObject    handle to guardar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global datos
nom=get(handles.nombre,'String');
if not(strcmp(nom,''))
    nom=strcat(nom,'.xlsx');
    nom=strcat('datosPos/',nom);
    xlswrite(nom,datos)
    sms = msgbox('Datos Guardados','Exito');
```

```
else
 sms = msgbox('Ingrese un Nombre correcto','ERROR');
end



function grado_Callback(hObject, eventdata, handles)
% hObject    handle to grado (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
grado as text
%        str2double(get(hObject,'String')) returns
contents of grado as a double


% --- Executes during object creation, after setting
all properties.
function grado_CreateFcn(hObject, eventdata, handles)
% hObject    handle to grado (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)


% --- Executes on button press in pushbutton9.
```

```
function    pushbutton9_Callback(hObject,    eventdata,
handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)


% --- Executes on button press in pushbutton7.
function    pushbutton7_Callback(hObject,    eventdata,
handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)


% --- Executes on button press in buscar.
function buscar_Callback(hObject, eventdata, handles)
% hObject    handle to buscar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global T
val=str2num(get(handles.valor,'string'))
y=T(:,1);
k = find(abs(y-val) < 1);
Valores=T(k,:)
tabladatos(Valores)



function valor_Callback(hObject, eventdata, handles)
% hObject    handle to valor (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
valor as text
%        str2double(get(hObject,'String')) returns
contents of valor as a double
```

```
% --- Executes during object creation, after setting
all properties.
function valor_CreateFcn(hObject, eventdata, handles)
% hObject    handle to valor (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if   ispc   &&   isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

adxl.m

```
function varargout = adxl(varargin)
% ADXL MATLAB code for adxl.fig
%       ADXL, by itself, creates a new ADXL or raises
the existing
%      singleton*.
%
%      H = ADXL returns the handle to a new ADXL or the
handle to
%      the existing singleton*.
%
%       ADXL('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in ADXL.M with the given
input arguments.
%
%      ADXL('Property','Value',...) creates a new ADXL
or raises the
%       existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before adxl_OpeningFcn gets
called.  An
%     unrecognized property name or invalid value makes
property application
%      stop.  All inputs are passed to adxl_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
```

```
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
adxl

% Last Modified by GUIDE v2.5 22-Nov-2020 22:31:19

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',  @adxl_OpeningFcn,
...
                   'gui_OutputFcn',    @adxl_OutputFcn,
...
                   'gui_LayoutFcn',   [] , ...
                   'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]   =   gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before adxl is made visible.
function  adxl_OpeningFcn(hObject,  eventdata,  handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
%  varargin     command  line  arguments  to  adxl  (see
VARARGIN)

% Choose default command line output for adxl
handles.output = hObject;
axes(handles.ima1)
handles.imagen=imread('imagenes/espoch.jpg');
```

```
imagesc(handles.imagen)
axis off
axes(handles.ima2)
handles.imagen=imread('imagenes/escuela.png');
imagesc(handles.imagen)
axis off
puertos=getAvailableComPort();
set(handles.puertocom,'string',puertos)
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes adxl wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
function varargout = adxl_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in puertocom.
function     puertocom_Callback(hObject,     eventdata,
handles)
% hObject    handle to puertocom (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% Hints: contents = cellstr(get(hObject,'String'))
returns puertocom contents as cell array
%        contents{get(hObject,'Value')} returns selected
item from puertocom


% --- Executes during object creation, after setting
all properties.
```

```
function    puertocom_CreateFcn(hObject,    eventdata,
handles)
% hObject    handle to puertocom (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: popupmenu controls usually have a white
background on Windows.
%       See ISPC and COMPUTER.
if  ispc  &&  isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function tinicio_Callback(hObject, eventdata, handles)
% hObject    handle to tinicio (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
tinicio as text
%            str2double(get(hObject,'String')) returns
contents of tinicio as a double


% --- Executes during object creation, after setting
all properties.
function tinicio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tinicio (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if  ispc  &&  isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function nombre_Callback(hObject, eventdata, handles)
% hObject    handle to nombre (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
nombre as text
%        str2double(get(hObject,'String')) returns
contents of nombre as a double


% --- Executes during object creation, after setting
all properties.
function nombre_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nombre (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in graficar.
function graficar_Callback(hObject, eventdata, handles)
% hObject    handle to graficar (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global datos

cla(handles.axesP)
cla(handles.axesP2)
cla(handles.barra)
com=get(handles.puertocom,'String');
puerto=com{get(handles.puertocom,'Value')}
hq=str2num(get(handles.muestras,'String'))
```

```
delete(instrfind({'Port'},{puerto}));
%crear objeto serie
s                                                    =
serial(puerto,'BaudRate',9600,'Terminator','CR/LF');
set(s,'Timeout',5); % 5 segundos de tiempo de espera
warning('off','MATLAB:serial:fscanf:unsuccessfulRead')
;

%abrir puerto
fopen(s);
x1=[];
x2=[];
y1=[];
y2=[];
z1=[];
z2=[];
i = 1;
msgbox('Espere hasta que la aplicacion tome los
datos','AVISO');
h=handles.barra;
set(h,'Visible','on','color',[0.7 0.7 0.7]);
axes(h);
axis([0,1,0,1]);
while i<=hq
    % leer del puerto serie
    a = fscanf(s,'%d,%d,%d,%d,%d,%d')
    x1=[x1 a(1)];
    y1=[y1 a(2)];
    z1=[z1 a(3)];
    x2=[x2 a(4)];
    y2=[y2 a(5)];
    z2=[z2 a(6)];
    i = i+1;
    datos=[x1' y1' z1' x2' y2' z2'];
    set(handles.uitable1,'data',datos)
    pause(0.3)
    value=(i-1)/hq;

patch([0,value,value,0],[0,0,1,1],'g','Parent',handles
.barra);

set(handles.nivel,'String',strcat(num2str(floor(value*
100)),'%'));
    drawnow
end
msgbox('Toma de datos terminada','AVISO');
delete(instrfind({'Port'},{puerto}));
```

```
delete(instrfind({'Port'},{'COM6'}));

axes(handles.axesP)
plot(y1,x1,'r')
hold on
plot(y1,z1,'b')
title('Grafica Sensor 1')
legend('Eje X','Eje Z')
axes(handles.axesP2)
plot(y2,x2,'r')
hold on
plot(y2,z2,'b')
title('Grafica Sensor 2')
legend('Eje X','Eje Z')



% --- Executes on button press in resultado.
function resultado_Callback(hObject, eventdata,
handles)
% hObject    handle to resultado (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global datos
nom=get(handles.nombre,'String');
if not(strcmp(nom,''))
    nom=strcat(nom,'.xlsx');
    nom=strcat('datosAdxl/',nom);
    xlswrite(nom,datos)
    sms = msgbox('Datos Guardados','Exito');
else
 sms = msgbox('Ingrese un Nombre correcto','ERROR');
end



% --- Executes on button press in menu.
function menu_Callback(hObject, eventdata, handles)
% hObject    handle to menu (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
puerto=get(handles.puertocom,'String');
```

```
delete(instrfind({'Port'},{puerto}));
guide=get(0,'CurrentFigure');
delete(guide);
menuP



function muestras_Callback(hObject, eventdata, handles)
% hObject    handle to muestras (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
muestras as text
%        str2double(get(hObject,'String')) returns
contents of muestras as a double


% --- Executes during object creation, after setting
all properties.
function muestras_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to muestras (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

getAvailableComPort.m

```
function lCOM_Port = getAvailableComPort()
% function lCOM_Port = getAvailableComPort()
% Return a Cell Array of COM port names available on
your computer

try
    s=serial('IMPOSSIBLE_NAME_ON_PORT');fopen(s);
catch
```

```
    lErrMsg = lasterr;
end

%Start of the COM available port
lIndex1 = findstr(lErrMsg,'COM');
%End of COM available port
lIndex2 = findstr(lErrMsg,'Use')-3;

lComStr = lErrMsg(lIndex1:lIndex2);

%Parse the resulting string
lIndexDot = findstr(lComStr,',');

% If no Port are available
if isempty(lIndex1)
    lCOM_Port{1}='';
    return;
end

% If only one Port is available
if isempty(lIndexDot)
    lCOM_Port{1}=lComStr;
    return;
end

lCOM_Port{1} = lComStr(1:lIndexDot(1)-1);

for i=1:numel(lIndexDot)+1
    % First One
    if (i==1)
        lCOM_Port{1,1} = lComStr(1:lIndexDot(i)-1);
    % Last One
    elseif (i==numel(lIndexDot)+1)
        lCOM_Port{i,1}      =      lComStr(lIndexDot(i-
1)+2:end);
    % Others
    else
        lCOM_Port{i,1}      =      lComStr(lIndexDot(i-
1)+2:lIndexDot(i)-1);
    end
end
```

SixDofAnimation.m

```
function fig = SixDOFanimation(varargin)

    %% Create local variables
```

```
    % Required arguments
    p = varargin{1};                % position of body
    R = varargin{2};                 % rotation matrix
of body
    [numSamples dummy] = size(p);

    % Default values of optional arguments
    SamplePlotFreq = 1;
    Trail = 'Off';
    LimitRatio = 1;
    Position = [];
    FullScreen = false;
    View = [30 20];
    AxisLength = 1;
    ShowArrowHead = 'on';
    Xlabel = 'X';
    Ylabel = 'Y';
    Zlabel = 'Z';
    Title = '6DOF Animation';
    ShowLegend = true;
    CreateAVI = false;
    AVIfileName = '6DOF Animation';
    AVIfileNameEnum = true;
    AVIfps = 30;

    for i = 3:2:nargin
        if     strcmp(varargin{i},   'SamplePlotFreq'),
SamplePlotFreq = varargin{i+1};
        elseif  strcmp(varargin{i}, 'Trail')
            Trail = varargin{i+1};
            if(~strcmp(Trail, 'Off') && ~strcmp(Trail,
'DotsOnly') && ~strcmp(Trail, 'All'))
                error('Invalid argument.  Trail must be
''Off'', ''DotsOnly'' or ''All''.');
            end
        elseif     strcmp(varargin{i},   'LimitRatio'),
LimitRatio = varargin{i+1};
        elseif      strcmp(varargin{i},    'Position'),
Position = varargin{i+1};
        elseif     strcmp(varargin{i},   'FullScreen'),
FullScreen = varargin{i+1};
        elseif    strcmp(varargin{i},  'View'),  View  =
varargin{i+1};
        elseif     strcmp(varargin{i},   'AxisLength'),
AxisLength = varargin{i+1};
        elseif   strcmp(varargin{i},  'ShowArrowHead'),
ShowArrowHead = varargin{i+1};
```

```
        elseif  strcmp(varargin{i}, 'Xlabel'), Xlabel =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'Ylabel'), Ylabel =
varargin{i+1};
        elseif  strcmp(varargin{i}, 'Zlabel'), Zlabel =
varargin{i+1};
        elseif   strcmp(varargin{i}, 'Title'), Title =
varargin{i+1};
        elseif     strcmp(varargin{i},   'ShowLegend'),
ShowLegend = varargin{i+1};
        elseif      strcmp(varargin{i},   'CreateAVI'),
CreateAVI = varargin{i+1};
        elseif    strcmp(varargin{i},  'AVIfileName'),
AVIfileName = varargin{i+1};
        elseif  strcmp(varargin{i}, 'AVIfileNameEnum'),
AVIfileNameEnum = varargin{i+1};
        elseif  strcmp(varargin{i}, 'AVIfps'), AVIfps =
varargin{i+1};
        else error('Invalid argument.');
        end
    end;

    %% Reduce data to samples to plot only

    p = p(1:SamplePlotFreq:numSamples, :);
    R   =   R(:,   :,   1:SamplePlotFreq:numSamples)   *
AxisLength;
    if(numel(View) > 2)
        View = View(1:SamplePlotFreq:numSamples, :);
    end
    [numPlotSamples dummy] = size(p);

    %% Setup AVI file

    aviobj                      =                      [];
% create null object
    if(CreateAVI)
        fileName = strcat(AVIfileName, '.avi');
        if(exist(fileName, 'file'))
            if(AVIfileNameEnum)
% if file name exists and enum enabled
                i = 0;
                while(exist(fileName,          'file'))
% find un-used file name by appending enum
                    fileName    =    strcat(AVIfileName,
sprintf('%i', i), '.avi');
                    i = i + 1;
```

```
                end
            else
% else file name exists and enum disabled
                fileName                =               [];
% file will not be created
            end
        end
        if(isempty(fileName))
            sprintf('AVI  file  not  created  as  file
already exists.')
        else
            aviobj  =  avifile(fileName,  'fps',  AVIfps,
'compression', 'Cinepak', 'quality', 100);
        end
    end

    %% Setup figure and plot

    % Create figure

    set(gca, 'drawmode', 'fast');
    lighting phong;
    set(gcf, 'Renderer', 'zbuffer');
    hold on;
    axis equal;
    grid on;
    view(View(1, 1), View(1, 2));
    title(i);
    xlabel(Xlabel);
    ylabel(Ylabel);
    zlabel(Zlabel);

    % Create plot data arrays
    if(strcmp(Trail,   'DotsOnly')   ||   strcmp(Trail,
'All'))
        x = zeros(numPlotSamples, 1);
        y = zeros(numPlotSamples, 1);
        z = zeros(numPlotSamples, 1);
    end
    if(strcmp(Trail, 'All'))
        ox = zeros(numPlotSamples, 1);
        oy = zeros(numPlotSamples, 1);
        oz = zeros(numPlotSamples, 1);
        ux = zeros(numPlotSamples, 1);
        vx = zeros(numPlotSamples, 1);
        wx = zeros(numPlotSamples, 1);
        uy = zeros(numPlotSamples, 1);
```

```
            vy = zeros(numPlotSamples, 1);
            wy = zeros(numPlotSamples, 1);
            uz = zeros(numPlotSamples, 1);
            vz = zeros(numPlotSamples, 1);
            wz = zeros(numPlotSamples, 1);
        end
        x(1) = p(1,1);
        y(1) = p(1,2);
        z(1) = p(1,3);
        ox(1) = x(1);
        oy(1) = y(1);
        oz(1) = z(1);
        ux(1) = R(1,1,1:1);
        vx(1) = R(2,1,1:1);
        wx(1) = R(3,1,1:1);
        uy(1) = R(1,2,1:1);
        vy(1) = R(2,2,1:1);
        wy(1) = R(3,2,1:1);
        uz(1) = R(1,3,1:1);
        vz(1) = R(2,3,1:1);
        wz(1) = R(3,3,1:1);

        % Create graphics handles
        orgHandle = plot3(x, y, z, 'k.');
        if(ShowArrowHead)
            ShowArrowHeadStr = 'on';
        else
            ShowArrowHeadStr = 'off';
        end
        quivXhandle = quiver3(ox, oy, oz, ux, vx, wx,  'r',
'ShowArrowHead',    ShowArrowHeadStr,    'MaxHeadSize',
0.999999, 'AutoScale', 'off');
        quivYhandle = quiver3(ox, oy, oz, uy, vy, wy,  'g',
'ShowArrowHead',    ShowArrowHeadStr,    'MaxHeadSize',
0.999999, 'AutoScale', 'off');
        quivZhandle = quiver3(ox, ox, oz, uz, vz, wz,  'b',
'ShowArrowHead',    ShowArrowHeadStr,    'MaxHeadSize',
0.999999, 'AutoScale', 'off');

        % Create legend
        if(ShowLegend)
            legend('Origin', 'X', 'Y', 'Z');
        end

        % Set initial limits
        Xlim   =   [x(1)-AxisLength   x(1)+AxisLength]   *
LimitRatio;
```

```
    Ylim   =   [y(1)-AxisLength   y(1)+AxisLength]   *
LimitRatio;
    Zlim   =   [z(1)-AxisLength   z(1)+AxisLength]   *
LimitRatio;
    set(gca, 'Xlim', Xlim, 'Ylim', Ylim, 'Zlim', Zlim);

    % Set initial view
    view(View(1, :));

    %% Plot one sample at a time

    for i = 1:numPlotSamples

        % Update graph title
        if(strcmp(Title, ''))
            titleText  =  sprintf('Sample  %i  of  %i',
1+((i-1)*SamplePlotFreq), numSamples);
        else
            titleText    =    strcat(Title,    '    (',
sprintf('Sample  %i  of  %i',  1+((i-1)*SamplePlotFreq),
numSamples), ')');
        end
        title(titleText);

        % Plot body x y z axes
        if(strcmp(Trail,  'DotsOnly')  ||  strcmp(Trail,
'All'))
            x(1:i) = p(1:i,1);
            y(1:i) = p(1:i,2);
            z(1:i) = p(1:i,3);
        else
            x = p(i,1);
            y = p(i,2);
            z = p(i,3);
        end
        if(strcmp(Trail, 'All'))
            ox(1:i) = p(1:i,1);
            oy(1:i) = p(1:i,2);
            oz(1:i) = p(1:i,3);
            ux(1:i) = R(1,1,1:i);
            vx(1:i) = R(2,1,1:i);
            wx(1:i) = R(3,1,1:i);
            uy(1:i) = R(1,2,1:i);
            vy(1:i) = R(2,2,1:i);
            wy(1:i) = R(3,2,1:i);
            uz(1:i) = R(1,3,1:i);
            vz(1:i) = R(2,3,1:i);
```

```
            wz(1:i) = R(3,3,1:i);
        else
            ox = p(i,1);
            oy = p(i,2);
            oz = p(i,3);
            ux = R(1,1,i);
            vx = R(2,1,i);
            wx = R(3,1,i);
            uy = R(1,2,i);
            vy = R(2,2,i);
            wy = R(3,2,i);
            uz = R(1,3,i);
            vz = R(2,3,i);
            wz = R(3,3,i);
        end
        set(orgHandle, 'xdata', x, 'ydata', y, 'zdata',
z);
        set(quivXhandle,  'xdata',  ox,  'ydata',  oy,
'zdata', oz,'udata', ux, 'vdata', vx, 'wdata', wx);
        set(quivYhandle,  'xdata',  ox,  'ydata',  oy,
'zdata', oz,'udata', uy, 'vdata', vy, 'wdata', wy);
        set(quivZhandle,  'xdata',  ox,  'ydata',  oy,
'zdata', oz,'udata', uz, 'vdata', vz, 'wdata', wz);

        % Adjust axes for snug fit and draw
        axisLimChanged = false;
        if((p(i,1) - AxisLength) < Xlim(1)), Xlim(1) =
p(i,1) - LimitRatio*AxisLength; axisLimChanged = true;
end
        if((p(i,2) - AxisLength) < Ylim(1)), Ylim(1) =
p(i,2) - LimitRatio*AxisLength; axisLimChanged = true;
end
        if((p(i,3) - AxisLength) < Zlim(1)), Zlim(1) =
p(i,3) - LimitRatio*AxisLength; axisLimChanged = true;
end
        if((p(i,1) + AxisLength) > Xlim(2)), Xlim(2) =
p(i,1) + LimitRatio*AxisLength; axisLimChanged = true;
end
        if((p(i,2) + AxisLength) > Ylim(2)), Ylim(2) =
p(i,2) + LimitRatio*AxisLength; axisLimChanged = true;
end
        if((p(i,3) + AxisLength) > Zlim(2)), Zlim(2) =
p(i,3) + LimitRatio*AxisLength; axisLimChanged = true;
end
        if(axisLimChanged),  set(gca,  'Xlim',  Xlim,
'Ylim', Ylim, 'Zlim', Zlim); end
        drawnow;
```

```
        % Adjust view
        if(numel(View) > 2)
            view(View(i, :));
        end

        % Add frame to AVI object
        if(~isempty(aviobj))
            frame = getframe(fig);
            aviobj = addframe(aviobj, frame);
        end

    end

    hold off;

    % Close AVI file
    if(~isempty(aviobj))
        aviobj = close(aviobj);
    end

end
```

tabladatos.m

```
function varargout = tabladatos(varargin)
% TABLADATOS MATLAB code for tabladatos.fig
%      TABLADATOS, by itself, creates a new TABLADATOS
or raises the existing
%      singleton*.
%
%        H = TABLADATOS returns the handle to a new
TABLADATOS or the handle to
%      the existing singleton*.
%
%
TABLADATOS('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in TABLADATOS.M with the
given input arguments.
%
%      TABLADATOS('Property','Value',...) creates a new
TABLADATOS or raises the
%       existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before tabladatos_OpeningFcn
gets called.  An
```

```
%     unrecognized property name or invalid value makes
property application
%              stop.     All   inputs   are   passed   to
tabladatos_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
tabladatos

% Last Modified by GUIDE v2.5 31-Oct-2020 00:44:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@tabladatos_OpeningFcn, ...
                   'gui_OutputFcn',
@tabladatos_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]   =   gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before tabladatos is made visible.
function   tabladatos_OpeningFcn(hObject,   eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
% varargin   command line arguments to tabladatos (see
VARARGIN)

% Choose default command line output for tabladatos
handles.output = hObject;
set(handles.data,'data',varargin{1})
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes tabladatos wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
function  varargout  =  tabladatos_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in ok.
function ok_Callback(hObject, eventdata, handles)
% hObject    handle to ok (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
guide=get(0,'CurrentFigure');
delete(guide);
```

AHRS.M

```
classdef AHRS < handle

    %% Public properties
    properties (Access = public)
        SamplePeriod = 1/50;
```

```
        Quaternion = [1 0 0 0];     % output quaternion
describing the sensor relative to the Earth
        Kp = 2;                     % proportional gain
        Ki = 0;                     % integral gain
        KpInit = 200;               % proportional gain
used during initialisation
        InitPeriod = 5;                % initialisation
period in seconds
    end

    %% Private properties
    properties (Access = private)
        q = [1 0 0 0];             % internal quaternion
describing the Earth relative to the sensor
        IntError = [0 0 0]';        % integral error
        KpRamped;                              % internal
proportional gain used to ramp during initialisation
    end

    %% Public methods
    methods (Access = public)
        function obj = AHRS(varargin)
            for i = 1:2:nargin
                if                  strcmp(varargin{i},
'SamplePeriod'), obj.SamplePeriod = varargin{i+1};
                elseif              strcmp(varargin{i},
'Quaternion')
                    obj.Quaternion = varargin{i+1};
                    obj.q                                =
quaternConj(obj.Quaternion);
                elseif     strcmp(varargin{i},   'Kp'),
obj.Kp = varargin{i+1};
                elseif     strcmp(varargin{i},   'Ki'),
obj.Ki = varargin{i+1};
                elseif  strcmp(varargin{i}, 'KpInit'),
obj.KpInit = varargin{i+1};
                elseif              strcmp(varargin{i},
'InitPeriod'), obj.InitPeriod = varargin{i+1};
                else error('Invalid argument');
                end
                obj.KpRamped = obj.KpInit;
            end;
        end
        function    obj    =    Update(obj,    Gyroscope,
Accelerometer, Magnetometer)
            error('This method is unimplemented');
        end
```

```
        function obj = UpdateIMU(obj, Gyroscope,
Accelerometer)

            % Normalise accelerometer measurement
            if(norm(Accelerometer) == 0)
% handle NaN
                warning(0, 'Accelerometer magnitude is
zero.  Algorithm update aborted.');
                return;
            else
                Accelerometer = Accelerometer /
norm(Accelerometer);                        % normalise
measurement
            end

            % Compute error between estimated and
measured direction of gravity
            v = [2*(obj.q(2)*obj.q(4) -
obj.q(1)*obj.q(3))
                2*(obj.q(1)*obj.q(2) +
obj.q(3)*obj.q(4))
                obj.q(1)^2 - obj.q(2)^2 - obj.q(3)^2 +
obj.q(4)^2];                    % estimated direction of
gravity
            error = cross(v, Accelerometer');

%                % Compute ramped Kp value used during
init period
%             if(obj.KpRamped > obj.Kp)
%                 obj.IntError = [0 0 0]';
%                     obj.KpRamped = obj.KpRamped -
(obj.KpInit - obj.Kp) / (obj.InitPeriod /
obj.SamplePeriod);
%                                             else
% init period complete
%                 obj.KpRamped = obj.Kp;
                obj.IntError = obj.IntError + error;
% compute integral feedback terms (only outside of init
period)
%             end

            % Apply feedback terms
            Ref = Gyroscope - (obj.Kp*error +
obj.Ki*obj.IntError)';

            % Compute rate of change of quaternion
```

```
            pDot = 0.5 * obj.quaternProd(obj.q, [0
Ref(1) Ref(2) Ref(3)]);             % compute rate of
change of quaternion
            obj.q = obj.q + pDot * obj.SamplePeriod;
% integrate rate of change of quaternion
            obj.q    =    obj.q    /    norm(obj.q);
% normalise quaternion

            % Store conjugate
            obj.Quaternion = obj.quaternConj(obj.q);
        end
        function obj = Reset(obj)
            obj.KpRamped = obj.KpInit;       % start Kp
ramp-down
            obj.IntError = [0 0 0]';           % reset
integral terms
            obj.q = [1 0 0 0];                   % set
quaternion to alignment
        end
%         function obj = StepDownKp(obj, Kp)
%             obj.KpRamped = Kp;
%             obj.Kp = Kp;
%         end
    end

    %% Get/set methods
    methods
        function obj = set.Quaternion(obj, value)
            if(norm(value) == 0)
                error('Quaternion magnitude cannot be
zero.');
            end
            value = value / norm(value);
            obj.Quaternion = value;
            obj.q = obj.quaternConj(value);
        end
    end

    %% Private methods
    methods (Access = private)
        function ab = quaternProd(obj, a, b)
            ab(:,1)  =  a(:,1).*b(:,1)-a(:,2).*b(:,2)-
a(:,3).*b(:,3)-a(:,4).*b(:,4);
            ab(:,2)                                    =
a(:,1).*b(:,2)+a(:,2).*b(:,1)+a(:,3).*b(:,4)-
a(:,4).*b(:,3);
```

```
            ab(:,3)          =          a(:,1).*b(:,3)-
a(:,2).*b(:,4)+a(:,3).*b(:,1)+a(:,4).*b(:,2);
            ab(:,4)   =  a(:,1).*b(:,4)+a(:,2).*b(:,3)-
a(:,3).*b(:,2)+a(:,4).*b(:,1);
        end
        function qConj = quaternConj(obj, q)
            qConj = [q(:,1) -q(:,2) -q(:,3) -q(:,4)];
        end
    end
end
```

prueba.m

```
fs = 100;
t = 0:1/fs:1;
x = sin(2*pi*t*3)+0.25*sin(2*pi*t*40);

y = medfilt1(x,5);

plot(t,x,t,y)
legend('Original','Filtered')
legend('boxoff')
```

Script.m

```
clear;
close all;
clc;
addpath('Quaternions');
addpath('ximu_matlab_library');
startTime = 6;
stopTime = 26;
samplePeriod = 1/256;
[datos,txt,cell] = xlsread('datos/datos.xlsx');
time = datos(:,1);
gyrX = datos(:,2);
gyrY = datos(:,3);
gyrZ = datos(:,4);
accX = datos(:,5);
accY = datos(:,6);
accZ = datos(:,7);
indexSel    =    find(sign(time-startTime)+1,    1)    :
find(sign(time-stopTime)+1, 1);
time = time(indexSel);
gyrX = gyrX(indexSel, :);
gyrY = gyrY(indexSel, :);
gyrZ = gyrZ(indexSel, :);
accX = accX(indexSel, :);
```

```
accY = accY(indexSel, :);
accZ = accZ(indexSel, :);
acc_mag = sqrt(accX.*accX + accY.*accY + accZ.*accZ);

% HP filter accelerometer data
filtCutOff = 0.001;
[b, a] = butter(1, (2*filtCutOff)/(1/samplePeriod),
'high');
acc_magFilt = filtfilt(b, a, acc_mag);

% Compute absolute value
acc_magFilt = abs(acc_magFilt);

% LP filter accelerometer data
filtCutOff = 5;
[b, a] = butter(1, (2*filtCutOff)/(1/samplePeriod),
'low');
acc_magFilt = filtfilt(b, a, acc_magFilt);

% Threshold detection
stationary = acc_magFilt < 0.05;

% ------------------------------------------------
--------------------
% Plot data raw sensor data and stationary periods

figure('Position', [9 39 900 600], 'NumberTitle',
'off', 'Name', 'Sensor Data');
ax(1) = subplot(2,1,1);
    hold on;
    plot(time, gyrX, 'r');
    plot(time, gyrY, 'g');
    plot(time, gyrZ, 'b');
    title('Gyroscope');
    xlabel('Time (s)');
    ylabel('Angular velocity (^\circ/s)');
    legend('X', 'Y', 'Z');
    hold off;
ax(2) = subplot(2,1,2);
    hold on;
    plot(time, accX, 'r');
    plot(time, accY, 'g');
    plot(time, accZ, 'b');
    plot(time, acc_magFilt, ':k');
    plot(time, stationary, 'k', 'LineWidth', 2);
    title('Accelerometer');
    xlabel('Time (s)');
    ylabel('Acceleration (g)');
```

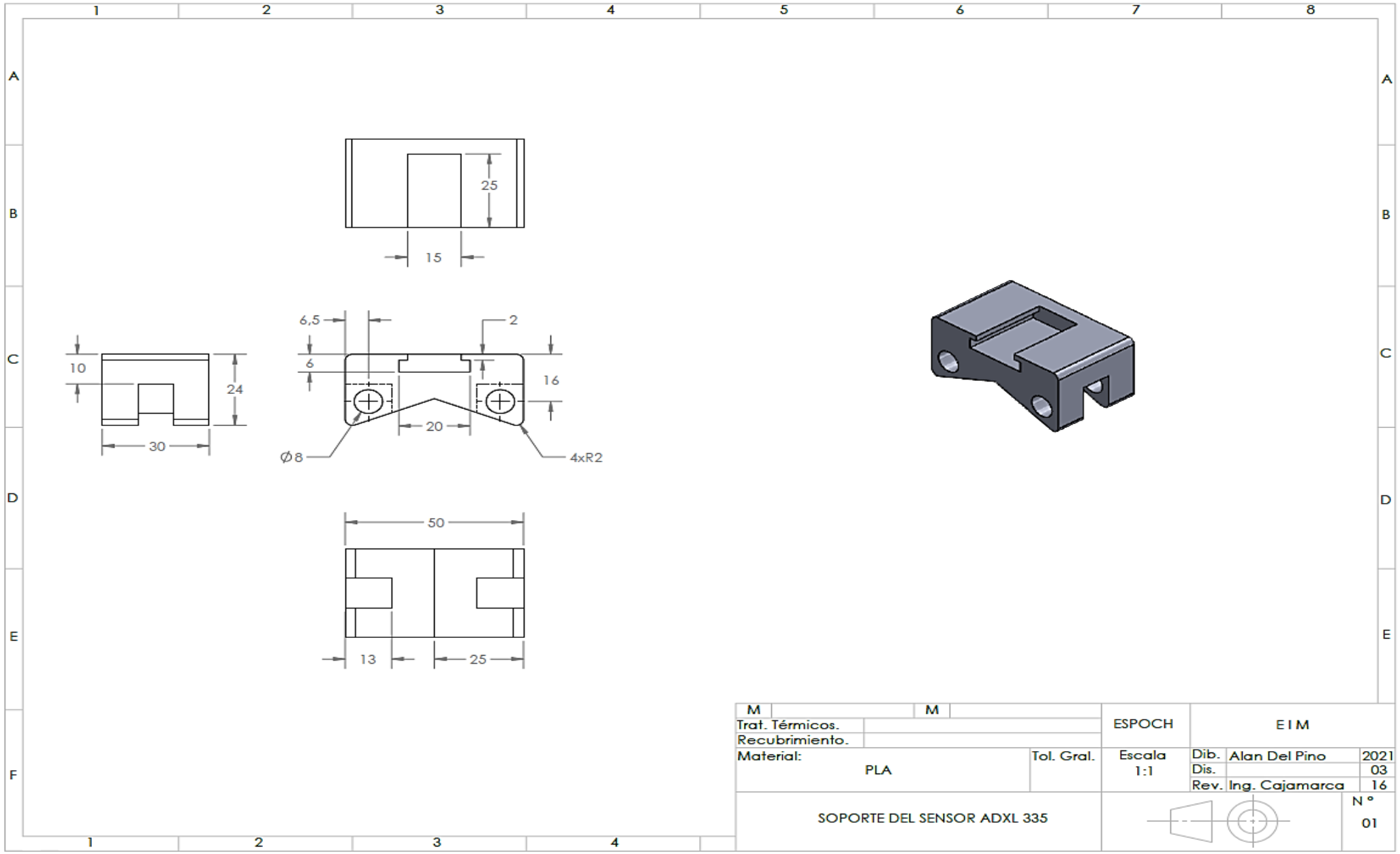
```
    legend('X', 'Y', 'Z', 'Filtered', 'Stationary');
    hold off;
linkaxes(ax,'x');

% ------------------------------------------------------------
% Compute orientation

quat = zeros(length(time), 4);
AHRSalgorithm = AHRS('SamplePeriod', 1/256, 'Kp', 1, 'KpInit', 1);

% Initial convergence
initPeriod = 2;
indexSel = 1 : find(sign(time-(time(1)+initPeriod))+1, 1);
for i = 1:2000
    AHRSalgorithm.UpdateIMU([0          0          0], [mean(accX(indexSel))             mean(accY(indexSel))  mean(accZ(indexSel))]);
end

% For all data
for t = 1:length(time)
    if(stationary(t))
        AHRSalgorithm.Kp = 0.5;
    else
        AHRSalgorithm.Kp = 0;
    end
    AHRSalgorithm.UpdateIMU(deg2rad([gyrX(t)    gyrY(t)  gyrZ(t)]), [accX(t) accY(t) accZ(t)]);
    quat(t,:) = AHRSalgorithm.Quaternion;
end

% ------------------------------------------------------------
% Compute translational accelerations

% Rotate body accelerations to Earth frame
acc      =      quaternRotate([accX      accY      accZ], quaternConj(quat));

% % Remove gravity from measurements
% acc = acc - [zeros(length(time), 2) ones(length(time), 1)];      % unnecessary due to velocity integral drift compensation

% Convert acceleration measurements to m/s/s
```

```
acc = acc * 9.81;

% Plot translational accelerations
figure('Position', [9 39 900 300], 'NumberTitle',
'off', 'Name', 'Accelerations');
hold on;
plot(time, acc(:,1), 'r');
plot(time, acc(:,2), 'g');
plot(time, acc(:,3), 'b');
title('Acceleration');
xlabel('Time (s)');
ylabel('Acceleration (m/s/s)');
legend('X', 'Y', 'Z');
hold off;

% ------------------------------------------------
---------------------
% Compute translational velocities

acc(:,3) = acc(:,3) - 9.81;

% Integrate acceleration to yield velocity
vel = zeros(size(acc));
for t = 2:length(vel)
    vel(t,:) = vel(t-1,:) + acc(t,:) * samplePeriod;
    if(stationary(t) == 1)
        vel(t,:) = [0 0 0];      % force zero velocity
when foot stationary
    end
end


% Compute integral drift during non-stationary periods
velDrift = zeros(size(vel));
stationaryStart = find([0; diff(stationary)] == -1);
stationaryEnd = find([0; diff(stationary)] == 1);
for i = 1:numel(stationaryEnd)
    driftRate    =    vel(stationaryEnd(i)-1,    :)    /
(stationaryEnd(i) - stationaryStart(i));
    enum = 1:(stationaryEnd(i) - stationaryStart(i));
    drift   =   [enum'*driftRate(1)   enum'*driftRate(2)
enum'*driftRate(3)];
    velDrift(stationaryStart(i):stationaryEnd(i)-1, :)
= drift;
end

% Remove integral drift
vel = vel - velDrift;
```

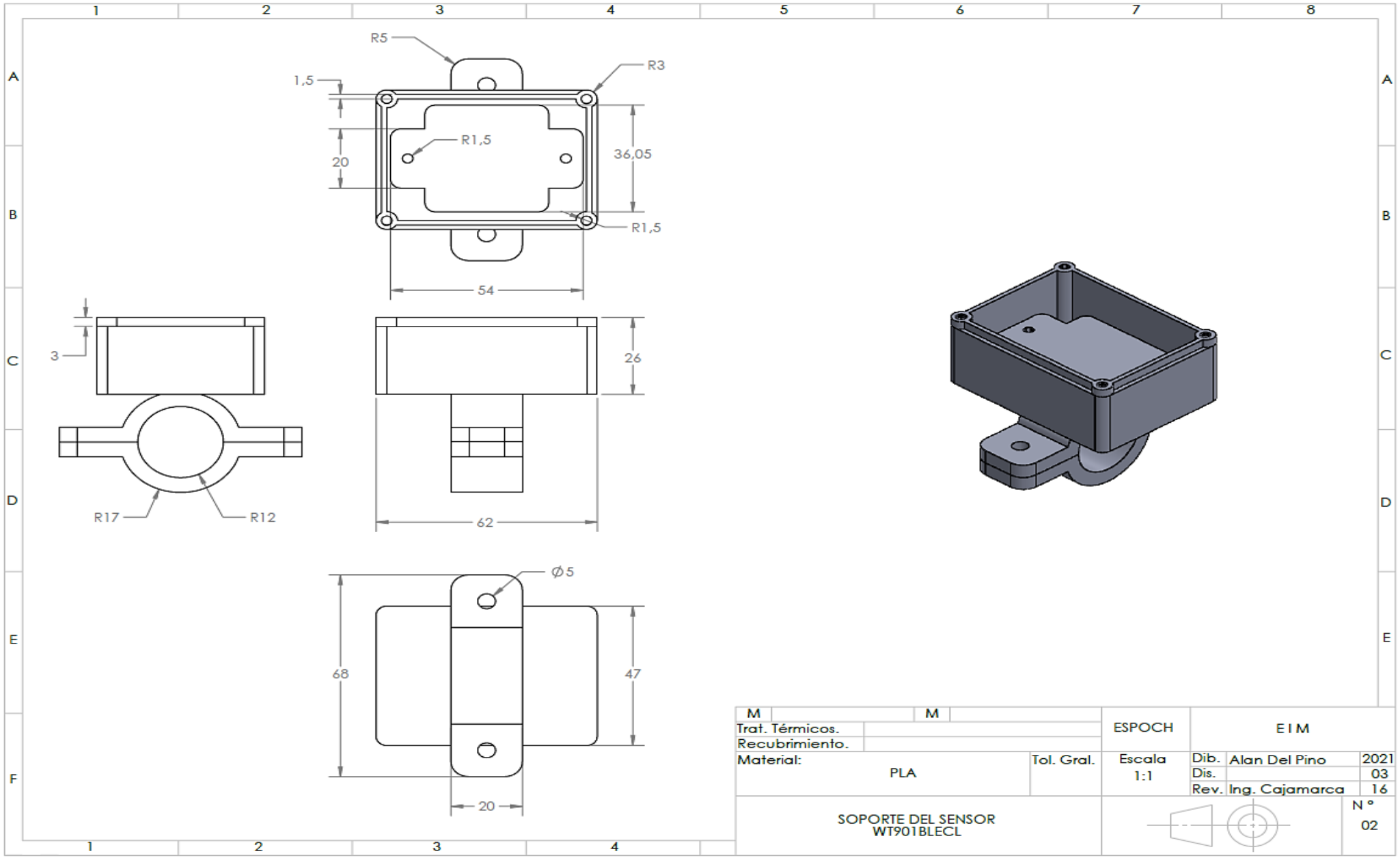
```
% Plot translational velocity
figure('Position', [9 39 900 300], 'NumberTitle',
'off', 'Name', 'Velocity');
hold on;
plot(time, vel(:,1), 'r');
plot(time, vel(:,2), 'g');
plot(time, vel(:,3), 'b');
title('Velocity');
xlabel('Time (s)');
ylabel('Velocity (m/s)');
legend('X', 'Y', 'Z');
hold off;

% ------------------------------------------------
--------------------
% Compute translational position

% Integrate velocity to yield position
pos = zeros(size(vel));
for t = 2:length(pos)
    pos(t,:) = pos(t-1,:) + vel(t,:) * samplePeriod;
% integrate velocity to yield position
end

% Plot translational position
figure('Position', [9 39 900 600], 'NumberTitle',
'off', 'Name', 'Position');
hold on;
plot(time, pos(:,1), 'r');
plot(time, pos(:,2), 'g');
plot(time, pos(:,3), 'b');
title('Position');
xlabel('Time (s)');
ylabel('Position (m)');
legend('X', 'Y', 'Z');
hold off;

% ------------------------------------------------
--------------------
% Plot 3D foot trajectory

% % Remove stationary periods from data to plot
% posPlot = pos(find(~stationary), :);
% quatPlot = quat(find(~stationary), :);
posPlot = pos;
quatPlot = quat;
```

```
% Extend final sample to delay end of animation
extraTime = 20;
onesVector = ones(extraTime*(1/samplePeriod), 1);
posPlot = [posPlot; [posPlot(end, 1)*onesVector,
posPlot(end, 2)*onesVector, posPlot(end,
3)*onesVector]];
quatPlot = [quatPlot; [quatPlot(end, 1)*onesVector,
quatPlot(end, 2)*onesVector, quatPlot(end,
3)*onesVector, quatPlot(end, 4)*onesVector]];

% Create 6 DOF animation
SamplePlotFreq = 4;
Spin = 120;
SixDofAnimation(posPlot, quatern2rotMat(quatPlot), ...
                'SamplePlotFreq', SamplePlotFreq,
'Trail', 'All', ...
                'Position', [9 39 1280 768], 'View',
[(100:(Spin/(length(posPlot)-1)):(100+Spin))',
10*ones(length(posPlot), 1)], ...
                'AxisLength', 0.1, 'ShowArrowHead',
false, ...
                'Xlabel', 'X (m)', 'Ylabel', 'Y (m)',
'Zlabel', 'Z (m)', 'ShowLegend', false, ...
                'CreateAVI', false, 'AVIfileNameEnum',
false, 'AVIfps', ((1/samplePeriod) / SamplePlotFreq));
```

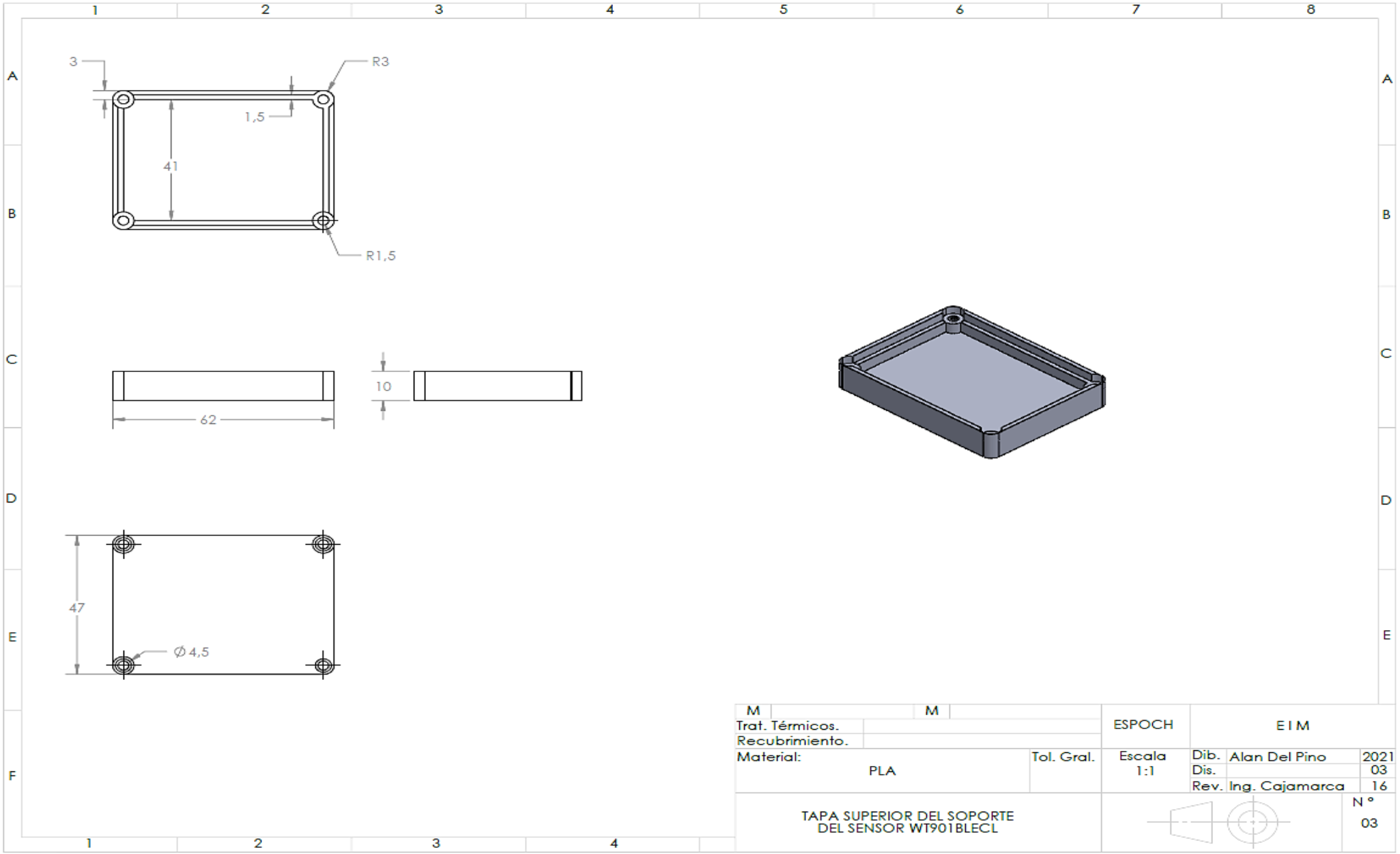
ANEXO B: PERSPECTIVA ISOMÉTRICA DEL SOPORTE DEL SENSOR ADXL335.



ANEXO C: PERSPECTIVA ISOMÉTRICA DEL SOPORTE DEL SENSOR WT901BLECL.



ANEXO D: PERSPECTIVA ISOMÉTRICA DE LA TAPA SUPERIOR DEL SOPORTE DEL SENSOR WT901 BLECL.



ANEXO E: SENSOR ADXL335.

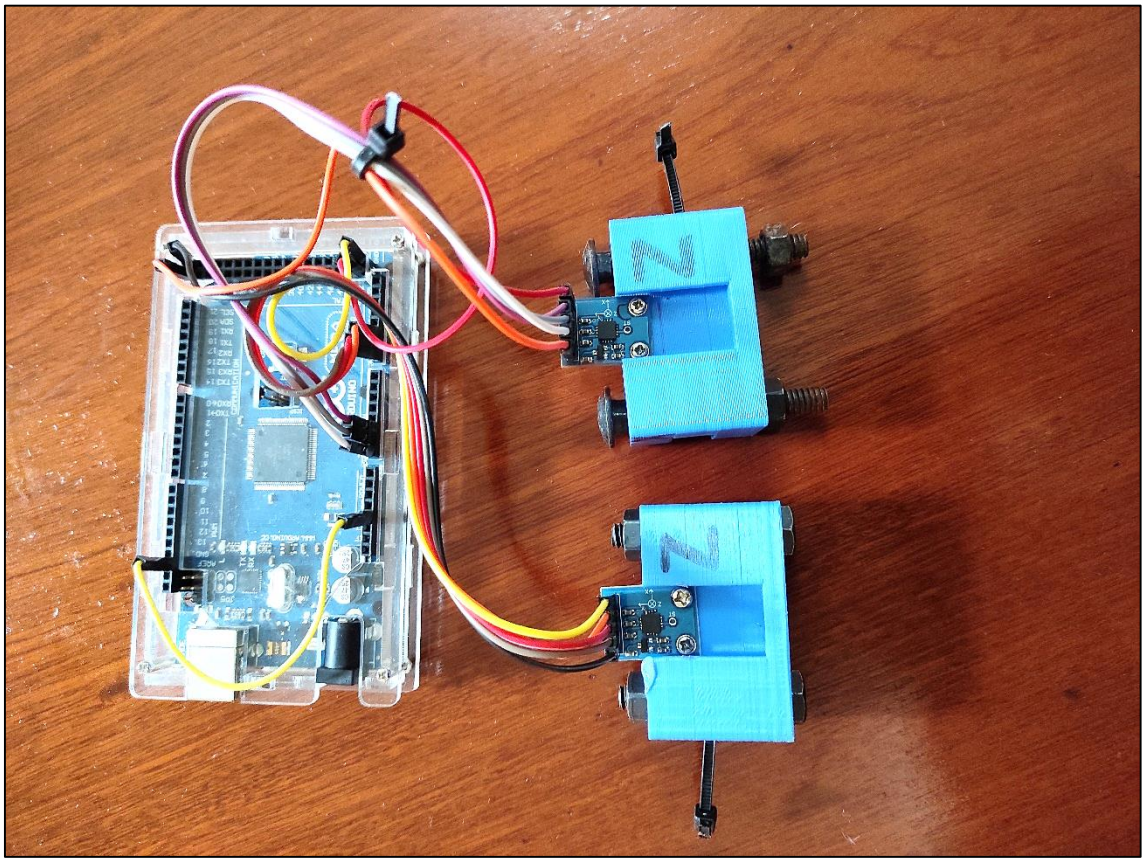


Figura 1. Sensor ADXL335 con el microcontrolador Arduino Mega 2560

Este es el prototipo del primer sensor que se utilizo para realizar las pruebas y ver como se comporta el sensor.

ANEXO F: SENSOR WT901 BLECL.

Figura 2. Sensor WT901BLECL con el adaptador bluettoth 5.0

Este fue el segundo sensor que se utilizó para las pruebas, y con este se pudo encontrar la curva caractetistica del eje de un motor eléctrico.

ANEXO G: SOPORTE PARA EL SENSOR WT901BLECL.

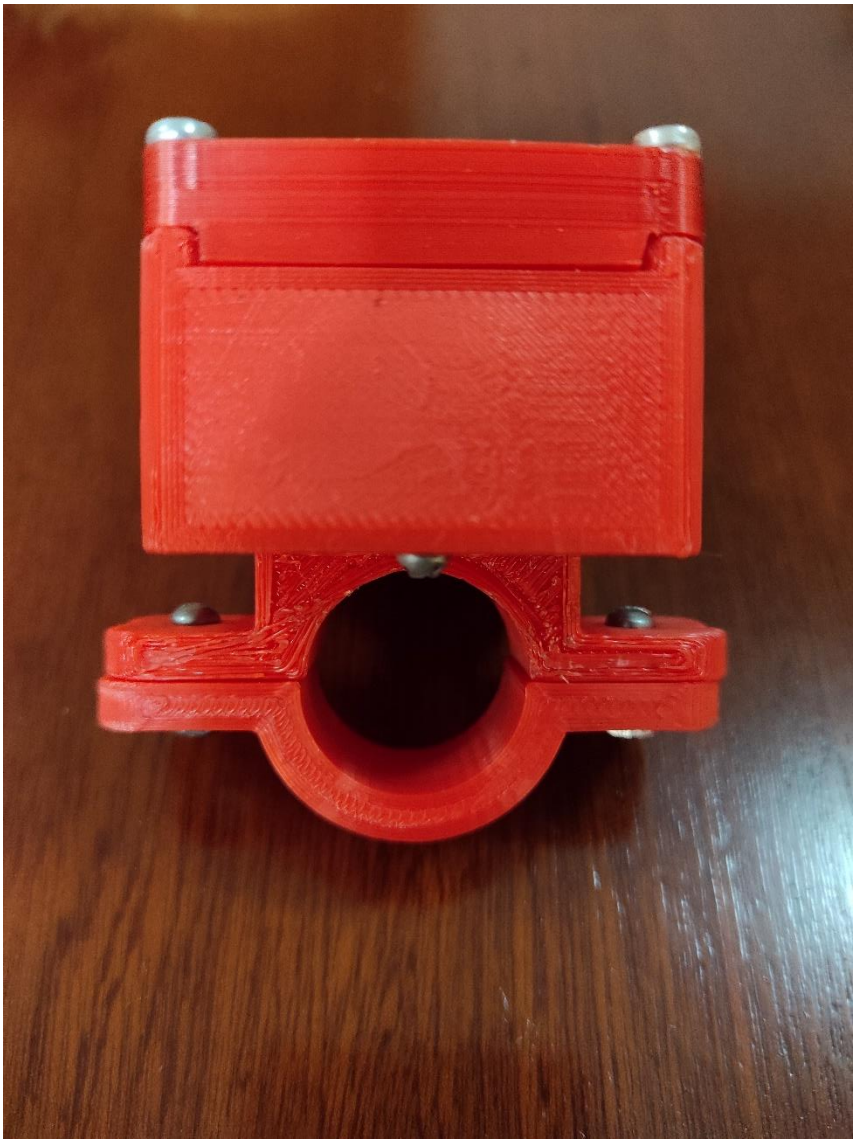


Figura 3. Soporte impreso en PLA para el sensor WT901BLECL

Este fue el disño del sooporte para colocar el sensor WT901BLECL sobre el eje de pruebas

ANEXO H: EJE DE PRUEBAS.

Figura 2. Eje de pruebas para los sensores ADXL335 Y WT901BLECL

Este fue el eje de pruebas que se diseño para realizar las pruebas a los dos sensores, y provocar las desviaciones y ver si los sensores lograban detectar esas desviaciones para poder encontrar la curva característica.

ANEXO I: DATOS DE LA PRUEBA NÚMERO UNO DEL SENSOR ADXL335.

PRUEBA 1 SENSOR ADXL335						
MUESTRAS	**ANGULO X (°)**	**ANGULO Y (°)**	**ANGULO Z (°)**	**ANGULO X1 (°)**	**ANGULO Y2 (°)**	**ANGULO Z2 (°)**
1	6	5	50	14	13	47
2	6	5	53	15	13	49
3	3	326	174	18	338	139
4	11	284	176	46	296	151
5	169	238	173	132	238	145
6	173	195	157	157	186	104
7	170	158	23	159	149	31
8	167	135	12	158	125	16
9	136	100	10	111	94	11
10	46	79	11	41	76	12
11	19	65	9	21	64	11
12	12	45	11	17	47	16
13	8	31	14	15	35	21
14	5	10	29	14	16	39
15	5	6	39	14	14	46
16	5	7	39	14	14	45
17	5	6	39	14	14	45
18	5	6	39	14	14	45
19	5	7	36	14	14	45
20	6	6	45	14	14	46

ANEXO J: DATOS DE LA SEGUNDA PRUEBA DEL SENSOR ADXL335.

PRUEBA 2 SENSOR ADXL335						
MUESTRAS	**ANGULO X (°)**	**ANGULO Y (°)**	**ANGULO Z (°)**	**ANGULO X1 (°)**	**ANGULO Y2 (°)**	**ANGULO Z2 (°)**
1	356	7	333	5	14	19
2	356	7	337	6	14	22
3	357	20	353	5	25	11
4	1	57	0	7	57	4
5	170	99	1	135	94	4
6	180	134	360	171	126	6
7	182	174	333	170	165	31
8	194	237	189	156	237	164
9	284	272	189	53	279	166
10	343	303	190	13	315	166
11	351	335	196	7	346	151
12	354	357	246	5	5	45
13	356	8	335	5	16	17
14	356	12	343	4	18	13
15	356	12	345	5	18	15
16	357	12	348	6	18	18
17	356	12	345	5	18	15
18	356	12	342	4	18	13
19	356	12	342	5	18	15
20	356	12	342	5	18	15

ANEXO K: DATOS DE LA TERCERA PRUEBA DEL SENSOR ADXL335.

PRUEBA 3 SENSOR ADXL335						
MUESTRAS	**ANGULO X (°)**	**ANGULO Y (°)**	**ANGULO Z (°)**	**ANGULO X1 (°)**	**ANGULO Y2 (°)**	**ANGULO Z2 (°)**
1	10	10	46	17	16	46
2	12	9	51	18	17	47
3	10	346	143	19	357	96
4	12	318	165	28	330	135
5	28	285	171	50	298	146
6	153	252	170	109	256	146
7	168	220	166	144	215	135
8	169	198	150	153	188	105
9	168	180	93	154	170	70
10	165	155	28	154	144	33
11	158	129	17	146	119	20
12	145	112	15	130	105	17
13	94	91	14	81	87	14
14	40	73	14	38	70	15
15	23	59	14	26	59	16
16	15	41	17	21	43	21
17	12	12	45	18	19	44
18	10	9	48	18	16	47
19	10	10	46	18	17	45
20	11	9	49	18	17	46

ANEXO L: DATOS DE LA CUARTA PRUEBA DEL SENSOR ADXL335.

PRUEBA 4 SENSOR ADXL335						
MUESTRAS	**ANGULO X (°)**	**ANGULO Y (°)**	**ANGULO Z (°)**	**ANGULO X1 (°)**	**ANGULO Y2 (°)**	**ANGULO Z2 (°)**
1	6	13	27	14	20	35
2	8	12	31	14	20	35
3	7	9	38	15	16	43
4	5	329	170	19	341	132
5	6	296	176	31	309	152
6	153	263	176	84	272	152
7	173	215	171	152	209	137
8	171	186	127	156	176	83
9	171	160	23	159	149	32
10	164	133	14	153	124	18
11	153	111	11	139	104	12
12	90	90	10	72	86	11
13	30	73	9	30	70	11
14	14	52	11	18	53	14
15	9	36	12	16	39	19
16	7	13	28	14	20	33
17	5	2	66	14	11	52
18	6	3	65	15	11	53
19	5	3	60	14	11	52
20	5	2	66	14	11	51

ANEXO M: DATOS DE LA QUINTA PRUEBA DEL SENSOR ADXL335.

PRUEBA 5 SENSOR ADXL335						
MUESTRAS	**ANGULO X (°)**	**ANGULO Y (°)**	**ANGULO Z (°)**	**ANGULO X1 (°)**	**ANGULO Y2 (°)**	**ANGULO Z2 (°)**
1	5	2	63	14	10	53
2	5	1	69	14	9	56
3	5	357	113	15	6	68
4	6	339	164	19	350	115
5	6	304	175	29	316	149
6	7	286	177	41	298	154
7	175	216	173	152	210	138
8	172	174	53	160	163	49
9	171	154	17	159	143	26
10	167	137	13	155	126	18
11	162	122	11	148	113	14
12	141	101	9	112	94	11
13	30	73	9	32	71	11
14	10	46	10	15	48	14
15	8	19	22	16	25	31
16	6	9	34	13	16	39
17	5	9	29	14	16	40
18	5	9	30	14	16	40
19	5	9	30	14	16	40
20	5	9	30	14	16	39