Interfaz gráfica de usuario para el monitoreo de un banco de evaluación cilindro de desgaste

Hermann Augusto Galvis Santamaría

Trabajo de Grado presentado como requisito para optar al título de Ingeniero Mecánico

Director

Alberto David Pertuz Comas

Doctor en Ingeniería Mecánica

Codirector

Oscar Rodolfo Bohórquez Becerra

Ingeniero Mecánico

Universidad Industrial De Santander

Facultad de Ingenierías Físico-Mecánicas

Escuela De Ingeniería Mecánica

Bucaramanga

2022

Tabla de Contenido

Lista de tablas

Lista de figuras

Lista de Apéndices

Resumen

Título: Interfaz gráfica de usuario para el monitoreo de un banco de evaluación cilindro de desgaste[*]

Autor: Hermann Augusto Galvis Santamaria[**]

Palabras Clave: Interfaz Gráfica, Monitoreo, Matlab, AppDesigner

Descripción: En este proyecto de investigación se realizó el diseño de una interfaz gráfica en la herramienta AppDesigner del software de Matlab. Con esta interfaz se busca monitorear el estado de un banco de laboratorio midiendo 3 variables de operación (RPM, Temperatura y vibraciones) con el fin de poder visualizar el estado del equipo en tiempo real. Para el desarrollo de este proyecto se hizo uso de la metodología propuesta por Winston Royce denominada "Metodología Incremental" o "Metodología en cascada" donde permite desarrollar el software por etapas implementando 5 fases metodológicas que involucra la evaluación de los requerimientos y el desarrollo de la parte backend y frontend. Durante el desarrollo de este proyecto se utilizaron software de modelamiento y simulación para calcular las frecuencias naturales de algunos de los componentes del banco para de esa forma estimar la frecuencia de muestreo usando el teorema de Nyquist. De igual forma, con ayuda de scripts de Matlab se aplicaron las transformadas rápidas de Fourier para el procesamiento de la señal de vibraciones en los 3 ejes independientes. La interfaz permite graficar y tabular los datos recolectados de los sensores y también permite la visualización en tiempo real de las 3 variables de operación. En este proyecto se hizo uso de hardware propio de Arduino y con ayuda de Simulink y el IDE de Arduino se programaron las funciones de la interfaz.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Físico-mecanicas. Escuela de Ingeniería Mecánica. Director: Alberto David Pertuz Comas. Ingeniero Mecánico, MSc PhD. Codirector: Oscar Rodolfo Bohorquez Becerra. Ingeniero Mecánico

Abstract

Title: Graphical user interface for monitoring a cylinder wear evaluation bench[*]

Author: Hermann Augusto Galvis Santamaria[**]

Key Words: GUI, Monitoring, Matlab, AppDesigner

Description: In this research project, the design of a graphical interface was carried out in the AppDesigner tool of the Matlab software. This interface seeks to monitor the state of a laboratory bench by measuring 3 operating variables (RPM, Temperature and vibrations) in order to be able to visualize the state of the equipment in real time. For the development of this project, the methodology proposed by Winston Royce called "Incremental Methodology" or "Cascade Methodology" was used, where it allows developing the software in stages, implementing 5 methodological phases that involve the evaluation of the requirements and the development of the backend and frontend part. During the development of this project, modeling and simulation software was used to calculate the natural frequencies of some of the components of the bank in order to estimate the sampling frequency using the Nyquist theorem. In the same way, with the help of Matlab scripts, the fast Fourier transforms were applied for the processing of the vibration signal in the 3 independent axes. The interface allows graphing and tabulating the data collected from the sensors and also allows real-time visualization of the 3 operating variables. In this project, Arduino's own hardware was used and with the help of Simulink and the Arduino IDE, the interface functions were programmed.

[*] Degree Work

[**] Faculty of Physical Engineering - Mechanics. Mechanical Engineering School.. Director: Alberto David Pertuz Comas. Mechanical Engineer, MSc PhD. Codirector: Oscar Rodolfo Bohorquez Becerra. Mechanical Engineer.

Introducción

En los años 80 la NASA comenzó a realizar simulaciones para observar las actividades de los equipos y de sus naves, con el fin de cuidar la integridad de la tripulación durante las misiones (Katia Hernández, 2021). En la actualidad, el monitoreo se ha convertido en una herramienta vital para el funcionamiento de muchas empresas, por lo que, el mercado necesita de sistemas robustos que permitan al usuario saber en qué momento pueden fallar sus equipos, proporcionándole la información necesaria para que puedan ser arreglados en el menor tiempo posible (Anabel Alarcón, 2005). Estas prácticas comenzaron a tener más impacto cuando se comenzaron a implementar junto con los avances de la big data, la computación en la nube y el internet de las cosas. La principal apuesta por este tipo de tecnología es por la realización de una réplica o copia virtual de una maquina o equipo, que simula su comportamiento para poder analizar su reacción frente a ciertas circunstancias y ambientes y de esa forma tomar acciones correctivas o preventivas para mejorar el ciclo de vida del equipo y mejorar su rendimiento sin correr riesgos.

Las principales industrias que se están viendo beneficiadas por el monitoreo de sus equipos son la automotriz y la industria de manufactura especialmente en países europeos y en Estados Unidos. Sin embargo, en Colombia es un tema muy poco conocido y aplicado. Hay varias tecnologías que están revolucionando el mundo y que prometen cumplir con la promesa de la famosa revolución 4.0.

Muchos de los accidentes laborales y fallos en los equipos son evitables, y es esa la intención de las actividades de monitoreo. Con ayuda de software y hardware adaptable para cada implementación los ingenieros son capaces de pronosticar y prevenir los riesgos del uso de aparatos industriales e incluso mejorar el comportamiento de los equipos desde la misma interfaz

sin intervención humana en sitio lo que también facilita una automatización de los procesos que vuelven mucho más eficiente el proceso productivo.

El proyecto se divide en varias partes importantes. Primero, se le da a conocer al lector un poco acerca de tecnologías innovadoras de este tema. Luego se procede al desarrollo de la interfaz gráfica la cual está construida metodológicamente según lo explicado. Después de eso, se conecta el hardware con el software y se codifican las acciones del software. Por último, se realizan las pruebas de funcionamiento correspondientes y se corrigen o mejoran las funciones de la interfaz.

1. Justificación del problema

Se estima que la digitalización genere un incremento de 120.000 millones de euros sobre el valor añadido bruto en 2025, (Dedios-Pleiten, 2019). Una interfaz gráfica de monitoreo consiste en exponer al usuario una herramienta donde pueda visualizar el comportamiento actual del equipo. Puede ser utilizada con varios propósitos aprovechando la sincronización en tiempo real de los datos recolectados originados en el sistema físico, pudiendo con los datos entregados por los sensores tomar decisiones sobre un conjunto de acciones con el objetivo de estructurar y asegurar el funcionamiento del sistema de una forma adecuada. Aplicando los conceptos y tecnologías de la Ingeniería mecánica se realizará este proyecto desarrollando una interfaz gráfica. Con la construcción del nuevo edificio de Ingeniería Mecánica se pretende instaurar diversas prácticas de laboratorio. Por lo tanto, la finalidad de este proyecto es contribuir en la adquisición y refuerzo de conocimientos de los estudiantes en el área de mantenimiento, con la implementación de una interfaz gráfica de monitoreo a partir de un modelo real de un banco de laboratorio de la materia de Ingeniería de mantenimiento.

La utilización de este tipo de herramienta en la industria moderna facilita la implementación de labores de mantenimiento (preventivo y predictivo). En la mayoría de las industrias el mantenimiento se realiza monitoreando el activo físico a través de diferentes instrumentos; este seguimiento muchas veces requiere la presencia física del ingeniero. Sin embargo, en muchas ocasiones esto no es posible. Con una interfaz gráfica de monitoreo estas dificultades se pueden aliviar, ya que para realizar estas acciones de seguimiento no se requiere de la presencia del ingeniero, sino que puede estudiar y analizar cada comportamiento del activo físico desde un lugar seguro.

2. Objetivos

2.1 Objetivo general

Desarrollar una interfaz gráfica utilizando el software MATLAB, para el monitoreo en tiempo real de un banco de laboratorio contribuyendo a la misión de la Universidad Industrial de Santander y la Escuela de Ingeniería mecánica en la formación de personas con alta calidad en el medio científico y tecnológico aportando al desarrollo de los laboratorios de la escuela.

2.2 Objetivos específicos

Diseñar una interfaz gráfica de usuario que permita el monitoreo en tiempo real de un banco usando el entorno de desarrollo de MATLAB siguiendo la metodología incremental propuesta por Winston Royce.

Conectar los sensores al banco de tal forma que permita medir variables de operación estableciendo comunicación con la interfaz gráfica.

Alimentar la interfaz o los datos recopilados del banco en tiempo real usando un medio que permita transformar e interpretar la información recolectada.

Realizar pruebas de funcionamiento bajo condiciones normales de operación y realizar los debidos ajustes siguiendo la metodología implementada.

Elaborar manuales de usuario y ayuda que incluyan la forma de uso de la interfaz y corrección de errores durante la ejecución de las funciones.

3 Marco Referencial

Con el fin de entender la importancia que tienen las interfaces graficas de monitoreo en la industria moderna, debemos iniciar por estudiar sus antecedentes, que significan, los tipos que hay, como es su funcionamiento, así como las áreas donde se enfocan este tipo de herramientas.

Finalmente se mostrará una breve descripción del banco de trabajo, junto con sus componentes mecánicos presentes.

El presente trabajo pretende desarrollar una interfaz gráfica de usuario en el software de MATLAB, lo que implica que se van a tomar los datos recopilados del activo físico y con ellos se va a alimentar el software.

3.1 Antecedentes

3.1.1 Industria 4.0

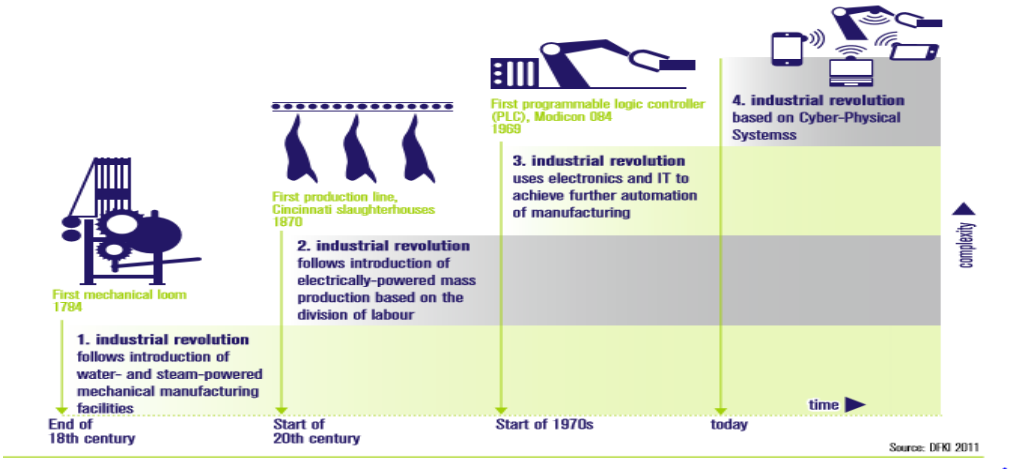
Actualmente nos encontramos en la llamada industria 4.0, la cual en 2013 fue definida por la ACATECH (academia nacional de ciencia e ingeniería) como “la integración técnica de las computadoras en la fabricación y la logística y el uso de Internet de las cosas y los servicios en los procesos industriales.”

Con el aumento del consumo de las herramientas tecnológicas portables se marcó una tendencia para crear nuevas infraestructuras y servicios TIC (tecnología de la información y la comunicación) a través de redes inteligentes (computación desde la nube). Y tras la introducción del nuevo protocolo de internet IPv6 en 2012, ahora hay suficientes direcciones electrónicas disponibles para permitir la conexión en red directa universal de objetos inteligentes a través de Internet, lo que nos lleva a estar viviendo una nueva revolución industrial. La primera revolución

industrial se llevó a finales del siglo XVIII con la introducción de la máquina de vapor y la mecanización de las tareas manuales para la fabricación industrial. Esta etapa es seguida por la producción de bienes con energía eléctrica a inicios del siglo XX. Esta fue reemplazada alrededor de 1970 con la introducción de la TI (tecnología de la información) para lograr una mayor automatización en los procesos industriales (German, 2013).

Figura 1

Las Cuatro Revoluciones Industriales



Nota. Tomado de Industrie 4.0 Working Group. “Recommendations for implementing the strategic initiative INDUSTRIE 4.0”. Securing the future of German manufacturing industry. Abril 2013. p 13-14.

En la figura 1 se muestra el pasar de las revoluciones industriales y su principal avance tecnológico. Este documento expresa que actualmente el 90% de todos los procesos industriales de fabricación son compatibles con las TIC. Lo que implica que la digitalización de las cosas está presente en nuestras vidas y es un cambio que todas las empresas deben realizar.

Con la llegada de las tecnologías digitales, se han realizado avances referentes a la integración de elementos inteligentes interconectados en la industria. Estas tecnologías permiten la detección remota, monitorización en tiempo real y control de dispositivos y elementos de producción ciber-físicos a través de las infraestructuras de red, proveyendo una integración y sincronización más directa desde el mundo físico hacia el virtual (Stefan Boschert, 2016).

El uso de tecnologías de digitalización a través de herramientas de simulación como los gemelos digitales han permitido la planificación virtual de productos y procesos.

3.2 Monitoreo

3.2.1 ¿Qué es el monitoreo?

Para empezar a aplicar este concepto dentro del proyecto es importante conocer y entender por completo la profundidad y alcance de esta palabra. La teoría de planificación del desarrollo define el seguimiento o monitoreo como un ejercicio destinado a identificar de manera sistemática la calidad del desempeño de un sistema, subsistema o proceso (Valle Otto, Idie) con el fin de realizar ajustes y los cambios pertinentes y oportunos para el logro de los resultados. El monitoreo permite analizar y estudiar el avance y proponer acciones a tomar para lograr los resultados. En la actualidad el monitoreo o seguimiento tiene dos enfoques, uno de ellos se centra en la coincidencia entre lo planificado y lo ocurrido y la otra en el conocimiento que se deriva de las actividades de seguimiento.

El primer enfoque descansa en una visión racional del proceso de planificación. De tal modo que se asume que dados ciertos insumos y recursos se obtendrán ciertos resultados. Así el foco de atención es la verificación si se ha cumplido lo planificado y sugerir cambios para reducir la discrepancia entre uno y otro momento.

En el segundo enfoque busca verificar la validez de algo propuesto, retroalimentado y consecuentemente tomar decisiones estratégicas y operativas fundamentadas, por lo que el monitoreo o seguimiento se traduce en un proceso productivo y gestión de los conocimientos empíricos y en una fuente de aprendizaje que contribuye a una mayor efectividad y un mayor funcionamiento.

3.2.2 Elementos del plan de monitoreo

Un plan de monitoreo está compuesto por una secuencia de acciones y actividades necesarias para la medición y el análisis de desempeño, dichas acciones incluyen el desarrollo de un plan o enunciado, un esquema de indicadores y un esquema de metas.

3.2.2.1 Plan o enunciado. Esta parte describe la racionalidad o el sentido que sustenta la iniciativa con respecto a la realidad que se pretende modificar. Dicho sentido se expresa en la manera en la que se plantean y se ejecutan las actividades, los resultados, los objetivos y los efectos buscados.

3.2.2.2 Esquema de indicadores. Todo objetivo, resultado o producto son medidos por una serie de indicadores con sus respectivas unidades de medida, también se miden por los responsables y las fuentes para la recopilación de los datos sobre el desempeño. Algunas veces los valores de los indicadores están expuestos en aspectos más específicos.

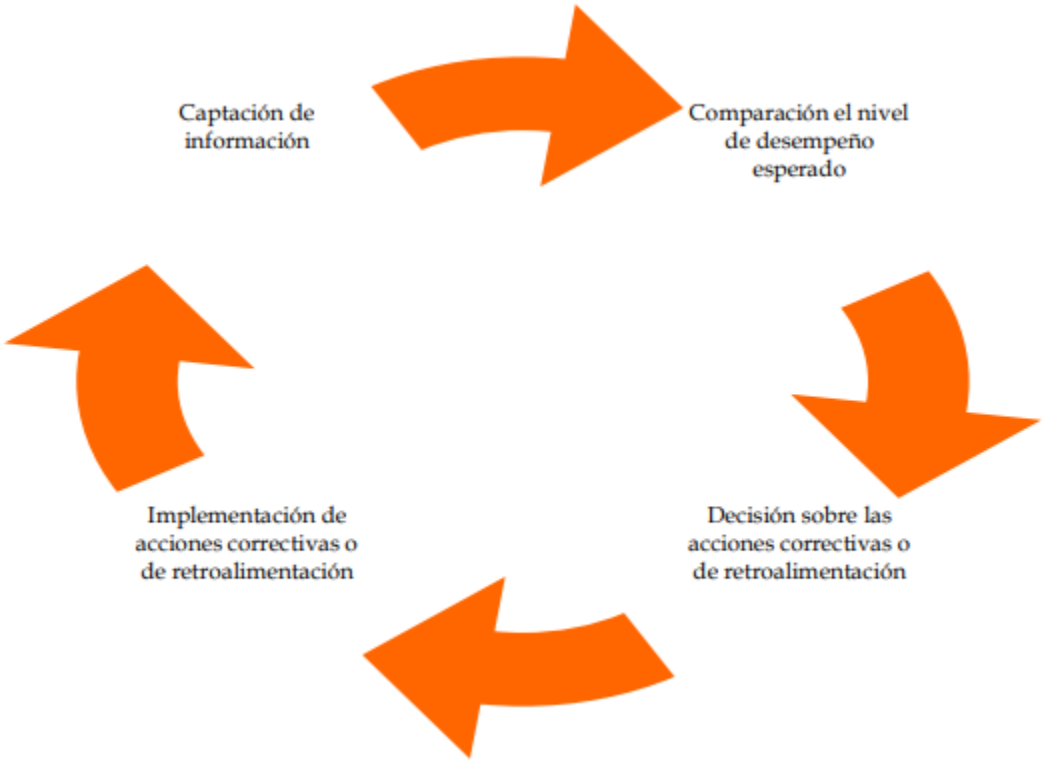
3.2.2.3 Esquema de metas durante el periodo. Este componente es el que permite identificar el comportamiento de los indicadores durante un determinado periodo de tiempo a definir (ya sea, diario, semanal, mensual, anual, etc). Los indicadores pueden medirse o definirse en base al pasado respecto a los valores de línea base o bien a futuro, con respecto a las metas definidas para el ciclo de tiempo definido.

3.2.3 El proceso de monitoreo es un proceso ciclico

El proceso de monitoreo rota constantemente en torno a diferentes énfasis funcionales – desde la toma de datos hasta las intervenciones de énfasis o reorientación.

Figura 2

Secuencia de monitoreo



Nota. Tomado de Idei (instituto para el desarrollo y la innovación educativa). Monitoreo e indicadores.

La figura 2 muestra el proceso de monitoreo, el cual es continuamente implementado, expone los elementos del ciclo de monitoreo, y las relaciones que guardan entre sí. Los elementos del ciclo de monitoreo son los siguientes. Captación de datos, de las fuentes como lo pueden ser sensores conectados al equipo y su registro y conexión en la herramienta respectiva. Comparación

de los datos contra el nivel esperado de cumplimiento. Decisión respecto a las decisiones correctivas y de retroalimentación necesarias de acuerdo a la información recolectada. Implementación que es cuando se ponen en practica las acciones correctivas encontradas en el paso anterior.

3.2.4 Indicadores

Los indicadores son información utilizada para dar seguimiento y ajustar las acciones que un sistema, subsistema o proceso necesita para alcanzar el cumplimiento de los objetivos y metas. Un indicador como unidad de medida permite el monitoreo y evaluación de las variables clave de un sistema organizacional, mediante su comparación, en el tiempo, con referentes externos e internos.

Se le pueden atribuir dos funciones básicas a los indicadores. La función descriptiva que consiste en el aporte de información sobre el estado real de una acción o proyecto, programa, política, etc. La función valorativa que consiste en añadir a la información descriptiva un juicio de valor, lo mas objetivo posible, sobre si el desempeño esta siendo o no el adecuado, para después de esto tomar decisiones que hacen parte del ciclo de monitoreo.

3.2.4.1 Características de los indicadores. No existe un grupo de “indicadores correctos” para medir el nivel actuación. Lo que existe es un rango de posibles señales para medir el cambio en las variables con grados diversos de certeza. El concepto y definición de indicadores varia mucho de un autor a otro y varían entre un mayor o menor número, pero de forma general un “buen indicador” se caracteriza por ser medible, preciso, consistente y sensible.

- **Medible.** Un indicador debe ser medible en términos cuantitativos o cualitativos. La mayor utilidad de un indicador es poder hacer una comparación entre la situación medida

y la esperada. Esto se facilita si durante la planeación, al formular los objetivos y fijar las metas, la definición de los indicadores se hace de tal forma que sea posible medirlo durante el monitoreo y la evaluación.

- **Preciso.** Un indicador debe definirse de forma precisa, debe ser inequivoco, que no permita interpretaciones ambiguas sobre el tipo de dato recolectado ni la forma de medirlo y evaluarlo.
- **Consistente** Un indicador debe ser consistente aun con el paso del tiempo. Es importante que los efectos observados se deban a los cambios reales de la condición de operación del equipo y no por cambios del propio indicador.
- **Sensible.** Es importante que un indicador sea sensible. Un indicador sensible cambiará proporcionalmente y en la misma dirección que los cambios en la condición o concepto que se está midiendo.

Figura 3

Ejemplo indicadores de desempeño

OEE	Calificativo	Consecuencias
OEE < 65%	Inaceptable	Importantes pérdidas económicas. Baja competitividad.
65% < OEE < 75%	Regular	Pérdidas económicas. Aceptable sólo si se está en proceso de mejora.
75% < OEE < 85%	Aceptable	Ligeras pérdidas económicas. Competitividad ligeramente baja.
85% < OEE < 95%	Buena	Buena competitividad. Entramos ya en valores considerados "World Class".
OEE > 95%	Excelente	Competitividad Excelente.

Tomado de:

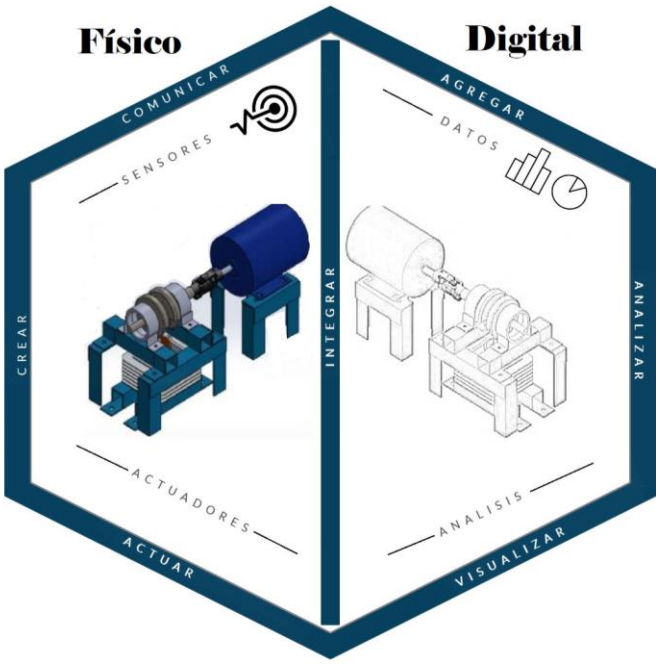
https://www.google.com/url?sa=i&url=https%3A%2F%2Fspcgroup.com.mx%2Foee-el-indicador-clave-del-rendimiento-de-un-proceso

En la figura 3 se muestra un ejemplo de indicadores de desempeño donde miden el OEE (eficacia de una maquina industrial) y lo comparan con un calificativo cualitativo y su respectiva descripción de las consecuencias.

En la figura 4 se muestra un esquema simple donde se evidencia la parte física y la parte digital que se tocará en el avance de este proyecto.

Figura 4

Esquema De La Creación De una interfaz gráfica



3.3 Áreas donde se enfocan las interfaces de monitoreo

Las interfaces graficas de monitoreo suelen utilizarse en varias áreas:

3.3.1 Optimización de operaciones

Mediante el uso de variables como el tiempo atmosférico, el tamaño de una flota, los costes energéticos o los factores de rendimiento, se activan modelos que ejecutan cientos o miles de simulaciones what-if para evaluar la aptitud de los puntos de control actuales del sistema o la necesidad de realizar ajustes en ellos. Esto permite optimizar o controlar las operaciones del sistema durante el funcionamiento y así mitigar riesgos, reducir costes o mejorar eficiencias.

3.3.2 Mantenimiento predictivo

En aplicaciones 4.0 del sector industrial, los modelos pueden determinar la vida útil restante e informar a los encargados de operaciones sobre el momento más oportuno de realizar tareas de mantenimiento o sustituir equipamiento.

3.3.3 Detección de anomalías

El modelo se ejecuta en paralelo a los activos reales y señala de inmediato cualquier comportamiento operativo que se desvíe del comportamiento (simulado) esperado. Por ejemplo, una empresa petrolera puede enviar por streaming datos de sensores procedentes de plataformas petroleras marinas en funcionamiento continuo. El modelo digital buscará anomalías en el comportamiento operativo para ayudar a evitar daños catastróficos.

3.3.4 Aislamiento de fallos

Las anomalías pueden activar una batería de simulaciones para aislar el fallo e identificar la causa raíz; de este modo, los ingenieros o el sistema podrán tomar las medidas adecuadas.

3.3.5 Planificación de diseño

La interfaz ayudará a la evaluación y planificación de sistemas de producción continuos, junto a la adquisición de datos y variables de entorno de forma automática sin necesidad de usar aplicaciones dedicadas.

3.4 Descripción del banco "Evaluación de desgaste"

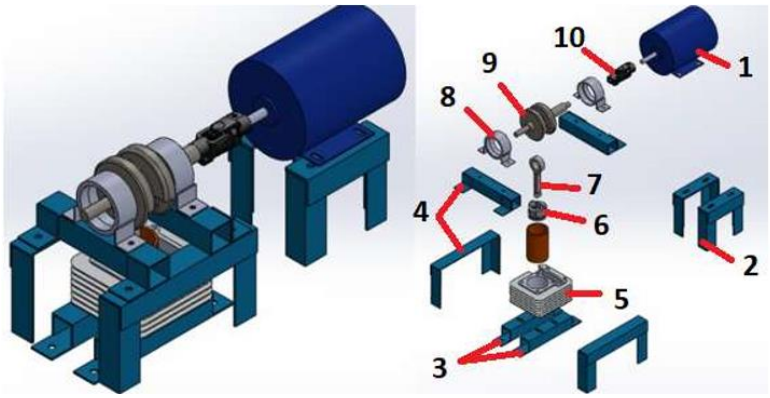
Este proyecto se realizó con la finalidad de analizar el desgaste en la camisa de un motor de combustión interna por el contacto que existe con los anillos del pistón, causado por el constante contacto entre las piezas mecánicas. Para esto se diseñó y construyó un banco de pruebas con un funcionamiento similar al de uno de combustión interna. La investigación se basó en disminuir el desgaste directamente entre las superficies al aplicar un recubrimiento superficial de Nitruro de Zirconio (ZrN). Se analizó la morfología y composición porcentual de los elementos presentes en la superficie de la camisa con y sin recubrimiento mediante la técnica de Microscopia Electrónica de Barrido (MEB). Este banco está disponible en el laboratorio de máquinas térmicas alternativas. A continuación, se muestra el despiece explosionado del banco.

El banco consta de un motor de medio caballo de fuerza que funciona a 1800 RPM, el eje del motor va conectado a dos contrapesos de un cigüeñal que convierten el movimiento rotativo del motor en movimiento lineal de un pistón.

El bloque es de una motocicleta YBR 125 de una capacidad de 123cc de referencia 5VL00 con su perspectivo juego de pistón, anillos, camisa y accesorios y piezas del motor de la motocicleta. Los contrapesos tienen un diámetro de 10,2 cm con una distancia de 26 mm entre el centro del contrapeso y el centro del eje que va unido al brazo del pistón, lo que permite estimar la carrera del pistón como el doble de esta distancia, es decir 5,2 cm.

Figura 5

Despiece Explosionado Banco De Evaluación De Desgaste



Nota. Tomado de CARRILLO, Sergio; SALAMANCA, Miguel. Análisis de desgaste en la camisa de un motor de combustión interna con recubrimiento superficial, 2020. p.55.

1) Motor eléctrico.
2) Soporte del motor eléctrico.
3) Soporte del bloque.
4) Soporte de las chumaceras del cigüeñal.
5) Bloque.
6) Pistón o embolo.
7) Biela.
8) Chumacera del cigüeñal.
9) Cigüeñal.
10) Acople.

Se generan fuerzas y momentos los cuales producen desgaste, fricción que genera un aumento indeseado de la temperatura y vibraciones durante el movimiento alternativo del pistón

respecto a demás componentes. Con ayuda de instrumentos de medición se pueden determinar estos efectos y evaluar su impacto sobre el equipo.

Figura 6

Configuración Del Banco

4 Metodología

En esta sección se aplicará en detalle cómo se plantea la solución a los objetivos específicos, se mencionarán y aplicarán las estrategias, técnicas e instrumentos necesarios para la realización del proyecto.

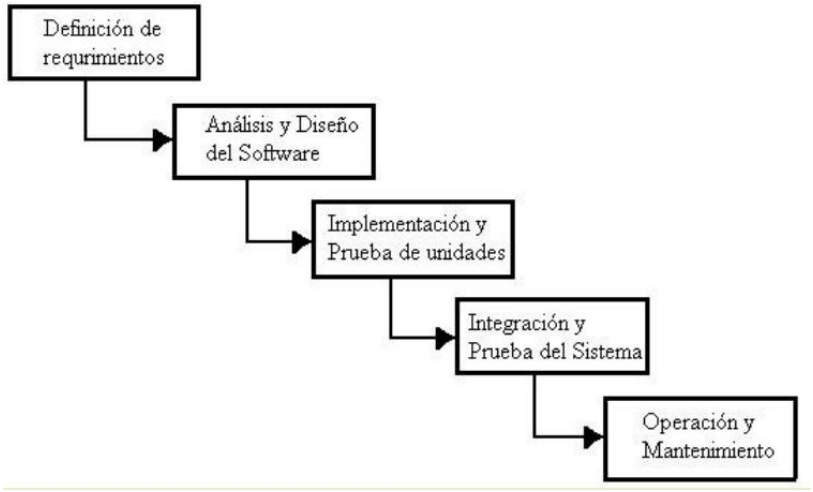
El proceso se dividirá en dos subsistemas, el hardware que lo conforman los elementos físicos implementados en el proyecto tales como, el computador, los sensores y la tarjeta de adquisición de datos, y el software conformado por la interfaz gráfica y el modelo CAD del banco "Evaluación de desgaste".

4.1. Metodología incremental

Para el subsistema del software vamos a implementar la metodología de cascada o metodología incremental que tiene su origen en el modelo ingeniado por Winston Royce, donde se presentan 5 fases metodológicas para el desarrollo de software visualizada en la Figura 7.

Figura 7

Metodología Incremental



Nota. Tomado de http://procesosdesoftware.blogspot.mx/. 4 de octubre de 2016

4.1.1 Definición de requerimientos

Son los requisitos que debe cumplir el software en presencia del operador.

4.1.2 Análisis y diseño de software

Estructura de los datos, la arquitectura del software, la presentación de la interfaz con las diferentes ventanas y funciones básicas del programa.

4.1.3 Implementación y prueba de unidades

Se conecta el diseño de tal forma que sea comprendido por la máquina. Se codifican y programan las acciones de la interfaz gráfica. También se hará la conexión del hardware.

4.1.4 Integración y prueba del sistema

Se realizan pruebas simuladas o reales controladas para comprobar las sentencias, que los procesos lógicos funcionen correctamente y se detecten errores.

4.1.5 Operación y mantenimiento

Se realizan los cambios necesarios para el óptimo funcionamiento de la interfaz gráfica, se añaden otras funciones que se vean necesarias sobre el código ya existente.

4.2 Desarrollo del software

4.2.1 Definición de requerimientos

4.2.1.1 Requerimientos generales. A continuación, se describirán los requerimientos generales que deben tener la interfaz y el software donde se realizará.

- **Interfaz intuitiva.** Para que cualquier usuario pueda usarla de forma rápida, el programa se hará con la intención de que sea sencillo entender el funcionamiento y el fin con el cual se realizó esta práctica, el contenido de la interfaz debe ser amigable, cómoda, intuitiva y eficaz. Se diseñará con distintos botones, menús, listas, gráficos, tablas y cualquier otra herramienta que esté disponible en el software para sintetizar y realizar las acciones de la forma más rápida y sencilla para el manejo del usuario.
- **Software asequible.** El software donde se va a realizar la interfaz debe ser uno con licencia gratuita para los estudiantes, para que la interfaz se pueda abrir desde cualquier ordenador. Es por esto por lo que se realizará la interfaz gráfica desde el software de MATLAB, ya que la universidad cuenta con licencia gratuita y los estudiantes podrán tener acceso desde su computador personal a esta interfaz y conectarla fácilmente al banco desde el laboratorio de mantenimiento de la escuela de ingeniería mecánica.
- **Acceso libre para los estudiantes.** La interfaz del software como ya se mencionó se encontrará escrita en lenguaje de programación de MATLAB, con ayuda de la herramienta AppDesigner de este programa se hará mucho más sencillo el diseño de la interfaz gráfica, y los usuarios que tengan conocimiento del programa podrán cambiar variables de mediciones tales como el número de muestras y el tiempo de lectura de las mismas para hacer más flexible el entendimiento y el funcionamiento de la interfaz gráfica.

En cuanto al apartado del software cabe mencionar que la App donde se diseñará la interfaz gráfica se realizará en la versión de Windows 10 y en MATLAB R2021a.

4.2.1.2 Requisitos funcionales. Los requisitos funcionales son aquellos donde se manipulan datos para realizar cálculos, algunos detalles técnicos y otras funciones específicas que

el sistema debe cumplir, para este proyecto se requiere que la interfaz cumpla los siguientes requisitos.

El sistema debe permitir al usuario visualizar las variables independientes medidas del banco de laboratorio, es decir, el usuario puede visualizar cada variable por separado y tener la capacidad de tomar acciones tales como, graficar los datos que están siendo medidos, guardar datos de interés y exportarlos a un programa externo para su análisis correspondiente y establecer un punto de referencia para que la interfaz genere una alarma visual y sonora en caso de pasar dicho límite.

La interfaz debe tener un tablero fijo donde se puedan manipular el estado del motor del banco (prenderlo y apagarlo según convenga), la visualización de la fecha y el estado de la interfaz, es decir, si la interfaz gráfica se encuentra o no recibiendo los datos entregados por los sensores conectados al banco.

La interfaz tendrá un espacio fijo donde se visualice el estado de las mediciones, es decir, que el usuario sepa si las mediciones están activas o no en ese momento.

El sistema tendrá un apartado donde se puedan tabular los valores máximos alcanzados por las mediciones en el tiempo que estuvo activa la interfaz gráfica, en esta misma tabla el usuario podrá seleccionar si el activo físico necesita revisión o no dependiendo de los valores máximos obtenidos en dichas mediciones.

4.2.1.3 Requisitos no funcionales. Este apartado especifica criterios que se usan para juzgar la operación de un sistema. Se refieran a todos los requisitos que no describen información a guardar ni funciones a realizar sino características de funcionamiento, para este proyecto se requiere que la interfaz cumpla los siguientes requisitos no funcionales.

El sistema debe garantizar la correcta utilización de la interfaz, es decir, que el usuario no cometa ningún error al momento de usarla, tal como, cerrar la interfaz por accidente sin guardar los datos medidos, limpiar las gráficas dándole al botón por accidente, entre otras.

Para la utilización de la interfaz se realizará una guía de laboratorio para que los estudiantes puedan aprender a usar el banco de forma sencilla, y desde ella tendrán acceso a dicho archivo desde la pestaña “ayuda”.

La interfaz debe ser fluida y muy intuitiva, con ayudas visuales y sonoras para que sea muy sencillo la utilización por parte del usuario independientemente de si tiene conocimiento o alguna capacitación para su uso.

4.2.2 Análisis del diseño.

Se pensó hacer el diseño de la interfaz sencilla y fácil de entender para los usuarios. El usuario puede monitorear las variables por separado, cada pestaña con su respectivo grafico que lee los datos entregados por los sensores en tiempo real con un máximo de 10 min por toma, un botón para guardar los datos en una tabla que se encuentra en la última pestaña de la interfaz, un medidor analógico que muestra el valor de la medida, al igual que un cuadro de texto que muestra el valor exacto de la medida en el momento. Además, la pestaña de temperatura contará con alarmas que se activarán una vez sobrepase el valor de referencia. La interfaz contará también con un cuadro fijo (no se altera al cambiar de pestaña) donde el usuario puede seleccionar la fecha, un botón donde se activen las funciones, es decir, el software se activará solo si el botón está en “ON”, además un botón intuitivo donde el usuario puede visualizar el estado el motor y una imagen del modelo del banco que interactúa con la temperatura actual. Y por último en la parte de arriba de la ventana de la interfaz se encontrarán los botones de inicio o el “Toolbar”, donde se encuentran los botones “Exportar” y “Ayuda”.

4.2.3 Diseño del software

El diseño y programación de la interfaz se hará en el software de MATLAB con la ayuda de la herramienta AppDesigner, así que los botones, gráficos, switches y todo lo visible que contenga el programa serán componentes de librerías propias del software.

4.2.3.1 Función pestañas. El usuario puede navegar entre 4 pestañas diferentes "temperatura", "Velocidad", "Vibraciones" y "Tablas" (Figura 8) donde podrá monitorear las variables por separado o visualizar sus variaciones en una solo pestaña, para esto se utilizó la librería "Tab Group". Se adecuó el espacio de las ventanas para dejar un espacio para el cuadro fijo de la interfaz, mostrado en la Figura 8.

Figura 8

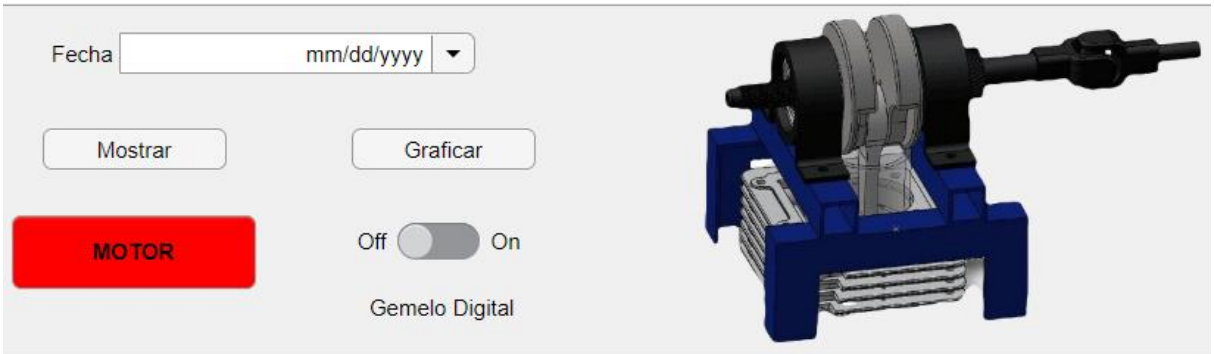
Pestañas De La Interfaz

4.2.3.2 Cuadro Fijo. Aquí como se dijo anteriormente se añadirán los botones de fecha, de encendido y apagado de la interfaz gráfica, de graficar, de mostrar las mediciones, de visualizar el estado del motor y una imagen del modelo del banco donde se realizan las mediciones. Para esto, se insertó un "Date picker" para seleccionar de una manera fácil y rápida la fecha, un "Switch" para encender y apagar el software, un "State Button" para el estado del motor, un "Button" para mostrar y otro para graficar y una "Image" para cargar las imágenes. Para el botón del motor se estableció el color "Verde" para el estado del motor encendido y "Rojo" para el estado apagado y de esta forma hacerlo más intuitivo para el usuario. El switch de inicio y el botón del estado del motor empiezan en estado "OFF", es decir, el usuario debe activarlos para activar la interfaz y cambiar el estado del botón del motor respectivamente. La imagen predeterminada es una del

banco sin movimiento, una vez el usuario active el botón del estado del motor esta imagen cambiará a un gif para visualizar el movimiento del banco, además esta imagen se actualizará dependiendo de la temperatura medida con un color específico en la camisa para simular la gama de colores que se establecerá dependiendo de la temperatura de referencia elegida por el usuario. El diseño del cuadro fijo se muestra en la Figura 9.

Figura 9

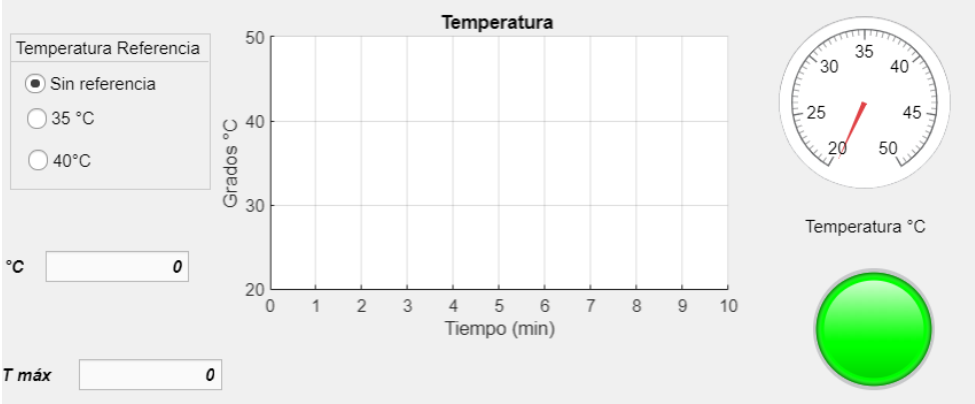
Cuadro Fijo



4.2.3.3 Pestaña "temperatura". La pestaña contiene un "Radio Button Group" que permite seleccionar la temperatura de referencia para la alarma, una "Lamp" que cambia de color dependiendo si la medición sobrepasó o igualó la temperatura de referencia seleccionada, un "Gauge" que muestra el valor de la temperatura en un medidor estilo analógico, un "Edit Field (Numeric)" para el valor de la temperatura medido en tiempo real y otro para el valor máximo de las mediciones y un "Axes" para el grafico de las mediciones. El diseño de la pestaña de temperatura se muestra en la Figura 10.

Figura 10

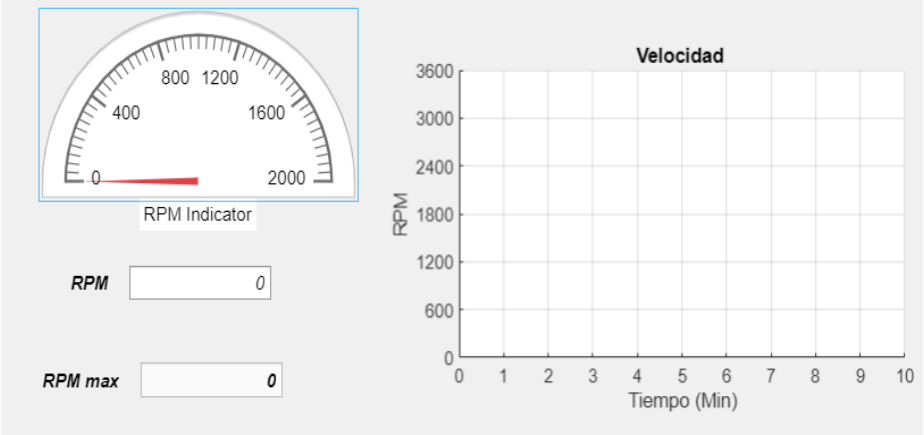
Pestaña Temperatura



4.2.3.4 Pestaña "Velocidad". Esta pestaña contiene toda la información necesaria para la medición de las RPM. Contiene un "Gauge" que muestra el valor de las RPM en un medidor estilo analógico, un "Edit Field(Numeric)" para el valor de la velocidad medido en tiempo real, otro "EdiT Field(Numeric)" para el valor máximo de las mediciones y un "Axes" para el grafico de la medición. El diseño de la pestaña de RPM se muestra en la Figura 11.

Figura 11

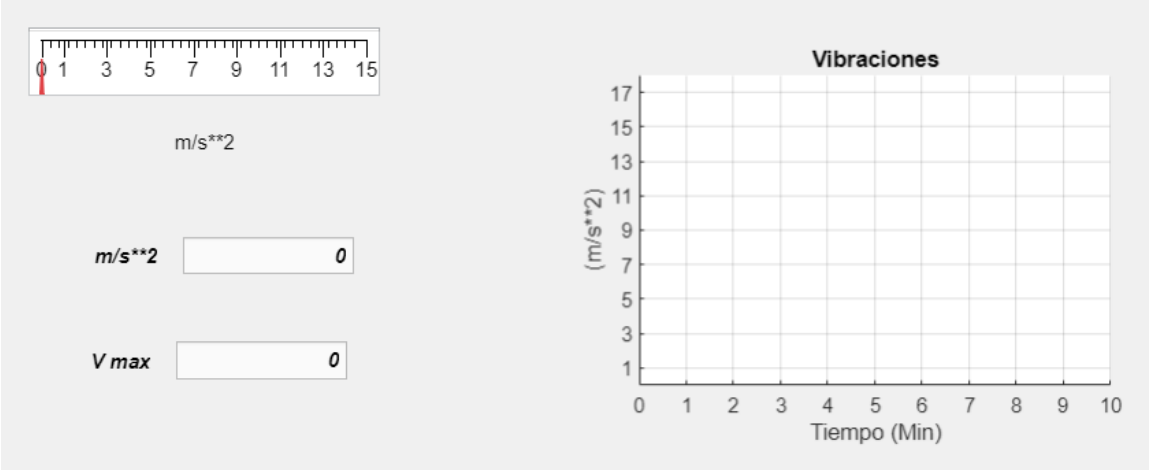
Pestaña RPM



4.2.3.5 Pestaña "Vibraciones". Esta pestaña contiene toda la información necesaria para la medición de las vibraciones, la pestaña contiene "Linear Gauge" que muestra el valor de las vibraciones en un medidor estilo analógico, un "EdiT Field(Numeric)" para el valor medido en tiempo real, otro "EdiT Field(Numeric)" para visualizar el valor máximo de las mediciones y un "Axes" para visualizar el grafico. El diseño de la pestañan de vibraciones se muestra en la Figura 12.

Figura 12

Pestaña Vibraciones

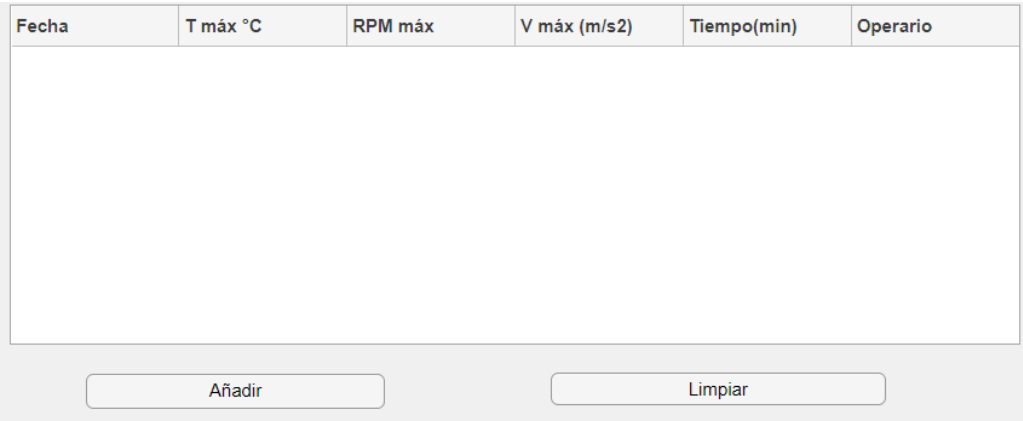


4.2.3.6 Pestaña "Tablas". En esta pestaña el usuario podrá visualizar los valores de las mediciones que guardó anteriormente de cada variable, se usó la librería "Table" y esta tabla contiene un espacio para la fecha que se llena automáticamente con la seleccionada en el "Date Picker" en el cuadro fijo, 3 cuadros que muestran los valores máximos de las mediciones, una celda desplegable para seleccionar el operario que trabajó durante el tiempo de medición (Simulando el proceso en la industria) y una celda donde el usuario puede marcar si el banco necesita revisión, dependiendo si sobrepasó los límites de referencia definidos anteriormente

durante el proceso. La pestaña contiene el botón “Añadir” que autocompletará una fila de datos, este también reiniciará los vectores de las mediciones máximas y del tiempo, por lo que, se visualizarán los valores máximos entre el tiempo que se presiona el botón y se vuelve a presionar. Además, se cuenta con un botón “Limpiar” que lo que hace es limpiar la tabla, es decir, elimina todas las celdas. El usuario tendrá la posibilidad de exportar estos datos de la tabla a un archivo Excel o a un documento tipo texto desde un botón definido en el “Tool Bar” de la interfaz gráfica. El diseño de la pestaña de tabla se muestra en la Figura 13.

Figura 13

Pestaña Tabla



4.2.3.7 Función barra de herramientas. El usuario puede elegir entre 2 herramientas diferentes, “Exportar” donde puede exportar los datos obtenidos en la tabla en un archivo de Excel o en un bloc de notas y la herramienta “Ayuda” donde encontrará principalmente el archivo PDF de la guía de laboratorio para que pueda usar el banco de la manera correcta y obtener información adicional sobre el funcionamiento de la interfaz gráfica. Además, tendrá ayuda para corregir algunos posibles errores que se presenten durante la ejecución de la práctica de laboratorio. Para esto se utilizó la librería “Tool Bar”. El diseño se muestra en la figura 14.

Figura 14

Tool Bar

Se realizó una encuesta descriptiva a 5 personas 3 de ellas estudiantes de la universidad donde se les preguntó sobre el uso y manejo de la interfaz. Arrojando los siguientes resultados.

Tabla 1.

Resultados encuesta uso de la interfaz

	Muy bueno	Bueno	Regular	Malo	Muy malo
Experiencia de uso		1	2	2	
Facilidad de uso		4	1		
Tiempo de respuesta			2	3	
Sencillez de la interfaz		3	2		

4.2.4 Implementación y prueba de unidades

En esta parte se mencionará el procedimiento de conexión del hardware con la interfaz gráfica. Luego, se hará énfasis en la codificación de las acciones de la interfaz, es decir, la programación de las funciones de la interfaz.

4.2.4.1 Selección del hardware. Para la selección de los componentes se realizaron matrices PUGH para los sensores de Temperatura y de Vibraciones y para la tarjeta de adquisición de datos, con la característica general de que todos ellos son energizados con 5V.

Tarjeta de adquisición de datos: Se compararon 2 de las tarjetas más usadas para temas de aprendizaje, las cuales son Arduino UNO y MYDAQ de NI. Arrojando la matriz PUGH con los siguientes resultados.

Tabla 2.

Matriz PUGH análisis Adquisición de datos

Criterios	Arduino Uno	MyDAQ
Compatibilidad con el Hardware	1	0
Puertos	1	0
Durabilidad	0	1
Soporte	1	1
Programación	1	1
Precio	1	-1
Total	5	2

De esta matriz se puede concluir que la tarjeta Arduino UNO es la mejor alternativa con un mayor número de puntos positivos en comparación a su rival, debido al precio y a la compatibilidad con la mayoría de los sensores del mercado. En la figura 14 se muestra la placa se utilizará.

Figura 15

Arduino UNO

Sensor de temperatura: Para el sensor de temperatura se evaluaron 3 alternativas, el sensor analógico LM35, y los sensores digitales DS18B20 y el sensor TC74. De igual forma se construyó la matriz PUGH para estas alternativas arrojando los siguientes resultados.

Tabla 3.

Matriz PUGH selección sensor de temperatura

Criterios	LM35	DS18B20	TC74
Compatibilidad con el Hardware	1	1	1
Puertos	-1	1	0
Durabilidad	1	1	1
Soporte	-1	1	1
Programación	1	1	-1
Precio	1	0	1
Total	2	5	3

De esta matriz se puede concluir que la mejor opción es el sensor digital DS8B20 debido a que supera a sus rivales en muchos aspectos, principalmente la precisión de las medidas la durabilidad y al ser un sensor digital hay muchos menos errores en las medidas comparados con sensores analógicos como el LM35. En la figura 15 se muestra una imagen del sensor utilizado.

Figura 16

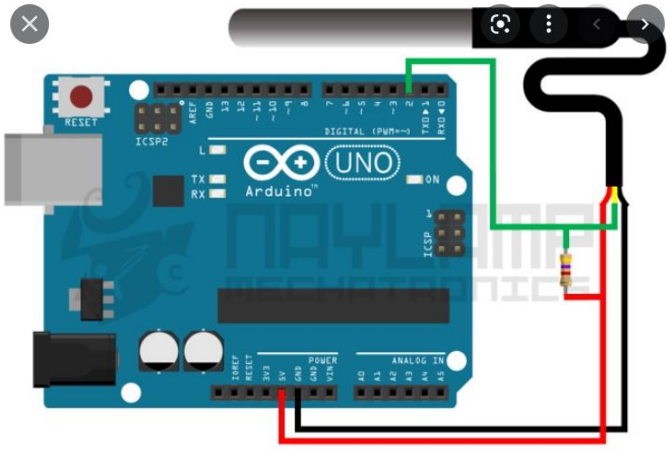
Sensor DS18B20

Para la conexión de este sensor se utilizará una resistencia de 4.7k ohmios que permitirán recibir la señal en grados centígrados para el largo de cable menor a 5 metros. Se conecta el pin de alimentación de 5V a la resistencia en paralelo junto con el pin de la señal digital.

Este sensor usa el protocolo 1-wire que significa un cable, haciendo referencia a que se pueden conectar más de un sensor de estas características con el mismo código. En la figura 17 se muestra el esquema de conexión de este sensor.

Figura 17

Conexión DS18B20



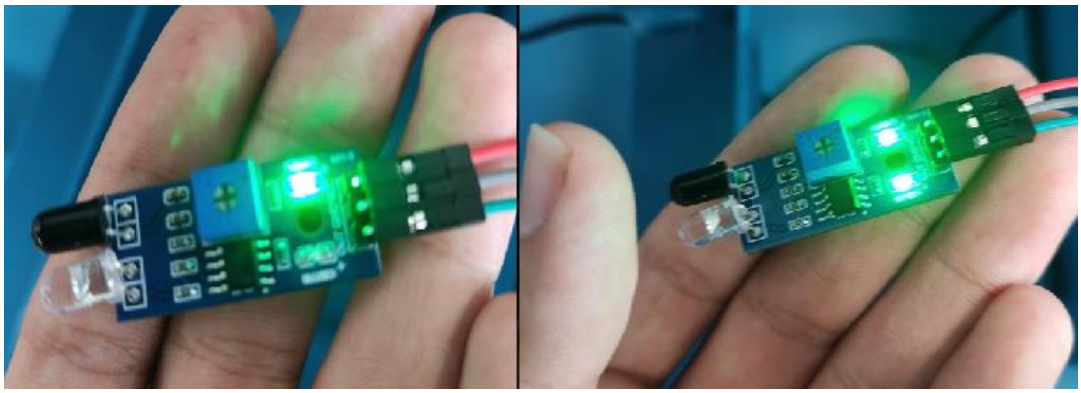
Nota. Tomado de https://naylampmechatronics.com/blog/

Sensor de RPM: Para esta parte no se utilizará un sensor nativo de RPM, sin embargo, con el conocimiento adquirido en clases de diseño se realizará una lectura tipo *encoder* con un sensor *infrarrojo* o también conocido como sensor *detector de obstáculos*. El sensor utilizado es de referencia *SENOBST-2* que lo que hace es mandar una señal alta (1 lógico) si detecta un obstáculo o una señal baja (0 lógico) si no detecta nada. La funcionalidad es bastante simple, cuenta con dos LED's, uno infrarrojo que emite luz infrarroja y un receptor, esa señal infrarroja al entrar

en contacto con un obstáculo rebota y el LED receptor lo detecta y envía una señal alta por el pin de señal. Mas adelante en la codificación de los sensores se explicará mejor la lectura que se realiza. El sensor tiene 3 pines, *VCC* de alimentación de 5V, *GND* de la conexión a tierra y el pin *OUT* de señal digital. Sensor mostrado en la figura 18.

Figura 18

Sensor Detector De Obstáculos



La conexión de este sensor es bastante más sencilla comparada con conexión del sensor de temperatura. Se conecta el pin de alimentación a la *protoboard* en la línea de 5V, de igual forma se conecta el pin *GND* a la línea de tierra y por último el pin de señal se conecta al pin digital "8" del *Arduino.*

Sensor de vibraciones: Para el sensor de vibraciones se evaluaron 3 alternativas de acelerómetros, MPU6050, MPU9250 y el sensor ADLX345. Se realizó de igual forma la matriz PUGH arrojando los siguientes resultados.

Tabla 4.

Matriz PUGH selección del acelerómetro

Criterios	ADXL345	MPU6050	MPU9250
Compatibilidad con el Hardware	0	1	1
Puertos	0	0	0
Durabilidad	1	1	1
Soporte	0	0	0
Programación	1	1	1
Precio	-1	1	0
Total	1	4	3

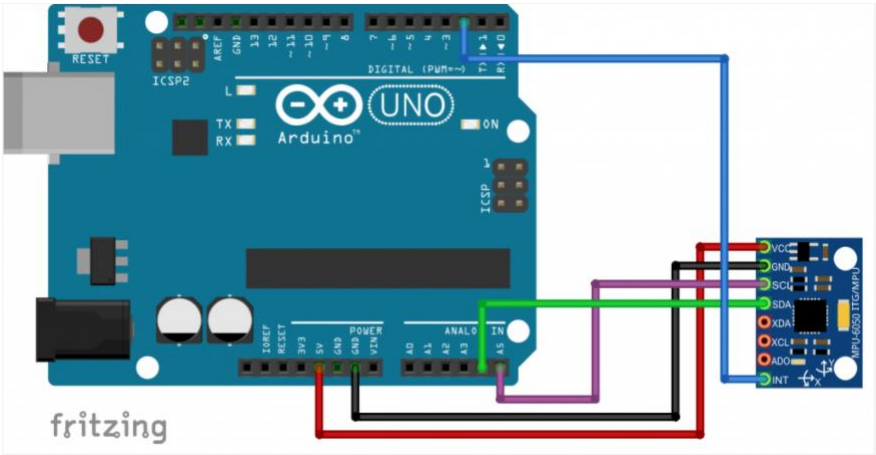
De esta matriz se puede concluir que la mejor opción es el sensor MPU6050, el cual obtuvo una puntuación muy similar a la del MPU9250 pero es un poco más económico.

Este sensor *MPU6050*, un sensor analógico con características sobresalientes para su precio. Este sensor es un acelerómetro y giroscopio, el cual funciona para los 3 ejes de rotación, además sirve como sensor de temperatura y algunas referencias tiene medidor de humedad. Para esta práctica solo se utilizará la parte del acelerómetro para visualizar las vibraciones de banco (en m/s2). El acelerómetro mide la aceleración, inclinación o vibración y transforma la magnitud física de aceleración en otra magnitud eléctrica que será la que emplearemos en los equipos de adquisición estándar. Las mediciones van desde decimas de g, hasta los miles de g (HERPRO)

El *MPU6050* contiene tanto acelerómetro como giroscopio en un solo empaque, es de 16 bits de resolución, lo que significa que divide el rango dinámico en 65536 fracciones. Este sensor es bastante utilizado en robótica en medición de vibraciones y en sistemas de medición inercial. Sensor y conexión mostrado en la Figura 18.

Figura 19

Conexión Sensor Vibraciones



Nota. Tomado de *Usando el MPU6050 | Tienda y Tutoriales Arduino (prometec.net)*

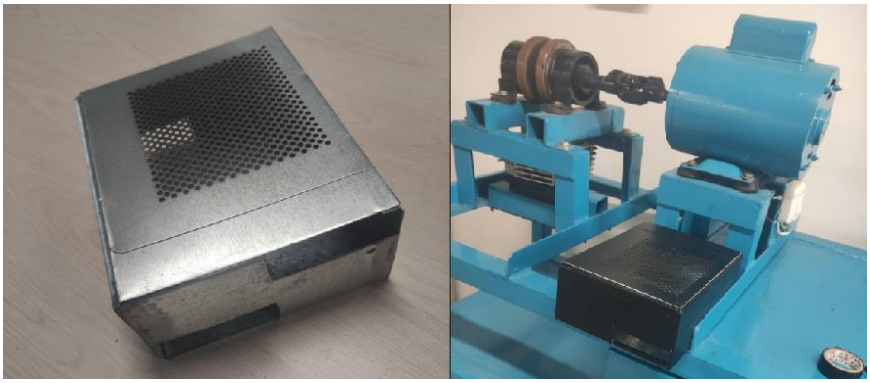
4.2.4.2 Conexión del hardware en el banco. En esta parte se mostrará la posición de los sensores en el banco de laboratorio. Se evaluaron varias posibilidades y se estudió el mejor posicionamiento con los resultados de las pruebas.

Arduinos. Para este proyecto se utilizaron dos tarjetas *Arduino* una que será la encargada de enviar la información de los sensores de RPM y vibraciones al computador a través de MATLAB y Simulink. El otro *Arduino* será el encargado de enviar la información de las mediciones de temperatura por medio del IDE de *Arduino* a la interfaz de MATLAB, ya que el sensor DS18B20 utilizado no tiene compatibilidad con las librerías de Simulink.

La parte de adquisición de datos estará dentro de la caja de control Figura 20, allí irán las dos tarjetas *Arduino,* la protoboard y las conexiones con los sensores.

Figura 20

Caja de control

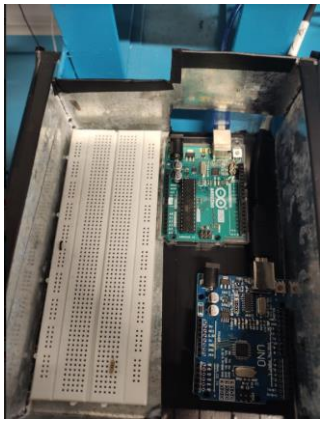


Como se mencionó anteriormente dentro de esta caja irán todas las conexiones de los sensores junto con las dos tarjetas *Arduino.* La caja se diseñó de forma que entrara a presión en el banco y con las medidas justas para que su contenido quedara bien ubicado en su interior. Tiene 3 ranuras ubicadas de forma que los cables USB de los *Arduinos* y los cables de conexión salgan y se conecten con facilidad al computador y a los sensores, respectivamente. La caja fue forrada con cinta aislante para que los componentes electrónicos, principalmente los pines inferiores de la tarjeta *Arduino CH341* (Color más oscuro) no hiciera tierra con la caja metálica, al igual que algunos otros componentes donde se ubicaron los sensores.

A la *protoboard* se conectaron dos líneas principales, la de tierra (la línea externa de la *proto* más alejada de los *Arduinos)* y la línea de 5V (la línea externa de la *proto* más cercana a los Arduinos). Estas dos líneas salen del *Arduino UNO r3* (el azul aguamarina) y son usadas por los sensores de RPM y de vibraciones, y al otro *Arduino* se conectan los pines de alimentación y de tierra del sensor de temperatura directamente a la tarjeta Figura 20.

Figura 21

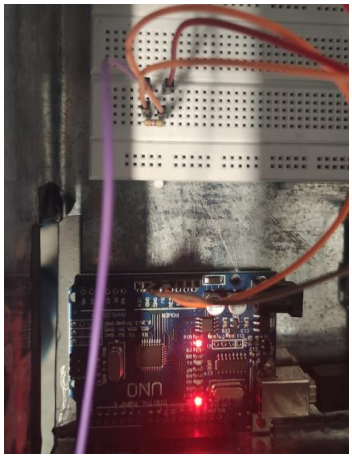
Ubicación De Los Arduinos



Sensor de temperatura. El sensor DS18B20 debe medir la temperatura de la camisa, sin embargo, durante las pruebas con ayuda de la cámara termográfica se puede visualizar que la temperatura en la parte exterior no es uniforme, y la conexión en la parte interior la hace imposible el movimiento del pistón, por lo que se decidió poner el sensor dentro del bloque, en una parte donde la temperatura es uniforme y de esta forma las lecturas son más precisas. La figura 21 muestra la conexión del sensor en la protoboard.

Figura 22

Conexión En El Banco Sensor Temperatura



En las Figura 23 y 24 se observan las imágenes de la cámara termográfica, de las distribuciones de temperatura en la camisa y las distribuciones de temperatura en el bloque del motor, respectivamente. Se observa que en la parte exterior de la camisa la temperatura es mucho menos uniforme y en el bloque del motor es constante en toda su superficie, además, por temas de geometría del sensor es más cómodo ubicarlo en esta parte. Por último, en la Figura 25 se muestra la ubicación final del sensor de temperatura.

Figura 23

Cámara Termográfica Camisa

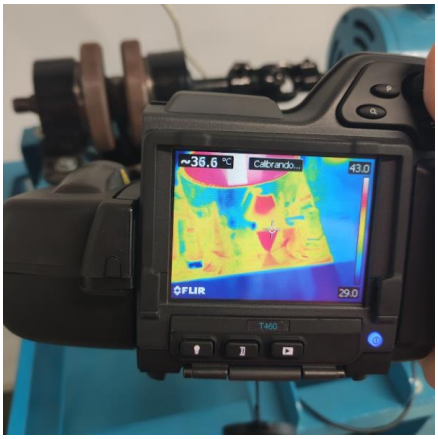


Figura 24

Cámara Termográfica Bloque Del Motor

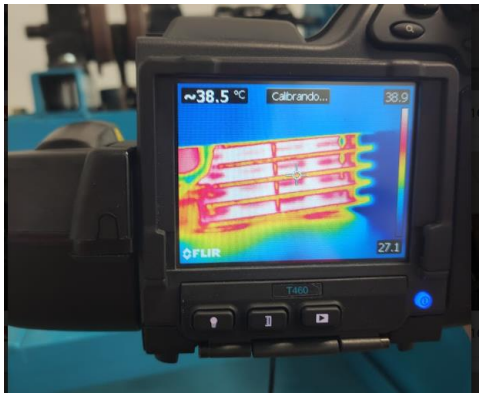
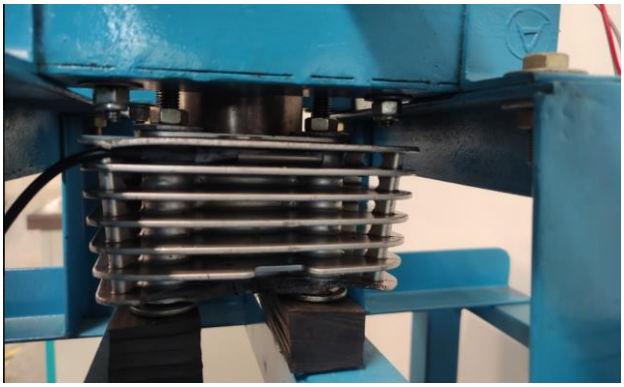


Figura 25

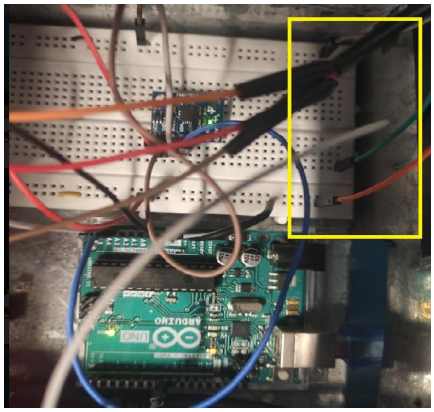
Ubicación Del Sensor De Temperatura



Sensor de RPM. Este sensor debía ubicarse en un lugar donde el eje o algún elemento de la maquina funcionara como un encoder, es decir, por cada revolución del motor este elemento pasara por el frente del sensor y así poder realizar los cálculos respectivos con la señal recibida. Se evaluaron varias partes del banco para ponerlo, principalmente en el eje de salida del motor, sin embargo, el eje está muy separado de las superficies del banco lo que hace difícil ubicar el sensor de manera que tome las medidas correctamente. Finalmente, se diseñó un puente que permitiera colocar el sensor frente a la unión del brazo del pistón y los contrapesos, un lugar bastante seguro y con muy poco error en la medición. La conexión de este sensor se muestra en la Figura 26.

Figura 26

Conexión Sensor Infrarrojo



El puente del sensor se diseñó de tal forma que entrará a presión en el banco, está hecho del mismo material de la caja y de igual forma se forró en cinta aislante para evitar que se conectara a tierra por los pines exteriores del sensor. La ubicación se muestra en la figura 27.

Figura 27

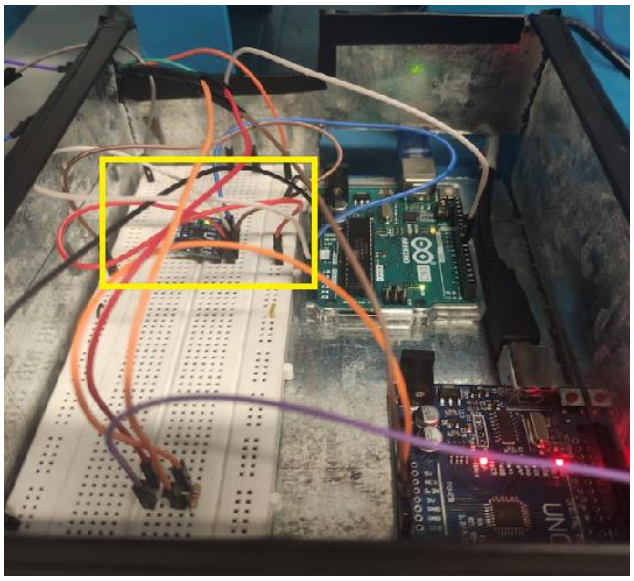
Ubicación Sensor De Obstáculos

Sensor de vibraciones. Hubo varios problemas para la ubicación de este sensor, ya que principalmente se quería medir vibración en las chumaceras como normalmente se hace en las

maquinas rotativas, sin embargo, fue imposible debido a varios factores. Primero, la geometría de las chumaceras y del sensor. Segundo, el material metálico del sensor hace que los pines exteriores del sensor se conecten a tierra y dañe todo el proceso de ejecución de la interfaz. Y, por último, al ser una maquina con vibraciones bastante elevadas hace que la comunicación con el sensor se pierda en algunas ocasiones y las medidas se vean afectadas. Por lo que, se optó por ubicar el sensor dentro de la *caja de control,* de esta manera se tomarán las vibraciones del banco. En las recomendaciones del proyecto se mencionará esta parte, para que aquel que siga con este proyecto mejore este apartado. La ubicación de este sensor se muestra en la Figura 28.

Figura 28

Conexión Y Ubicación Del Sensor De Vibraciones



4.2.4.3 Codificación de las acciones de la interfaz gráfica. En este apartado se mostrarán todas las codificaciones de cada pestaña y de las funciones de las librerías usadas. Se explicará muy brevemente la línea de código, que función cumple y algunas relaciones entre

librerías. Para esto es necesario entender que la línea de código correspondiente a las librerías solo se ejecuta si hay un cambio en el estado del *Widget,* por ejemplo, la línea del código de los botones se ejecuta si se da clic en dicho botón, y en el código está implícito que hay un cambio en el valor y se ejecuta.

Empezaremos por mostrar las propiedades de la interfaz (Figura 29), estos son valores globales que nos ayudarán durante todo el código, se llaman con el prefijo *app.*

Figura 29

Propiedades Globales De La Interfaz

```
properties (Access = private)
    Datostabla % Description
    datos
    tiempo
    datosrpm
    datosv
    pserial
    k=0

    maxt
    maxrpm
    maxv
    mr=0
end

methods (Access = private)

    function results = func(app)

    end
end
```

Se declararon como variables globales los vectores de los datos de temperatura (*datos),* RPM (*datosrpm),* vibraciones (*datosv),* y sus correspondientes vectores para almacenar los valores máximos *maxt, maxrpm* y *maxv.* Además, el vector para almacenar la cantidad de mediciones *tiempo,* la variable *pserial* para la comunicación serial con Arduino y las constantes *k y mr* que servirán de referencia para almacenar los datos de los vectores.

Luego, viene el *StartUpFunction* que básicamente es todas aquellas instrucciones que se ejecutan una vez se abra la interfaz. En él van a estar todas las conexiones externas de la interfaz al igual que la inicialización de la tabla. Se muestra en la Figura 30.

Figura 30

StartUpFunction

```
function startupFcn(app)

    app.tabla.Data = cell(0,6);
    app.tabla.ColumnFormat = {'char','numeric','numeric','numeric','numeric',
        {'Operario 1','Operario 2','Operario 3'},'logical'};

   load_system('Simulink.slx')
   find_system
   set_param(gcs,'SimulationMode','external');
   delete(instrfind({'port'},{'COM5'}));
   app.pserial=serial('COM5','BaudRate',115200);
   fopen(app.pserial);


end
```

Aquí se define el tamaño de las celdas de la tabla al igual que el formato de los valores que contiene. El primer dato que recibe la tabla es tipo *char* o cadena de texto, más adelante en el botón de añadir se hará el casteo del tipo de dato *fecha* a *char,* es por esto por lo que se define así. Se tienen 4 campos numéricos que serán los correspondientes a las mediciones. Un arreglo con los 3 operarios y un campo lógico que corresponde al *check box* de la revisión del banco.

También se tienen las instrucciones de inicialización de las conexiones tanto con el diagrama de bloques de Simulink llamado *Simulink.xls* y con el IDE de Arduino.

Cuadro Fijo: En el cuadro fijo tenemos 6 librerías que componen la fecha, el botón de encendido y apagado de la interfaz, el botón de mostrar las mediciones, el botón de graficar y del botón del estado del motor. Para la fecha se utilizó la librería “Date Picker” y no se crearon líneas de código directas en el callback, sin embargo, se utilizará en la pestaña “Tablas” para auto llenar

la celda “Fecha”. Para el Botón *Switch* se generó la siguiente línea de código en su callback (Figura 31).

Figura 31

Callback Switch de inicio

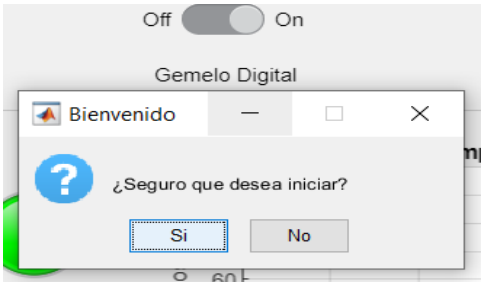
```
function GemeloDigitalSwitchValueChanged(app, event)

    value = app.GemeloDigitalSwitch.Value;
    if strcmp (value,'On')
        seleccion=questdlg('¿Seguro que desea iniciar?','Bienvenido','Si','No','Si');
        if strcmp(seleccion,'Si')
            set_param(gcs,'SimulationCommand','start');
            app.Image.ImageSource='Gif Gemelo-unscreen.gif';
        end
        if strcmp(seleccion,'No')
            app.GemeloDigitalSwitch.Value = 'Off';
        end
    end
    if strcmp (value,'Off')
        seleccion=questdlg('¿Seguro que desea Apagar?','Bienvenido','Si','No','Si');
        if strcmp(seleccion,'Si')
            set_param(gcs,'SimulationCommand','stop');
        end
        if strcmp(seleccion,'No')
            app.GemeloDigitalSwitch.Value = 'On';
        end
    end

end
```

En esta línea de código se observa la variable *value* que se usará para que la interfaz detecte cuando su valor cambie, es decir, cambie de *OFF* a *ON* o, al contrario. Los condicionales *if* (encerrados en naranja) es para que cuando el usuario presione el Switch el programa responda al cambio, las líneas *seleccion=questldlg* son para que aparezca un cuadro de aviso donde se le pregunte si realmente quiere iniciar o no la interfaz. En el primer condicional (naranja) si el usuario responde *Si* (condicional en verde) a este cuadro entonces la interfaz se conecta con el diagrama de bloques de Simulink y lo inicia en modo externo *set_param(gcs,’SimulationCommand’,’start’)*. Si el usuario responde *NO* (condicional en verde) al cuadro entonces el *switch de inicio* vuelve a su estado *OFF*. El segundo condicional (naranja) funciona muy similar al primero, solo que este se ejecuta cuando hay una acción de apagado, es decir, pase de *ON a OFF*. Si el usuario da *Si*

(condicional en verde) al cuadro de dialogo entonces el diagrama de bloques en Simulink se detiene. Si presiona *NO* (condicional en verde) el botón vuelve a su estado *ON*.

Figura 32

Cuadro De Dialogo Inicio de la interfaz



De aquí en necesario mencionar que en algunas de las librerías, principalmente en los botones se tiene un condicional que evalúa si el *Switch de la interfaz* está encendido, para que se puedan ejecutar las acciones del *Widget*, es decir, solo funcionan si el software está encendido. A este condicional le llamaremos el *if de la interfaz (Figura 33).*

Figura 33

If de la interfaz

```
if strcmp (app.GemeloDigitalSwitch.Value,'On')
```

El botón del estado del motor se programó para que se visualizara un cambio significativo. Como en la acción de la librería “State Button” no se evidencian cambios notorios para el estado del botón, se agregaron colores, verde (que representa que el motor está prendido) y rojo (representa que el motor está apagado) para hacer más fácil al usuario saber si el motor está prendido o no. Por ahora este botón no tiene más funcionalidades, sin embargo, se recomendará añadir funciones como encendido y apagado del motor en esta parte. Podremos ver la línea de código del callback de este botón (Figura 34).

Figura 34

Callback Botón De Estado Del Motor

```
function MOTORButtonValueChanged(app, event)
seleccion=questdlg('¿Desea continuar?','¡Atención!','Si','No','Cancelar','Si');
 if strcmp(seleccion,'Si')
    value=app.MOTORButton.Value;
    if value == 1
    app.MOTORButton.BackgroundColor=[ 0 1 0];
    app.Image.ImageSource='Normal.gif';
    else
    app.MOTORButton.BackgroundColor=[ 1 0 0];
    app.Image.ImageSource='NormalFoto.png';
    end
 end
end
```

Una vez el usuario presione el botón salta nuevamente un cuadro de aviso donde pregunta si desea continuar. Si el usuario presiona *Si* dependiendo del valor de la variable *value* que toma el valor 1 si está encendido o 0 si está apagado. Si el valor es 1 se carga una imagen animada *gif* al *Widget image* y el fondo del botón toma el color verde ([0 1 0] en código rgb) o se carga una imagen estática del banco y el fondo del botón toma el color rojo ([1 0 0]) si el valor es 0.

El botón “Mostrar” tiene la mayor funcionalidad. Una vez el usuario presione este botón los *Edit field* de las variables comenzarán a mostrar las mediciones, también estos valores se irán añadiendo a sus vectores correspondientes, que más adelante se utilizarán para graficar y tabular en las distintas pestañas de la interfaz. En esta parte también están los condicionales que evalúan la cantidad de elementos dentro de los vectores de las variables, para que cuando se alcance un valor de 600 muestras automáticamente la interfaz grafique y tabule dichos valores. A continuación, se muestra el código implementado en esta parte (figura 35).

Figura 35

Callback Botón "Mostrar"

```
if strcmp (app.GemeloDigitalSwitch.Value,'On')
       %rto1= get_param('Simulink/Divide2','RuntimeObject');
       rto= get_param('Simulink/Divide1','RuntimeObject');
       rto2= get_param('Simulink/Divide3','RuntimeObject');

while strcmp (app.GemeloDigitalSwitch.Value,'On')
       app.C.Value=str2double(fscanf(app.pserial));
       app.TemperaturaCGauge.Value=str2double(fscanf(app.pserial));
       app.RPM.Value=rto.OutputPort(1).Data;
       app.RPMGauge.Value=rto.OutputPort(1).Data;
       app.ms2EditField.Value=rto2.OutputPort(1).Data;
       app.ms2Gauge.Value=rto2.OutputPort(1).Data;
       %x=rto1.OutputPort(1).Data;
       app.k=app.k+1;

       x=str2double(fscanf(app.pserial));


       y=rto.OutputPort(1).Data;
       z=rto2.OutputPort(1).Data;
       app.datosrpm(app.k)=y;
       app.datos(app.k)=x;
       app.datosv(app.k)=z;

       app.mr=app.mr+1;
       app.maxt(app.mr)=x;
       app.maxrpm(app.mr)=y;
       app.maxv(app.mr)=z;
       app.TmxEditField.Value=max(app.maxt);
       app.RPMmax.Value=max(app.maxrpm);
       app.Vmax.Value=max(app.maxv);
```

```
       app.tiempo=1:1:app.k;
        if app.k==600

            plot(app.UIAxes,app.tiempo,app.datos);
            plot(app.UIAxes2,app.tiempo,app.datosrpm);
            plot(app.UIAxes3,app.tiempo,app.datosv)
            app.datos=0:0;
            app.tiempo=0:0;
            app.datosrpm=0:0;
            app.k=0;

       end
       pause(1);
   V1=app.SinreferenciaButton.Value;
   V2=app.CButton.Value;
   V3=app.CButton_2.Value;

 if V2 == 1
       if str2double(fscanf(app.pserial))>=35
       app.AlarmaLamp.Color=[ 1 0 0];
           for i=0:15
           sound(sin(1:3000));
            pause(0.5);
           end
       else
           app.AlarmaLamp.Color=[ 0 1 0];
       end
elseif V3 == 1
       if str2double(fscanf(app.pserial))>=40
       app.AlarmaLamp.Color=[ 1 0 0];
            for i=0:15
             sound(sin(1:3000));                    /
```

Como se ve en el código, la primera parte (en amarillo), debajo de *if de la interfaz* se definen las variables *rto* y *rto2* las cuales son las encargadas de obtener los parámetros del diagrama de

bloques de Simulink, precisamente a la salida de los bloques *Divide1* para la medición de las RPM y *Divide3* para la medición de las vibraciones. Después, se inicia todo el proceso, en el ciclo *while* declaramos que *mientras el botón de inicio se mantenga encendido* continúe ejecutando las instrucciones debajo. El bloque de instrucciones en verde es el encargado de mostrar los valores recogidos tanto del diagrama de bloques de Simulink como el IDE de Arduino en los diferentes *widgets,* los *Edit fields* que muestran los valores de las mediciones directamente y los diferentes *Gauges* que muestran estos valores en forma de medidor analógico, también se incrementa en 1 el valor de la variable *app.k* para que sirva como referencia al momento de agregar el valor a los vectores. Más abajo en el bloque de color azul se declaran 3 variables *x (temperatura), y (RPM)* y *z(vibraciones)* donde se almacenan los valores de las mediciones momentáneamente, después estos valores son guardados en sus vectores correspondientes en la posición *app.k.*

En el cuadro naranja se tiene un condicional, el cual evalúa si la variable global *app.k* iguala las 600 muestras, recordemos que esta variable se incrementa en 1 cada ciclo del *while.* Una vez alcanzado este valor se ejecutan 3 instrucciones *plot* las cuales grafican los valores de los vectores vs el vector tiempo. Al mismo tiempo reinicia dichos vectores para empezar de nuevo. Más adelante cuando veamos el callback del botón *Graficar* observaremos que estos vectores también se reinician.

En el cuadro café se ejecuta una pausa de 1 segundo y se definen 3 variables *V1, V2 y V3* los cuales son los valores de referencia del *Radio Button group* de la pestaña de temperatura. En la parte final se tienen 3 condicionales los cuales evalúan si la variable correspondiente está activa y al sobrepasar el valor de referencia de temperatura prende la alarma sonora (el ciclo *for* con la instrucción *sound)* y cambia de color la bombilla de la pestaña de temperatura.

Figura 36

Callback Botón Graficar

```
    function graficartButtonPushed(app, event)
if strcmp (app.GemeloDigitalSwitch.Value,'On')
    plot(app.UIAxes,app.tiempo,app.datos);
    plot(app.UIAxes2,app.tiempo,app.datosrpm);
    plot(app.UIAxes3,app.tiempo,app.datosv);
    app.datosrpm=0:0;
    app.datos=0:0;
    app.datosv=0:0;
    app.tiempo=0:0;
    app.k=0;
end
    end
```

En la figura 36 se muestra el callback del botón graficas, es muy parecido al condicional del botón *Mostrar* donde se evalúa cuando la variable *app.k* llega a 600 muestras, solo que, esta vez el usuario decide cuando graficar por lo que el número de muestras es siempre menor a 600. Una vez el usuario presione este botón se grafican los valores guardados en los vectores de las variables vs el vector tiempo, siguiente a esto, se reinician los contadores y los vectores.

El *widget image* también tiene su callback, el cual se activa al presionarlo. En este callback se cargan las imágenes del modelo del banco, dependiendo del valor de la temperatura se carga una imagen diferente, con un color en la camisa del motor que se asemeja al código de colores usado en las cámaras termográficas.

Figura 37

Callback Image

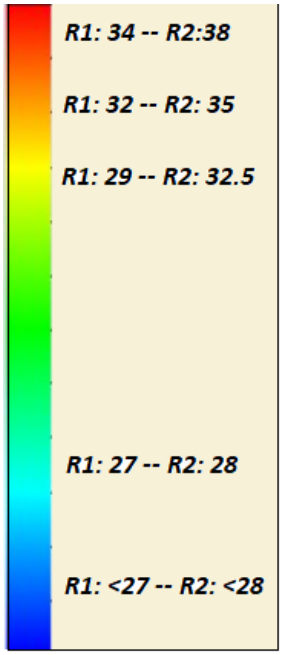
```
        function ImageClicked(app, event)
            if app.MOTORButton.Value==1
                if app.CButton.Value==1
                    if str2double(fscanf(app.pserial))>=34
                        app.Image.ImageSource='Rojo.gif';
                    elseif str2double(fscanf(app.pserial))>= 32
                        app.Image.ImageSource='Naranja.gif';
                    elseif str2double(fscanf(app.pserial))>=29
                        app.Image.ImageSource='Amarillo.gif';
                    elseif str2double(fscanf(app.pserial))>=27
                        app.Image.ImageSource='Azulado.gif';
                    else
                        app.Image.ImageSource='AzulS.gif';
                    end
                elseif app.CButton_2.Value==1
                    if str2double(fscanf(app.pserial)) >=38
                        app.Image.ImageSource='Rojo.gif';
                    elseif str2double(fscanf(app.pserial))>= 35
                        app.Image.ImageSource='Naranja.gif';
                    elseif str2double(fscanf(app.pserial))>=32.5
                        app.Image.ImageSource='Amarillo.gif';
                    elseif str2double(fscanf(app.pserial))>=28
                        app.Image.ImageSource='Azulado.gif';
                    else
                        app.Image.ImageSource='AzulS.gif';
                    end
                end
            end
        end
```

En la Figura 37 se muestra el callback del widget *Image*, este bloque de código se tienen dos condicionales principales, donde se evalúa cuál de las dos temperaturas de referencia está activa. Cada condición tiene 5 condicionales que corresponden al código de colores.

A continuación, se presenta una imagen (Figura 37) con el código de colores utilizado. Se recuerda que se utilizaron un total de 5 colores para no sobrecargar el código con instrucciones y que el flujo de ejecución sea lo más ligero posible. *R1* corresponde a la temperatura de referencia de 35° y *R2* a la de 40°.

Figura 38

Código De Colores



En las pestañas de las variables se tienen varias librerías, sin embargo, ninguna de estas tiene callback directo por lo que son en su mayoría para visualizar. Por lo que, pasaremos directamente a la pestaña *Tabla*.

En esta pestaña tenemos dos botones. El primero es el de *Añadir*, que una vez se presione añadirá automáticamente los valores máximos de las mediciones. A continuación, se muestra el callback de este botón.

Figura 39

Callback Botón Añadir

```
function AadirButtonPushed(app, event)
    a={datestr(app.FechaDatePicker.Value),app.TmxEditField.Value,app.RPMmax.Value,
        app.Vmax.Value,app.mr/60,'Operario 1','False'};

    [m,n]=size(app.tabla.Data);
    app.Datostabla = app.tabla.Data;
    d=cell(m+1,n);
    d(1:m,1:n)=app.Datostabla;
    d(m+1,1:7)=a;
    app.tabla.Data=d;
    app.mr=0;
    app.maxt=0:0;
    app.maxrpm=0:0;
    app.maxv=0:0;

end
```

En el callback del botón añadir (figura 39) lo primero que se hace es crear un arreglo *a* con 7 elementos. Se crea un arreglo con las filas y columnas existentes en ese momento, seguido se guardan los datos de la tabla en el arreglo *app.Datostabla*, se crea otro arreglo de celdas *d* con la misma cantidad de columnas pero con una fila de más. Luego, se guardan los datos existentes *app.Datostabla* en el arreglo *d.* Después de esto, se iguala la última fila del arreglo *d* a los datos capturados en *a.* De esta forma se añade una fila sin sobrescribir los datos que se hayan capturado anteriormente. Por último, se reinician los vectores y las variables para capturan las mediciones de nuevo. Por lo que, cada vez que el usuario presione este botón el campo *Tiempo* que se muestra es desde la última vez que se presionó hasta cuando se vuelve a presionar.

El botón *Limpiar* se usa para limpiar todos los datos de la tabla. Para que no haya la posibilidad de que el usuario elimine estos datos sin estar seguro se mostrará un cuadro de dialogo para que se confirme si realmente desea limpiar la tabla. Una vez el usuario confirme se limpiará por completo y los datos no se podrán recuperar, es por esto, que en el *ToolBar* se le da

la opción de exportar los datos a un archivo externo ya sea de Excel (.xls) o a un archivo de texto (.txt).

Figura 40

Cuadro De Dialogo Limpiar

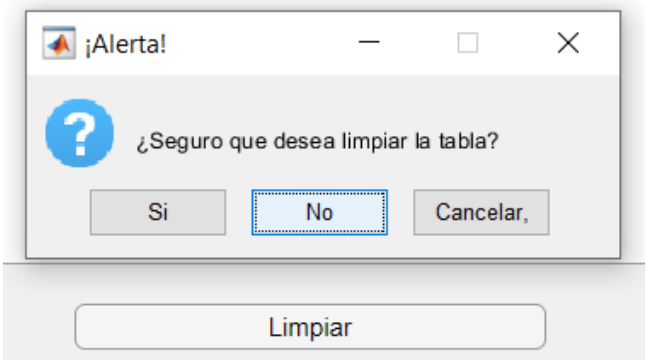


Figura 41

Callback Botón Limpiar

```
function LimpiarButtonPushed(app, event)

    seleccion=questdlg('¿Seguro que desea limpiar la tabla?','¡Alerta!','Si','No','Cancelar,','No');
    if strcmp(seleccion,'Si')
        app.tabla.Data=zeros;
        app.tabla.Data = cell(0,6);
        app.tabla.ColumnFormat = {'char','numeric','numeric','numeric','numeric',
            {'Operario 1','Operario 2','Operario 3'},'logical'};
    end
end
```

En el callback del botón *Limpiar*, para reiniciar la tabla lo primero que se hace es igualar los datos de la tabla a *zeros*, borrando por completo el contenido de esta. Luego, se vuelve a inicializar la tabla, es decir, se crea nuevamente un arreglo de celda de 7 espacios y se declaran el tipo de variable en cada campo de dicho arreglo.

Luego, tenemos los callbacks de la *ToolBar* correspondiente a los botones primarios de *Exportar* y *Ayuda*, empezaremos definiendo los botones de *Hoja de cálculo .xls y Block de notas .txt* para los que se tiene las siguientes líneas de código.

Figura 42

Callback Hoja De Cálculo .Xls

```
    function HojadeclculoxlsMenuSelected(app, event)
seleccion=questdlg('¿Desea exportar los datos a Excel?','¡Atención!','Si','No','Cancelar','Si');
        if strcmp(seleccion,'Si')
            seleccion2=questdlg('¡Se sobreexcribirán los datos!','¿Continuar?','Si','No','Cancelar','No');
            if strcmp(seleccion2,'Si')
            writecell(app.tabla.Data,'archivoexcel.xls')
            end
        end
    end
```

Figura 43

Callback Block De Notas .Txt

```
function BlocdenotastxtMenuSelected(app, event)
    seleccion=questdlg('¿Desea exportar los datos a Block de notas?','¡Atención!','Si','No','Cancelar','Si');
    if strcmp(seleccion,'Si')
        seleccion2=questdlg('¡Se sobreexcribirán los datos!','¿Continuar?','Si','No','Cancelar','No');
        if strcmp(seleccion2,'Si')
        writecell(app.tabla.Data,'archivonotas.txt')
        end
    end
end
```

En los callback de la pestaña *Exportar* (figura 42 y 43) primero, se tiene el cuadro de dialogo con su mensaje de confirmación correspondiente para que el usuario esté seguro de que quiere exportar los datos. Luego, nuevamente aparece un cuadro de dialogo advirtiendo que los datos que estén actualmente en los archivos destino se sobreescribirán, por lo que los datos que contengan estos serán eliminados y reemplazados por las nuevas mediciones. Una vez el usuario confirme

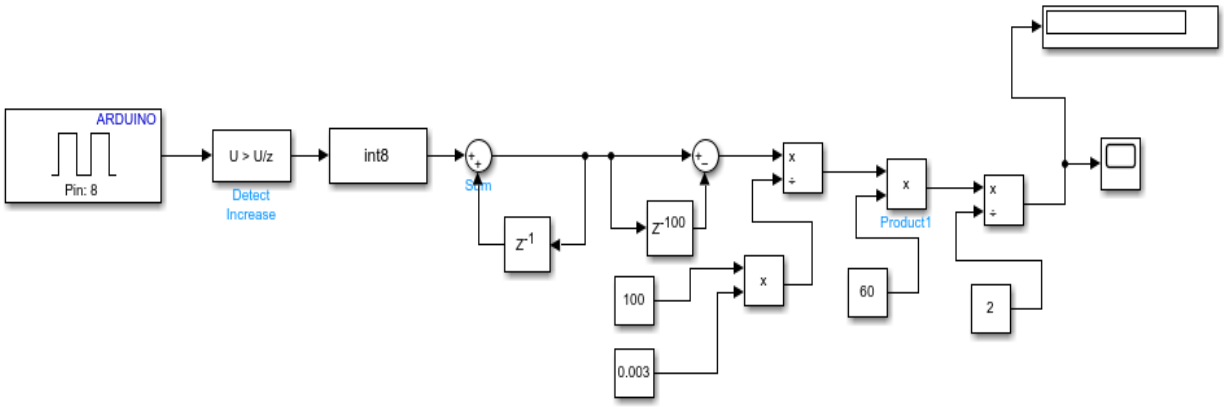
los valores se cargan a los archivos *archivoexcel.xls* y *archivonotas.txt* correspondientemente. Mas adelante en el apartado de las pruebas se evidenciarán estos archivos una vez contengan estos datos.

Se tiene el callback de la pestaña *Ayuda* el cual le permitirá al usuario disponer de distintos archivos de ayuda para la implementación satisfactoria del banco. Aquí se podrá encontrar desde archivos de configuración, hasta archivos de ayuda para la correcta implementación del banco y información de corrección de algunos errores que pueden saltar con la ejecucion de la interfaz.

Por último, se mostrarán las codificaciones y el diagrama de bloque utilizados para la toma de las muestras, que luego serán conectados directamente con la interfaz gráfica.

Figura 44

Diagrama De Bloques RPM



En la figura 44 primeramente, se tiene un bloque *Digital Input* propio del paquete de Simulink para Arduino, seguido va un *Detect Increase* que permitirá detectar los flancos provenientes del sensor, luego se tiene un *Date Conversion* que cambiará los flancos positivos en unos (1) para que se puedan operar mas adelante, luego tenemos dos *Delays* o retardos que permitirán capturar solamente los ultimos 100 datos. Luego se hacen las operaciones respectivas que permiten

relacionar el tiempo en que se toman esos 100 datos para hallar la velocidad angular y luego las RPM, como sigue:

$$\frac{\Delta pulsos}{N_{muestras} \times T_{muestreo}} \times \frac{60seg}{1min} \times \frac{1rev}{Resolución\ encoder}$$

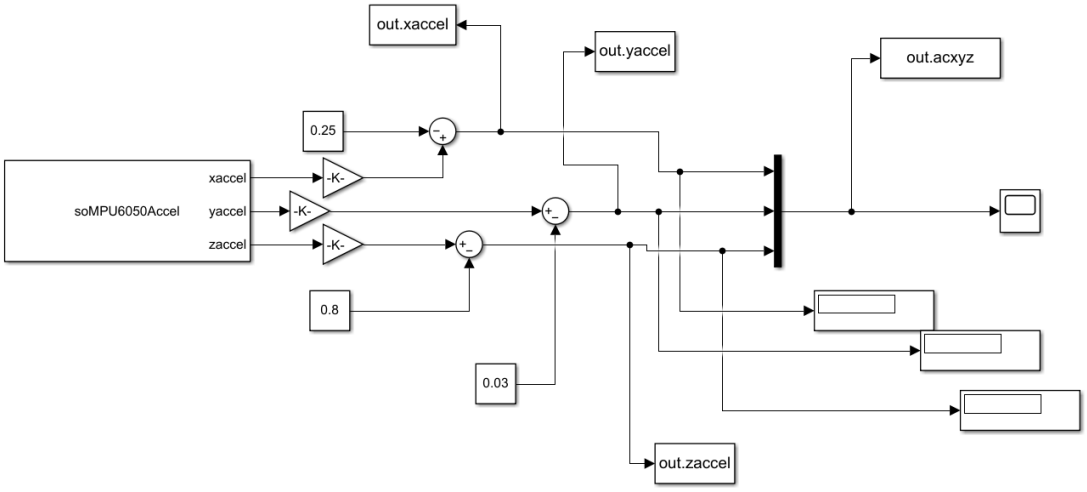
$$\frac{pulsos}{100(M) \times 0.003(\frac{S}{M})} \times \frac{60(s)}{1(\text{min})} \times \frac{rev}{2(pulsos)}$$

Donde, el T de muestreo está en muestras por segundo y lo definimos en las configuraciones de nuestro diagrama de bloques y la opciones de ejecución en modo externo de Simulink. Al final de las operaciones, el resultado será en RPM.

Para la parte de las vibraciones se tiene el siguiente diagrama de bloques.

Figura 45

Diagrama De Bloques Vibraciones



El acelerometro entrega las 3 aceleraciones correspondientes a cada uno de los ejes. En el diagrama de bloques (figura 45). Primero se realiza la conversión de RAW entregados por el sensor a G con la conversión 1/16384. Luego, se suman las constantes para corregir los valores iniciales no despreciables generados por la inclinación del sensor y se unen las tres señales en un vector con ayuda del bloque MUX, los bloque To Workspace nos ayudará a exportar los datos recolectados de las mediciones al espacio de trabajo de MATLAB donde posteriormente las señales serán filtradas y se aplicará las transformadas de Fourier. Para esta parte se usa un script donde permite iniciar la simulación de 30 segundos en simulink y posterior a eso realizar el correspondiente manejo de la señal, el script se evidencia en la figura 46

Figura 46

Script manejo de la señal vibraciones

Editor - D:\Descargas\DigitalTwin\pruebas.m

Script2.m | FFT.m | pruebas.m

```
%Inicio de simulación en simulink
load_system('sensor_yz.slx')
set_param(gcs,'SimulationMode','external')
set_param(gcs,'Simulationcommand','start')
pause(30)
set_param(gcs,'Simulationcommand','stop')
%Gráfica de recolección de señales del sensor sin implementación de fft
figure()
plot(acxyz.time,acxyz.signals.values)
title('Lecturas de sensor MPU-6050 sin fft')
xlabel('tiempo (s)')
ylabel('Aceleración (G)')
grid on
x=acxyz.signals.values;
%obtener la frecuencia de muestreo para f 1 ft
[N,m]=size(acxyz.signals.values);
Fs=1000; %obtener la frecuencia de muestreo para fft
fprintf('%12.0f data points\n',N);
xdft=fft(x);
xdft = 1/length(x).*xdft; %Normalize
xdft(2:end-1) = 2*xdft(2:end-1);

%Gráfica de fft para eje x
ejx=acxyz.signals.values(:,1);
%Establecer el tiempo de la muestra para fft
Fsx=1000; %obtener la frecuencia de muestreo para fft
[N,m]=size(acxyz.signals.values(:,1));
fprintf('%12.0f data points\n',N);
freqx = 0:Fsx/length(ejx):Fsx/2; %frequency array for FFT
```

Editor - D:\Descargas\DigitalTwin\pruebas.m

Script2.m | FFT.m | pruebas.m

```
freqx = 0:Fsx/length(ejx):Fsx/2; %frequency array for FFT
ejxdft=fft(ejx);
ejxdft = 1/length(ejx).*ejxdft; %Normalize
ejxdft(2:end-1) = 2*ejxdft(2:end-1);

figure()
plot(freqx,abs(ejxdft(1:floor(N/2)+1)))
xlabel('Frequency (Hz)');
ylabel('Accel (g)');
title('eje x');
grid on
%Gráfica de fft para eje Y
ejy=acxyz.signals.values(:,2);
Fsy=1000; %obtener la frecuencia de muestreo para fft
[N,m]=size(acxyz.signals.values(:,1));
fprintf('%12.0f data points\n',N);
freqy = 0:Fsy/length(ejy):Fsy/2; %frequency array for FFT
ejydft=fft(ejy);
ejydft = 1/length(ejy).*ejydft; %Normalize
ejydft(2:end-1) = 2*ejydft(2:end-1);
figure();
plot(freqy,abs(ejydft(1:floor(N/2)+1)));
xlabel('Frequency (Hz)');
ylabel('Accel (g)');
title('eje y');
grid on
%Gráfica de fft para eje z
ejz=acxyz.signals.values(:,3);
tz=acxyz.time(:,1); %Establecer el tiempo de la muestra para fft
```

```
xlabel('Frequency (Hz)');
ylabel('Accel (g)');
title('eje y');
grid on
%Gráfica de fft para eje z
ejz=acxyz.signals.values(:,3);
tz=acxyz.time(:,1); %Establecer el tiempo de la muestra para fft
Fsz=1000; %obtener la frecuencia de muestreo para fft
[N,m]=size(acxyz.signals.values(:,1));
fprintf('%12.0f data points\n',N)
freqz = 0:Fsz/length(ejz):Fsz/2; %frequency array for FFT
ejzdft=fft(ejz);
ejzdft = 1/length(ejz).*ejzdft; %Normalize
ejzdft(2:end-1) = 2*ejzdft(2:end-1);
figure()
plot(freqz,abs(ejzdft(1:floor(N/2)+1)))
xlabel('Frequency (Hz)');
ylabel('Accel (g)');
title('eje z')
grid on
```

Para la parte de la temperatura se utilizó el IDE de arduino y sus lineas de codigo correspondientes se encuentran a continuación en la figura 47.

Figura 47

Script Código Temperatura IDE Arduino

Sensor_Temperatura §

```
#include <OneWire.h>
#include <DallasTemperature.h>
// Pin donde se conecta la senal
const byte pinDatosDQ=2;

//Instancias a las clases
OneWire oneWireObjeto(pinDatosDQ);
DallasTemperature sensorDS18B20 (&oneWireObjeto);

void setup() {
  Serial.begin(115200);
  sensorDS18B20.begin();
}

void loop() {
  sensorDS18B20.requestTemperatures();
  Serial.println(sensorDS18B20.getTempCByIndex(0));
}
```

El lenguaje de programación del código es basado en lenguaje C. En él se importan las librerias de *OneWire y DallasTemperature* que serán quienes hagan el proceso interno para obtener las mediciones del sensor. Se establece el pin de conexión (pin 2), se instancian las clases (termino de programación orientado a objetos), se definen los baudios (115200 en este caso) y por último se establece un *loop* o ciclo que mostrará las mediciones infinitamente (o hasta que se detenga el programa). Despues de esto, se exportán las mediciones directamente a la interfaz. Es importante entender que la conexión serial de Arduino con MATLAB solo permite transportar datos, por lo que, el proceso de recolectar las medidas las hace Arduino y no MATLAB.

4.2.4.4 Manejo de señales

En este apartado se enunciará el manejo que se le dio a las señales y sus configuraciones correspondientes en los entornos donde se programaron.

Señal de temperatura: Como se ha mencionado anteriormente las funciones del sensor DS18B20 se programaron en el entorno de desarrollo de Arduino (IDE) con ayuda de la librería *DallasTemperature*. En la tabla 5 se evidencian las caracteriticas principales del manejo y configuración de la señal de temperatura.

Tabla 5.

Caracteristicas señal de temperatura

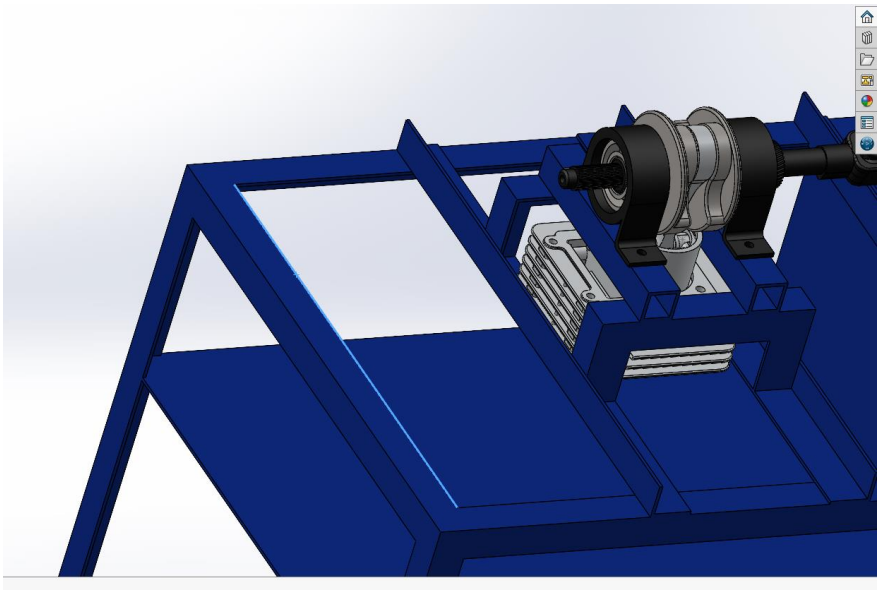
Sensor	Frecuencia señal	Tiempo de señal	Frecuencia de muestreo	Baudios
DS18B20	1,33 Hz	0,75 s	Implicita en la librería	115200

La frecuencia de muestreo se adapta automaticamente a la frecuencia de la señal gracias a unos metodos incluidos en la librería *DallasTemperature*.

Señal de vibraciones: Para este apartado, primero se halló la frecuencia Natural del sistema por medio de la Herramienta SOLIDWORKS modelando el sistema completo del banco de laboratorio y hallando las 5 primeras frecuencias naturales para luego aplicar el teorema de Nyquist para determinar la frecuencia de muestreo. En la figura 47 se evidencia el modelo CAD del sistema

Figura 48

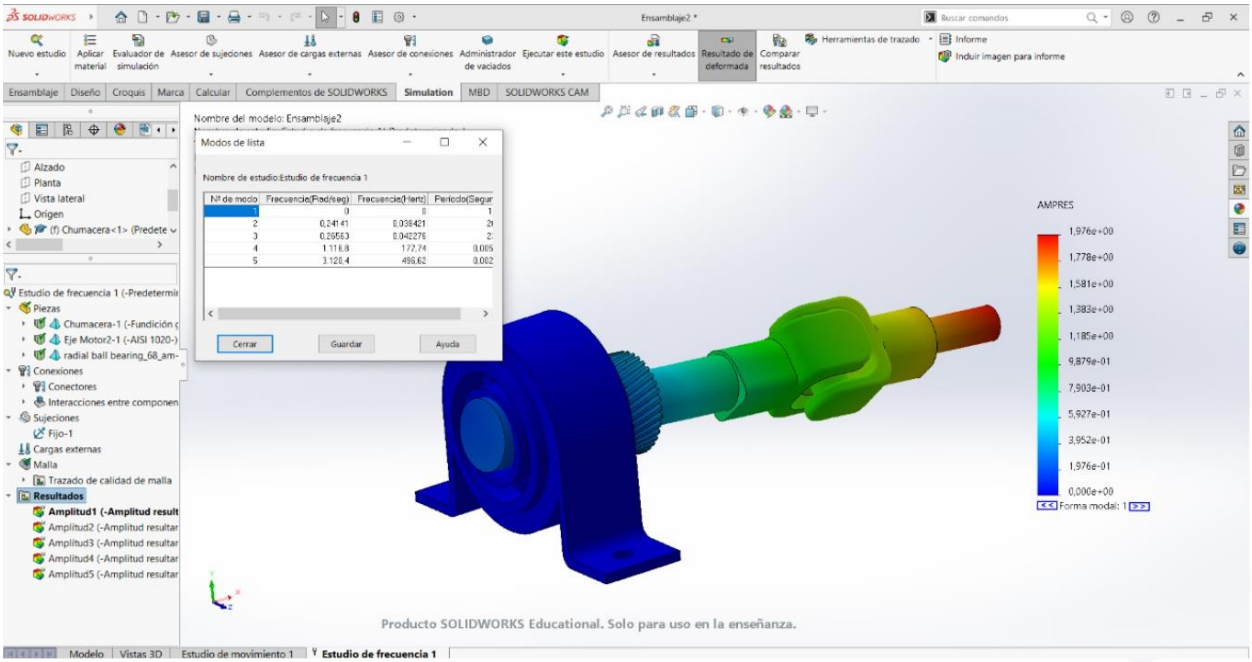
Modelo CAD del banco de laboratorio



Se realiza el estudio de frecuencia de los componentes donde se realizan las mediciones de vibraciones, en este caso las chumaceras y se realiza el estudio de frecuencia arrojando los resultados mostrados en la figura 48.

Figura 49

Estudio de frecuencia SOLIDWORKS



De allí podemos observar que la frecuencia natural Maxima es de 496.62 Hz. Aplicando el teorema de Nyquist para hallar la frecuencia de muestreo el cual denota que "La minima frecuencia teorica de muestreo fs necesaria para conseguir recuperar la señal original es igual al doble de la frecuencia máxima t que contenga la señal a muestrear" esta espresión se muestra en la Ec. 1.

Ec. 1) $$\boldsymbol{f_s = 2t}$$

fs -> es la frecuancia de muestreo

t -> frecuancia del sistema

Cumpliendo este criterio garantizamos que si la señal contiene frecuencias elevadas se precisará de mas muestras para poder ser recuperada. Aplicando este criterio y realizando los calculos correspondientes se establecio una frecuencia de muestreo igual a 1000 Hz:

Tabla 6

Caracteristicas señal de vibraciones

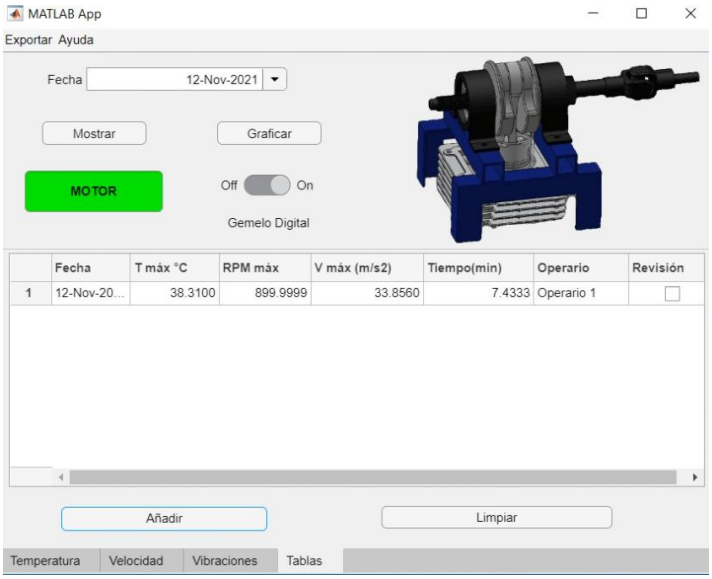
Sensor	Frecuencia señal	Tiempo de señal	Frecuencia de muestreo	Baudios	Señal
MPU6050	496,62 Hz	0,002	1000	115200	Filtrada

4.2.5 Integración y prueba del sistema

En esta parte se realizaron pruebas de funcionamiento de la interfaz gráfica. Se contabilizó manualmente el tiempo para comparar con el estimado. Es importante mencionar que el botón *Añadir* de la tabla de valores se presionó cada 10 minutos, mientras que las graficaciones se hicieron de forma automática por parte del software y se tomó el tiempo que se demoraron en activarse. Pasados los primeros 10 min reales la tabla mostró los siguientes valores, figura 46.

Figura 50

Valores Tabla 10 Minutos Reales



Vemos los valores de las mediciones de los primeros 10 minutos, se evidencia que el tiempo estimado en la simulación es diferente a los 10 minutos medidos manualmente, al final de las pruebas se hará una tabla de comparación entre estos valores de tiempo. Luego de esto, se tomaran cada 10 minutos las mediciones en la tabla, que se irán mostrando en las imágenes siguientes. Para el caso de las gráficas, estas se grafican automaticamente, por lo que, se tomarán 10 minutos estimados de simulación y se compararán con los obtenidos manualmente.

Figura 51

Gráficas 10 Minutos

En la figura 47 se muestran los pantallazos de las 2 pestañas de las variables pasados 10 minutos de simulación

Figura 52

Tabla De Mediciones 30 Minutos

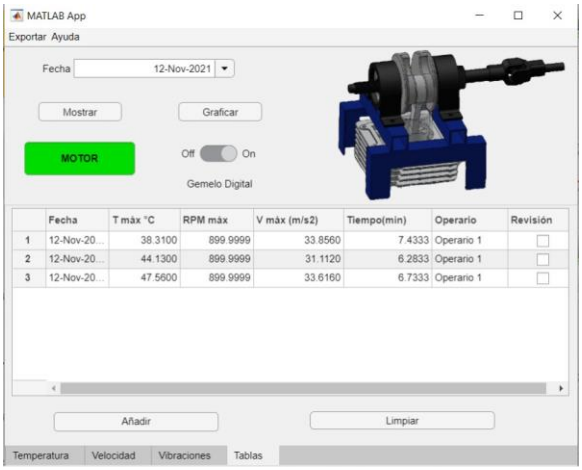
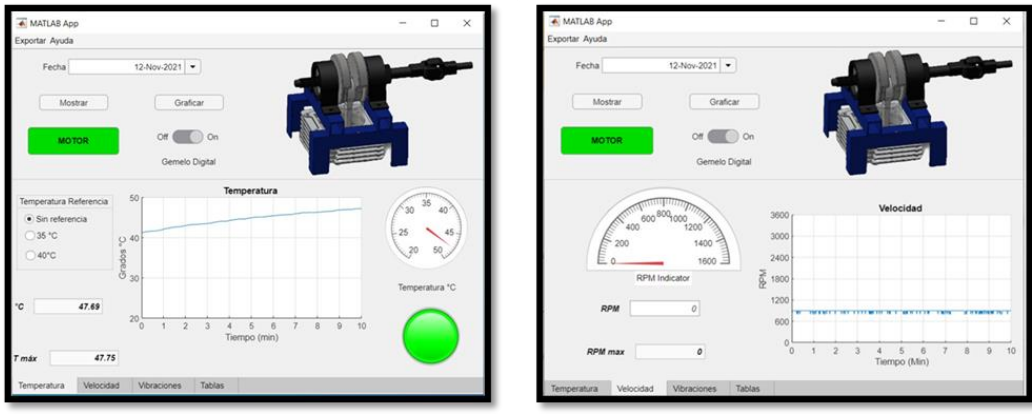


Figura 53

Gráficas 20 Minutos



Las figuras 49 y 50 mostradas anteriormente corresponden a las mediciones de las tres variables mientras el motor del banco estaba encendido. Se tomaron estos datos durante un total de 30 minutos. Despues de esto, se procede a apagar el banco y se siguen midiendo los valores hasta que se completen las graficas.

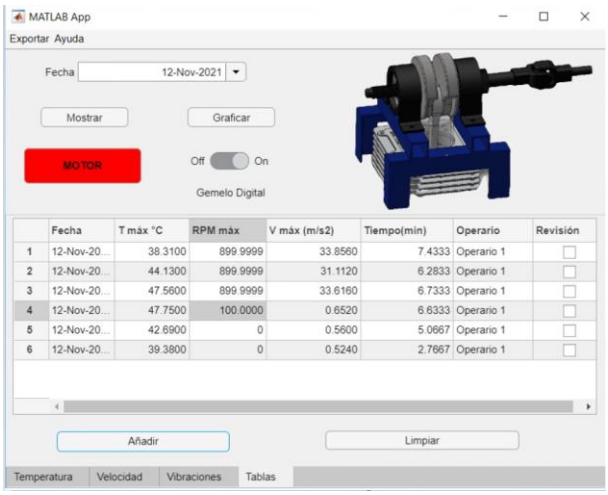
En esta parte se evidenció un error en la medición de las RPM, ya que el valor teorico es de 1800 y las mediciones arrojan 900 aproximadamente. Esto se debe a que en el diagrama de bloques se cometió un error en el valor correspondiente a la resolucion del encoder, ya que este no es de 2 sino de 1, por este motivo las mediciones dan la mitad.

Figura *54*

Gráficas 30 Minutos

Figura 55

Tabla De Mediciones 60 Minutos



	Fecha	T máx °C	RPM máx	V máx (m/s2)	Tiempo(min)	Operario	Revisión
1	12-Nov-20...	38.3100	899.9999	33.8560	7.4333	Operario 1	
2	12-Nov-20...	44.1300	899.9999	31.1120	6.2833	Operario 1	
3	12-Nov-20...	47.5600	899.9999	33.6160	6.7333	Operario 1	
4	12-Nov-20...	47.7500	100.0000	0.6520	6.6333	Operario 1	
5	12-Nov-20...	42.6900	0	0.5600	5.0667	Operario 1	
6	12-Nov-20...	39.3800	0	0.5240	2.7667	Operario 1	

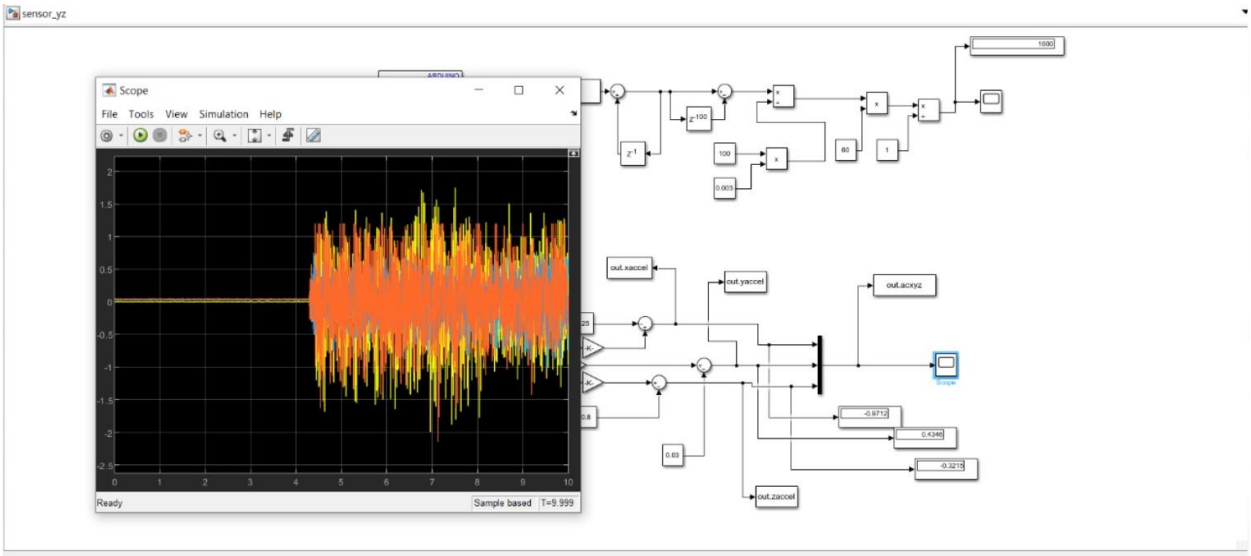
Figura 56

Gráficas 40 Minutos

Despues se realizaron las pruebas para las vibraciones en el cual se ejecutó el script, primero se evidencian las graficas sin aplicar el filtro y el FFT Figura 57.

Figura 57

Gráfica vibraciones sin FFT



Después de esto se realizó la ejecución del script arrojando los siguientes resultados para los 3 ejes independiente

Figura 58

Gráfica vibraciones con FFT eje X

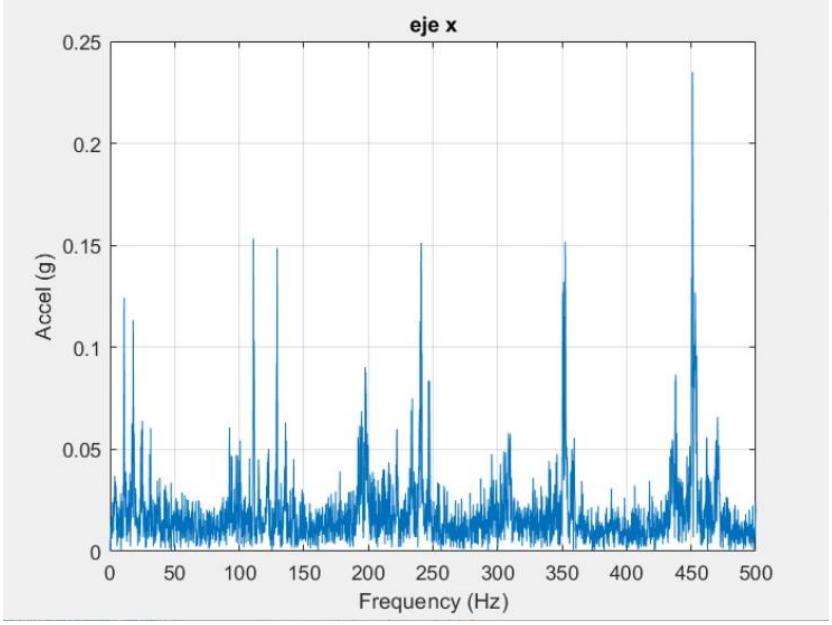


Figura 59

Gráfica vibraciones con FFT eje Y

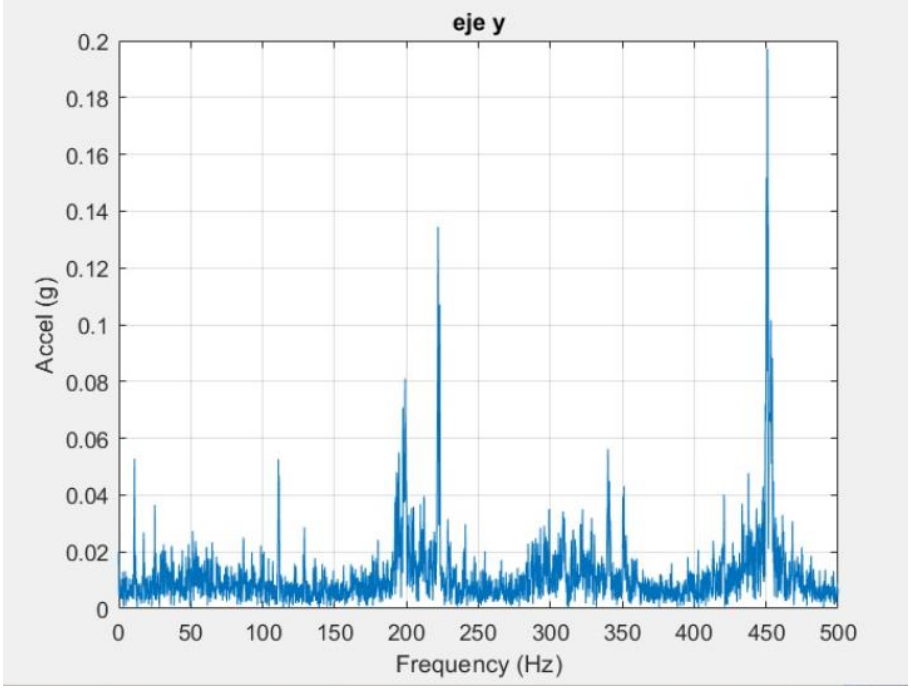


Figura 60

Gráfica vibraciones con FFT eje Z

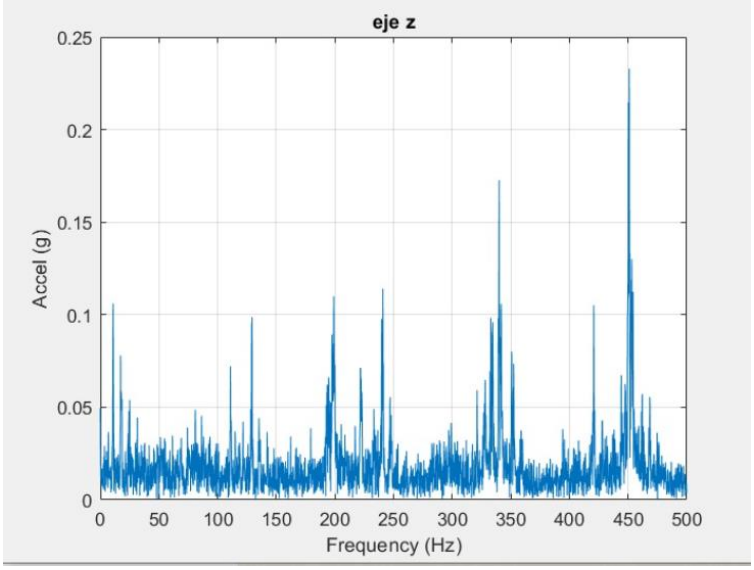
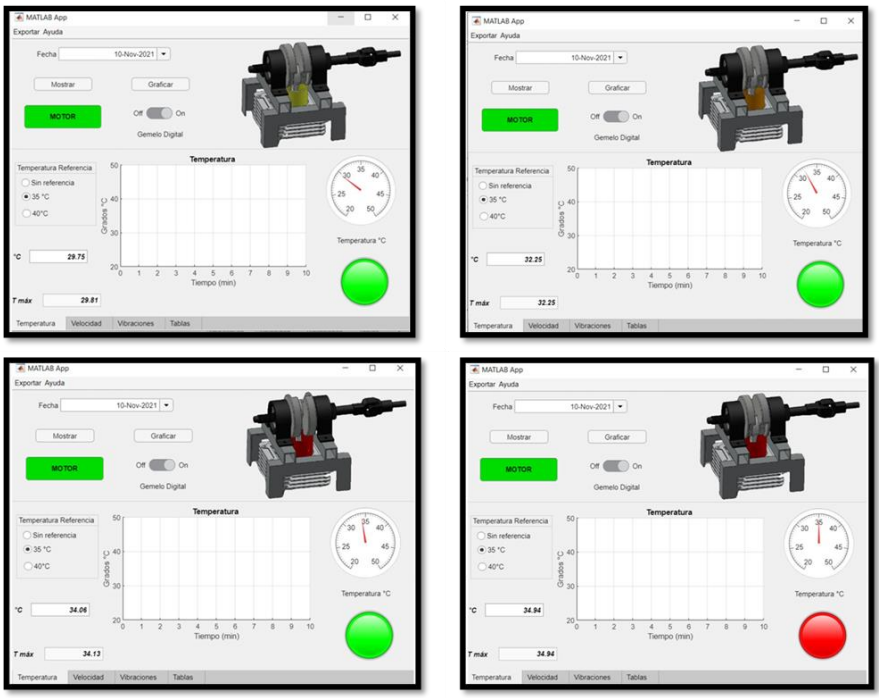


Figura 61

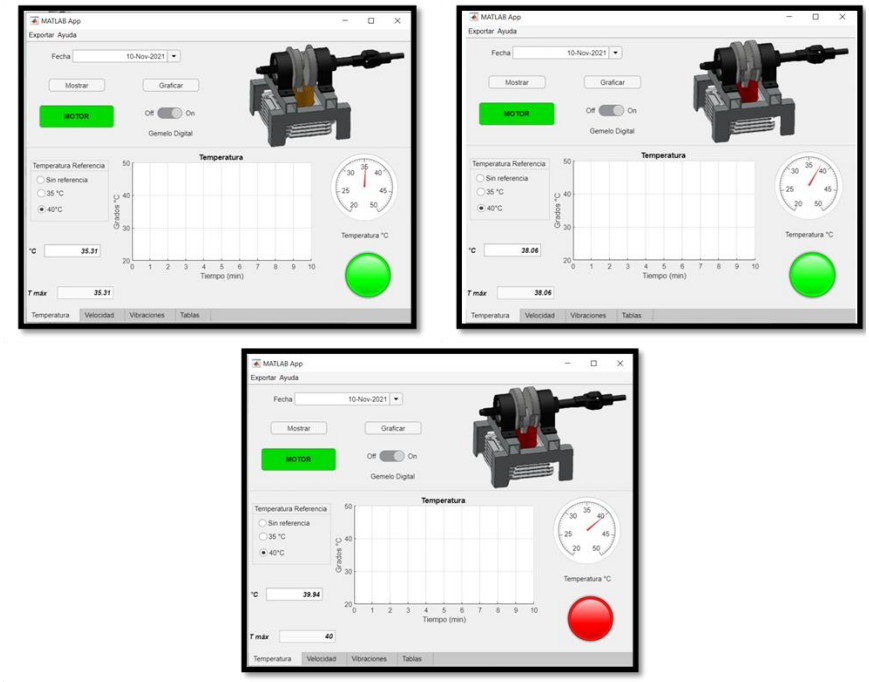
Pruebas de Temperatura Referencia 35°



En las figura 54 se observa el comportamiento de la interfaz usando la temperatura de referencia de 35° C., vemos que los colores de la camisa cambian de acuerdo con el código de colores que se estableció anteriormente en la figura 38. También se pudo observar que la alarma se encendió justo cuando se alcanzó la temperatura de referencia, aunque la interfaz muestra la medición justo antes de alcanzar dicho valor (34.94° en este caso). De igual manera, una vez alcanzado este valor de referencia se hace el cambio en el *Radio Button* para establecer la segunda temperatura de referencia (40°). Se obtuvieron las siguientes imágenes.

Figura 62

Pruebas Temperatura Referencia 40 Código de Colores



En la figura 55 podemos observar que una vez se cambie la temperatura de referencia, la *AlarmLamp* se torna de color verde, indicando que aún no hemos llegado a la temperatura de referencia, una vez la temperatura va aumentando, de igual manera va cambiando el color de la camisa del motor de acuerdo con lo establecido anteriormente en la figura 38. Una vez se

cumpla la condición de T > 40° se observa el cambio en el estado de la *AlarmLamp* junto con la alarma sonora programada. Se puede concluir que las pruebas salieron de acuerdo con lo esperado.

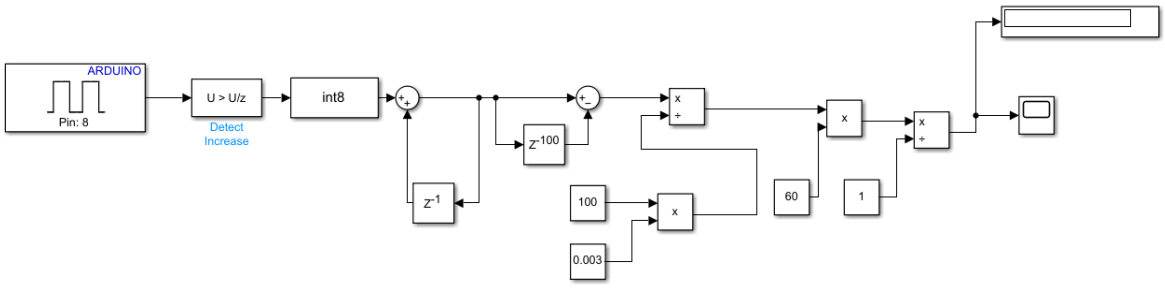
4.2.6 Operación y mantenimiento

En esta parte se van a analizar los resultados obtenidos de las pruebas de funcionamiento del apartado anterior. Se harán los cambios al software que se crean necesarios, para hacerlo más limpio y óptimo.

4.2.6.1. Corrección Diagrama RPM. Empezaremos corrigiendo la parte del diagrama de bloques de la medición de RPM, recordemos que la resolución del encoder es de 1 y por motivos del posicionamiento del sensor, que al principio se quería instalar apuntando al eje, el cual tiene una resolución de 2 se cometió ese pequeño error en la codificación de esta variable. Se muestra en la figura 56.

Figura 63

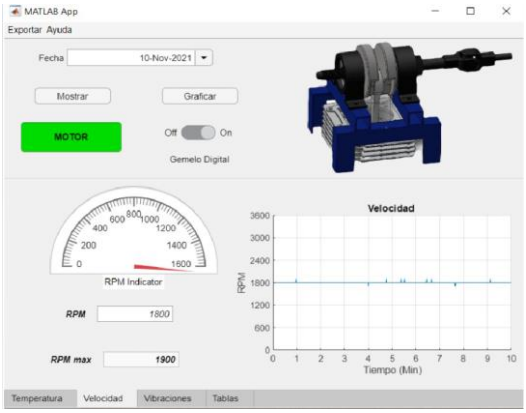
Diagrama De Bloques RPM Corregido.



Después de corregido este valor se volvió a hacer la prueba para comprobar que las mediciones den de forma indicada.

Figura 64

Prueba De Medición De RPM Corregido



4.2.6.2. Corrección indicador RPM. El indicador analógico de la pestaña de RPM iba de 0 a 1600, sin embargo, el banco presenta unas 1800 RPM constantes por lo que, el indicador no es suficiente para mostrar este valor.

Figura 65

Tabla De La Interfaz Corregida

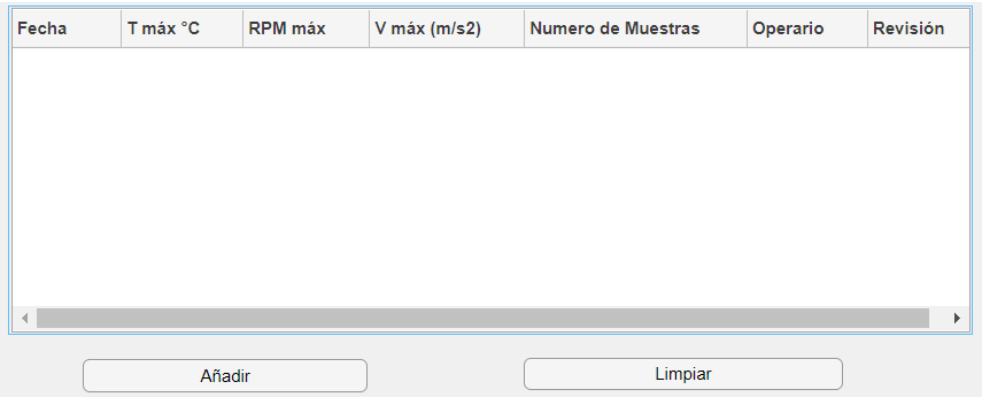


Figura 66

Callback Botón Añadir Corregido

```
function AadirButtonPushed(app, event)
    a={datestr(app.FechaDatePicker.Value),app.TmxEditField.Value,app.RPMmax.Value,
        app.Vmax.Value,app.mr,'Operario 1','False'};

    [m,n]=size(app.tabla.Data);
    app.Datostabla = app.tabla.Data;
    d=cell(m+1,n);
    d(1:m,1:n)=app.Datostabla;
    d(m+1,1:7)=a;
    app.tabla.Data=d;
    app.mr=0;
    app.maxt=0:0;
    app.maxrpm=0:0;
    app.maxv=0:0;

end
```

De esta manera las mediciones serán mucho más generales y la interfaz gráfica podrá ser usada en cualquier computador sin la confusión de los tiempos. De esta forma se termina el proceso de corrección y diseño de la interfaz. Sin embargo, aplicando un proceso y una mentalidad de mejora continua se recomienda evaluar y optimizar aún más el diseño y la implementación de la interfaz.

5 Costos

En la siguiente tabla se presentan los costos del hardware implementado en la realización del proyecto, que van desde la compra de jumpers para las conexiones hasta el costo del equipo de cómputo.

Tabla 7.

Costos del hardware.

Ítem	Descripción	Cantidad	Valor C/U	Valor
1	ARDUINO UNO R3 CH340G	1	49.000	49000
2	ARDUINO UNO R3	1	127.800	127.800
3	SENSOR IR OBSTÁCULO	1	8.000	8.000
4	PROTOBOARD WB102	1	12.000	12.000
5	JUEGO CABLE M/M	4	7.000	28.000
6	JUEGOS DE CABLES M/H	4	7.000	28.000
7	RESISTENCIA 4.7K	1	200	200
8	CABLE USB	1	3.500	3.500
9	ACELERÓMETRO MPU-6050	1	7.900	7.900
10	CABLE ADAPTADOR ARDUINO	1	5.000	5.000
11	SENSOR TEMPERATURA DS18B20	1	35.000	35.000
12	LAMINA METAL	1	20.000	20.000
13	MANUFACTURA CAJA METÁLICA	1	10.000	10.000
14	COMPUTADOR PORTÁTIL	1	2.300.000	2.300.000
			TOTAL	2.634.400

6 Conclusiones

Fue necesario implementar la conexión de los sensores con dos (2) placas Arduino debido a la incompatibilidad del sensor DS18B20 con la herramienta Simulink.

La interfaz es capaz de recopilar los datos de las mediciones desde el IDE (entorno de desarrollo) de Arduino y la herramienta Simulink en tiempo real simultáneamente con los métodos fscanf y getParam().OutputPort respectivamente.

Las mediciones obtenidas por medio del sensor DS18B20 comparadas con las obtenidas por la cámara termográfica (configurada con el factor de emisividad del aluminio 0.02) fueron levemente superiores (0.49°C), lo cual está dentro del rango especificado por el Datasheet de la pieza (0.5 °C entre 25° - 85°) a 9 bits.

Las mediciones obtenidas de RPM con ayuda del sensor de obstáculos *SENOBST-2* comparadas con la especificación del banco se mantuvieron en 1800 constante, con algunas pequeñas fluctuaciones, lo cuál es un acierto debido a que no es un sensor nativo para este tipo de mediciones.

Los sensores se conectaron a 115200 baudios con un tiempo de muestreo de 0.003 y 0.001 s por muestra para el sensor de RPM y vibraciones respectivamente cumpliendo con el criterio de Nyquist para recuperar la señal en caso de obtener frecuencias de señal elevadas.

Se diseñó la interfaz gráfica en la herramienta AppDesigner con un total de 21 Widgets y 813 líneas de código para un total de 60Mb. De igual forma se diseñaron manuales de usuario de acceso mediante la interfaz que permiten corregir errores que se puedan presentar durante la ejecución de las actividades, una guía de laboratorio y una guía de configuración.

7 Observaciones

En este proyecto se estableció un montaje y conexión muy rudimentarios con un hardware asequible, sin embargo, para futuros proyectos es importante implementar hardware de mejor calidad como los utilizados en distintos laboratorios de la escuela generalmente de la marca National Instruments, ya que esto marcara una mejora sustancial en el tiempo de muestreo y la capacidad de procesamiento del software.

Durante el desarrollo del proyecto se tenía la idea de implementar el código de colores automático, es decir, que el usuario no tenga que presionar en la imagen del gif para que este cambie de color según la medición de temperatura, sin embargo, se saturaba demasiado el ciclo while del botón mostrar, ya que en cada ciclo tenía que evaluar 5 sentencias adicionales correspondientes a cada color y los tiempos de ejecución aumentarían considerablemente. Por esta razón se dejó de modo manual.

Se observó que mientras más se tuviera prendido el software el tiempo en el que se tardaba en ejecutar en ciclo while aumentaba, incluso llegando a tener un comportamiento exponencial, es por esto que se recomienda usar un equipo de cómputo con una capacidad de procesamiento alta.

Podrían aumentarse las variables medidas, tomando como referencia la codificación de este proyecto, es importante optimizar el código para que la velocidad de procesamiento mejore y los tiempos de medida disminuyan.

8 Recomendaciones

En caso de querer seguir con el proyecto se recomienda implementar unos sensores de mayor calidad para que la velocidad de procesamiento de las medidas mejore y por lo tanto disminuya el tiempo de medición, también es importante utilizar un equipo de cómputo con alta capacidad de procesamiento para el mismo propósito.

Se recomienda desarrollar una interfaz que no solo reciba la información, sino que también actúe según unos parámetros, es decir, que tome acciones dependiendo del valor de las mediciones que recolecte. Por ejemplo, que al llegar a cierta temperatura el motor se detenga o disminuya las RPM con un variador de frecuencia conectado al software.

Este tipo de proyecto es posible aplicarlo a cualquier tipo de máquina, desde motores eléctricos como en este caso hasta sistemas hidráulicos de potencia, donde se pueden controlar variables como el caudal y la presión además de las utilizadas en este proyecto y que regule válvulas según esas mediciones.

Es recomendable hacer un estudio del código, para encontrar posibilidades de mejora, es decir, para hacer el código más eficiente y entendible, quizá utilizar algún paradigma de programación como la Programación Orientada a Objetos para hacer un código modular y que sea más mantenible, modificable y escalable.

Referencias Bibliografícas

BeJob. (2017) ¿Qué es la programación con Arduino y para qué sirve? https://www.bejob.com/que-es-la-programacion-con-arduino-y-para-que-sirve/

Alarcón, Anabel, Ball Carlos (2005). Interfaz grafica de monitoreo y configuracion de redes WAN. p 2-3.

Barragán, Diego (2008). Manual interfaz Grafica de usuario en MATLAB. p 8-41.

Economía aplicada. (2019) ¿Cuántas empresas hay en Colombia? Disponible en http://economiaaplicada.co/index.php/10-noticias/1493-2019-cuantas-empresas-hay-en-colombia [Accedido 20 Junio]

El Economista.es. (2019). "Siemens propone su 'Gemelo digital' para reducir los costos de fabricación". Disponible en: https://www.eleconomista.es/empresas-finanzas/noticias/10148662/10/19/Siemens-propone-su-Gemelo-Digital-para-reducir-los-costes-de-fabricacion.html.

Hernández, K. (2021). Los Gemelos digitales y la promesa de transformar al futuro. Servnet. https://www.servnet.mx/blog/gemelos-digitales-la-promesa-del-futuro

Hetpro. Modulo acelerómetro y giroscopio MPU 6050. Disponible en: https://hetpro-store.com/TUTORIALES/modulo-acelerometro-y-giroscopio-mpu6050-i2c-twi/

Iberdrola. (2019). “Gemelos digitales, claves en la cuarta revolución industrial”. Disponible en: https://www.iberdrola.com/innovacion/gemelos-digitales.

Industrie 4.0 Working Group. (2013). “Recommendations for implementing the strategic initiative INDUSTRIE 4.0”. *Securing the future of German manufacturing industry*. Páginas 13-14.

Mathworks. (2021). Connect to MPU-6050 sensor on Arduino hardware I2C bus. Disponible en: Conexión al sensor MPU-6050 en el bus I2C de hardware Arduino - MATLAB (mathworks.com)

Mathworks. “¿Qué son los gemelos digitales?”. Disponible en https://la.mathworks.com/discovery/digital-twin.html

Muñoz, A. (2019). “Aplicación del concepto de gemelo digital a un SCADA Industrial”. Trabajo grado en ingeniería informática. Universitat Politècnica de València. Escola Tècnica Superior d’Enginyeria Informàtica.

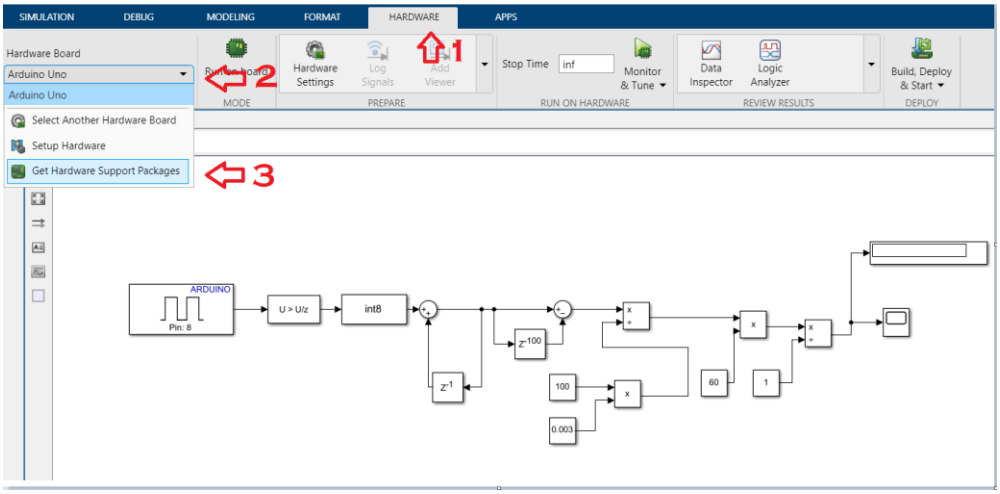
Manual de programación en Arduino Disponible: http://dfists.ua.es/~jpomares/arduino/page_01.htm

Parrot, A; Warshaw, L. (2017) Industry 4.0 and the digital twin: Deloitte University Press. Páginas 7-9.

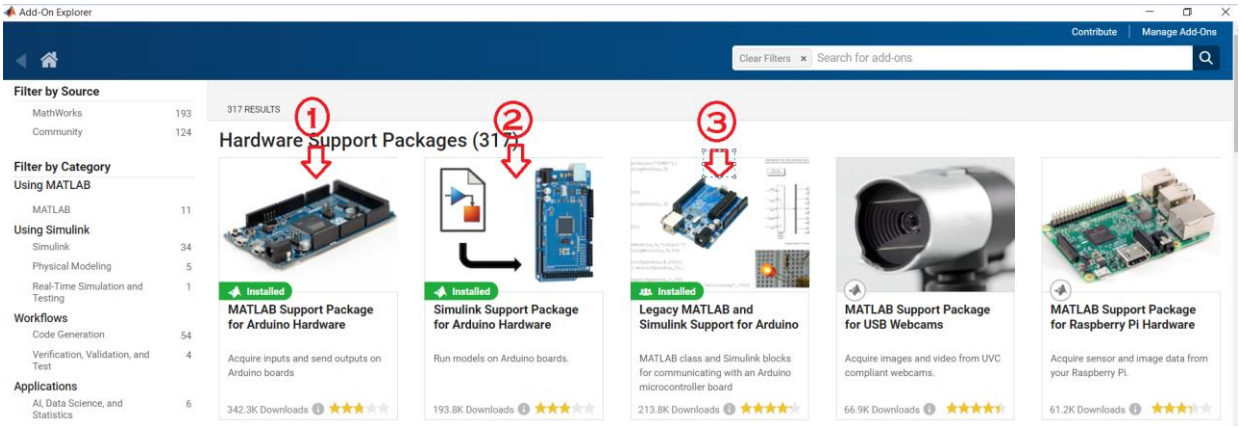
Stark, R; Damerau, T. (2019) “Digital Twin”. CIRP Encyclopedia of Production Engineering. Páginas 1-5.

Apéndice A Configuración Simulink

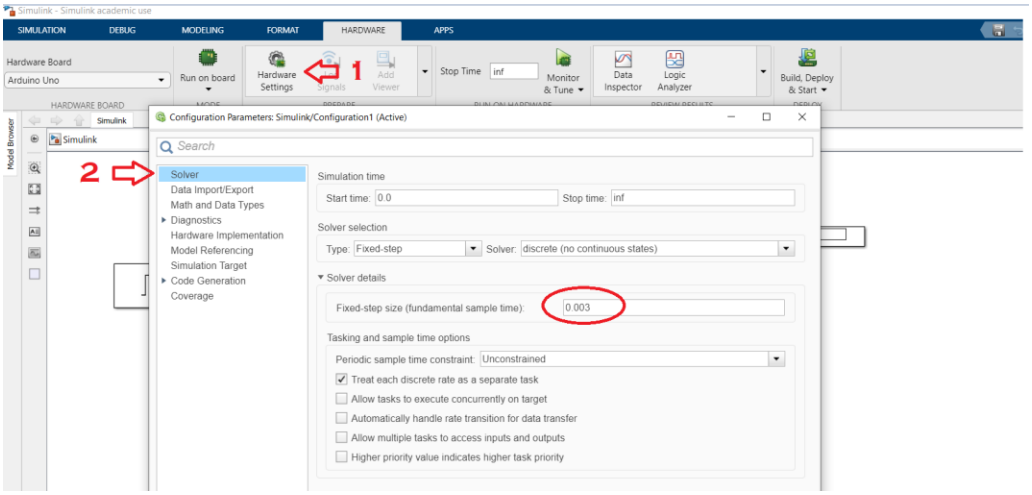
En este documento se muestra la configuración que debe tener la herramienta Simulink, tanto la conexión con Arduino como la toma de muestras. Lo primero que debemos hacer es descargar las librerías de Simulink Arduino Package siguiendo los pasos mostrados a continuación.



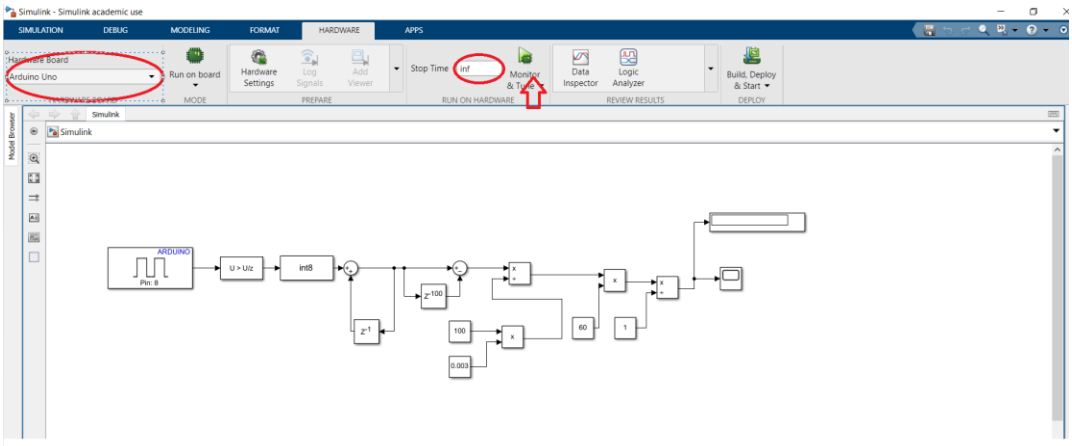
Luego descargamos los 3 paquetes que se muestran a continuación, lo único de debes hacer es darle click en la librería y luego *Install.* Aquí ya aparecen instalados, pero debes buscarlos e instalarlos uno por uno.



Después de esto vamos a configurar el Hardware, para esto damos click en Hardware Settings y configuramos la ventana de la siguiente forma. Es muy importante colocar los valores correctos, ya que los cálculos de las mediciones dependen de esto.



Después de esto podremos hacer la prueba manual de que la conexión funciona correctamente desde la ventana de Simulink de la siguiente forma. Recuerda poner Stop Time = inf para que el ciclo sea infinito y también que el *Hardware Board* corresponda a la placa de Arduino que estemos usando.

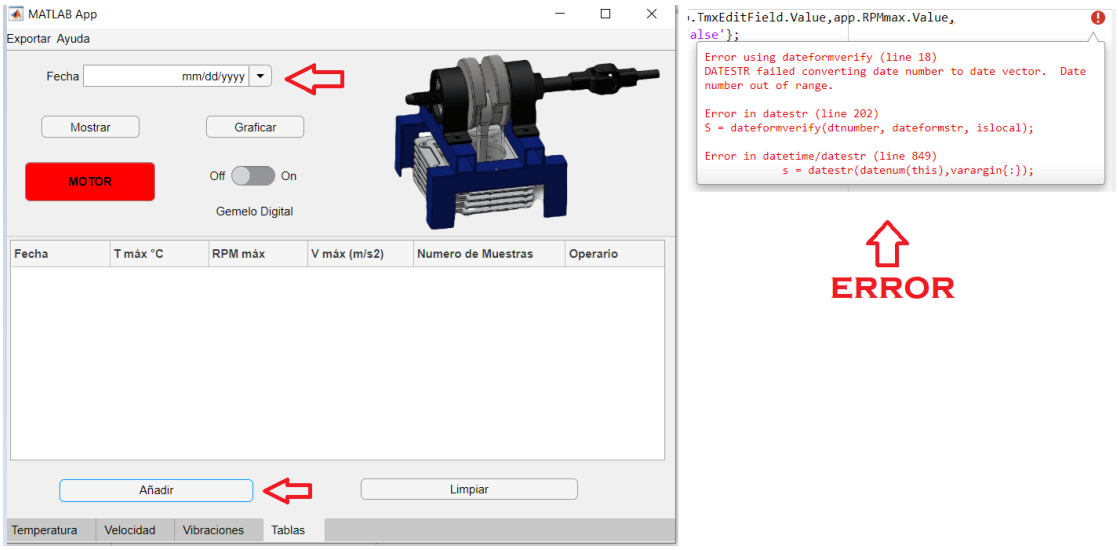


Apéndice B Correcciones de errores

En este documento encontraras solución a algunos de los errores que te pueden surgir durante la ejecución del software. Ten en cuenta que estos errores se generan la mayoría de las veces por una mala configuración o por la falta de algún dato necesario.

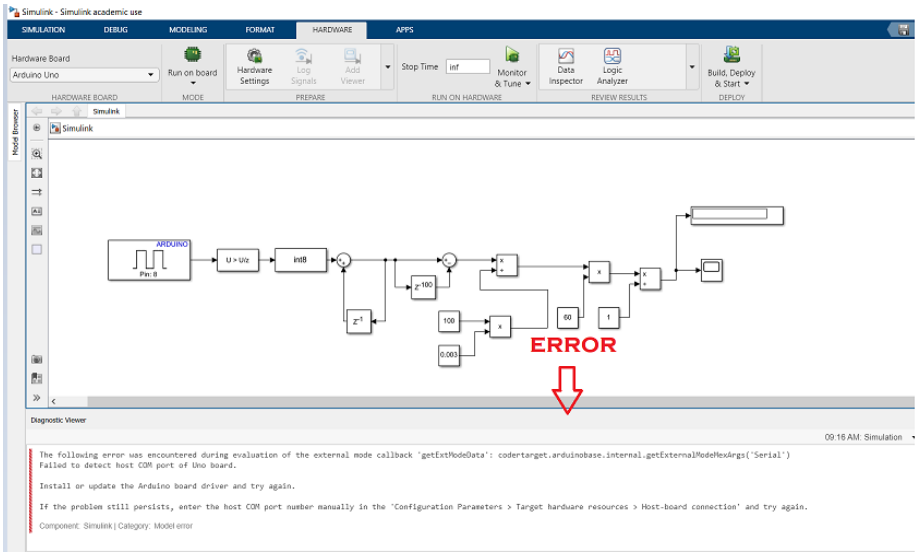
Error al presionar *Añadir* en la tabla

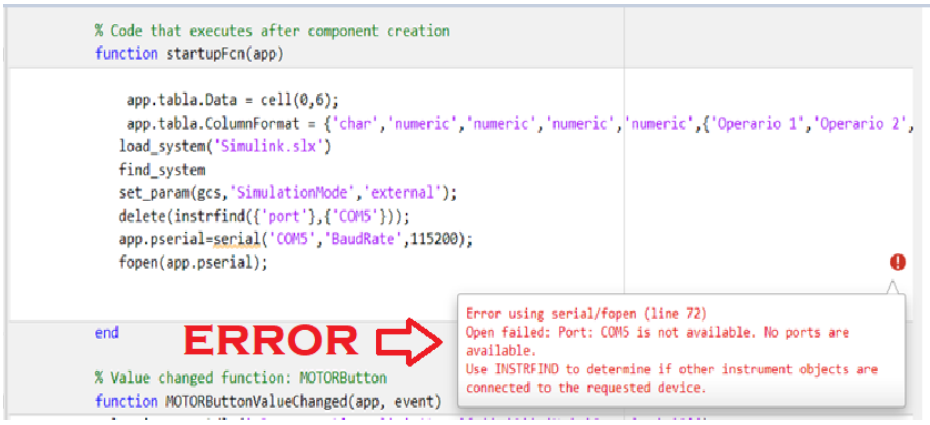
Este error se genera porque No se pone la fecha en la parte del cuadro fijo de la interfaz, como el botón *Añadir* añade automáticamente estos valores al no ingresar nada en el campo de la fecha se genera una inconsistencia en el tipo de dato que extrae la tabla.



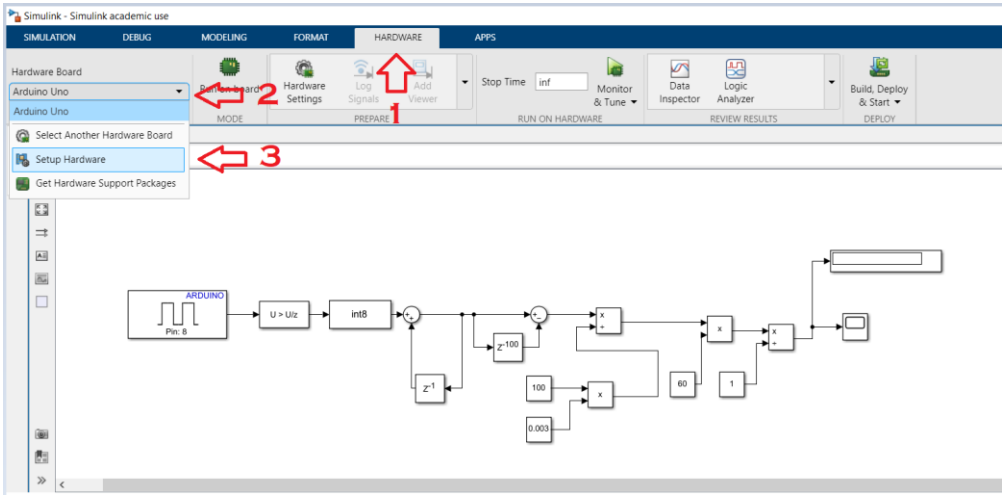
Matlab No reconoce Arduino

El siguiente error común es al momento de activar el software, correrlo o también al hacer la prueba del diagrama de bloques de Simulink. La corrección de este error está relacionada con la configuración general de Matlab, para esto vamos al diagrama de bloques de Simulink y seguimos los pasos que se muestran a continuación.

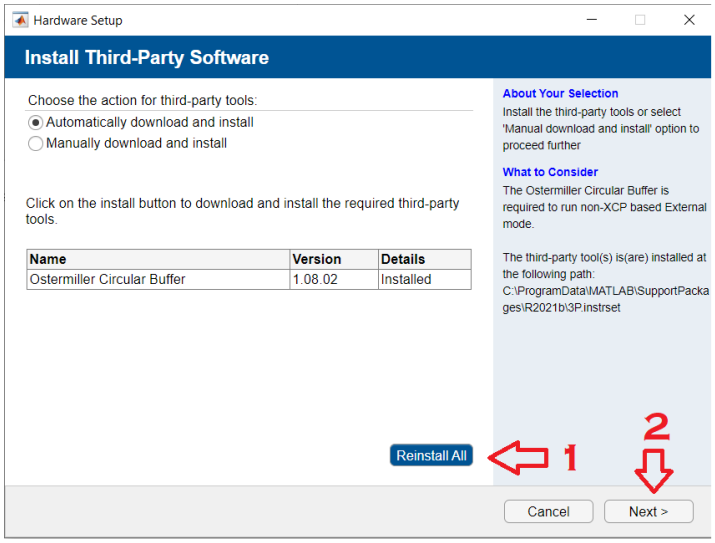

Simulink - Simulink academic use
SIMULATION
DEBUG
MODELING
FORMAT
HARDWARE
APPS
Hardware Board
Arduino Uno
Run on board
Hardware Settings
Log Signals
Add Viewer
Stop Time
inf
Monitor & Tune
Data Inspector
Logic Analyzer
Build, Deploy & Start
HARDWARE BOARD
MODE
PREPARE
RUN ON HARDWARE
REVIEW RESULTS
DEPLOY
Model Browser
Simulink
ARDUINO
Pin: 8
U > Uiz
int8
z-100
z-1
100
0.003
60
1
ERROR
Diagnostic Viewer
09:16 AM: Simulation
The following error was encountered during evaluation of the external mode callback 'getExtModeData': codertarget.arduinobase.internal.getExternalModeHexArgs('Serial')
Failed to detect host COM port of Uno board.
Install or update the Arduino board driver and try again.
If the problem still persists, enter the host COM port number manually in the 'Configuration Parameters > Target hardware resources > Host-board connection' and try again.
Component: Simulink | Category: Model error

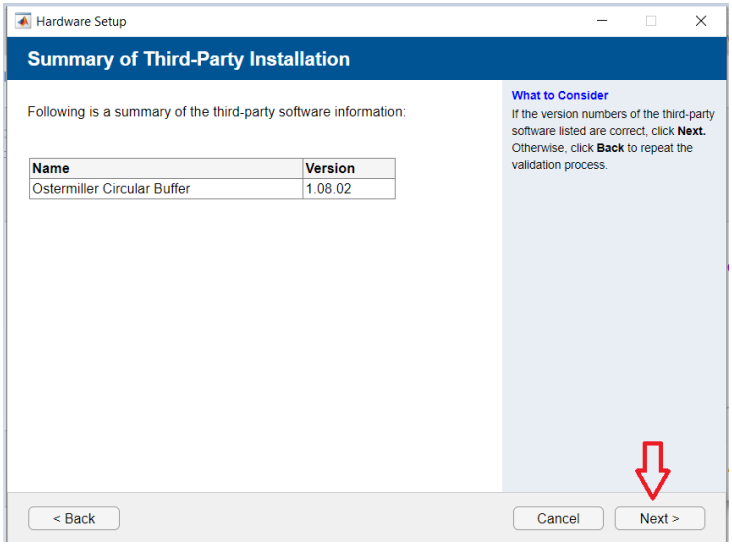

% Code that executes after component creation
function startupFcn(app)
app.tabla.Data = cell(0,6);
app.tabla.ColumnFormat = {'char','numeric','numeric','numeric','numeric',{'Operario 1','Operario 2',
load_system('Simulink.slx')
find_system
set_param(gcs,'SimulationMode','external');
delete(instrfind({'port'},{'COM5'}));
app.pserial=serial('COM5','BaudRate',115200);
fopen(app.pserial);
end
ERROR
% Value changed function: MOTORButton
function MOTORButtonValueChanged(app, event)
Error using serial/fopen (line 72)
Open failed: Port: COM5 is not available. No ports are available.
Use INSTRFIND to determine if other instrument objects are connected to the requested device.

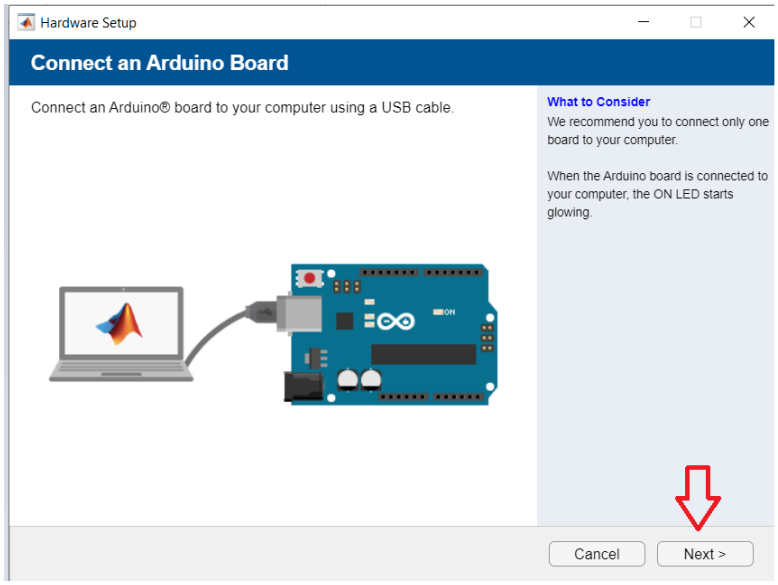
Solución


Simulink - Simulink academic use
SIMULATION
DEBUG
MODELING
FORMAT
HARDWARE
APPS
Hardware Board
Arduino Uno
Arduino Uno
Select Another Hardware Board
Setup Hardware
Get Hardware Support Packages
2
1
3
Hardware Settings
Stop Time
inf
Monitor & Tune
Data Inspector
Logic Analyzer
Build, Deploy & Start
MODE
PREPARE
RUN ON HARDWARE
REVIEW RESULTS
DEPLOY
ARDUINO
Pin: 8
U > Uiz
int8
z-100
z-1
100
0.003
60
1

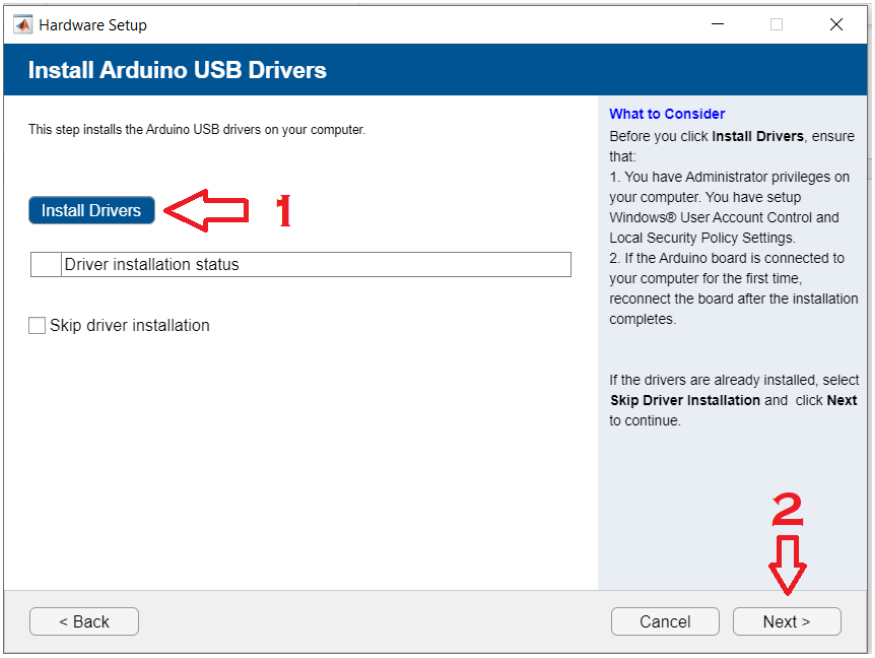
Después de esto, salta la siguiente ventana. Damos en Reinstall all y luego Next.



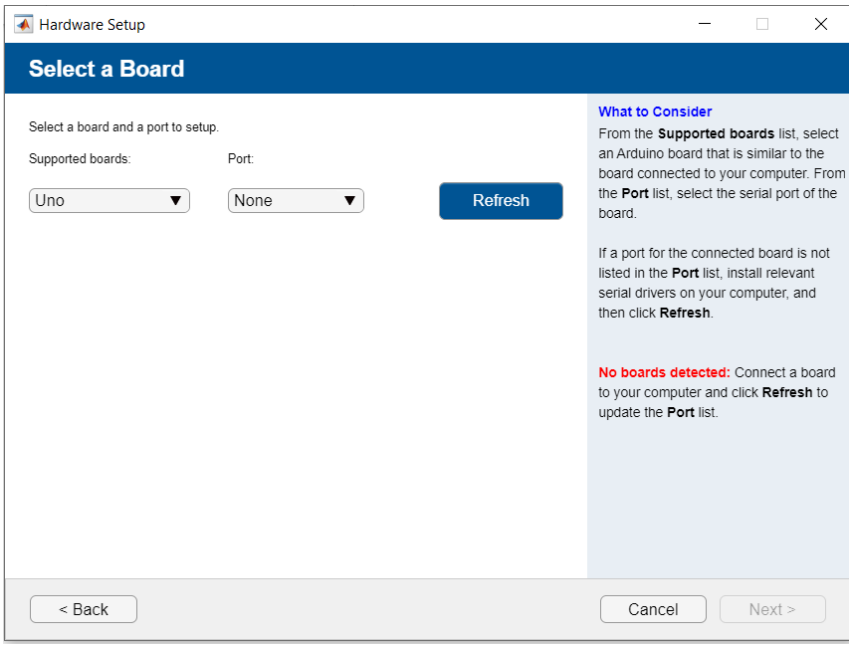




Una vez hecho esto, MATLAB detectará automáticamente los puertos donde esté conectada la placa de Arduino. Luego vamos a reinstalar los drivers del controlador y daremos Next.



Una vez hecho esto MATLAB mostrará las placas detectadas.

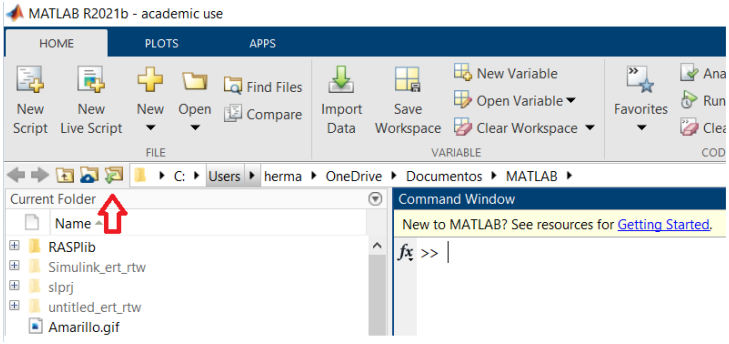


Si No reconoce la placa, desconecta el Arduino vuélvelo a conectar y presiona *Refresh.* Una vez reconocida la paca damos next y *Aceotar*. Y ya quedaría solucionado el problema de que No reconozca la placa.

Apéndice C Guía de laboratorio

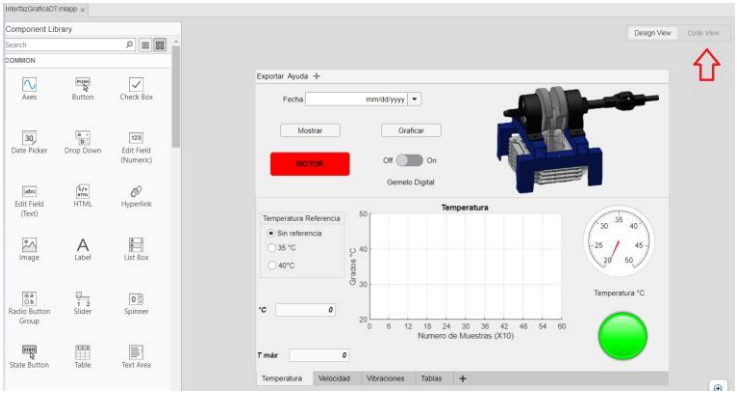
1. Abrir MATLAB

2. Clic en Browser for folder y abrimos la carpeta donde están los archivos del proyecto.

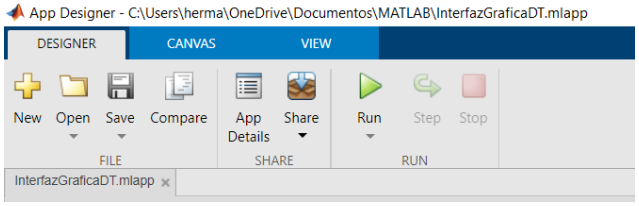


3. Abrimos el archivo *InterfazGraficaDT.mlapp*

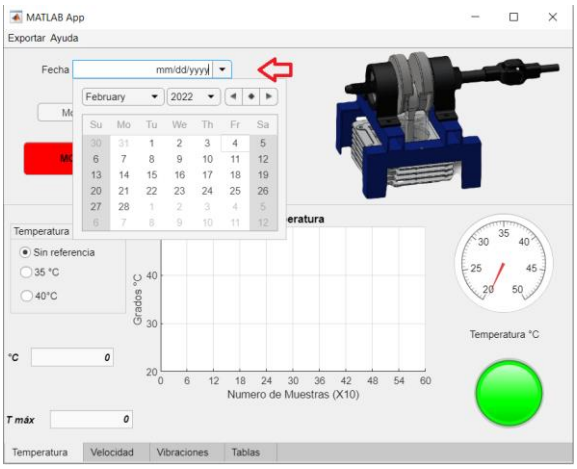
4. Una vez abierta podemos observar la herramienta AppDesigner y la construcción de la interfaz, puede dar clic en Code View si desea ver la codificación del software.



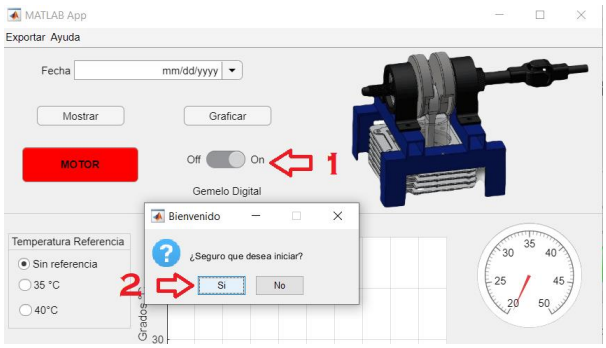
5. Hecho esto damos en el botón *Run* para correr la interfaz en la parte izquierda superior.



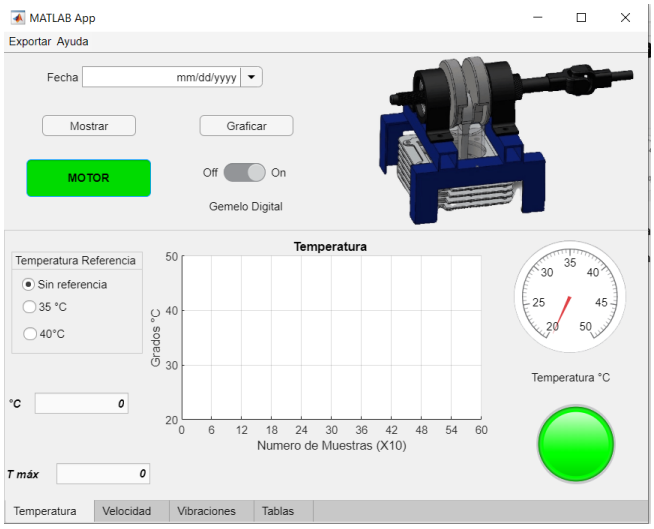
6. Una vez hecho esto se abrirá la interfaz, esto puede tardar un tiempo ya que allí se realizan las conexiones con las distintas herramientas que permiten capturar las mediciones.



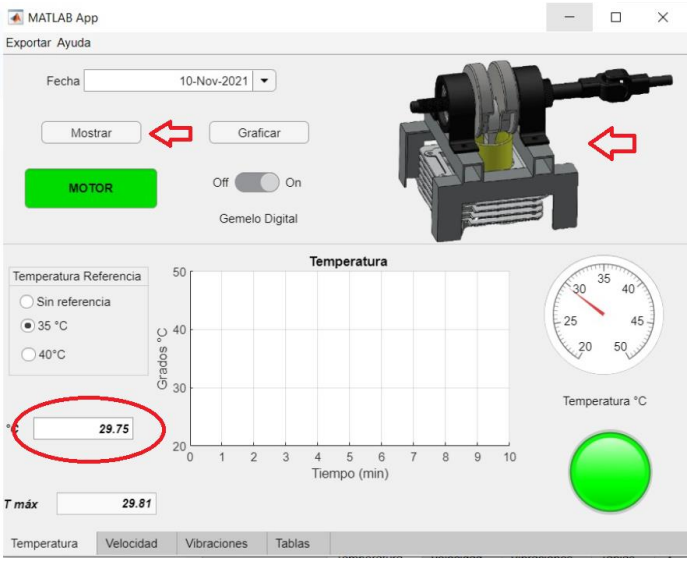
7. Una vez abierta ingresamos la fecha para que no ocurran errores más adelante. Después damos en el botón *inicio Digital* y luego *SI*. Recuerde que sin este botón activado no podrá visualizar las mediciones.



8. Si desea puede encender el botón del *Motor* para que observe el movimiento de la imagen, sin embargo, este botón aún no tiene funcionalidad.

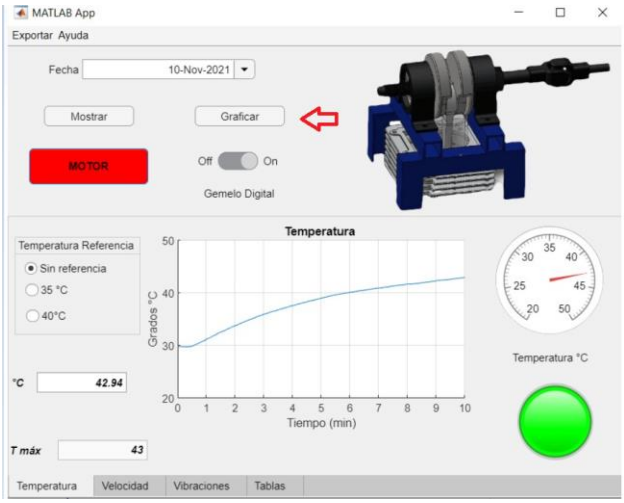


9. Una vez hecho esto damos en el botón *Mostrar* para visualizar las mediciones. Después de esto de clic en la imagen gif para visualizar el código de colores.

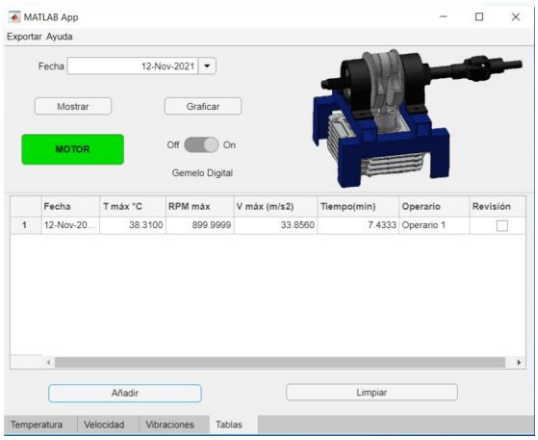


10. Navegue por las pestañas para visualizar las mediciones de todas las variables.

11. Ahora si desea puede graficar las mediciones dando clic en el botón *Graficar*, hágalo un tiempo después de haber presionado *Mostrar* para que la interfaz pueda almacenar una cantidad considerable de datos, o si desea espere a que el software grafique automáticamente (después de 600 muestras)

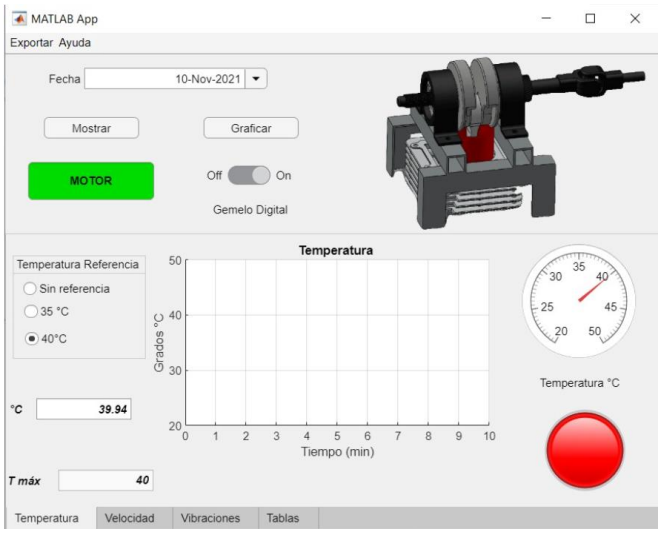


12. De igual forma en la pestaña de *Tablas*, puede tabular en cualquier momento para recoger los valores máximos de las mediciones o dejar que la interfaz lo haga por usted después de las 600 muestras.



13. Y listo, navegue, exporte, grafique e interactúe con la interfaz.

14. En la pestaña de temperatura elija una temperatura de referencia, observe lo que ocurre cuando la medición sobrepasa dicho valor



RECUERDE QUE EN LA PESTAÑA DE *AYUDA* ENCONTRARÁ ESTE DOCUMENTO Y OTROS MÁS QUE LE SERVIRÁN DE AYUDA EN CASO DE ERRORES Y DE CONFIGURACIONES NECESARIAS.

Apéndice D Encuestas

Nombre: Santiago Mendez	Edad: 24	Ocupación: Estudiante UIS			
	Muy bueno	Bueno	Regular	Malo	Muy malo
Esperiencia de uso		X			
Facilidad de uso	X				
Tiempo de respuesta de la Interfaz		X			
Sencillez de la interfaz	X				

Nombre: Jeison Olegario Cisneros	Edad: 23	Ocupación: Estudiante UIS			
	Muy bueno	Bueno	Regular	Malo	Muy malo
Esperiencia de uso	X				
Facilidad de uso	X				
Tiempo de respuesta de la Interfaz		X			
Sencillez de la interfaz	X				

Nombre: Andres Parmenio Mantilla Corredor	Edad: 23	Ocupación: Estudiante UIS			
	Muy bueno	Bueno	Regular	Malo	Muy malo
Esperiencia de uso		X			
Facilidad de uso	X				
Tiempo de respuesta de la Interfaz			X		
Sencillez de la interfaz		X			

Nombre: Gladys Santamaria Ariz	Edad: 55	Ocupación: Empleada			
	Muy bueno	Bueno	Regular	Malo	Muy malo
Esperiencia de uso			X		
Facilidad de uso	X				
Tiempo de respuesta de la Interfaz			X		
Sencillez de la interfaz		X			

Nombre: Carlos Augusto Galvis Gonzalez	Edad: 54	Ocupación: Empleado			
	Muy bueno	Bueno	Regular	Malo	Muy malo
Esperiencia de uso			X		
Facilidad de uso		X			
Tiempo de respuesta de la Interfaz			X		
Sencillez de la interfaz	X				