Implementación de la Interfaz de Usuario para el túnel de viento de la Universidad del Valle de Guatemala

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de la Interfaz de Usuario para el túnel de viento de la Universidad del Valle de Guatemala

Trabajo de graduación en modalidad de Tesis presentado por
Roberto Moreno Cáffaro
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,
2013

Vo. Bo. :

(f) 

Ing. Roberto Delgado

Tribunal Examinador:

(f) 

Ing. Roberto Delgado

(f) 

Ing. Víctor Hugo Ayerdi

(f) 

Ing. Sergio Izquierdo

Fecha de aprobación: Guatemala, 4 de diciembre 2013

ÍNDICE

Página

LISTA DE CUADROS

LISTA DE FIGURAS

RESUMEN

El propósito de este proyecto fue el diseño de una interfaz de usuario para el túnel de viento en la Universidad del Valle. Se manejaron protocolos de comunicación serial y programación en NI Labview para implementar esta interfaz y realizar cálculos con valores proporcionados por sensores. Se determinó en este proyecto que las velocidades de operación de los protocolos de comunicación permiten envíos carácter por carácter, y que el uso del Arduino UNO y Labview habilita una gran variedad de facilidades para la programación de interfaces. Sin embargo, la gran variedad de medidas posibles en un túnel de viento garantiza que aún con los logros de este proyecto es posible obtener mucho más.

I. INTRODUCCIÓN

El propósito de este trabajo fue implementar una interfaz de usuario para un túnel de viento en la manera más económica, práctica, y sencilla posible, comenzando desde cero y utilizando programación gráfica y de microcontroladores. Se trató de determinar la compatibilidad entre los dos últimos mediante la integración de Labview y el software de Arduino. El proyecto se enfocó en el envío y recepción de valores provenientes de cuatro módulos de sensores por el puerto serial de la computadora. Aunque se consideró mucha de la teoría de dinámica de fluidos externos para el túnel, el objetivo principal del proyecto fue obtener los valores relevantes de los sensores, sin manipularlos más allá de lo necesario, para después, a futuro, discutir los detalles de la didáctica que se implementará con el túnel. Al final se observó principalmente que es enteramente posible integrar Labview con Arduino en una manera económica y eficiente, que no es siempre necesario enviar la menor cantidad de bytes posible en una trasmisión serial, particularmente en el caso de una GUI, y que el uso de Labview y Arduino permite muchísima versatilidad a la hora de realizar cálculos.

II. OBJETIVOS

A. Generales

1. Instalar un túnel de viento funcional con la capacidad de realizar una variedad útil de medidas en la Universidad.
2. Integrar componentes mecánicos y electrónicos en una manera práctica, económica, ordenada, y fácil de modificar para cumplir los objetivos de la interfaz.
3. Crear una interfaz de usuario fácil de utilizar por el usuario.
4. Aprender más sobre la programación de interfaces para cualquier sistema.

B. Específicos

1. Elaborar la interfaz de usuario de tal forma que esté programada en una manera eficiente por si se requieren modificaciones, y que esté bien documentada.
2. Seleccionar un programa eficiente y práctico para realizar la interfaz.
3. Permitir por medio de comandos en la interfaz variar el ángulo de modelo, y la velocidad del fluido en el túnel.
4. De los valores obtenidos por los sensores, obtener las fuerzas en el modelo y sus coeficientes de arrastre y sustentación, desplegándolos.
5. Asegurar la portabilidad del programa con las computadoras de la universidad.
6. Diseñar con microcontroladores un sistema de manejo de datos para la interfaz del túnel de viento.
7. Permitir la obtención de datos por el usuario a un programa como Microsoft Excel.

III. JUSTIFICACIÓN

La instalación del túnel de viento con una interfaz de usuario eficiente y representativa es de gran utilidad para la Universidad. En la actualidad, para cursos que involucren la mecánica de fluidos no hay mucho material existente para mostrar los conceptos enseñados en clase en una manera práctica y accesible. Para mantener estándares altos de educación en la Universidad, considerando la necesidad cada vez más grande de tener tecnología didáctica para enseñar los conceptos relevantes a las carreras de Ingeniería, la instalación de este túnel de viento es una inversión rentable e incluso necesaria.

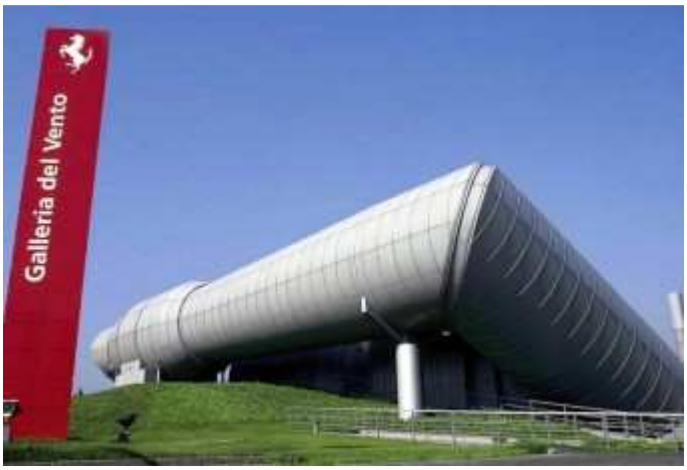
Como un estudiante a punto de graduarse en Ingeniería Mecatrónica, este proyecto también sirvió para alcanzar muchas finalidades personales. Una de las razones por las cuales este proyecto fue de gran utilidad para mí fue el hecho de que afiancé mi conocimiento de microcontroladores junto con los integrantes de mi grupo. Tuve la fortuna de poder irme de intercambio y este proyecto permitió unir los conocimientos adquiridos en el extranjero junto con algunos de los temas que se trataron en la UVG que no se estudiaron afuera. Para mí es importante destacar que lidiar con las complicaciones relacionadas a un proyecto real es una destreza que me interesa mejorar, ya que me hará un ingeniero más competitivo y con mayor capacidad para realizar trabajo de calidad en la industria o en procesos de investigación. Otra razón por la cual considero que este proyecto será importante para mí es el hecho de que día a día se desarrollan cada vez más lenguajes de programación, y es muy importante no sólo actualizarse sino también tener la capacidad de aprender éstos lenguajes rápido. La rápida evolución del poder de cómputo y de la tecnología garantiza que un ingeniero que no es capaz de actualizarse rápido se atrasará y no podrá ser de calidad mundial. La última destreza que busqué desarrollar un poco más en este proyecto fue mi capacidad de investigación. No todos los lenguajes de programación tienen una buena base de ayuda, y es importante tener la capacidad de poder analizar códigos encontrados en el internet para resolver los problemas que puedan presentarse al trabajar.

IV. MARCO TEÓRICO

A. Flujo sobre Cuerpos Inmersos:

Existen muchísimos ejemplos diarios del flujo de fluidos sobre cuerpos inmersos en ellos, incluyendo pero no reduciéndose al flujo de aire alrededor de las alas de aviones, el flujo de aire alrededor del chasís de automóviles, y el flujo de agua alrededor de barcos y submarinos. A todos los flujos anteriores se les conocen como flujos externos, ya que el objeto en cuestión se encuentra completamente sumergido en el fluido y este es completamente "externo" al objeto. El análisis de flujos externos se ha vuelto cada vez más importante: el análisis de los efectos de fluidos externos, principalmente todo lo que involucra arrastre y sustentación, permite mejorar drásticamente la economía de muchísimos vehículos, y su facilidad de manejo. Aunque existen formas teóricas de calcular estos factores, la mayoría del tiempo los flujos externos se analizan experimentalmente en túneles de viento, donde se utilizan modelos o a veces el propio objeto a analizarse en sí para observar el efecto del fluido sobre éste. Los túneles de viento pueden llegar a comprender inversiones multimillonarias, como el túnel mostrado en la Figura 1, que generan ahorros de enormes magnitudes al emplearse en el proceso de diseño. (Munson, Bruce R., 2013)

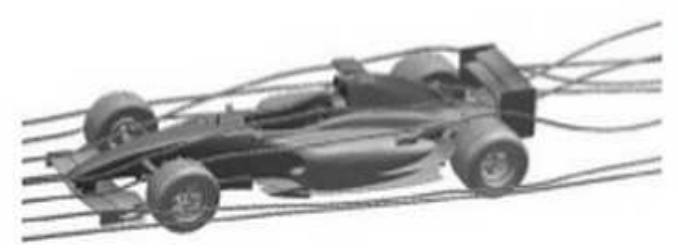
Figura 1. Túnel de Viento Maranello de Ferrari



Un cuerpo inmerso en un fluido en movimiento siempre experimenta una fuerza resultante de la interacción entre este cuerpo y el fluido alrededor de él. Es posible que el cuerpo

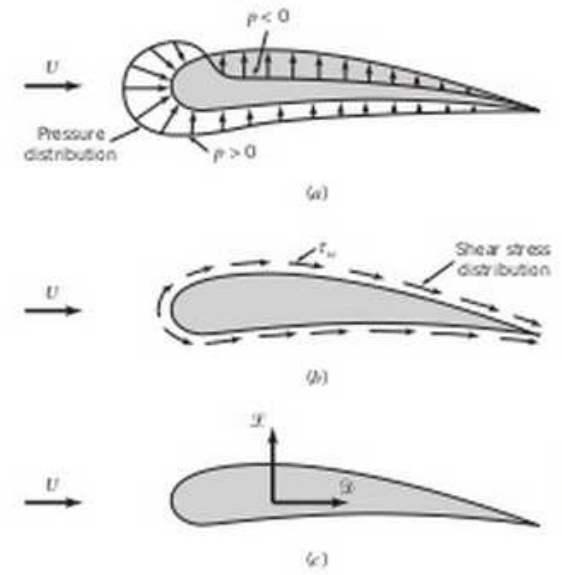
esté estacionario y el fluido se mueva alrededor de él, o que el fluido lejos del objeto se encuentre estacionario y el objeto se mueva con una velocidad relativa al fluido – casi siempre se fija el eje de coordenadas sobre el modelo u objeto a analizarse y se toma como un objeto fijo. En el caso de análisis mencionado anteriormente, surge un importantísimo valor de mérito en el análisis de túneles de viento, que es el valor de la velocidad ascendente o “upstream” del fluido en el túnel. Por convención se asume que esta velocidad es constante y laminar alrededor del modelo. Este, sin embargo, no necesariamente es el caso, ya que aún con un flujo que parezca poseer estas características es posible que se presenten zonas con flujo turbulento cerca del modelo analizándose. A pesar de esta propiedad, la consideración anterior no tiene mucha importancia en la mayoría de los casos. (Munson, Bruce R., 2013)

Figura 2. Líneas de flujo en un túnel de viento



(Munson, Bruce R., 2013)

Figura 3.Distribución de presión y estrés y sus resultantes en un alerón



(Munson, Bruce R., 2013)

Cuando un cuerpo se encuentra en flujo externo, se presentan siempre dos fuerzas provenientes de la interacción entre éste y el fluido alrededor de él. Estas fuerzas son, netamente, los esfuerzos cortantes sobre el cuerpo analizándose debido a los efectos de la viscosidad del fluido pasando sobre éste, y esfuerzos normales debido a la presión presente en los alrededores del cuerpo en movimiento. Es difícil obtener información detallada del esfuerzo cortante y presión alrededor de un modelo – casi siempre se trata con las fuerzas resultantes de arrastre -en la dirección de la velocidad ascendente- y la fuerza de sustentación direccionada hacia la normal del modelo analizándose. En la Figura 3 puede verse un alerón con los parámetros resultantes anteriormente descritos, y las distribuciones de estrés y presión de las que éstas resultantes provienen. (Munson, Bruce R., 2013)

Ecuación 1. Fórmula adimensional para el Coeficiente de Sustentación (Munson, Bruce R., 2013)

$$C_L = \frac{L}{\frac{1}{2}\rho U A}$$

Ecuación 2. Fórmula adimensional para el coeficiente de Arrastre. (Munson, Bruce R., 2013)

$$C_D = \frac{D}{\frac{1}{2}\rho U A}$$

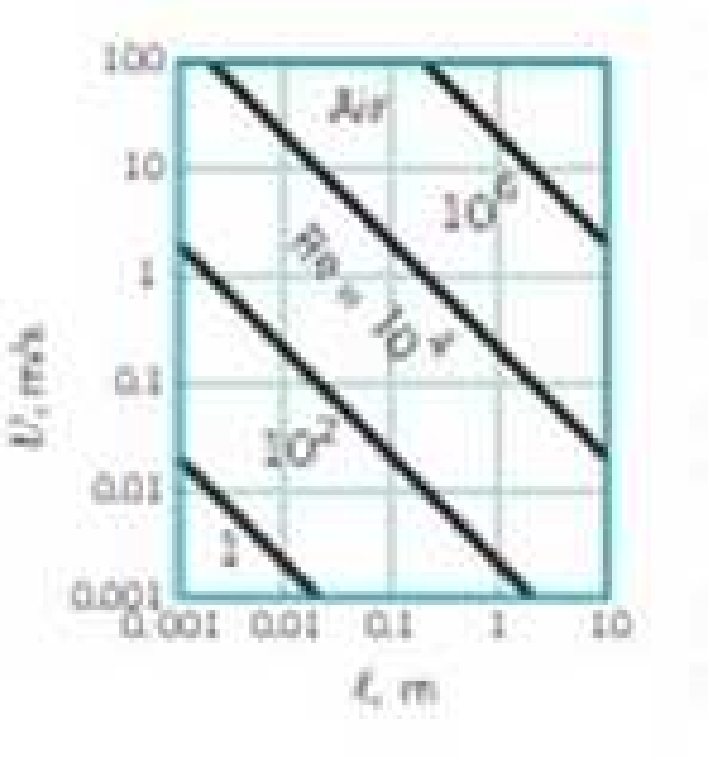
La extrema dificultad para comprobar exactamente la distribución de fuerzas alrededor de modelos complicados llevó a que se desarrollaran las ecuaciones mostradas en las Ecuaciones 1 y 2, para los coeficientes de sustentación y arrastre, respectivamente, donde A es el área característica de los objetos analizados, D es la fuerza de arrastre (Drag), L es la fuerza de sustentación (Lift), ρ es la densidad del fluido en el túnel y U es la velocidad ascendente alrededor del fluido. Para el área característica, se analiza ya sea el área frontal (el área proyectada vista directa desde una dirección paralela a la velocidad ascendiente) o el área planiforme, que es el área proyectada vista desde una dirección perpendicular a la velocidad, o desde arriba del objeto). Estas áreas no deben mezclarse en el cálculo y sólo una debe tomarse en consideración para el análisis, y las dos fórmulas. (Munson, Bruce R., 2013)

Ecuación 3. Fórmula para el número de Reynolds (Munson, Bruce R., 2013)

$$Re = \frac{\rho U l}{\mu} = \frac{U l}{v}$$

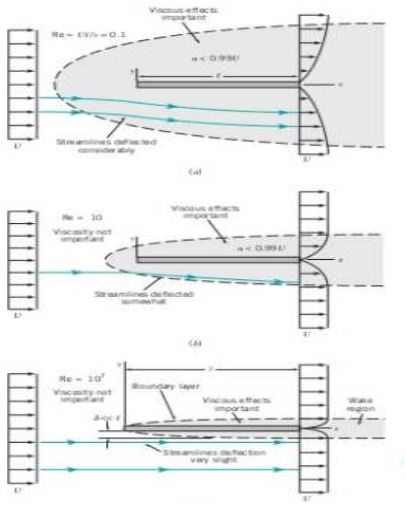
El flujo externo alrededor de un objeto depende de muchos parámetros, incluyendo su tamaño, orientación, velocidad, y las propiedades del fluido alrededor de él. El uso de análisis dimensional sobre el flujo externo, siguiendo los criterios del primero, establece que el flujo depende de los parámetros adimensionales que sean relativos a él. Los más importantes en el caso del flujo externo son el número de Reynolds, el número Mach, y, en el caso de que existan flujos con interfaces entre dos fluidos, el número de Fraude. El análisis de flujo externo se basa casi exclusivamente en el análisis del número de Reynolds, cuya fórmula puede observarse en la Ecuación 3, y sus efectos sobre el arrastre y la sustentación sobre el modelo.

Figura 4. Número de Reynolds para largos característicos y velocidad ascendente



(Munson, Bruce R., 2013)

Figura 5. Distribuciones de flujo y capas límite para números bajos, medianos, y altos de Reynolds



(Munson, Bruce R., 2013)

El número de Reynolds representa la relación entre los efectos de inercia de un fluido y los efectos de la viscosidad de este mismo. Basado en la fórmula, cuando la viscosidad es casi inexistente (μ=0), el número de Reynolds se aproxima al infinito, mientras que cuando hay una masa casi completamente despreciable, la densidad en la fórmula se aproxima a cero y no existen efectos de inercia. En la práctica, todos los flujos poseen números de Reynolds entre estos dos extremos. Existen otras consideraciones que afectan el análisis con el número de Reynolds. Una de estas es que la mayoría de flujos se asocian con objetos que poseen un largo característico (l) entre .01 y 10m. Adicionalmente a esto, las típicas velocidades ascendentes oscilan entre .01m/s a 100m/s. Estos dos rangos mencionados anteriormente llevan a que el típico número de Reynolds a analizarse se encuentra entre 10 y 10^9. Se pueden ver varios rangos de número de Reynolds para diferentes largos característicos y velocidades ascendentes en la Figura 4. (Munson, Bruce R., 2013)

Existen ciertas situaciones que pueden observarse al analizar el valor del número de Reynolds. Si el número de Reynolds es mayor a 1, por ejemplo, y mientras más incrementa el número de Reynolds, el objeto siendo analizado es cada vez menos afectado por la viscosidad, como se muestra en la Figura 5. Sin embargo, cabe destacarse que, como se mencionó anteriormente, no es teóricamente posible que el número de Reynolds sea igual al infinito por lo cual siempre existe algún efecto de viscosidad en cualquier modelo analizado. El efecto más

visible de esta propiedad es la existencia de capas límite. La capa límite para un modelo, también mostrada en la Figura 5 (boundary layer) disminuye mientras aumenta el número de Reynolds, y representa la región alrededor de un modelo en la cual, por la viscosidad del fluido, se presenta una velocidad de cero alrededor del objeto en flujo externo. Un parámetro final que es importante considerar es la transición de una capa límite laminar a una capa límite turbulenta, que ocurre para números de Reynolds entre 20,000 y 300,000, dependiendo de la superficie del objeto en flujo externo y la turbulencia en el flujo analizado. Esta transición ocurre gradualmente en el flujo, y no es instantánea. (Munson, Bruce R., 2013)

Ecuación 4. El coeficiente de arrastre y su dependencia en forma, Reynolds, Mach, Froude, y rugosidad relativa (Munson, Bruce R., 2013)

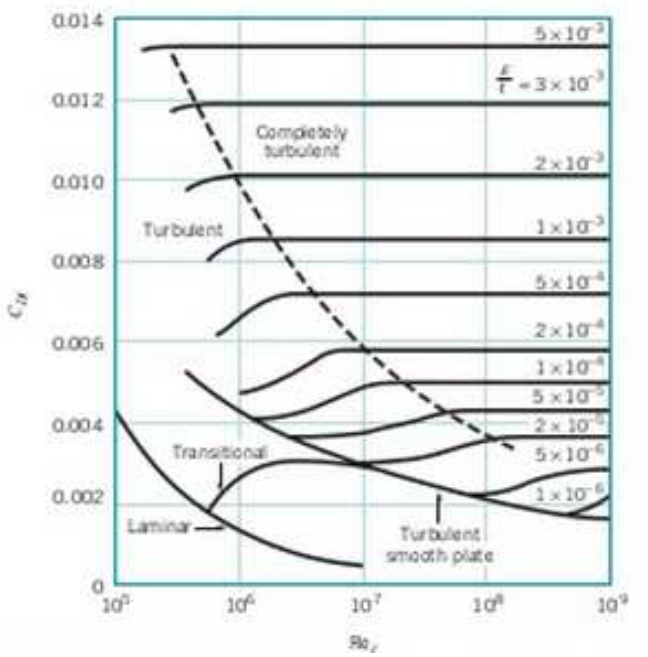
$$C_D = \varphi(forma, Re, Ma, Fr, \frac{\varepsilon}{l})$$

Ecuación 5. Arrastre de fricción (Munson, Bruce R., 2013)

$$D_f = \frac{1}{2}\rho U^2 blC_{Df}$$

El uso de las fórmulas empíricas mostradas en las Ecuaciones 1 y 2 no es tan preciso, ya que existen factores que tienen más relevancia en el análisis de flujo que otros, cuando se van cambiando las formas de los modelos utilizados. Para que el coeficiente de arrastre sea en verdad representativo, se requieren muchos experimentos y pruebas específicas a cada modelo, al igual que para el cálculo de sustentación. En el caso de una placa plana de largo *l*, por ejemplo, se toma en cuenta su rugosidad relativa y el número de Reynolds para observar el efecto en el arrastre, aunque también depende de otros factores, como se puede observar en la Ecuación 4. Un parámetro adicional que se define típicamente es el arrastre de fricción, cuya fórmula puede observarse en la Ecuación 5, y que corresponde al efecto de las fuerzas cortantes sobre el objeto en flujo externo. Este valor varía de acuerdo a parámetros del fluido pero también depende fuertemente del número de Reynolds por medio del coeficiente de arrastre de fricción Cdf, cuyas fórmulas pueden verse en la Tabla 1 para diferentes condiciones de flujo (y números de Reynolds). En la Figura 6 pueden verse gráficas para el coeficiente de arrastre de una placa plana para diferentes números de Reynolds y rugosidades relativas. (Munson, Bruce R., 2013)

Figura 6. Coeficiente de arrastre de fricción para diferentes rugosidades y números de Reynolds, en una placa plana



(Munson, Bruce R., 2013)

Cuadro 1. Ecuaciones experimentales para el coeficiente de arrastre de fricción para una placa plana

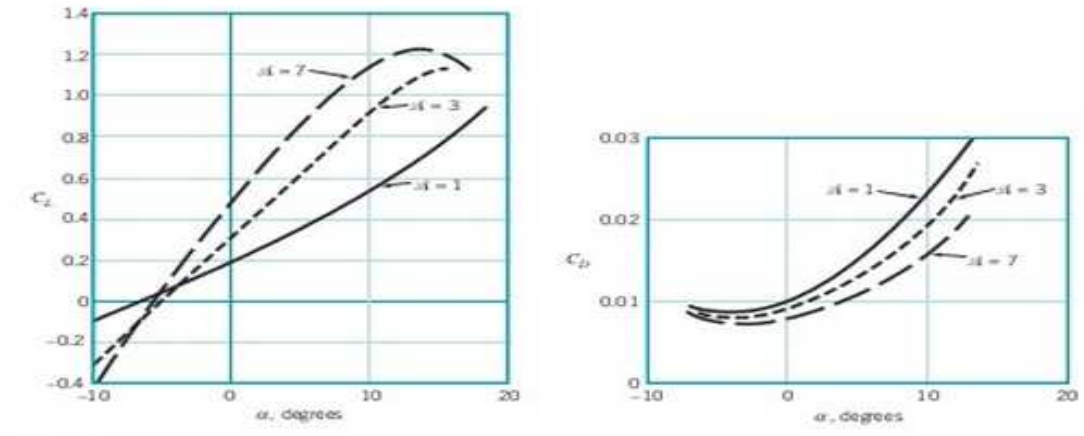
Empirical Equations for the Flat Plate Drag Coefficient (Ref. 1)

Equation	Flow Conditions
$C_D = 1.328/(\mathrm{Re}_L)^{0.5}$	Laminar flow
$C_D = 0.455/(\log \mathrm{Re}_L)^{2.58} - 1700/\mathrm{Re}_L$	Transitional with $\mathrm{Re}_{xc} = 5 \times 10^5$
$C_D = 0.455/(\log \mathrm{Re}_L)^{2.58}$	Turbulent, smooth plate
$C_D = [1.89 - 1.62 \log(\varepsilon/\ell)]^{-2.5}$	Completely turbulent

(Munson, Bruce R., 2013)

Otros ejemplos de análisis de arrastre y sustentación que se pueden realizar se pueden ver en la Figura 7, que muestra las gráficas de arrastre y sustentación para diferentes ángulos de ataque y relaciones de aspecto de un alerón. La fórmula para la relación de aspecto de un alerón puede verse en la ecuación 5. Se pueden sacar otras graficas adicionales a las de la Figura 7 de acuerdo a estándares de alerones, como puede verse en la Figura 8. Se puede obtener una infinidad de diferentes parámetros si se desea - casi siempre se obtienen los valores adimensionales de coeficientes de arrastre y sustentación para tener algo a que referirse. (Munson, Bruce R., 2013)

Figura 7. Sustentación y arrastre en función a ángulo de ataque y relación de aspecto en un alerón

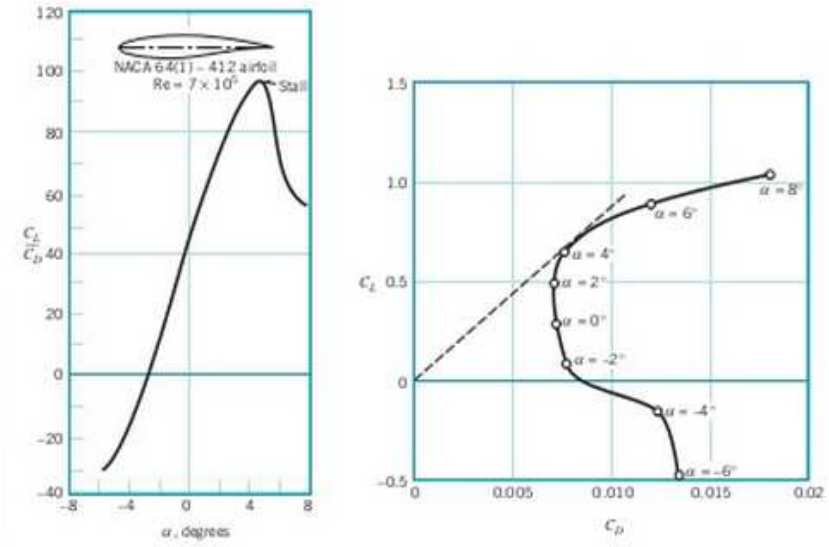


(Munson, Bruce R., 2013)

Ecuación 6. Fórmula para la relación de aspecto (Munson, Bruce R., 2013)

$$A = \frac{b^2}{A_{planiforme}}$$

Figura 8. Coeficiente de sustentación sobre arrastre vs. grados y coeficiente de sustentación vs. coeficiente de arrastre, respectivamente, para un alerón NACA 64.



(Munson, Bruce R., 2013)

B. Protocolos de Comunicación Serial

La comunicación desde microcontroladores hacia computadoras para el procesamiento de señales se realiza utilizando los múltiples diferentes tipos de buses que están presentes en el hardware de la computadora. El uso de la interfaz serial de una computadora es conveniente en muchas aplicaciones, ya que puede implementarse con sólo una línea de entrada y salida, a diferencia de los buses ISA y PCI. El puerto serial también permite la distribución de memoria y recursos compartidos entre diferentes procesadores. Por estos motivos, es un tipo de comunicación conveniente si se requieren diseños simples y con poco hardware involucrado. (Patrick, John, 2002)

El uso de comunicación serial tan proliferado en la industria ha llevado a que muchas compañías implementen diferentes protocolos de comunicación para diferentes usos del puerto serial, y con una infinidad de diferentes propiedades, facilidades, y posibilidades. La variedad de protocolos de comunicación disponible es algo importantísimo de analizar si se desean efectuar comunicaciones entre microcontroladores y con una computadora. Existen algunos términos importantes a definir para este fin, que se muestran a continuación: (Patrick, John, 2002)

Bus síncrono/asíncrono: Un bus síncrono es un bus que envía información de acuerdo a señales de reloj, y un bus asíncrono no depende de un reloj para el envío de datos. (Patrick, John, 2002)

Full Duplex y Half Duplex: Un protocol de comunicación full-duplex permite enviar y recibir información por el puerto serial simultáneamente. La comunicación half-duplex permite enviar o recibir, pero sólo se puede realizar una de las dos en cualquier momento. (Patrick, John, 2002)

Bus Master/Slave: Un bus master/slave es un bus que contiene un master que controla una serie de otros componentes, conocidos como “slaves”. Esta configuración de buses es casi siempre síncrona ya que es conveniente para estas configuraciones que el master maneje tiempos para el envío de datos, y use un reloj. (Patrick, John, 2002)

Bus Multi-Master: Un bus multi-master es un bus master/slave que permite múltiples masters en control del mismo bus. Para saber qué máster rige el proceso en cada momento se realizan pruebas de arbitración entre los masters, que determinan cuál controlará el proceso y por lo tanto los comandos a ejecutarse. (Patrick, John, 2002)

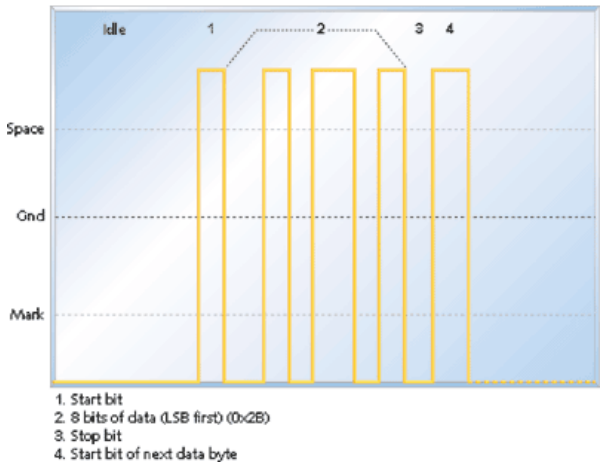
Interfaces punto a punto o “peer”: Esta configuración de comunicación involucra a dos componentes de hardware que no son ni masters ni slaves, y tienen la misma precedencia. Este tipo de interfaces son típicamente asíncronas por la poca importancia de manejar tiempos. (Patrick, John, 2002)

Interfaz Multi-Drop: Este término se refiere a una interfaz en la que existen múltiples receptores y sólo un transmisor de información. (Patrick, John, 2002)

Bus Multi-punto: Esta configuración de buses involucra a más de dos peers, y se diferencia de la interfaz Multi-Drop por el hecho que permite la comunicación en ambas direcciones (enviar o recibir). Estos buses son half dúplex. (Patrick, John, 2002)

1. RS-232: El protocolo RS-232 es extremadamente común y encontrado en casi todas las computadoras. Es una interfaz punto a punto capaz de manejar velocidades desde 20Kbps hasta 115.2 Kbps, siempre que se puenteen bien las tierras de los cables y éstos sean de corta distancia. Se pueden alcanzar distancias de hasta 200 pies para utilizar este protocolo si los cables utilizados son de baja capacitancia. El RS-232 permite comunicación full duplex entre dos pares de trasmisores/receptores: el DTE (data terminal equipment, que en el caso de una interfaz correspondería a una computadora) y el DCE (data communication equipment, que corresponde a los periferales utilizados para transferir información. En la Figura 9 puede verse una señal de envío típica de este protocolo. (Patrick, John, 2002)

Figura 9. Configuración estándar de envío para el RS-232



(Patrick, John, 2002)

Cada uno de los pares de trasmisores y receptores conectan su señal de trasmición a su señal de recepción y viceversa para el envío de datos. El envío de datos se basa en un rango de voltajes de -3 a 3V, donde un valor mayor a 3V es un zero binario y un valor menor a -3V es un uno binario, sin utilizar los valores entre -3 y 3V. Para convertir a este rango de voltajes es necesario el uso de convertidores como el MAX232 para operar el típico rango de voltaje lógico proporcionado por una computadora de 0 a 5V. La típica configuración de bits del RS-232 al comunicarse con una PC es ocho bits de información con un stop bit, 8N1, aunque este protocolo no define un valor máximo de bits en su envío, ni la necesidad de usar bits de inicio o final. Adicionalmente a lo ya mencionado, se utilizan comúnmente UARTs (Universal Asynchronous Receiver Transmitters) para lograr alcanzar mayores velocidades de operación cercanas a los 115.2 Kbps. (Patrick, John, 2002)

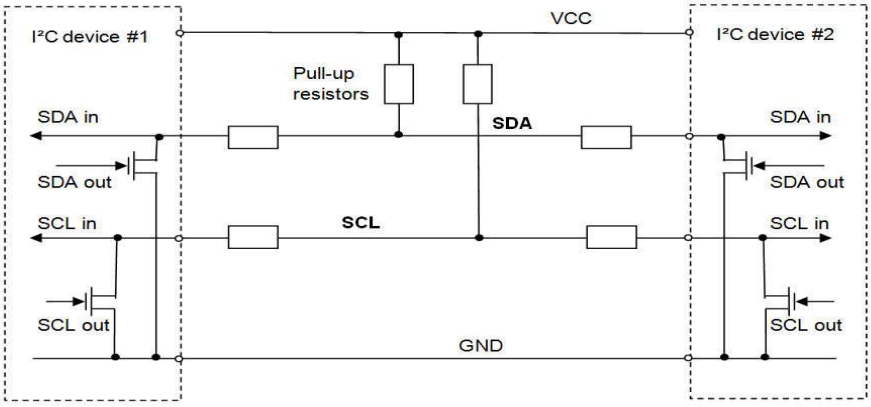
2.RS-422 y RS-485: A estos protocolos también se les conoce como TIA/EIA 422-B y TIA/EIA-485A, respectivamente. Pueden alcanzar velocidades de hasta 10Mbps y sus cables alcanzan distancias muy grandes, hasta 4,000 pies. Esto se debe a que usan buses diferenciales con señales de 1.5V a 6V, y tienen muy alta imunidad al ruido del ambiente. La RS-422 es una interfaz multi-drop con configuración half duplex estándar, con un máximo de 10 unidades. Si se desea comunicación full duplex con este protocolo es posible de implementarse, pero se requiere que hayan buses individuales entre cada receptor y el trasmisor, lo que representa un incómodo problema de hardware y por lo cual típicamente sólo se usa el RS-422 entre dos nodos. El RS-485, por otra parte, está configurado en buses multi-puntos con un máximo soportado de 32 trasmisores y/o receptores. Ambos protocolos de comunicación utilizan el mismo tipo de envío discutido anteriormente para el RS-232, mostrado en la Figura 9. Es posible utilizar convertidores para cambiar del protocolo RS-232 hacia cualquiera de estos dos protocolos, aunque debe tomarse en cuenta que la comunicación sólo es full duplex para el RS-232 y algunas configuraciones del RS-422. Finalmente, también existe la posibilidad de usar UARTs especializados para mejorar el funcionamiento del RS-485. (A Quick Comparison, 2013)

3. I2C: El Inter-Integrated Circuit (I2C) fue diseñado por Phillips en 1982. Es un protocolo de comunicación síncrono, half-duplex, y multi-master. El protocolo I2C sólo utiliza dos líneas además de una tierra común: SDA para los datos de envío/recepción, y SCL para el reloj. Estas líneas son manipuladas por resistores pull-up y esencialmente funcionan como un AND cableado en todo momento. El I2C permite el uso de direcciones de 7 ó 10 bits para comunicarse desde el master a los slaves presentes en el sistema. Típicamente se utilizan sólo 7 bits, pero la gran

variedad de elementos en el mercado con direcciones I2C específicas creó la necesidad de permitir la implementación de direcciones de hasta 10 bits para ajustarse al número limitado que existe con sólo 7 bits. (Patrick, John, 2002)

En la comunicación en I2C, el master se encarga de todos los cambios en el reloj para el envío de datos. La comunicación comienza cuando cualquiera de los aparatos inicia una transferencia de datos, siendo designado el aparato en hacer esto como el “master”. Primero se genera una condición de inicio, que consiste en una transición de high a low en la línea SDA mientras que la SCL se encuentre high.. Después de esto, se envía la dirección del slave con el que se desea comunicarse, ya sea en 7 bits ó 10 bits dependiendo del caso. Inmediatamente después de esta señal se envía un bit para designar si la operación deseada es leer o enviar valores a cualquiera de los slaves (read/write). El master después espera un bit de “acknowledge” de parte de la dirección que envió – si no recibe nada se pasará a una condición de stop después de cierto tiempo. Si el bit de acknowledge sí se obtiene de parte de la dirección del slave enviada, se trasmite un byte de información. Después cada byte trasmitido se le envía un acknowledge al master si se está escribiendo a un slave, o se le envía un acknowledge al slave por el master si se está leyendo la información, para confirmar que sí fue recibido el byte. Finalmente una vez todos los bytes deseados son trasmitidos, se envía un bit de stop que para la trasmisión de datos termine. La forma en la que el protocolo I2C transiciona entre cada uno de los parámetros anteriormente mencoinados es enviando cada bit o byte cuando SCL se encuentra low, como puede verse en la Figura 11. En la Figura 12 se pueden ver la estructura de envíos de I2C resumida, al igual que los envíos y acknowledges enviados por el master o el slave cuando se recibe información o se escribe, respectivamente. (Patrick, John, 2002)

Figura 10. Esquemática de conexión en I2C



(Byte Paradigm, 2013)

Figura 11. SDA y SCL en I2C

A start condition, a one data, a zero data bit, and a stop condition

(Patrick, John, 2002)

Figura 12. Orden de envíos en el protocolo I2C

START	Slave address	Rd/nWr	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 1: writing 2 byte to a slave. The data put on the bus by the master are shaded.

START	Slave address	0	0	Data	0	Data	0	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Example 2: reading 2 bytes from a slave. The data put on the bus by the master are shaded.

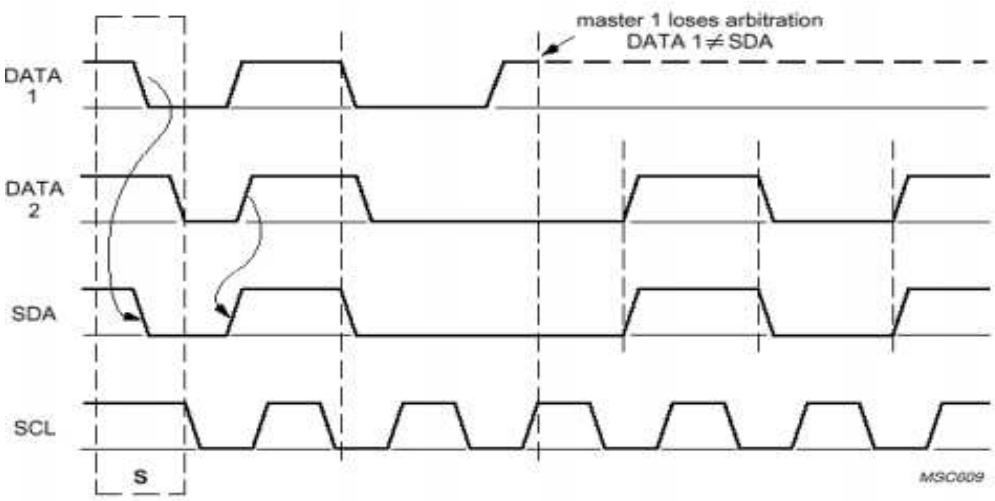
START	Slave address	1	0	Data	0	Data	1	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

(Byte Paradigm, 2013)

El I2C es considerado un protocolo algo elegante por su estabilidad. Esta estabilidad viene de que la naturaleza de los envíos mostrados en la Figura 4 garantiza que cualquier máster en el sistema recibe retroalimentación de sus envíos, y del hecho que todos los aparatos conectados en un bus I2C siempre monitorean esta línea permanentemente. El hecho que todos los aparatos en el I2C estén consistentemente observando la línea es importante porque permite el uso anterior mencionado de un start y stop bit para controlar el flujo de datos, y asegura que los componentes a los que el master no les habla no interfieren con el envío de datos. De igual forma, esta propiedad permite “arbitrar” entre masters, ya que si dos masters tratan de escribir en la línea, sólo uno tendrá precedencia ya que el otro master detectará el start bit enviado de primero y no actuará sobre la línea. En la Figura 13 puede verse la arbitración entre múltiples

masters en I2C, debido a que el master cuya línea de envío es DATA 1 no ve reflejado su envío sobre el bus compartido de SDA. (Byte Paradigm, 2013)

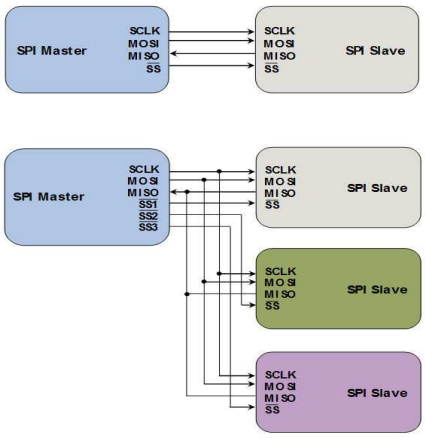
Figura 13. Arbitración entre másters



(Byte Paradigm, 2013)

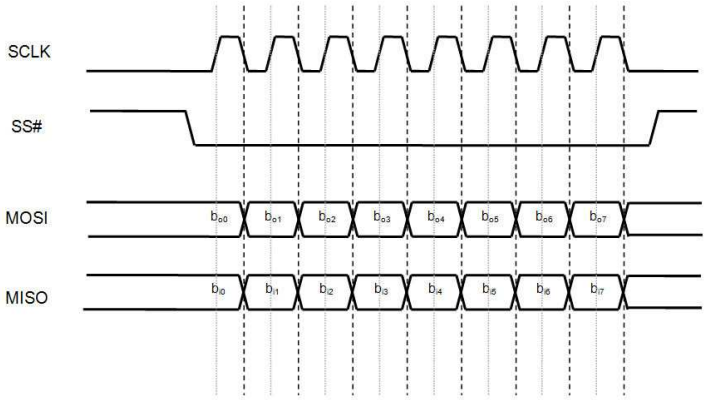
4. SPI: El protocolo de comunicación SPI fue desarrollado por Motorola en 1979. Es un protocolo full-duplex y master-slave, que utiliza 4 señales: master out slave in (MOSI), master in slave out (MISO), serial clock (SCLK), y active-low slave select (/SS). Se requiere una línea diferente de slave select por cada esclavo en el sistema, lo cual hace este sistema un poco aparatoso. En la Figura 14 puede verse esta característica. (Patrick, John, 2002)

Figura 14. Configuración de SPI para un slave y múltiples slaves



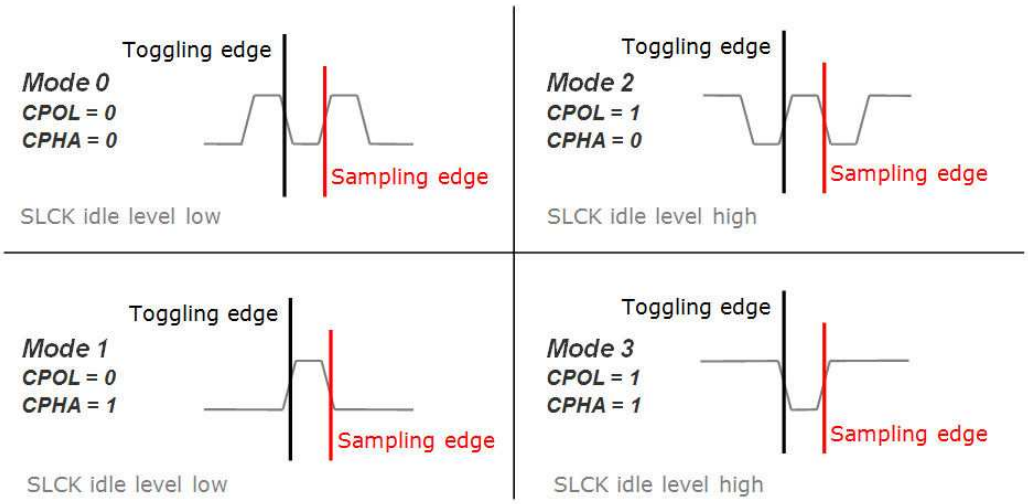
(Byte Paradigm, 2013)

Figura 15. Envío de datos en SPI en Modo 0



(Byte Paradigm, 2013)

Figura 16. Modos de comunicación en SPI



(Byte Paradigm, 2013)

Cuando el máster desea enviar información a un slave en particular, se baja la línea SS de ese slave y se activa el reloj a una velocidad mutuamente aceptable para el master y el slave. El máster genera información en la línea MOSI, y recibe información en la línea MISO. Existen cuatro modos de comunicación (Modo 0, 1, 2, y 3) disponibles en SPI, que definen en qué esquina/tramo de SCLK se alterna la línea MOSI, en qué tramo de la SCLK muestrea el master, y el nivel “idle” del SCLK – el tramo high o low del reloj donde el bus no se encuentra

comunicándose. La frecuencia de SCLK, y los valores CPOL y CPHA (que fijan las esquinas de muestreo y envío de datos en el SCLK para MOSI y MISO) deben ser iguales entre el máster y el slave. En la Figura 15 se pueden ver las señales enviadas en un envío convencional en Modo 0, mientras que la Figura 16 muestra los cuatro diferentes modos de operación y las esquinas del toggle para envío y recibir. El SPI no define una velocidad máxima de transmisión de datos y tampoco usa acknowledges, por lo cual no presenta control de flujo. Es un protocolo de muy alta velocidad y comúnmente utilizado en la industria.

Cuadro 2. Comparación entre parámetros de operación de diferentes protocolos

Name	Sync /Async	Type	Duplex	Max devices	Max speed (Kbps)	Max distance (Kbps)	Pin count(1)
RS-232	async	peer	full	2	20(2)	30(3)	2(4)
RS-422	async	multi-drop	half	10(5)	10,000	4,000	1(6)
RS-485	async	multi-point	half	32(5)	10,000	4,000	2
I^2C	sync	multi-master	half	-7	3,400	<>	2
SPI	sync	multi-master	full	-7	>1,000	<>	3+1(8)
Microwire	sync	master/slave	full	-7	>625	<>	3+1(8)
1-Wire	async	master/slave	half	-7	16	1,000	1s
Notes							
-1	Not including ground.						
-2	Faster speeds available but not specified.						
-3	Dependent on capacitance of the wiring.						
-4	Software handshaking. Hardware handshaking requires additional pins.						
-5	Device count given in unit loads (UL). More devices are possible if fractional-UL receiv						
-6	Unidirectional communication only. Additional pins needed for each bidirectional commu						
-7	Limitation based on bus capacitance and bit rate.						
-8	Additional pins needed for every slave if slave count is more than one.						

(Patrick, John, 2002)

Existen muchísimos protocolos de comunicación que pueden utilizarse. Por lo general, en el caso de comunicaciones a bajas velocidades, es común encontrarse con RS-232, RS-422, RS-485, I2C, y SPI, ya que muchos microcontroladores tienen programación dedicada a ellos.

C. Comunicación serial y protocolo I2C en Arduino

El Arduino UNO se caracteriza por su simplicidad de código, la amplia variedad de documentación que posee por utilizar software open-source, y por el hecho que es un microcontrolador con un precio accesible en el mercado de Guatemala (menos de 300Q). El hecho que los creadores del Arduino implementaron una gran cantidad de funciones simplificadas para muchísimas diferentes aplicaciones ha contribuido a la popularidad de estos microcontroladores, ya que son muy accesibles. (What is Arduino?, 2013) La librería Serial en el Arduino UNO incluye todos los comandos relevantes al envío por los puertos seriales del Arduino hacia una computadora u otros dispositivos. Esta librería utiliza los pines digitales 0 (RX) y 1 (TX), al igual que una conexión USB para el envío a una computadora. (Serial, 2013) Los comandos más relevantes que se utilizan para programar envíos seriales en Arduino son:

Serial.begin(): Establece la velocidad en baudios a la que opera la trasmisión serial de la placa del Arduino. Utiliza valores de 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200 como Baud rates estándar. (Begin(), 2013)

Serial.end(): Desactiva el Puerto serial de la placa Arduino y permite usar los pines RX y TX como salidas digitales. (End(), 2013)

Serial.available(): Devuelve el número de bytes disponibles en el buffer serial del Arduino para lectura. Este buffer tiene una capacidad máxima de 128 bytes. (Available(),2013)

Serial.read(): Esta función lee los datos disponibles en el buffer serial de la placa Arduino. Devuelve un valor de -1 si no hay datos disponibles. (Read(), 2013)

Serial.print(): Esta función imprime datos al puerto serie como caracteres ASCII para cada dígito involucrado. Los valores tipo “float” son impresos con dos decimales en un envío estándar, los bytes se envían como un solo carácter, y cualquier string se envía en su estado original. SI se desea cambiar el número de puntos decimales impresos para un valor de punto flotante, existe un parámetro adicional que se puede utilizar para especificar el número de cifras decimales a imprimir. Serial.print(flotante) imprimiría “1.23” si flotante es igual a 1.234 en el puerto serial, por ejemplo, y Serial.print(flotante,3) imprimiría “1.234” al puerto serial. (Print(), 2013)

Serial.write(): Esta función imprime su parámetro de envío al puerto serial como un byte o una serie de bytes, dependiendo del tipo de valor, sin excepciones. (Write(), 2013)

La comunicación en I2C se implementa en el Arduino utilizando la librería Wire. El protocolo I2C utiliza modos de direccionamiento de 7 u 8 bits en la industria – en el caso del Arduino, sólo se utilizan 7 bits, lo cual es importante considerar ya que direcciones de ocho bits se truncan y producen errores. (Librería Wire, 2013) Los comandos relevantes al envío en I2C en Arduino son:

Wire.begin(): Inicia la librería Wire y configura al bus I2C como maestro o esclavo. Esta configuración se realiza especificando una dirección de 7 bits en el parámetro de ingreso de la función. Si esta dirección no se especifica, se configura el bus como un bus de maestro. (Wire.begin(), 2013)

Wire.requestFrom(address,quantity): Solicita al dispositivo cuya dirección se ingresa en "address" la cantidad de bytes especificada en "quantity". (Wire.requestFrom, 2013)

Wire.beginTransmission(address): Fija el bus I2C para iniciar comunicación con un esclavo cuya dirección es especificada en "address". (Wire.beginTransmission, 2013)

Wire.send(valor, número): Envía datos desde un esclavo siempre que se obtenga una petición de un maestro, o prepara datos de envío de un maestro hacia un slave. Se puede enviar un valor en bytes, un string, o una cantidad de bytes en un vector de datos, si se especifica con el segundo parámetro opcional el número de bytes del vector de datos que se desean enviar. (Wire.endTransmission, 2013)

Wire.endTransmission(): Finaliza el envío de datos en I2C y, en el caso del master, ejecuta el envío de datos hacia el slave al que se le desea hablar. (Wire.send(value), 2013)

Es importante notar que la implementación de todas estas funciones mencionadas anteriormente involucra cambios en registros y código más complicados que los resúmenes expuestos aquí. Para más información, el código fuente disponible en las librerías de Arduino debe ser estudiado. El código que consolida las funciones en I2C para el Arduino puede encontrarse en la sección de Anexos de este trabajo.

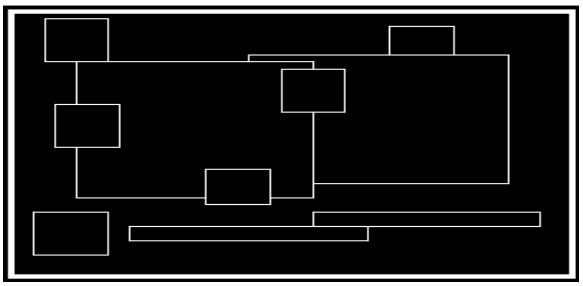
D. Diseño de Interfaces para el Usuario:

Muchísimas aplicaciones involucran el uso de interfaces con personas para relegar datos, permitir que éstas monitoreen u ordenen procesos, y otra gran infinidad de funciones. Las interfaces de datos muchas veces comprenden la gran mayoría del tamaño de un programa, ya que existen muchas consideraciones que deben tomarse en cuenta al diseñar una. Esto se debe a que un gran factor en vender un producto es su estética y facilidad de uso, los cuales no son necesariamente fáciles de considerar y trabajarse. (Martin, Suzanne, 2013)

Existen, a grandes rasgos, tres tipos de factores a tomarse en cuenta al diseñar una interfaz de usuario. Estos son el tipo de desarrollo de la interfaz, sus factores visuales, y sus factores de aceptación. El tipo de desarrollo de una interfaz se refiere a la plataforma en la que se elabora una interfaz, las herramientas y propiedades que presenta esta plataforma, y la versatilidad y personalización que permita esta plataforma. Los factores visuales de una interfaz de usuario se refieren a la representación general de la interfaz y su producto, particularmente la forma en la que conceptualizan al producto que representan. El último factor general, el factor de aceptación, incluye todo lo relacionado a la documentación de la interfaz y su mercado objetivo, y la capacidad de la interfaz para ser exitosa en este último. (Martin, Suzanne, 2013)

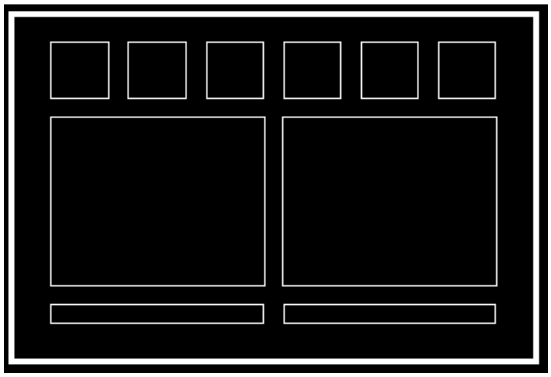
Una interfaz gráfica se centra en el uso efectivo de lenguaje visual. Es importante que los elementos en una interfaz gráfica tengan una distribución en la pantalla adecuada, que la fuente y tamaño de la letra utilizada en ésta esté bien, que colores e imágenes en la interfaz se utilicen adecuadamente, que animaciones, secuencias, y sonido se integren en una manera atractiva, y que se mantenga en la totalidad de la interfaz una identidad visual consistente. (Martin, Suzanne, 2013)

Figura 17. Distribución de elementos incorrecta



(Martin, Suzanne, 2013)

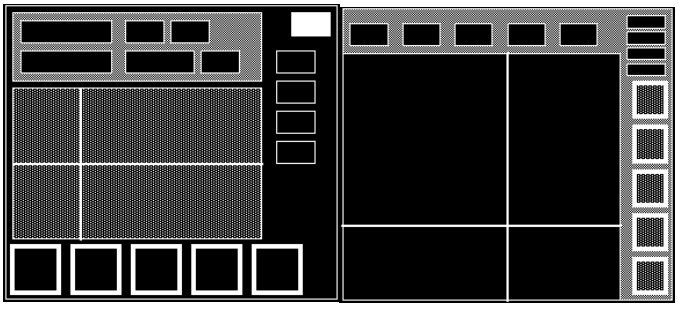
Figura 18. Distribución de elementos correcta



(Martin, Suzanne, 2013)

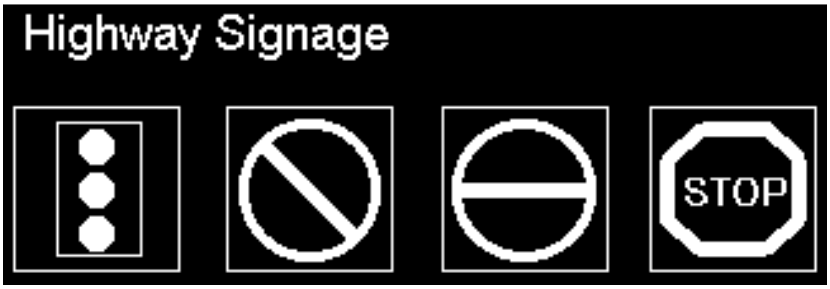
La distribución de elementos en una interfaz gráfica es muy importante. Una mala distribución de menús, gráficas, u otros elementos de interés pueden hacer que una interfaz sea poco atractiva y comprensible. En la Figura 17 puede observarse una mala distribución de elementos en una interfaz gráfica, ya que no sigue un claro patrón y puede confundir al usuario. La Figura 18 muestra una gran mejoría a la Figura 17, agrupando los elementos en una manera ordenada. Debe destacarse, sin embargo, que hay más relaciones que deben cumplirse adicionales a la agrupación por afinidad. Las relaciones visuales entre los elementos del mismo tipo deben ser consistentes: elementos del mismo tipo deben tener el mismo tamaño y estar agrupados en la misma ubicación. La Figura 19 muestra una distribución de elementos que cumple con afinidad pero aún es incorrecta en la izquierda, y una distribución que toma en cuenta el criterio mencionado anteriormente a la derecha. Es importante considerar que los elementos afines deben tener tamaños iguales ya que de lo contrario, la atención del usuario se dirige hacia los elementos más grandes o que más destacan, lo cual en muchos casos no se desea. (Martin, Suzanne, 2013)

Figura 19. Uso de relaciones visuales incorrectas (izq.) y correctas (der.)



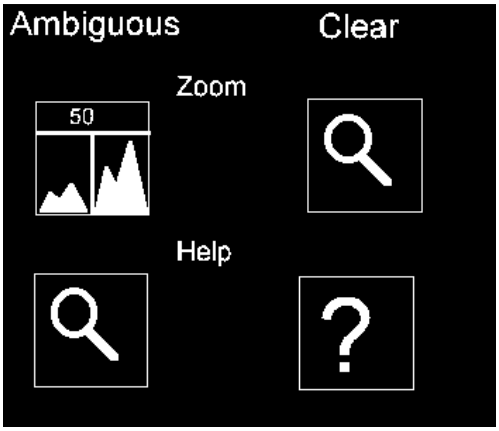
(Martin, Suzanne, 2013)

Figura 20. Algunos símbolos universales



(Martin, Suzanne, 2013)

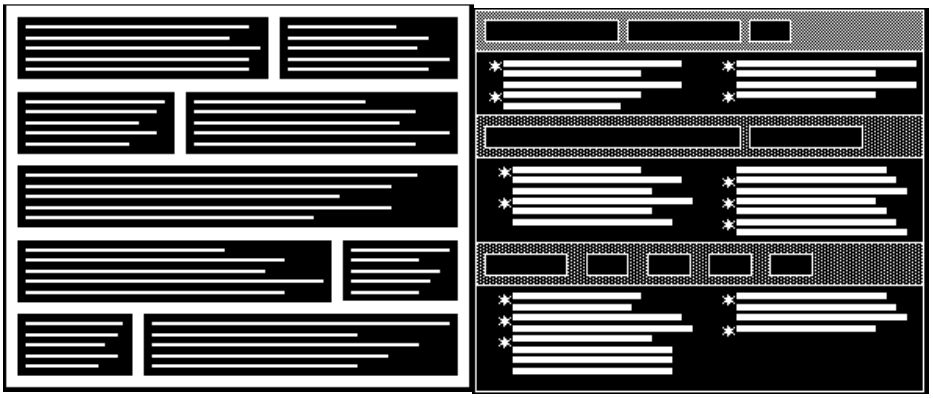
Figura 21. Formas ambiguas y claras de desplegar símbolos



(Martin, Suzanne, 2013)

En general, se busca que la interfaz gráfica agrupe los elementos de acuerdo a su afinidad y también de acuerdo a estándares culturales o la percepción del usuario. La Figura 20 muestra algunos símbolos que se mantienen iguales entre interfaces, siendo éstos símbolos extremadamente comunes en todo el mundo y entre culturas. Un símbolo de “Stop” utilizado para iniciar un proceso, por ejemplo, es confuso, poco práctico, y le resta mucha credibilidad a la calidad de una interfaz gráfica. En la Figura 21 se pueden ver dos ejemplos de cómo se deberían de utilizar Sin embargo, cabe destacarse que aunque los criterios mencionados anteriormente por lo general se cumplen, es posible romper estos estándares siempre y cuando sea muy claro que propósito sirve cada símbolo. Esto puede atribuirle cierto aire de originalidad a una GUI, de tal forma que la distinga de las demás. (Marai, Liz, 2013; Cornell University Ergonomics Web, 2013)

Figura 22. Despliegue sin prioridades de proceso (izquierda) y con prioridades (derecha)



(Martin, Suzanne, 2013)

Una vez ya se hayan establecido bien las relaciones entre elementos en la GUI, es importante dirigir la atención del usuario a elementos importantes por medio de señalización en la interfaz. El proceso de diseño de elementos busca que el usuario logre lo que desee en la forma menos confusa posible, y que involucre el menor esfuerzo de su parte posible. Para lograrse este propósito se busca mantener la información en la pantalla lo más concreta posible (pocos íconos), darle suficiente retroalimentación al usuario para que sepa en todo momento en que parte del proceso está, y mantener los menús y despliegues lo más simples posibles, creando secuencias lineales de comandos que sean simples y fáciles de comprender. En la Figura 6 se puede observar una interfaz con consistencia entre sus elementos pero ninguna relación entre elementos a la izquierda, y una interfaz con un claro patrón de relaciones a la derecha. El despliegue correcto en la Figura 22 logra establecer elementos que tienen prioridad en el proceso por medio de su uso de estímulos visuales, lo cual se desea no solo como una forma de retroalimentación sino también como una forma de mantener más sencillo el proceso para el usuario. (Marai, Liz, 2013; Cornell University Ergonomics Web, 2013)

Figura 23. Texto inadecuado y adecuado para utilizar, respectivamente

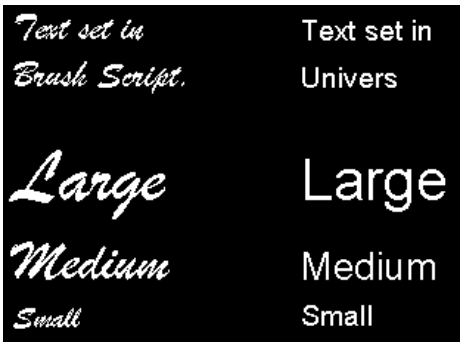


Figura 24. Uso incorrecto del color, de acuerdo a estándares culturales

Figura 25. Uso correcto de color de acuerdo a estándares culturales

Otros dos últimos elementos muy importantes en el diseño de interfaces son el tipo de texto utilizado y los colores implementados en ésta. Como se ha mencionado anteriormente, la claridad es extremadamente importante para la utilidad y éxito de una interfaz: en lo que respecta a los textos, se debe de evitar utilizar itálicas y letras que pueden ser difíciles de entender por el usuario. La Figura 23 ilustra ejemplos de fuentes y formatos que deben evitarse a la izquierda, y formatos más aceptables a la derecha. En lo que respecta al uso de colores, existen tres normas generales que se siguen para que se utilice correctamente: organización de colores, economía de colores, y comunicación de colores. El criterio de organización de colores estipula que un color consistente debe utilizarse entre elementos de la misma clase. El color siempre debe sugerir similitudes entre partes afines de la interfaz. El criterio de economía de colores, por otro lado, sugiere que el número límite de colores a utilizarse siempre supere los 3 colores pero nunca pase los 7. Adicionalmente, se sugiere, de acuerdo a este criterio, que se diseñe primero una interfaz en blanco y negro y que después se le vayan agregando colores sólo

para acentuar elementos de la interfaz, y no en una manera exagerada. El criterio final, la comunicación de colores, es el más complejo y se centra en las relaciones entre los colores y los efectos visuales que pueden tener en el usuario. El uso de rojo y verde, por ejemplo, debe de concentrarse en el centro de la visión del usuario, ya que el ojo es menos sensible en la periferia de su enfoque a notar estos colores. Otra característica importante del uso de colores es que áreas del mismo color pero diferentes tamaños dan la ilusión de que poseen diferentes tonalidades – esto se debe tomar en cuenta al utilizar color en una GUI. Es importante también evitar el uso de combinaciones de rojo con verde, azul con amarillo, verde con azul, y rojo con azul ya que pueden interferir con y cansar la vista del usuario. Se sugiere adicionalmente que para computadoras y estaciones de trabajo se utilicen fondos oscuros con textos claros, dado a que colores más claros brillan más y causan fatiga. Una consideración final muy importante que debe tomarse en cuenta es el simbolismo de diferentes colores a través del mundo. No siempre el mismo color representará el mismo sentimiento de acuerdo a donde sea utilizada la GUI diseñada. Es muy importante que se tome en cuenta esto y los estándares culturales presentes donde se utilice la interfaz: por ejemplo, debe utilizarse, de preferencia, el color verde para un botón de “ok” o “inicio” y el color rojo para un botón de “Stop”, ya que universalmente estos colores connotan esta simbología (ya que el usuario relacionaría los colores con, por ejemplo, el funcionamiento de un semáforo). En las Figuras 24 y 25 puede verse claramente un ejemplo del mal uso de estos colores y su buen uso, respectivamente. (Marai, Liz, 2013)

V. METODOLOGÍA Y ANÁLISIS DE RESULTADOS

A. Consideraciones Iniciales:

Existen, según la investigación de flujos externos, seis parámetros mínimos que deben ser obtenidos para analizar flujos efectivamente en un túnel de viento: ángulo del modelo analizado (que afecta el ángulo de ataque del flujo sobre el modelo), la fuerza de arrastre sobre este modelo, la fuerza de sustentación sobre este modelo, la velocidad ascendente sobre este modelo, y coordenadas en dos ejes (mínimo) para permitir observar cambios en líneas de flujo y la variación de la velocidad alrededor del modelo en análisis, para determinar las líneas de flujo alrededor de éste y sus capas límites.

Discutiendo con el grupo de trabajo de graduación se dividió la tarea para la instalación del túnel en cinco diferentes módulos:

1. Módulo de control para la velocidad del fluido en el túnel.

2. Módulo para colocar y permitir el cambio de grados en el modelo de estudio.

3. Módulo para la obtención de fuerzas de arrastre y sustentación.

4. Módulo para realizar medidas de velocidad ascendente en dos diferentes coordenadas.

5. Este módulo, el diseño de una interfaz de usuario para recopilar y desplegar los parámetros de estudio para el flujo en el túnel.

Ecuación 7. La ecuación de Bernoulli (Munson, Bruce R., 2013)

$$p + \frac{1}{2}\rho V^2 + \gamma z = c$$

Ecuación 8. Relación entre presión diferencial y velocidad en una línea de flujo (Munson, Bruce R., 2013)

$$p_2 - p_1 = p_d = \frac{1}{2}\rho V^2$$

Ecuación 9. Fórmula para velocidad en una línea de flujo de acuerdo a la presión diferencial y densidad del fluido utilizado (Munson, Bruce R., 2013)

$$\sqrt{\frac{2 * p_d}{\rho}} = V$$

Una vez se obtuvo una clara división de tareas para el diseño en el túnel, se comenzó a contemplar la forma de implementar la interfaz. Se determinó en el módulo para obtener la velocidad ascendente que la forma más práctica de realizar esta medición sería con un sensor de presión diferencial. Utilizando la ecuación de Bernoulli mostrada en la Ecuación 7, que es relevante para cualquier línea de flujo, se despejó la Ecuación 8 y esta se simplificó para obtener la Ecuación 9, que relaciona la presión diferencial en una línea de flujo con la velocidad de esta y la densidad del fluido en el túnel. Todas las demás mediciones y requisitos fueron investigadas por los encargados de los demás módulos, y se concluyó que los valores que se obtendrían para trabajarse en la interfaz serían las coordenadas de un sensor de presión diferencial en dos dimensiones (es decir, coordenadas en x y coordenadas en y de éste), el valor de la presión diferencial en este modelo, la fuerza de arrastre sobre el modelo y la fuerza de sustentación sobre el modelo. Adicionalmente, se decidió incluir otro sensor de presión diferencial para medir la velocidad relativamente lejos del modelo.

Los demás módulos del túnel de viento, se comprobó al poco tiempo, requerirían el uso de microcontroladores para realizar cálculos de referencia con los valores de los sensores, obteniendo retroalimentación por medio de monitores seriales en los diferentes módulos. Se decidió utilizar un protocolo de comunicación para controlar los datos en una manera más fácil de implementar y estandarizada. El análisis de los diferentes protocolos que se podrían usar se derivó en la etapa de investigación resumida en el Marco Teórico.

B. Selección del Protocolo de Comunicación Serial y lenguajes de programación:

Se buscó que el protocolo a utilizarse cumpliera con las siguientes propiedades:

1. Facilidad de uso
2. Poco cableado
3. Implementación simple
4. Precisión y control de flujo, de preferencia.

Las características anteriores redujeron las opciones para los protocolos de comunicación a I2C vs. SPI. No se consideraron protocolos multi-punto ya que en la interfaz no se necesitaría un muestreo continuo de información (el muestreo lo controlaría el usuario, ya que no se quiere gastar energía en el túnel por gusto). Adicionalmente, la velocidad del protocolo no tendría que ser alta – asumiendo una velocidad de transferencia de aproximadamente 400 kbps para el I2C o 1000 kbps para SPI, y una Baud Rate de 9600 para la Comunicación Serial, y considerando que sólo se necesitan mostrar valores a una velocidad lo suficientemente rápida para que el usuario perciba cambios en el muestreo (una vez cada cuarto de segundo, como mínimo), quedaría un amplio margen para lograr lo deseado. Aún a estas relativamente bajas velocidades, imprimir cada valor de los sensores como un punto flotante a 10 decimales en ASCII (es decir, enviando 72 bytes como mínimo) se enviaría todo muy rápido, incluso tomando en cuenta el tiempo requerido para realizar cálculos en el microcontrolador utilizado.

La ventaja de SPI sobre I2C es el hecho que permite mayores velocidades, puede ser más fácil de implementar, y que puede continuar muestreando si un slave no manda información (en I2C se traba la línea). La ventaja de I2C sobre SPI es que involucra menos cableado, el máster sí puede saber cuántos slaves hay en la línea e incluso sus direcciones sin hardware adicional, y que el I2C permite el uso de múltiples masters, con su altamente eficiente proceso de arbitración. Al final, como la velocidad de envío no era de principal importancia y se desea mayor control de flujo (aunque este viene con el precio que el bus se puede trabar), se seleccionó I2C como el protocolo que se usaría para integrar todos los sensores del túnel. [I2C vs. SPI]

En lo que respecta a la selección del microcontrolador a utilizarse, casi todos los módulos decidieron utilizar el Arduino UNO para obtener los datos de los sensores. Este microcontrolador tiene una librería de comandos dedicada a la programación en I2C al igual que muchos otros, por lo cual era una alternativa viable. Cabe destacarse que la programación en Arduino es mucho más simple que la programación en otros microcontroladores como los PIC, ya que sus desarrolladores programaron una multitud de funciones simplificadas para una gran cantidad de librerías, permitiendo así códigos bastante sencillos y con muchas menos instrucciones que otros microcontroladores. Bajo las librerías del software del Arduino se pueden encontrar los headers y funciones en C++ que corresponden a cada una de las funciones importantes a utilizarse – la documentación online del software sólo explica qué hacen los comandos relevantes en las librerías Seriales y de Wire (que corresponden a la comunicación serial con la computadora e I2C entre los slaves y el máster del túnel, respectivamente) – si en verdad se quiere saber qué está ejecutándose se debe investigar en las librerías disponibles, que son de fácil acceso. (What is Arduino?, 2013)

Al igual que en el caso de la selección de microcontroladores, existen una infinidad de programas que se pueden utilizar para crear interfaces gráficas. En el caso de este proyecto, se decidió utilizar NI Labview. Existen múltiples ventajas al utilizar un programa gráfico/dataflow como NI Labview para el desarrollo de interfaces. Para empezar, Labview hace muy fácil la creación de botones, gráficas, y controles en una interfaz, al igual que exportar los datos que se grafiquen en ella. Se pueden hacer muchos cambios estéticos e incluso profundos en poco tiempo, lo cual es extremadamente conveniente al probar la funcionalidad del túnel e ir haciendo cambios graduales. Labview también permite ejecutar cierto grado de paralelismo, lo cual puede ser conveniente si se quieren correr múltiples procesos en una interfaz al mismo tiempo que involucren las mismas variables. Otra ventaja del uso de Labview es el hecho que no se tiene que programar casi nada para, por ejemplo, configurar una trasmisión serial. El uso de bloques elimina la necesidad de programar a nivel de hardware, y permite la visualización de todo el proceso a programarse como un flujo de datos sin necesidad de escribir código (todos los cambios en parámetros se realizan modificando las propiedades de los bloques utilizados). (Advantages of Using Labview, 2013) Lo único necesario en el caso del diseño del túnel de viento es entender cómo configurar los módulos de trasmisión serial que posee el programa, y cómo realizar cálculos con los datos obtenidos por el puerto serial y el Arduino. Como punto final

se decidió fijar las direcciones a utilizar en el programa. Estas se pueden observar en el Cuadro 3.

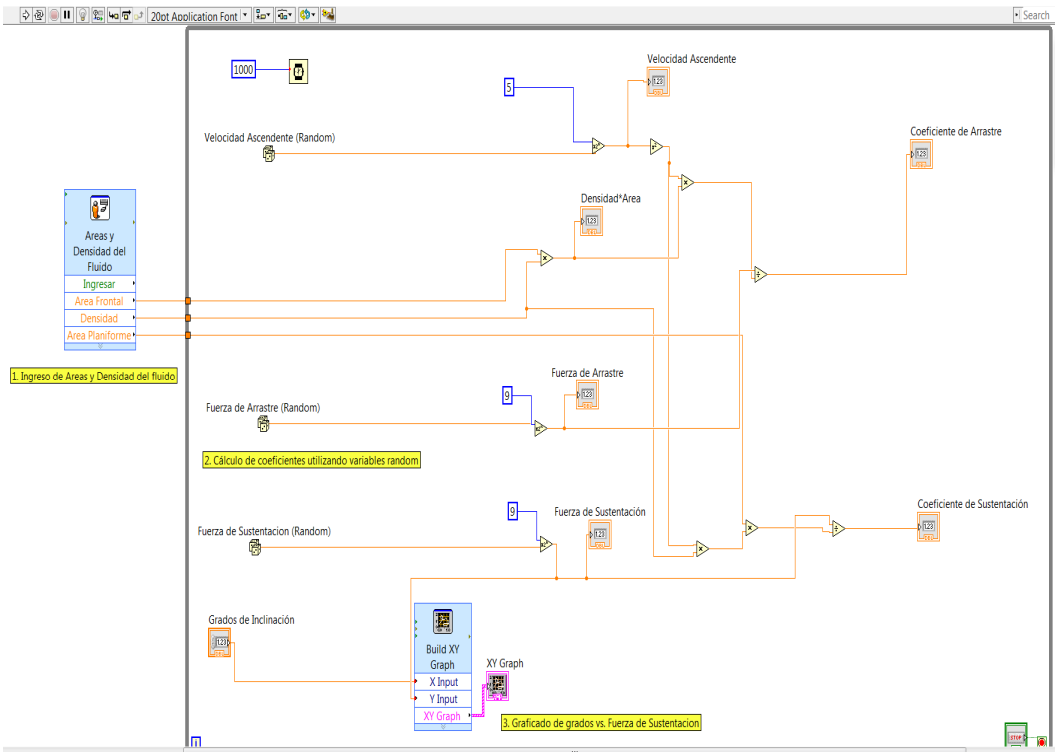
Cuadro 3. Direcciones de Envío Establecidas para la Comunicación en I2C.

módulo	Dirección
Grados	1
Fuerzas	2
Posicionamiento	3
Velocidad	4

C. Envío de Grados desde la Interfaz:

Una vez se seleccionó el Arduino UNO y Labview como el microcontrolador y programa para implementar la interfaz, respectivamente, se comenzaron a ver las funciones disponibles en Labview para realizar cálculos y despliegues. Para realizar algunos cálculos relacionados al túnel, principalmente los coeficientes adimensionales de sustentación y arrastre, se utilizó el diagrama de bloques mostrado en la Figura 26, que se programó en Labview utilizando una serie de números random que se multiplicaron por potencias de 2 para ponerlos en el orden de magnitud de valores típicos a observarse en el túnel. Como puede notarse, el programa es gráfico y no se involucra código. Todos los cálculos se realizan “on the wire”, y las variables se trasfieren de proceso en proceso mientras se va ejecutando el programa de izquierda a derecha.

Figura 26. Diagrama de bloques para la primera prueba en Labview



Para probar el uso de diálogos en Labview se programó al principio de todo el código, como puede verse en la Figura 26, un “prompt” para pedirle al usuario datos de ingreso para calcular los coeficientes. Se tomaron los valores de velocidad ascendente, fuerza de arrastre, y fuerza de sustentación como randoms porque en esta etapa del diseño todavía no se habían probado los sensores de los otros módulos e incluso no habían llegado algunos del extranjero. En la Figura 27 puede observarse el pánel frontal al ejecutarse el programa. Como se discutió anteriormente, el primer comando en ejecutarse a nivel de bloques fue el prompt, como esta figura evidencia. Posteriormente, una vez el usuario ingresó datos, se pasaba a un while loop indefinido donde cada segundo se generaban nuevos valores aleatorios para las variables random utilizadas en el problema, y se actualizaban los valores calculados para los coeficientes. El delay de un segundo entre iteraciones del ciclo while del programa, que en el caso de Labview es una caja con un marco sólido gris, se puede ver en la esquina superior del ciclo while en la Figura 26, como un símbolo de un reloj con un valor de 1000 (1000 milisegundos) cableado. En las Figuras 28 y 29 puede verse dos iteraciones diferentes del ciclo while, con los cambios respectivos en los indicadores de las variables calculadas e ingresadas, que se encuentran

directamente debajo de la gráfica. De acuerdo a los resultados obtenidos, en esta etapa del proceso se verificó que Labview, en efecto, si podía realizar los cálculos deseados para los coeficientes de arrastre y sustentación siempre que el ingreso numérico a las funciones de multiplicación y división fuera válido. Asimismo, otro resultado que se pudo observar fue que el uso de graficación en tiempo real no es fácil de implementar en Labview y que debería encontrarse una alternativa más adelante para poder desplegar bien las gráficas. Esta última característica puede observarse al comparar las Figuras 28 y 29, que sólo grafican un punto diferente cada una pero refrescan el punto graficado cada iteración del while loop. Este problema no se arregló inmediatamente pero se notó por su importancia.

Figura 27. Prompt para el primer programa de prueba

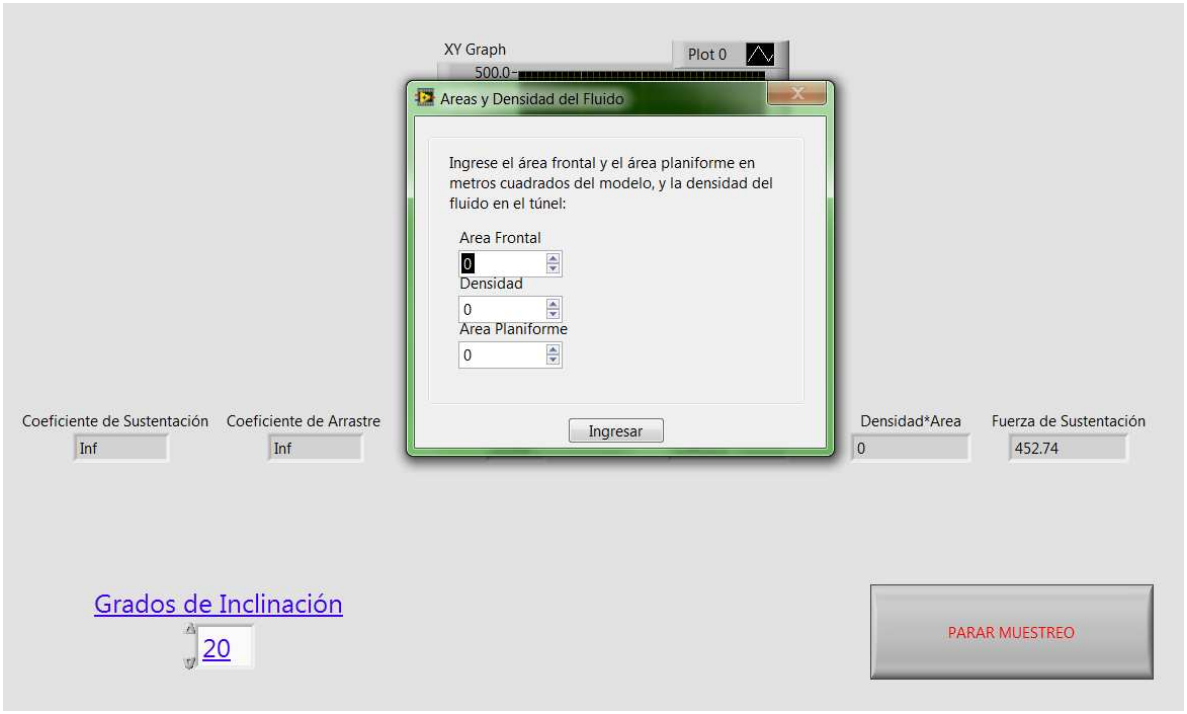


Figura 28. Cierre de prompt y comienzo del ciclo while en el primer programa de prueba

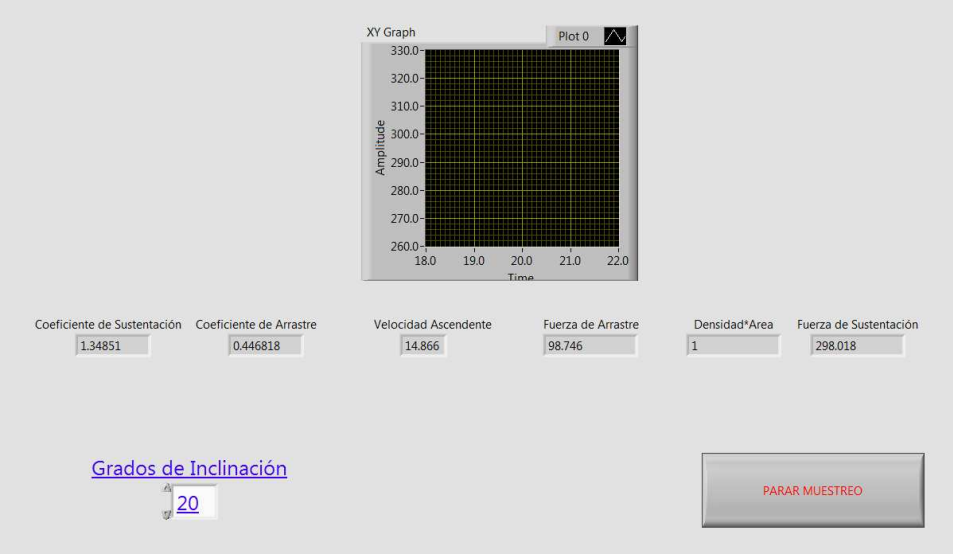
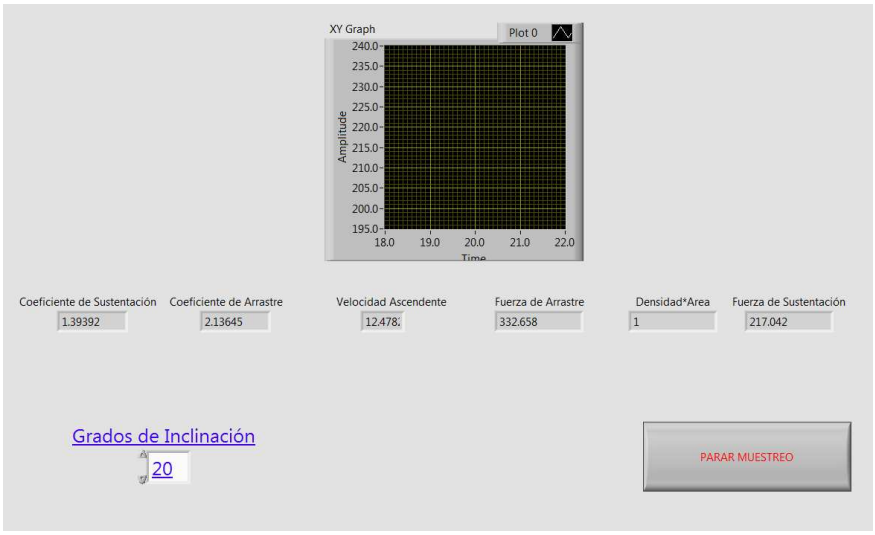


Figura 29. Cambio de valores en el ciclo while indefinido en el primer programa de prueba

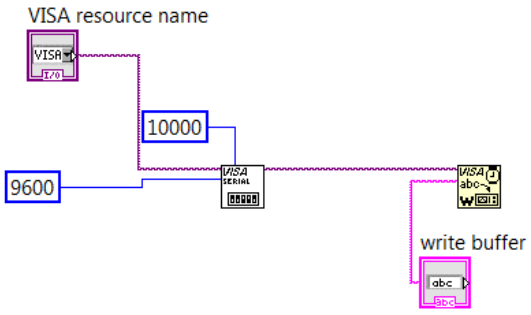


Después de las pruebas iniciales con el programa inicial de pruebas en Labview, el módulo de grados ya había mostrado progreso por lo que se comenzó a explorar cómo se enviarían valores a éste. Se comenzó investigando sobre la comunicación en I2C en Arduino. Se encontró un ejemplo online para la programación de un master, en la página de Arduino, y de allí se comenzó a familiarizarse con la librería Wire, que ya de por sí es muy simple de utilizar.

(Master Writer/Slave Receiver, 2013) El primer código de prueba puede verse en la sección de Anexos de este trabajo. Se especificó por parte del módulo de grados que se requería el envío de un byte con un rango de valores entre 80 y 124. Inicialmente, se programó una variable con un valor fijo de envío y se fue comprobando la capacidad del Arduino para enviar este byte.

No se presentó ningún problema al utilizar sólo el Arduino para el envío de datos. Sin embargo, ya que el control del túnel y el modelo deberían hacerse desde la interfaz, se comenzó a investigar qué funcionalidad presentaba Labview para trasmitir datos serialmente. Se averiguó que Labview cuenta con un módulo llamado VISA que se encarga de esta funcionalidad. En específico, el bloque VISA Serial Write fue el bloque que se comenzó a utilizar, al igual que el bloque de configuración de VISA. En la Figura 30 pueden verse ambos bloques. El primer bloque en la esquemática es el bloque de configuración de VISA. Como puede verse en la imagen, se configuró una Baud Rate de 9600 y un timeout de 10000 milisegundos en el caso de no enviar nada por el puerto serial. El bloque a la derecha es el VISA Serial Write, y es muy simple de utilizar. Lo único que se le tiene que cablear es la configuración del puerto serial anterior, y un buffer de escritura compuesto por un carácter o un string, siendo este un constante o ingresado por medio de un control, como en este caso.

Figura 30. Envío serial básico



Para probar el control de los envíos en el Arduino desde Labview se decidió utilizar el Serial Write para enviar caracteres de acuerdo al comando deseado de implementarse. En el

caso del envío de grados, se decidió enviar el carácter “2” en ASCII como un indicador al Arduino de que se deseaba enviar grados al módulo de grados del túnel. Se investigó la librería serial del Arduino, principalmente la funcionalidad de Serial.available(), y se probó ejecutar el código que ya se había probado y verificado para enviar grados, esta vez recibiendo un carácter por Labview de primero y posteriormente entrando a un if statement verificando que el carácter ingresado fuera “2” para ejecutar el envío del byte de grados fijado con anterioridad en el Arduino. Se probó el código modificado de esta manera y funcionó. Este sería un paso muy importante para la implementación del proyecto, ya que todos los envíos de datos posteriores a diferentes módulos se realizarían de la misma manera, cambiando sólo el carácter de envío. De la misma manera, la programación en el Arduino sólo cambiaría poco de acuerdo a lo requerido por el envío. Es importante notar que el comando recibido por el puerto serial fue fijado como una variable tipo char en el Arduino – el manejo de tipos de datos correcto fue importante en esta implementación, y llegaría a presentar muchos problemas cada vez que se obvió. El segundo código de prueba que se utilizó también puede observarse en la sección de Anexos de este proyecto.

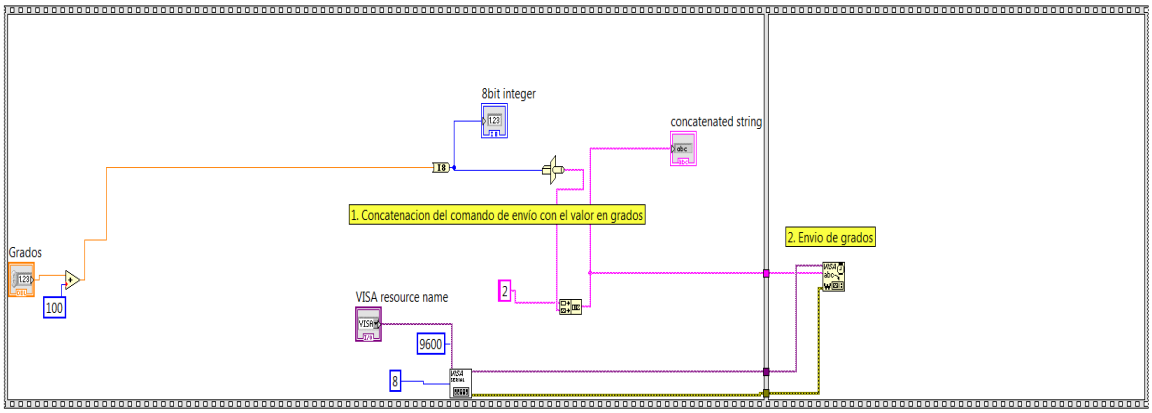
Es importante destacar que al mismo tiempo que se investigó la comunicación serial en Labview también se comenzó a analizar cómo se iba a implementar el envío de grados por parte del usuario, eliminando el valor constante de prueba que originalmente se utilizó en el Arduino. Al principio, se hicieron algunos programas de prueba con un indicador numérico que podía subirse y bajarse arbitrariamente cuando el usuario quisiera. Se investigó como podía realizarse el envío de datos desde Labview (enviando también el comando “2” en ASCII para señalizarle al Arduino el deseo de enviar datos) y se descubrió la estructura de eventos en Labview. La estructura de eventos en Labview permite, en resumidas cuentas, ejecutar series de comandos de acuerdo a “event cases” o casos de eventos. Un event case en Labview puede ser cualquier cosa desde apachar un botón hasta un cambio de valor en una variable o incluso pasar el mouse encima de cierta área del panel frontal. El uso de event cases terminaría siendo una parte integral de la interfaz, por lo cual esta etapa del proceso fue importante.

Inicialmente, se trataron de ejecutar casos de eventos de acuerdo a cambios de valor en el indicador de grados que puede observarse en las Figuras 27, 28, y 29. Sin embargo, rápidamente salió a luz que estos cambios podían forzar al sistema de envío de grados ya que el usuario podría continuamente cambiar valores de grados sin esperar a que se estabilizara la base del modelo, lo cual podría haber dañado al sistema. Esto podría haberse arreglado

poniendo un retraso de tiempo significativo después de cada cambio de valores en el indicador de grados (5 segundos de retraso de tiempo después de cada cambio de valor, por ejemplo), pero es importante notar que esto habría deshabilitado el panel frontal hasta que se parase de ejecutar este retraso de tiempo, una situación muy indeseable. Se decidió dejar el indicador de valor númerico de grados en el panel frontal, agregando un botón de envío para que el usuario enviase el valor de grados al módulo de ángulo cuando ingresase el número deseado en el indicador. El botón de envío fue el que se ligó a la siguiente versión de prueba para el módulo de ángulo del modelo, esta vez utilizando el Arduino sólo como el medio de transferencia de información (es decir, sin utilizar el valor fijo de prueba que se implementó anteriormente).

El módulo de ángulo, como se discutió anteriormente, se programó para operar en un rango de valor de byte entre 80 y 124, siendo este rango directamente correspondiente con valores de ángulo de -20 y 24 grados, respectivamente. Lo primero que se hizo con el programa original mostrado en las Figuras 27, 28, y 29 fue fijar el rango de valores a ingresar a este intervalo de -20 a 24 grados. Posteriormente, se agregó un botón de envío, el cual se ligó a una estructura de eventos. Inmediatamente después de programar esto, se tomó la salida del valor de indicador númerico de grados, se le sumó 100 (para obtener un valor númerico entre 80 y 124) y este valor pasó por un bloque de conversión a U8 (un bloque para convertir números a ints de un byte) y un type cast (para convertir este byte en un string). Posteriormente se concatenó un carácter constante de “2” al carácter ASCII (con un valor decimal en el rango de 80 a 124) obtenido por todas las conversiones anteriormente mencionadas, y esto se le alimentó a la función de VISA Serial Write. Se le agregaron indicadores a cada una de las salidas de operaciones en este proceso para verificar que, en efecto, el resultado final de todas las conversiones y la concatenación siempre sería un string de 2 bytes iniciando con el carácter “2”. En la Figura 31 se puede ver todo el proceso anteriormente descrito, comenzando con la suma de 100 al valor en el indicador numérico y terminando con un VISA Serial Write con un indicador de “return count” para verificar que siempre se estén escribiendo sólo 2 bytes.

Figura 31. Envío de grados finalizado



El programa de Labview mostrado en la Figura 6 se integró para finalizar la prueba con una versión modificada del programa en Arduino probado de último, que ya podía leer el comando enviado desde Labview utilizando la función de Serial.read(), y también ya podía escribir el valor fijo de prueba utilizando Wire.write(). El único cambio que se realizó en el tercer programa de prueba, que puede verse en la sección de anexos, fue que se agregó un case statement al código para poder recibir e enviar el valor numérico concatenado en Labview, y enviado desde la interfaz. La Figura 32 muestra el diagrama de flujo del programa en Arduino. En la Figura 33 puede observarse una toma de pantalla del pánel frontal del envío de grados como se observó en Labview.

Figura 32. Diagrama de flujo para el envío de grados desde Labview

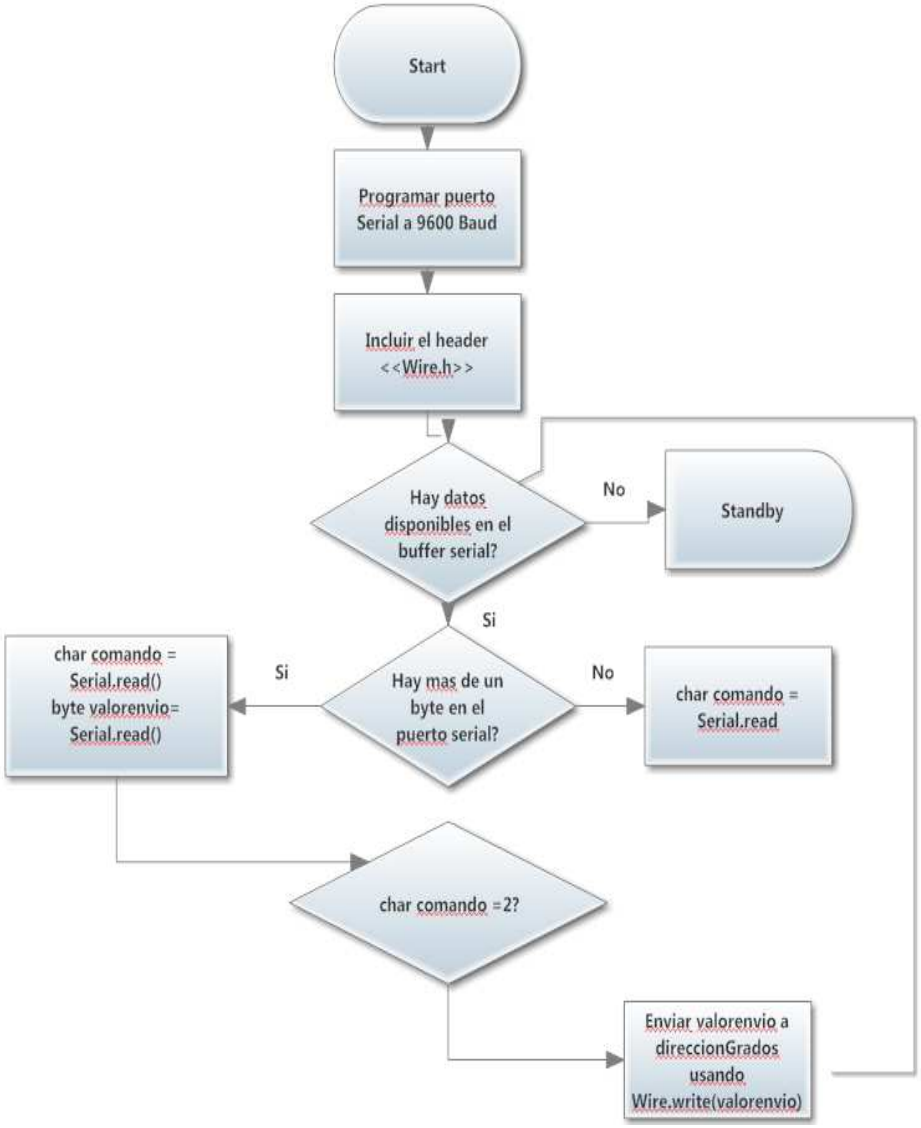
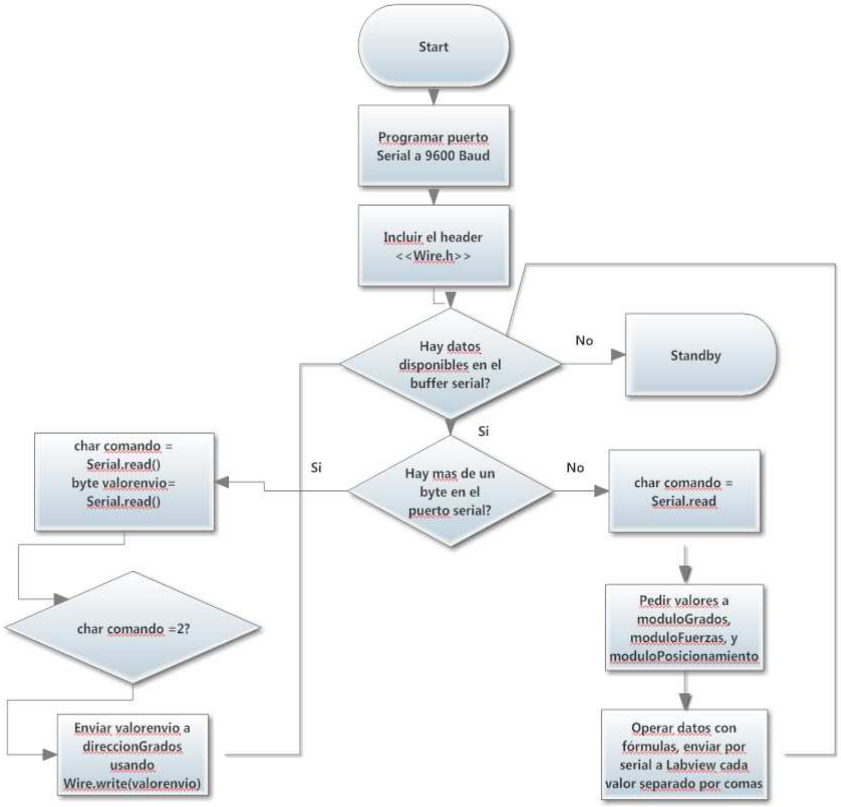


Figura 33. Botón de envío de datos en Labview

D. Recepción de datos general:

Casi inmediatamente después de la finalización del programa de grados se comenzó a implementar la recepción de datos de cada uno de los sensores, cuyos módulos ya casi estaban finalizados. Se discutió con los módulos de fuerza de arrastre, fuerza de sustentación, y presiones diferenciales que se requeriría para desplegar exactamente los valores de los sensores, y se obtuvo una serie de fórmulas para operar los datos de envío de cada sensor. El código en el Arduino se modificó para poder recibir dos comandos - un comando de recepción de datos correspondiente a todos los datos del sensor, y un comando para el envío de datos, correspondiente sólo al módulo de grados. Los comandos utilizados en ASCII para ambos casos fueron “1” y “2”, respectivamente. En la Figura 34 puede verse el diagrama de flujo para este código de prueba.

Figura 34. Diagrama de flujo tentativo para envío de grados y recepción de información



En lo que respecta a Labview, se utiliza el bloque de VISA Serial Read para recibir información por el puerto serial. Esta se cablea muy similarmente a VISA Serial Write, con una característica particular importantísima para el envío de datos: se debe especificar la cantidad de bytes a leer como un parámetro de configuración de este bloque. Esta característica resultó siendo extremadamente importante para el desarrollo del código en el Arduino ya que requirió que se conociera muy bien que clase de envío se realizaría desde el Arduino, y, como consecuencia, cuantos bytes terminarían trasmitiéndose serialmente. Inicialmente, se consideró recibir todos los valores por I2C y después enviarlos sin procesar a Labview para sus respectivas conversiones. Sin embargo, rápidamente se comprobó que los cambios requeridos (mapeos, conversiones de strings a bytes y después de bytes a valores numéricos) tomarían bastante tiempo de implementar correctamente por el hecho que Visa Serial Read siempre regresa strings en su buffer de salida, sin excepciones, lo cual volvía la manipulación de datos en Labview un proceso engorroso. Aunque en la industria casi siempre se busca que se envíe todo con el

menor número de bytes posibles (en el caso de este proyecto, el número mínimo en un envío sería de seis bytes, uno por cada valor de medición en el sistema), se decidió realizar toda la programación de mapeos y fórmulas en el Arduino para evitarse el mayor número de conversiones posibles en Labview, aunque involucrase imprimir cada valor de los sensores carácter por carácter. Esta decisión fue posible de implementar en el caso de este proyecto por el hecho que la velocidad de muestreo del despliegue de la interfaz (un máximo de 4 veces por segundo) dio un muy amplio margen para el número de bytes que se podía enviar, considerando los altos valores de Baud Rate (9600) y velocidad de trasmición de I2C (400 kbps), y el hecho que aún imprimiendo todos los valores carácter por carácter como números de punto flotante se obtendrían envíos mucho menores a los límites establecidos individualmente por cada método de comunicación. Una gran ventaja del envío de todos los valores carácter por carácter ASCII desde Arduino, debe destacarse, fue el hecho que ya de entrada al programa de Labview se podían ver los valores de los sensores, aunque fueran un array de caracteres en un string por el método de envío, en vez de una representación numérica en sí. Mientras se consideraba el método de envío, finalmente también se contempló establecer alguna forma de control para lograr diferenciar fácilmente en el buffer del VISA Serial Read los diferentes números siendo enviados. Se decidió utilizar el común procedimiento de separar los valores de envío por comas (Comma Separated Values) y después, en Labview, separar cada valor y modificarlo apropiadamente. Se contempló e implementó tambíen el uso de un byte de inicio para comprobar que siempre se enviase todo en el orden correcto, y corregirlo de no ser necesario.

Hubo un factor crítico adicional que se solucionó también en esta etapa importante del proceso de programación. En la prueba numérica inicial al principio de todo el proceso de programación (ver Figura 1) se había verificado que no se podía graficar más de un punto en tiempo real. Este era un serio problema ya que las gráficas son importantees para el análisis de fluidos, y eran una parte crítica del producto final que se buscaba al crear la interfaz. Para lograr solucionar este problema se investigaron más las propiedades de las gráficas en Labview. Se descubrió que el graficado en Labview se puede realizar utilizando arrays de valores, siempre que estos tengan el mismo índice. Adicionalmente a esto, se comprobó que un array se puede obtener en Labview fácilmente trazando cables que contengan valores al margen exterior de un for loop, siendo el array creado y modificado por cada iteración del for loop, y conteniendo el mismo índice que el for loop. Se decidió aprovechar esta característica en un event case

manejado por un envío de tiempo en minutos, para guardar cada valor de interés en el túnel en un array diferente, y guardarlo al final del proceso de muestreo.

Un ajuste final que fue importante en el código del Arduino fue la forma en la que se programó el envío de valores de punto flotante. Algunos valores presentaron variación en su cantidad de bytes, ya que habían ocasiones en las que presentaban valores con órdenes de magnitud mayores o menores de acuerdo a las fórmulas empleadas. Para ilustrar esto se pueden comparar, por ejemplo, los números 1.234 y 10.234. El envío de caracteres ASCII en puerto serial del segundo número involucraría un byte adicional. Considerando que, como se discutió anteriormente, VISA Serial Read en Labview requiere un buffer fijo de número de bytes en la entrada, este problema debió corregirse en el Arduino para que no ocurriera pérdida de información. Para hacer esto, se dividió cada valor por su orden de magnitud máximo de medición, y se enviaron estos valores modificados a mayor precisión decimal por el puerto serial. En el caso de la presión diferencial alrededor del modelo, por ejemplo, el valor máximo de medición superó en las pruebas del módulo los 1000 Bar – para enviar este dato correctamente por el puerto serial, entonces, se dividió el valor saliente de la fórmula proporcionada por 1000, para garantizar un número de envío con, a lo sumo, valores en unidades (y ya sin valores en decenas, centenas, y millares).

En resumen, el código en Arduino implementado presentó:

1. Control del orden de envío utilizando un start byte con el valor hex 0xFF (ya que este valor no se utilizaba por ningún sensor en sus envíos),
2. El uso de valores separados con comas en el envío para poder distinguir en el buffer de entrada en Labview entre los diferentes valores,
3. División en orden de magnitud máxima experimental para permitir envíos siempre constantes en número de bytes,
4. Y el envío de los datos carácter por carácter utilizando Serial.print(), para permitir observar los valores reales directamente en el buffer de entrada de VISA Serial Read.
5. Deshabilitar la comunicación en el Arduino hasta recibir un nuevo comando.

Los cambios en el código pueden consultarse en la sección de Anexos, para la prueba 3.

Es importante notar que al principio no se obtuvo sincronía con el Arduino. Esto se debió a que el Arduino estaba trasmitiendo continuamente ya que no se fijó el valor del comando en el Arduino a cero después de la recepción de datos. Este error fue pequeño pero importante porque es enteramente posible, si no se asegura que sólo un envío ocurre cuando se le pide al Arduino, que se comience a llenar el buffer y los datos no entren en el orden correcto.

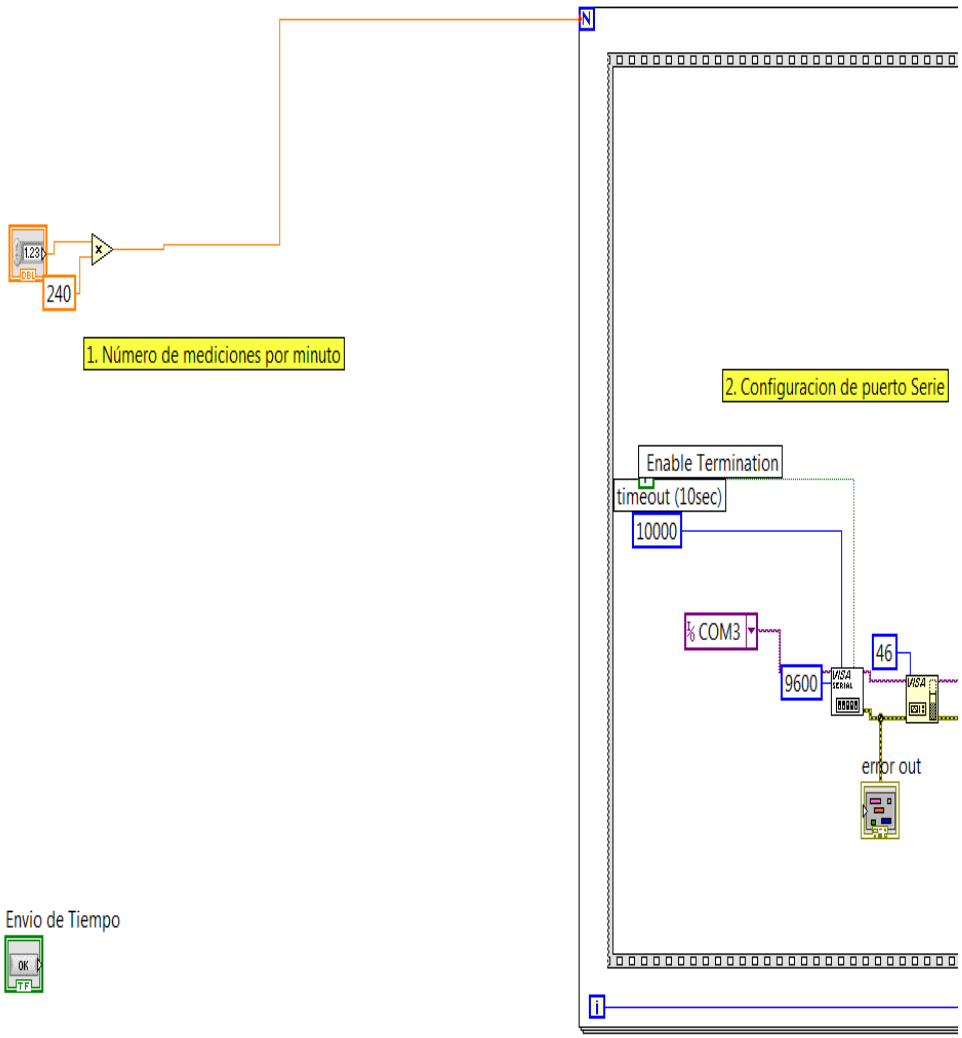
En resumen, para cumplir con lo requerido en Labview, se tuvo que:

1. Cablear el número de mediciones por minuto al for loop,
2. Configurar VISA con una Baud Rate de 9600,
3. Enviar un comando “1” para establecer el muestreo con el Arduino,
4. Configurar VISA Read apropiadamente (# de bytes adecuado)
5. Verificar el byte de inicio del buffer de lectura de VISA read como OxFF,
6. Eliminar las comas en el string de entrada para poder distinguir entre los valores de envío, separando cada valor y guardandolo en su array respectivo,
7. Y fijar un delay en microsegundos igual al tiempo entre muestras.

Las Figuras 35, 36, 37, y 38 resumen los métodos y pasos discutidos en los párrafos anteriores. En la Figura 9 puede observarse el número de iteraciones del loop de la prueba 3 y la configuración de VISA para el puerto serial. Las iteraciones fueron configuradas de tal forma que el ingreso de minutos de muestreo por el usuario podía variar de 1 a 10 minutos, y se toman cuatro muestras por minuto. En el código de la Figura 36 se programó el envío del comando “1” ASCII de muestreo al Arduino, al igual que el número total de bytes requeridos, de acuerdo a los cálculos realizados con la ayuda del Cuadro 4. Como se puede notar al comparar los dos, el valor calculado y el valor ingresado en Labview fueron los mismos, ya que tenían que serlo. En la Figura 37 puede verse el el uso del bloque “Match pattern” para separar cada valor de las comas en el envío. En el programa, se utilizó el Match Pattern junto con un shift register para separar el string original proveniente del Arduino en sus valores numéricos, los cuales se guardaron en un array. Éste array después se convirtió a un array de valores double para operarse en fórmulas utilizando Fraction/Exponiential to Double, un bloque de Labview que permite la conversión de strings a valores numéricos, como denota el paso 7. Finalmente, cada término individual se sacó del array para tenerlo en arrays individuales, utilizando el bloque de Index Array. El

funcionamiento de todos estos bloques está explicado en el “help” de Labview y en el Marco Teórico de este trabajo. En la Figura 38 se puede observar, para finalizar el análisis del código, el valor de retraso en tiempo fijado por este circuito para establecer el tiempo entre muestras. El código mostrado en las Figuras 35, 36, 37, y 38 se programó para muestrear una vez cada 250 milisegundos, o 4 veces por segundo aproximadamente, se puede ver en el indicador de delay en la Figura 38.

Figura 35. Pasos 1 y 2 del muestreo de sensores



Cuadro 4. Comandos para el envío de velocidades

Resumen de envio desde Arduino		
Valor	Rango de valores	# bytes
Start	XXXXXXXXXXXXXX	1
Presión diferencial (psi)	0-1	6
Coma	XXXXXXXXXXXXXX	1
Fuerza de sustentación	0-10	6
Coma	XXXXXXXXXXXXXX	1
Fuerza de arrastre	0-10	6
Coma	XXXXXXXXXXXXXX	1
Posicion en Y	0-100	6
Coma	XXXXXXXXXXXXXX	1
Posicion en X	0-100	6
Coma	XXXXXXXXXXXXXX	1
Presion Diferencial (Bar)	-1400	9
Suma total de Bytes		45

Figura 36. Pasos 3, 4, y 5 del muestreo de sensores.

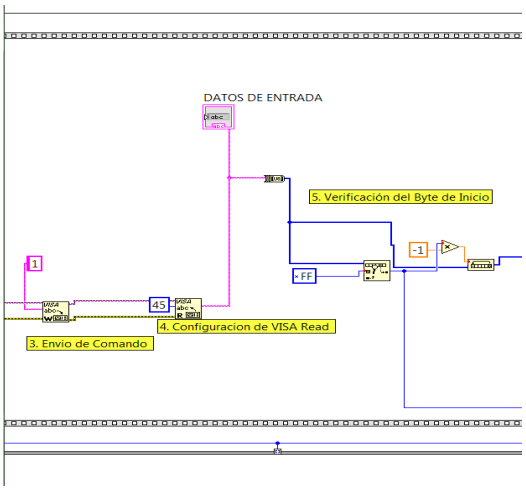


Figura 37. Pasos 6 y 7 del muestreo de sensores

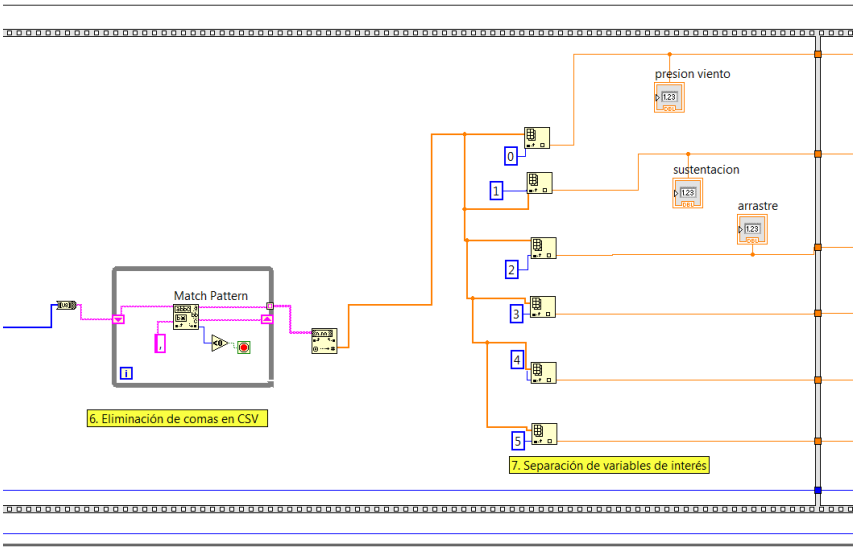
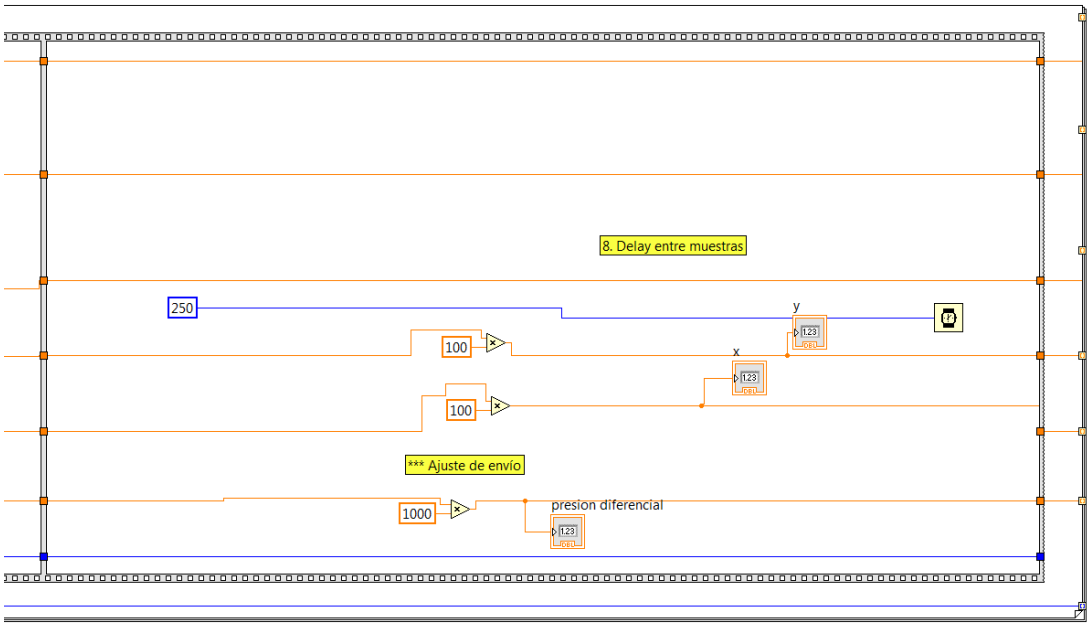


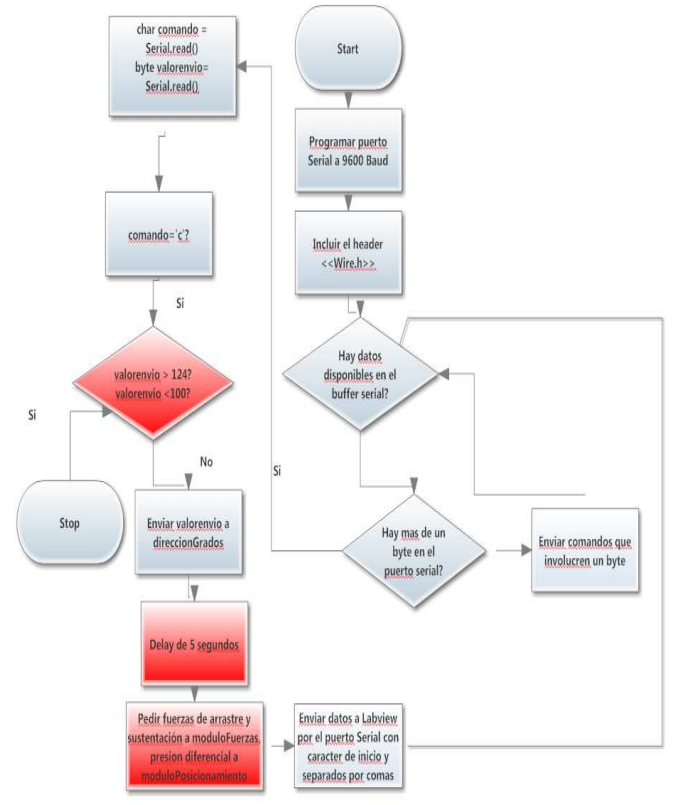
Figura 38. Paso 8 del muestreo de sensores, ajustando los valores de envío del Arduino a sus valores reales en Labview



E. Prueba de Barrido de Grados:

La tercera prueba exitosa explicada anteriormente logró obtener todos los valores de los sensores eficientemente y en una manera confiable. Sin embargo, lo que en verdad importa en el análisis de flujo externo, como se investigó, es la relación entre estos parámetros. Con esto en mente, se programó una modalidad de barrido de ángulo en el túnel de viento para permitir obtener gráficas de fuerza de arrastre y sustentación versus grados del modelo, y comparar éstas con sus equivalentes teóricos. Al referirse a un barrido de grados se está hablando de una prueba de mediciones de fuerza de arrastre y sustentación para un rango de 0 a 24 grados, el rango de grados positivos del modelo posibles de enviar en este sistema. No se tomó en cuenta grados negativos por la posibilidad de que la fuerza de sustentación cambie de orientación, como en el caso de un alerón.

Figura 39. Diagrama de Flujo para el barrido de grados



Para programar el barrido de grados en el Arduino se utilizó como referencia el diagrama de flujo mostrado en la Figura 39. Desde el principio se decidió que los valores de envío serían trasmitidos desde Labview y relegados al Arduino por el puerto serial, al igual que el envío de grados normal implementado anteriormente. Es importante notar que la naturaleza de la programación en el Arduino y en Labview permite libertad en el tema: es enteramente posible enviar un comando inicial en Labview y relegar el envío de grados automáticamente al Arduino, de tal forma que cada cierto tiempo se envíe un ángulo mayor y se regrese la información de fuerzas a Labview. Al final se obvió esta implementación ya que habría sido un poco confusa, porque no habría sido consistente con la programación de envío de grados anterior, cuyo funcionamiento ya se había comprobado. Es importante destacar como punto final que se decidió utilizar un código similar al ya funcional por el simple hecho que cada cambio en el código del Arduino requiere que se vuelva a programar el chip desde la computadora, y físicamente esto será un problema si en el futuro se decide por motivos didácticos que se quiere cambiar algo en el túnel.

Figura 40. Paso 1 para el barrido de grados

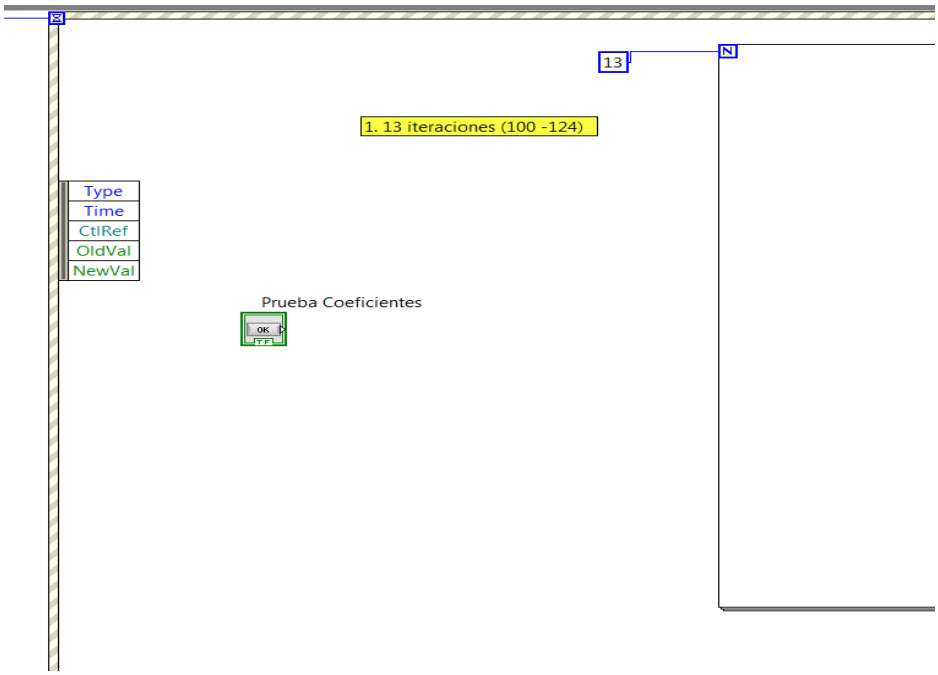


Figura 41. Pasos 2 y 3 para el barrido de grados

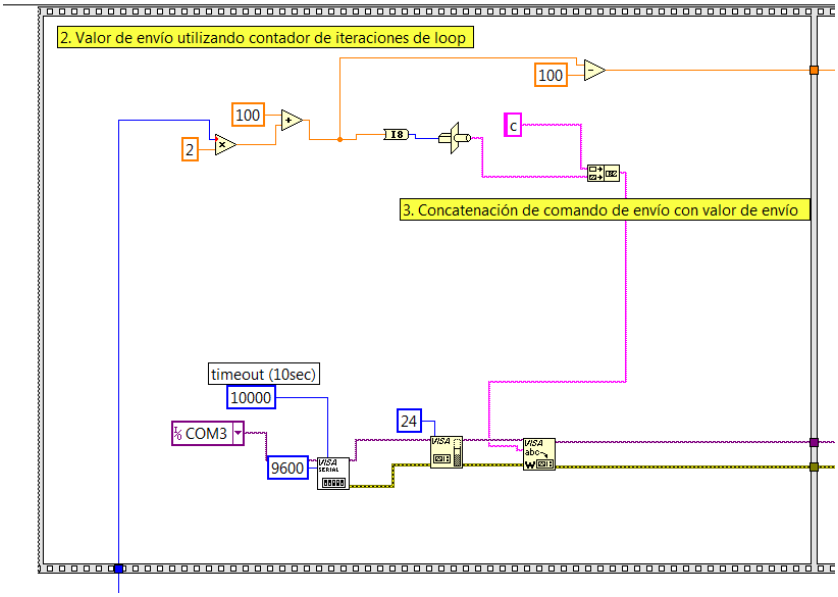


Figura 42. Pasos 4 y 5 para el barrido de grados

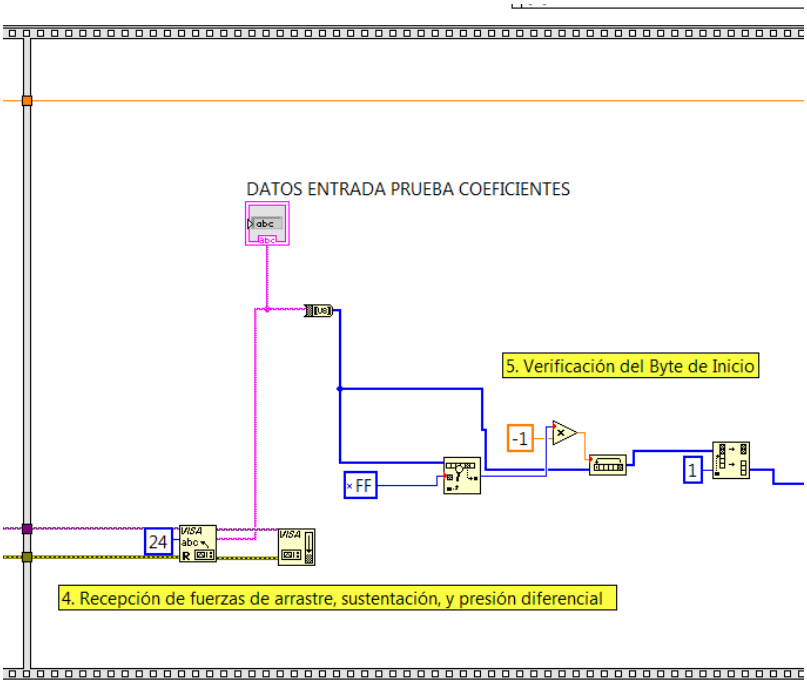


Figura 43. Pasos 6 y 7 para el barrido de grados

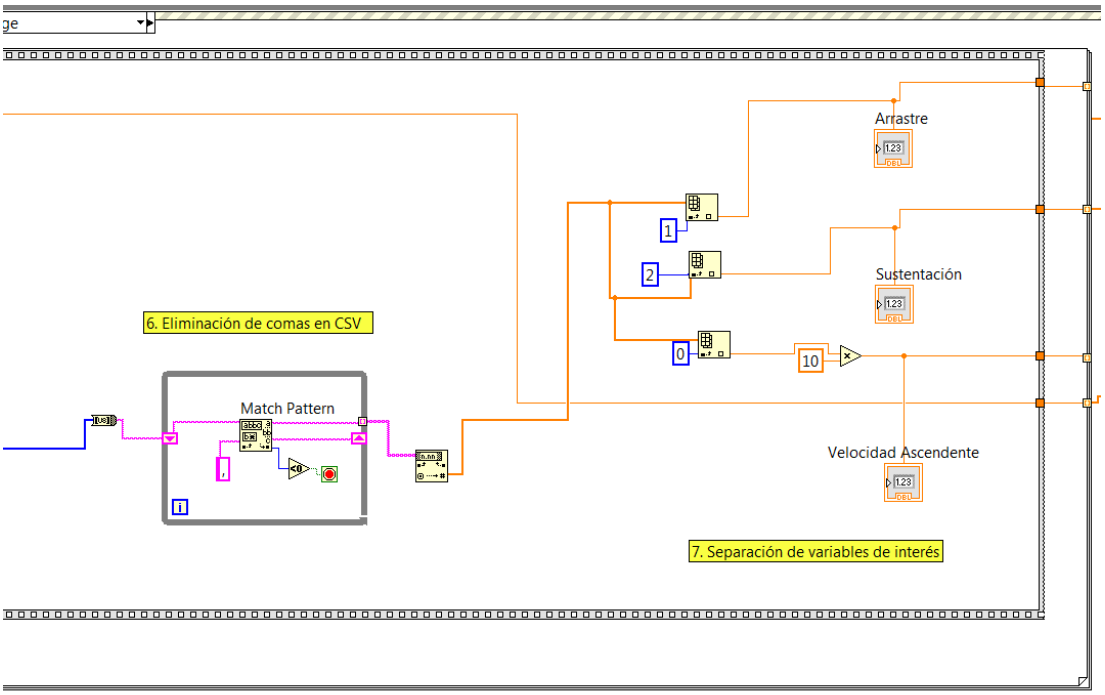
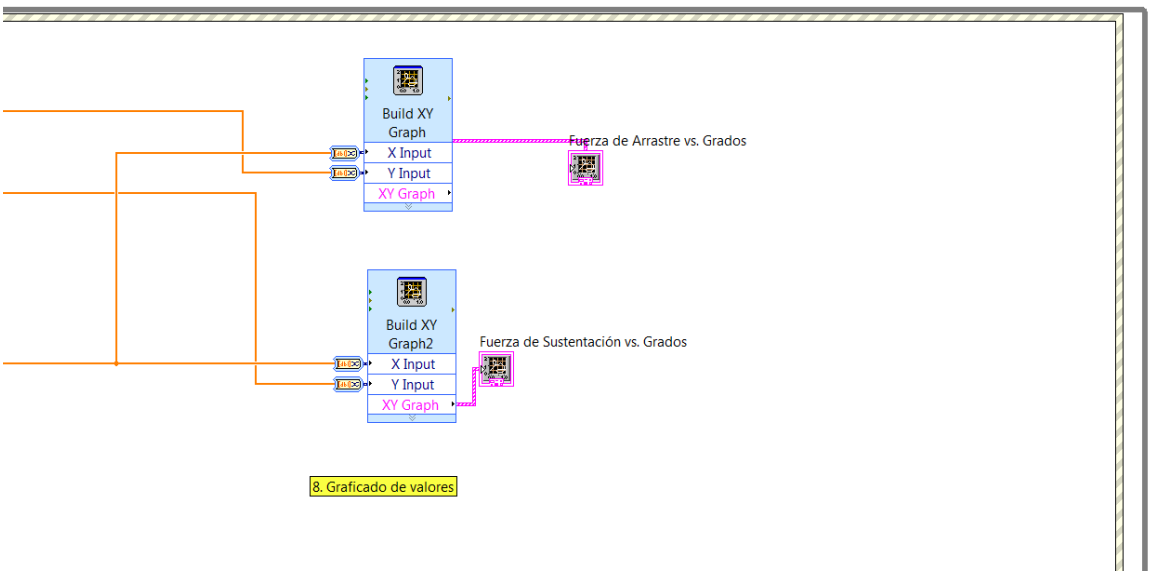


Figura 44. Paso 8 para el barrido de grados



Todos los parámetros importantes y que involucraron decisiones en la programación del barrido de ángulo se encuentran resaltados en rojo en la Figura 39. El valor máximo de envío se manejó como un límite en el for loop de envío para el barrido, como se puede notar en la Figura 40. Este valor está directamente ligado a la resolución del envío de grados, que se puede notar en la Figura 41 en el Paso 2, en el primer bloque de multiplicación: se requiere en el túnel un mínimo de diferencia de dos grados para enviar cambios de grados, ya que se determinó experimentalmente que el mecanismo utilizado para el movimiento del modelo se trababa al realizar envíos con un grado de diferencia. En el barrido estándar implementado, se decidió utilizar la resolución mínima posible de dos grados entre cambios – después de obtener los valores de fuerza de arrastre, sustentación, y presión diferencial en una posición del modelo, se le incrementan dos grados (de ser aplicable) a la posición del modelo y se vuelve a muestrear. Este cambio de grados se implementó en Labview utilizando el contador del loop de envío mostrado también en la Figura 41: una vez se ejecuta un envío de grados y se obtienen los valores deseados, el uso de este contador permite que se le sume un valor de dos al envío de grados siguiente, hasta parar cuando el valor de envío sea 124, por el límite impuesto por el número de iteraciones del for loop. En las Figuras 42 y 43 se puede ver un ejemplo de la consistencia de código mencionada anteriormente, ya que se utilizaron los mismos bloques implementados en el muestreo general de valores, con algunas modificaciones (sólo se reciben tres valores de interés, en vez de seis).

Figura 45. Fuerza de sustentación vs. grados

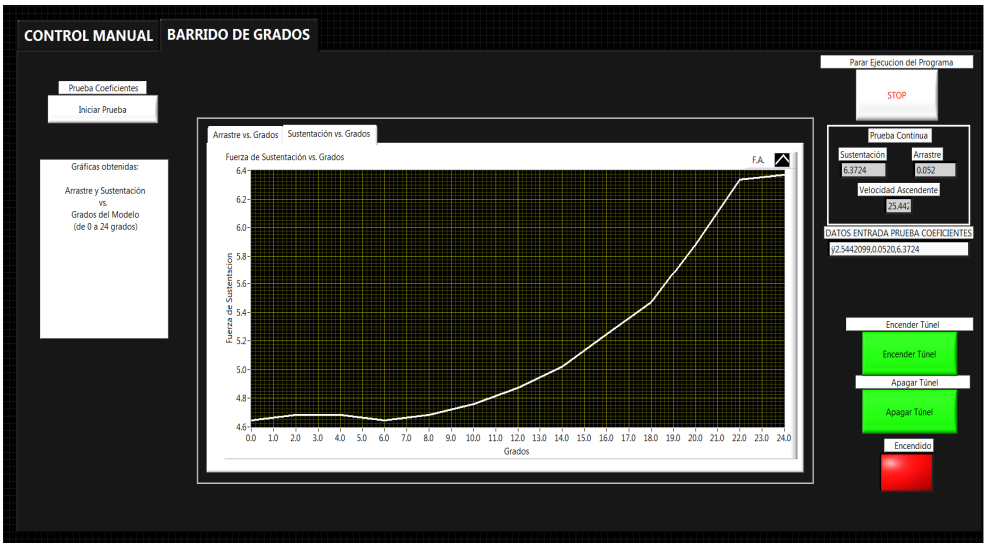
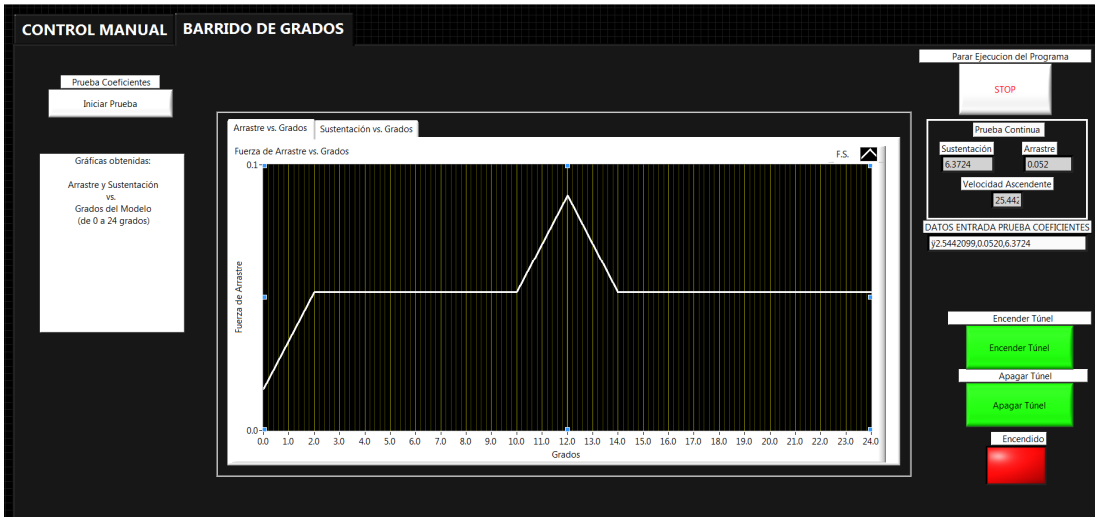


Figura 46. Fuerza de arrastre vs. grados



Finalmente, el Paso 8, mostrado en la Figura 44, utilizó por primera vez en el código el uso de los arrays guardados por el muestreo para graficar la relación entre grados y cada una de las fuerzas de arrastre. Se utilizaron arrays formados por el for loop para corregir la incapacidad de Labview de graficar más de un punto en tiempo real. Se pueden ver los resultados del despliegue de la prueba en las Figuras 45 y 46, para la prueba con el alerón disponible en el túnel. La fuerza de arrastre en el caso de la prueba, como evidencian los bajos valores medidos, fue despreciable. La fuerza de sustentación, sin embargo, mostró un comportamiento no lineal, lo cual es esperado de acuerdo a lo que se investigó para flujos externos. Al confirmar esto se tomó como finalizada la programación para el barrido de grados.

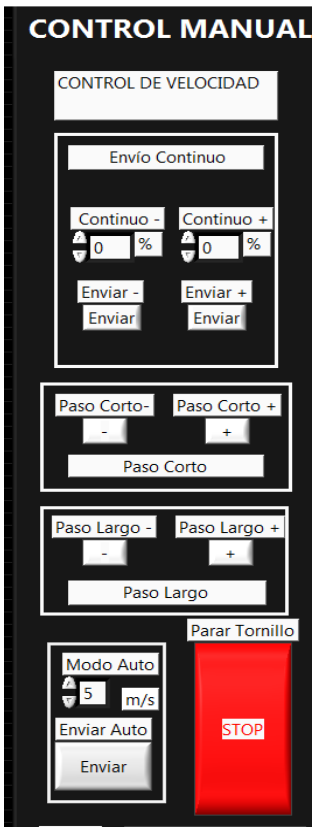
F. Variación de Velocidad de Viento en el Túnel:

Como parte de los objetivos de la instalación del túnel, se buscó poder variar la velocidad en el túnel de viento desde Labview además de los grados en éste. La calibración del módulo de velocidad en el túnel tomó mucho tiempo y pruebas, por lo cual se realizó al final. Para tener claridad y programar eficientemente, se estableció desde el principio una serie de comandos a implementar desde Labview para el control del sistema. En el Cuadro 5 pueden verse todos los comandos que se implementaron en el túnel para el envío de velocidades.

Cuadro 5. Comandos de envío de velocidades desde el Arduino hacia el módulo de velocidades

Comando	1er byte	2do byte
Paso corto cerrar túnel (1/4 de vuelta)	**-**	**no importa**
Paso corto abrir túnel (1/4 de vuelta)	**+**	**no importa**
Paso largo cerrar túnel (1 vuelta)	**n**	**no importa**
Paso largo abrir túnel (1 vuelta)	**m**	**no importa**
Moverse para cerrar continuo	**a**	**0-100 (% de velocidad)**
Moverse para abrir continuo	**d**	**0-100 (% de velocidad)**
Parar tornillo	**f**	**no importa**
Poner velocidad de viento automaticamente	**p**	**5-30 (velocidad m/s)**
Encender túnel	e	no importa
Apagar túnel	o	no importa
Calibrar	z	no importa
Subir velocidad del tornillo	w	no importa
Bajar velocidad del tornillo	s	no importa

Figura 47. Módulo de envío de velocidades en Labview



El envío de velocidades en Labview fue muy similar al envío de grados. Se tuvieron que agregar muchos botones y programación en Labview para permitir todos estos envíos. Sin embargo, el uso de eventos y el hecho que su programación fue muy similar a la ya probada para el envío de grados hizo que esto fuera bastante sencillo, ya que sólo se tuvo que copiar la programación original para el envío de grados, poniendo atención a los comandos acordados en el Cuadro 5. Los envíos para cambios de paso involucraron sólo un comando por el puerto serial. En la programación en Arduino se agregó un valor adicional que no sería utilizado ni por Labview ni por el módulo de velocidad – se acordó en enviar un byte con el valor ASCII de “x”, que no tendría relevancia para los rangos de valores de envío utilizados. En el caso del modo automático y el modo continuo se realizaron envíos utilizando los valores de cambio de indicación en la interfaz, sin realizar ningún cambio. Las Figuras 48 a 57 resumen los envíos desde Labview para realizar lo deseado. Los envíos estipulados por el Cuadro 5, junto con su revisión de estos envíos en las Figuras, puede verse en el código integrado en Anexos.

Figura 48. Envío de paso corto positivo

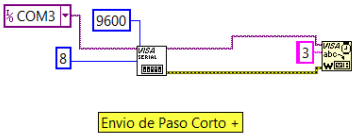


Figura 49. Envío de paso corto negativo

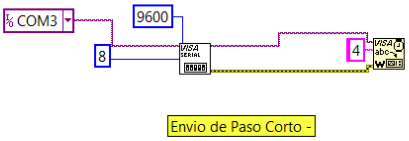


Figura 50. Envío de paso largo positivo

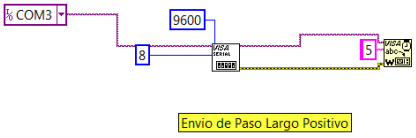


Figura 51. Envío de paso largo negativo

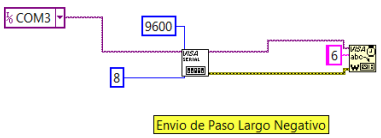


Figura 52. Envío de movimiento porcentual continuo positivo

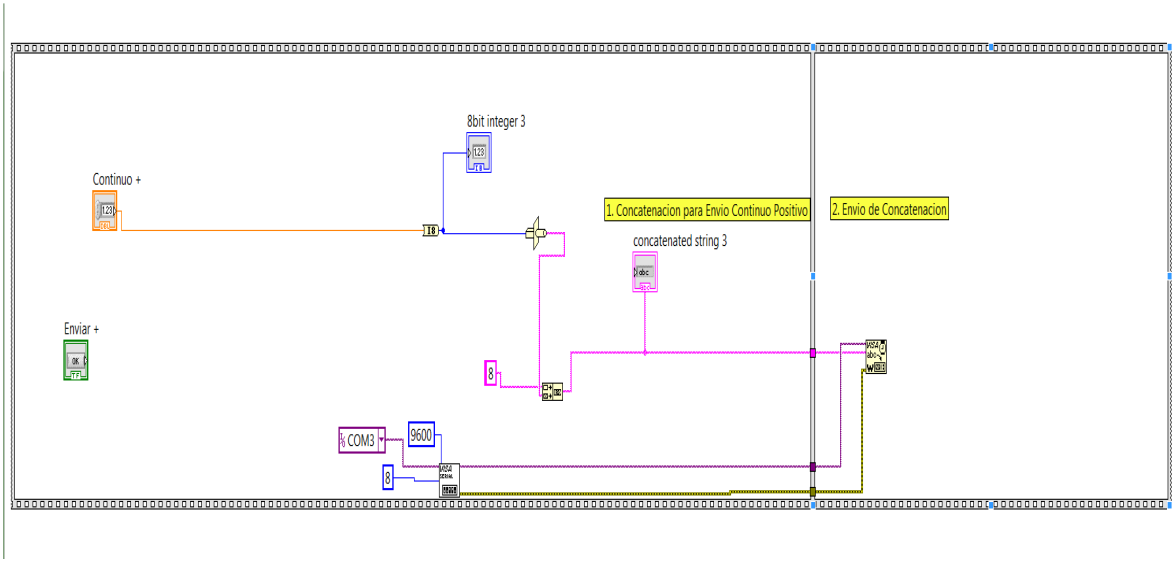


Figura 53. Envío de movimiento porcentual continuo negativo

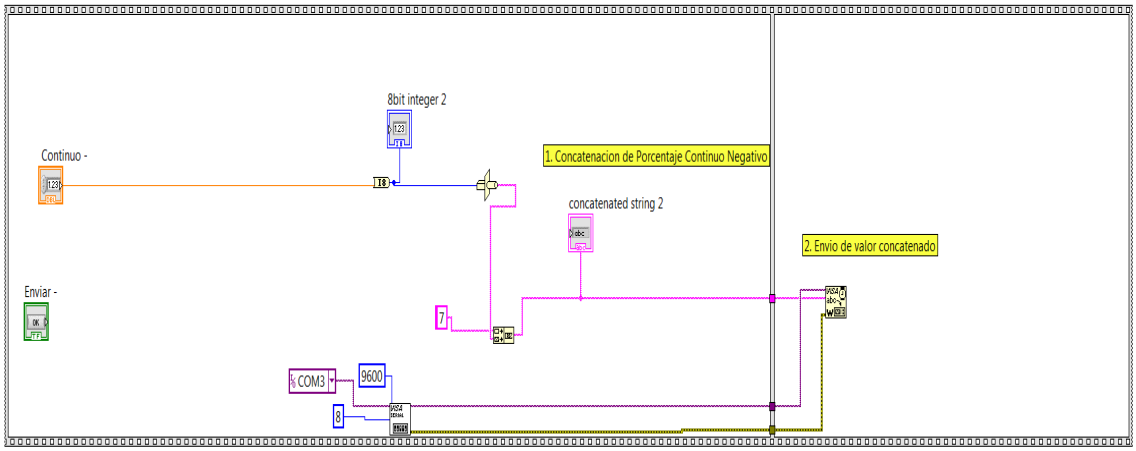


Figura 54. Envío de velocidad automática

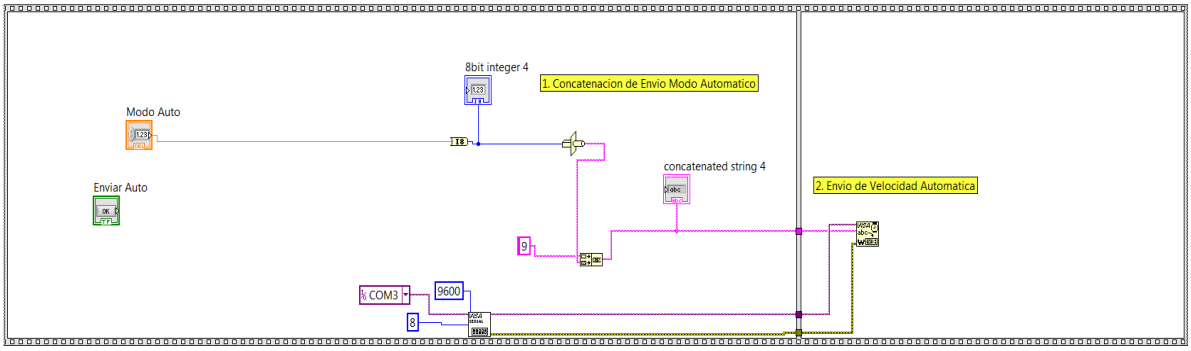


Figura 55. Envío para parar el tornillo en el túnel

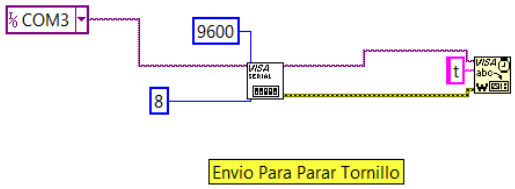


Figura 56. Envío para encender el túnel de viento

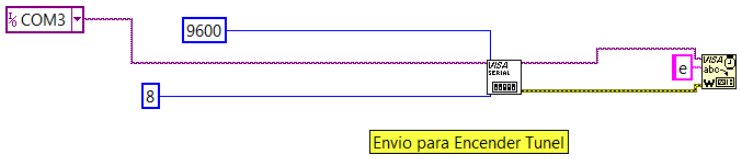
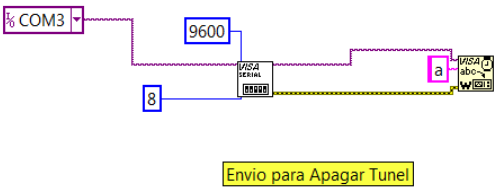


Figura 57. Envío para apagar el túnel de viento



G. Cambios Visuales en la Interfaz de Usuario:

Figura 58. Primera prueba en Labview

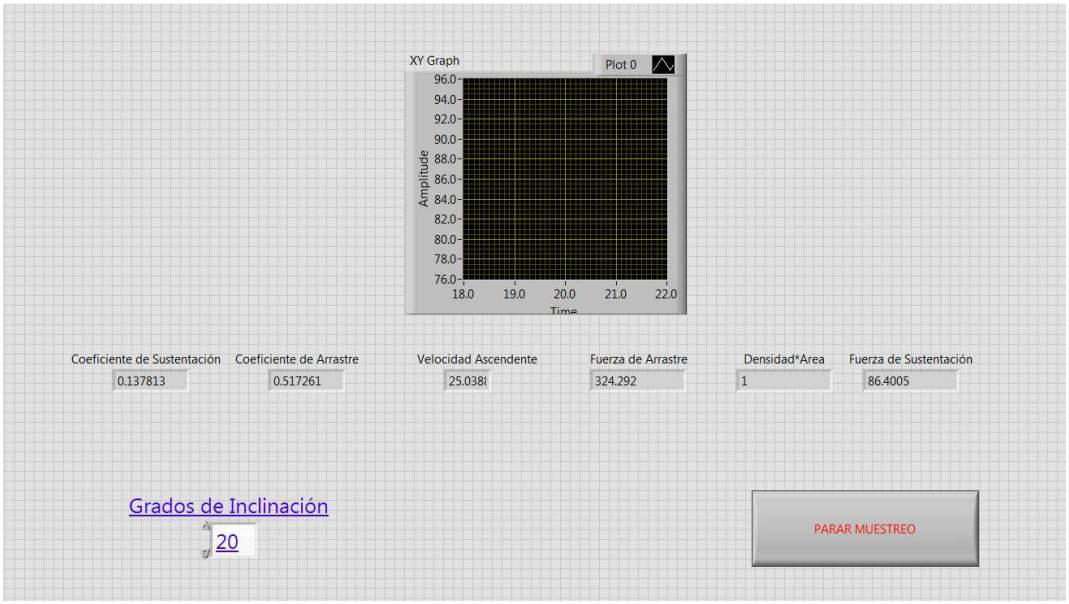


Figura 59. Prueba de grados inicial en Labview

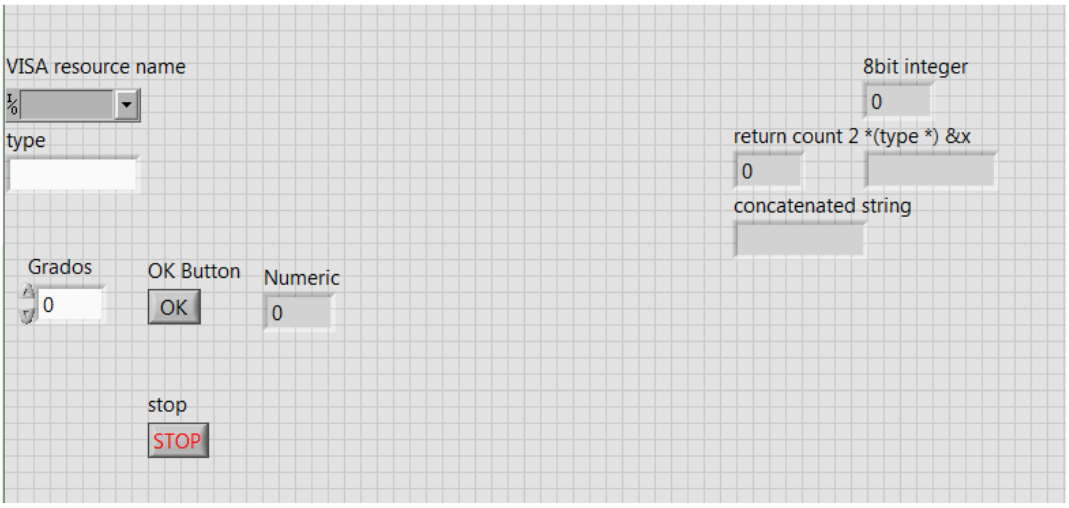


Figura 60. Interfaz con barras splitter y prueba de grados funcional

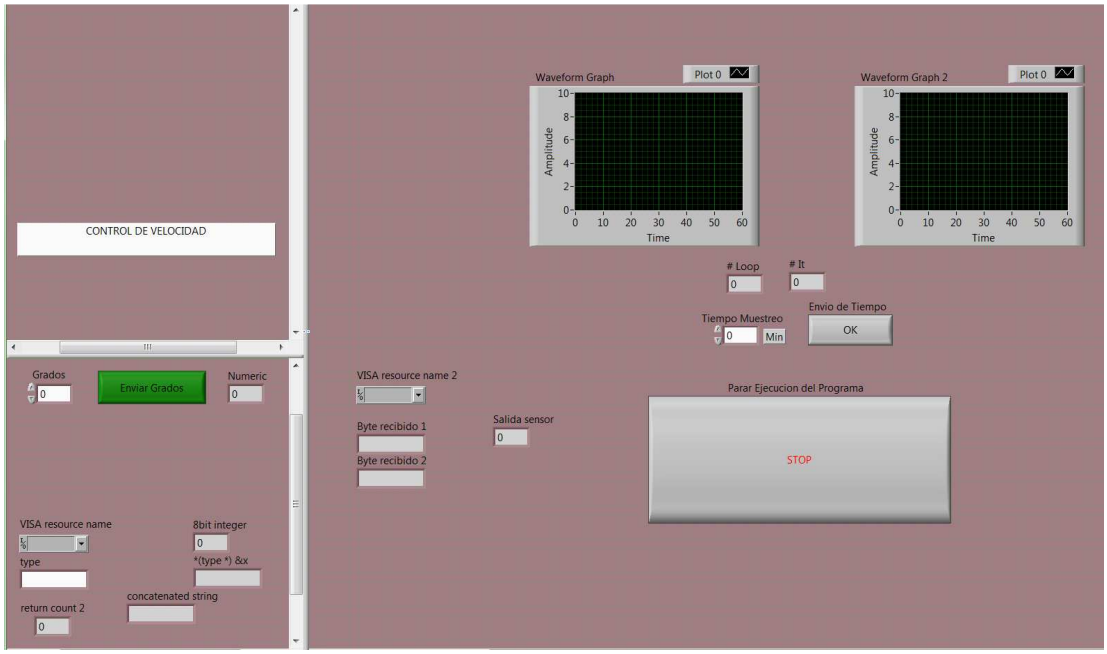


Figura 61. Interfaz con barras splitter y envío de grados y fuerzas funcional

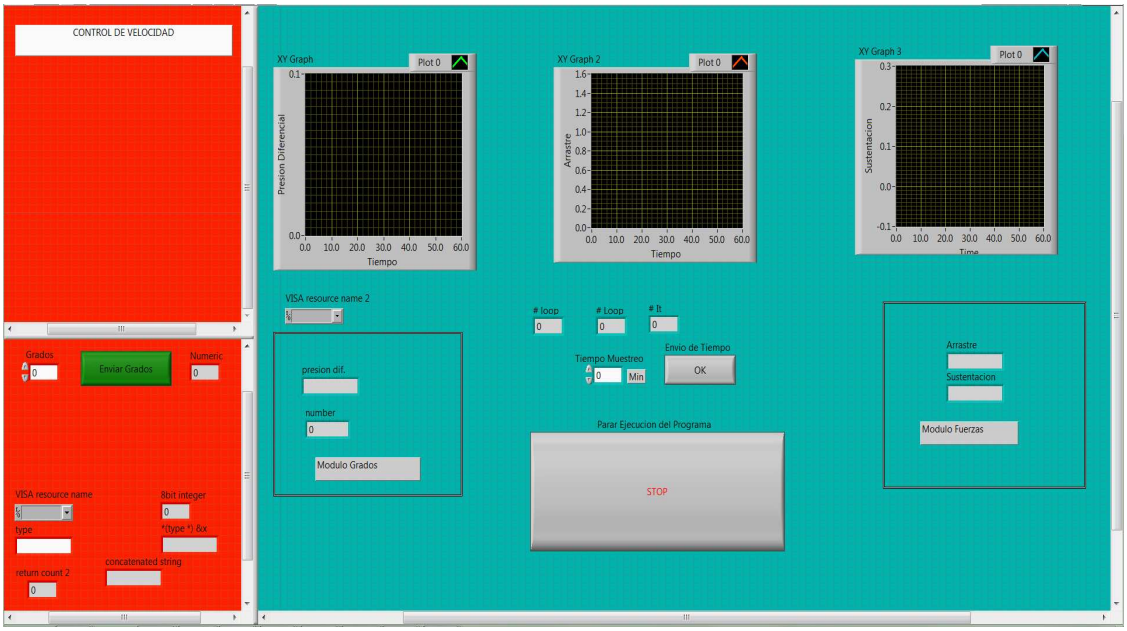


Figura 62. Prueba finalizada con CSV

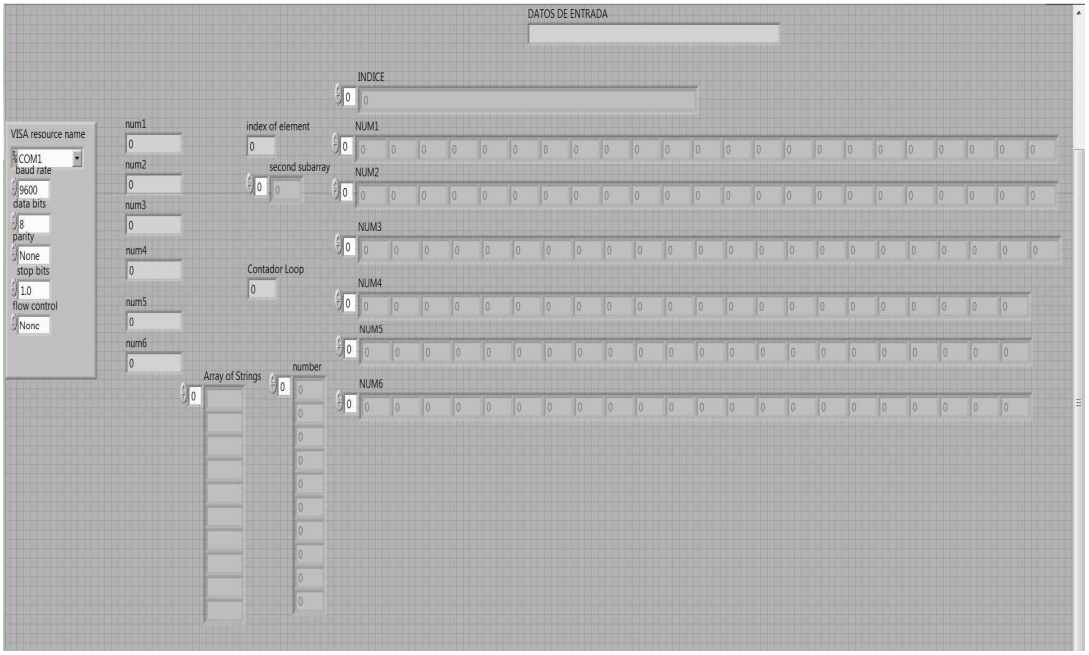


Figura 63. Control de envíos (esquina superior)

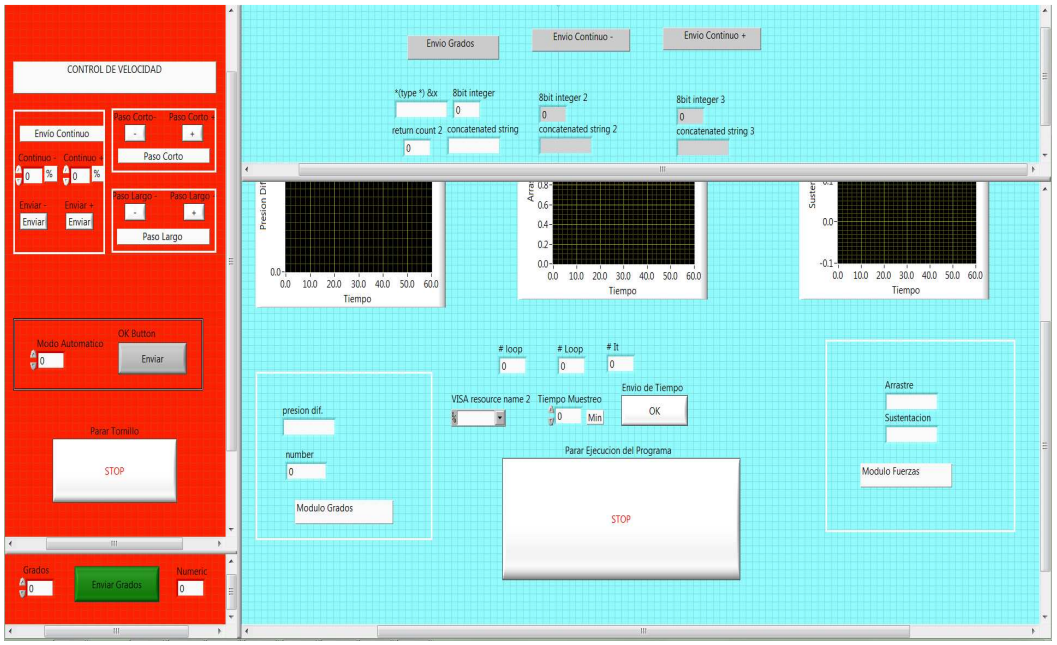


Figura 64. Interfaz con indicadores de envío

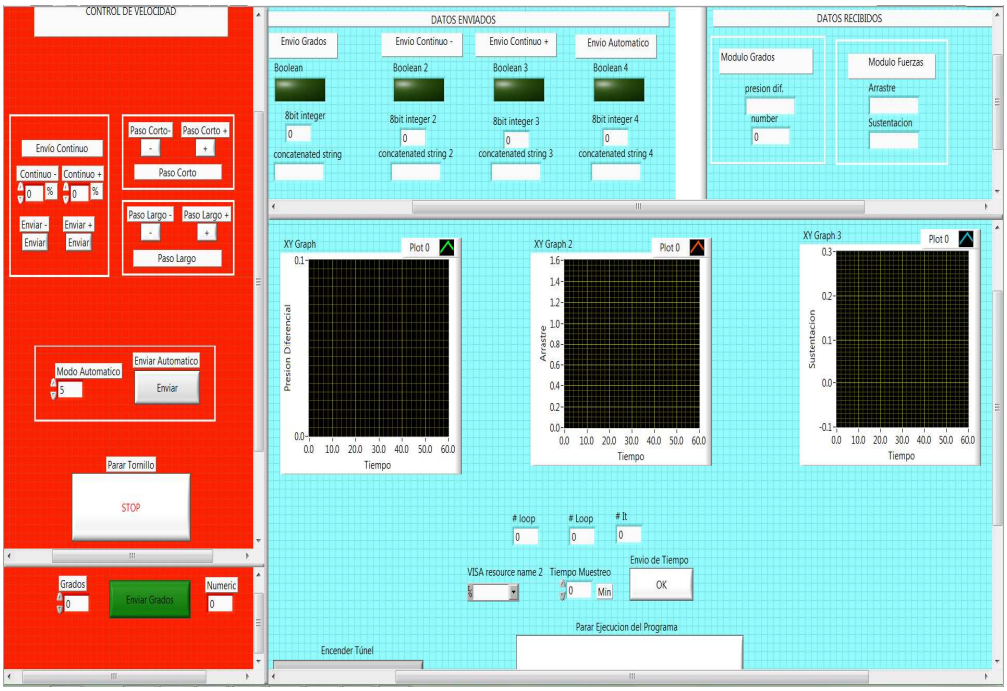


Figura 65. Interfaz con cambios de color, botón de encendido y apagado, y datos de prueba

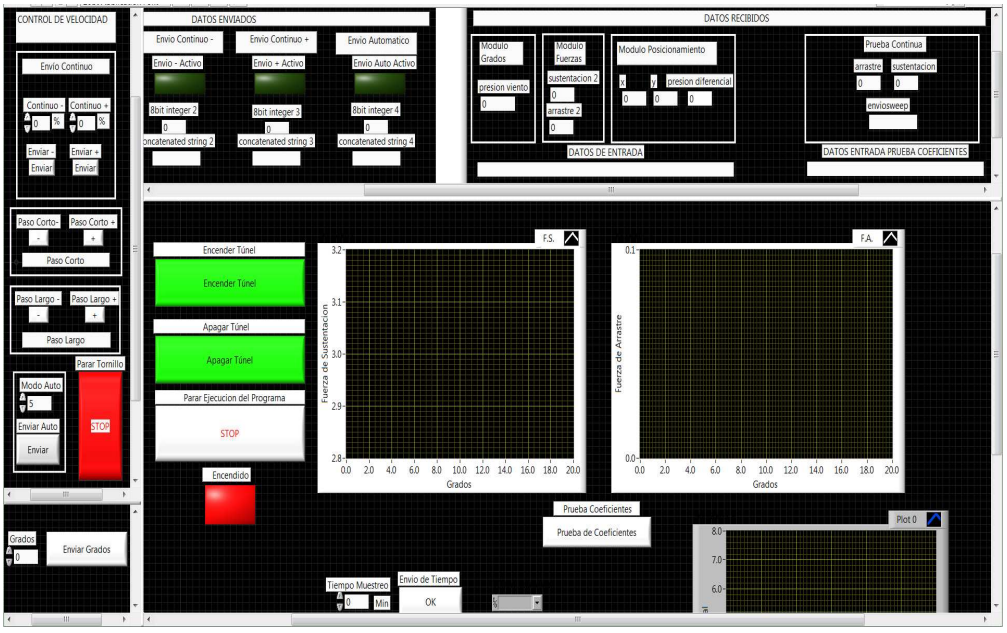


Figura 66. Uso de tabs, control manual

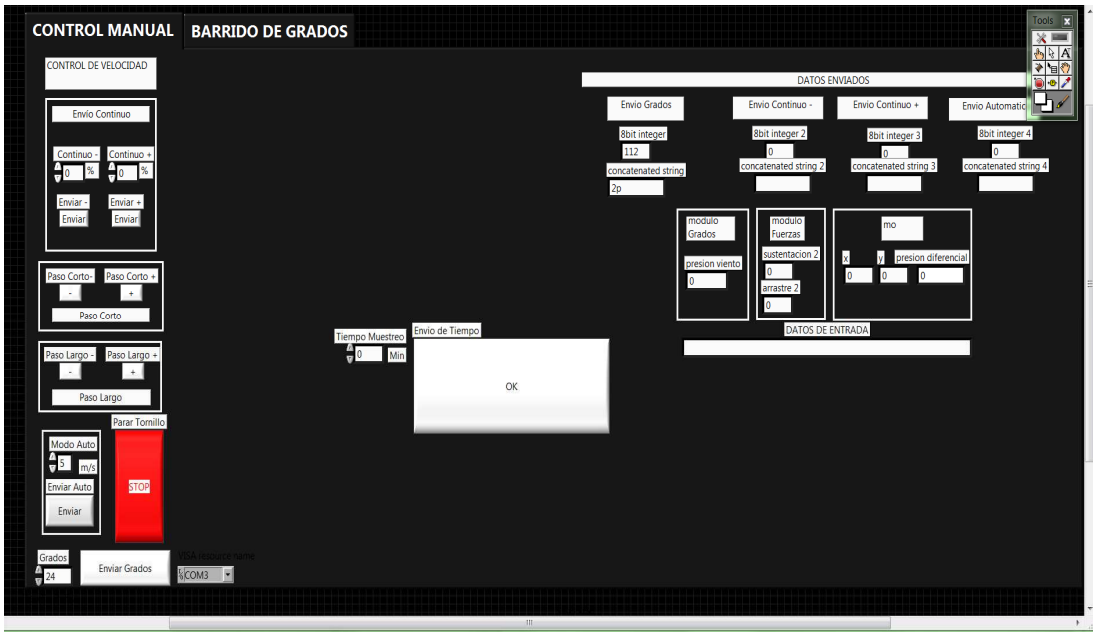
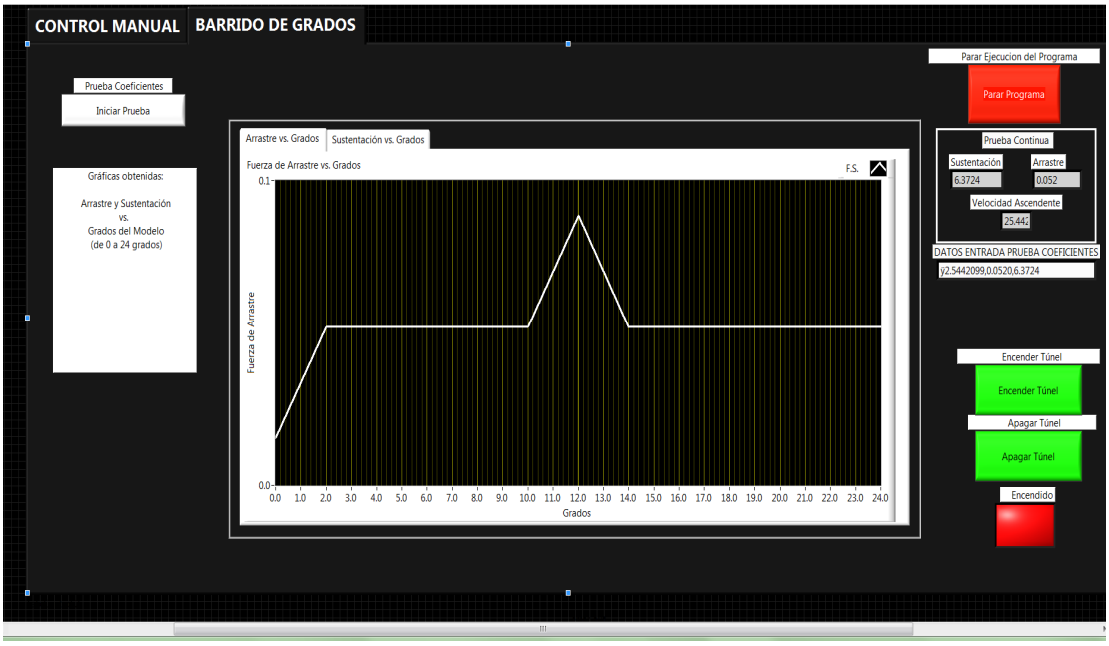


Figura 67. Uso de tabs, barrido de grados



Las Figuras 58 a 67 resumen todos los cambios de despliegue creados en la interfaz. Se comenzó muy simplemente, como puede notarse en las Figuras 58 y 59. En esta etapa del diseño de la interfaz visual no se buscó hacerla adecuada para el uso del usuario, ya que sólo se estaban realizando pruebas para comprobar la funcionalidad del código. Las Figuras 60 y 61 muestran ya el uso de colores en la interfaz como una forma de probar cómo implementarlos. Adicionalmente, en las figuras también puede observarse el uso de barras "splitter" para lograr dividir la interfaz en una manera inicial en áreas con elementos afines. La Figura 62 se agregó como referencia de los procesos adicionales de prueba siendo implementados en la interfaz mientras se realizaban los cambios estéticos en el código ya probado – siempre se buscó funcionalidad antes que estética en la programación de la interfaz, ya que lograr que el código funcionara fue la principal prioridad del proyecto. Las Figuras 63 y 64 comenzaron a evidenciar un problema que sería resuelto más adelante: el incremento indefinido de las áreas de trabajo que encapsulan las barras "splitter". Mientras se agregaban más elementos a cada área encapsulada, cada vez se volvió menos posible observarlos a todos al mismo tiempo, ya que Labview no fija límites en pantalla a las áreas entre barras "splitter": cuando se agregan elementos que superan el área visible establecida por estas barras, Labview simplemente corre los elementos adicionales al área no visible alrededor de estas barras. Esta propiedad esencialmente obligaría al usuario a tener que ir moviendo el botón de la barra para poder utilizar todos los elementos en la interfaz, lo cual contradice el principio de simplicidad deseado en una interfaz. Otro problema que se notó fue el impacto visual de los colores en la interfaz: las Figuras 61,62, 63, y 64 muestran un mal uso de colores ya que son demasiado brillantes para el despliegue en una computadora. Para resolver el problema de color, se decidió cambiar el fondo y la mayoría de los textos utilizados a blanco y negro, para ser consistentes con la convención de diseño de interfaces. Este cambio estético es evidente en las Figuras 65 a 67.

El cambio final y más visible en la progresión de diseño en la interfaz fue el cambio de distribución de elementos entre las Figuras 65, 66, y 67. Se notó al realizar el cambio de color mencionado anteriormente y agregar los elementos de velocidad e indicadores que la pantalla ya estaba exageradamente congestionada por elementos (ver Figura 35), y que era difícil establecer prioridades entre éstos y navegar cómodamente los menús e interfaces. Para corregir esto se descubrió el uso de tabs en Labview. El uso de tabs, mostrado en las Figuras 66 y 67, fue extremadamente útil ya que permitió que se separara el modo manual de envíos de Labview al modo de prueba de barrido sin la necesidad de utilizar barras "splitter". Este fue un gran salto de

diseño ya que se cumplió con virtualmente todas las características que faltaban en la interfaz: se redujo cada acción a pasos lineales simples, se redujo la congestión visual en la pantalla, se permitió aumentar el tamaño de las gráficas para la prueba de barrido (cumpliendo al mismo tiempo con la implementación de jerarquías visuales) y se logró, en general, obtener un diseño más compacto, económico, y atractivo.

Figura 68. Cambios posteriores a la distribución de la interfaz manual

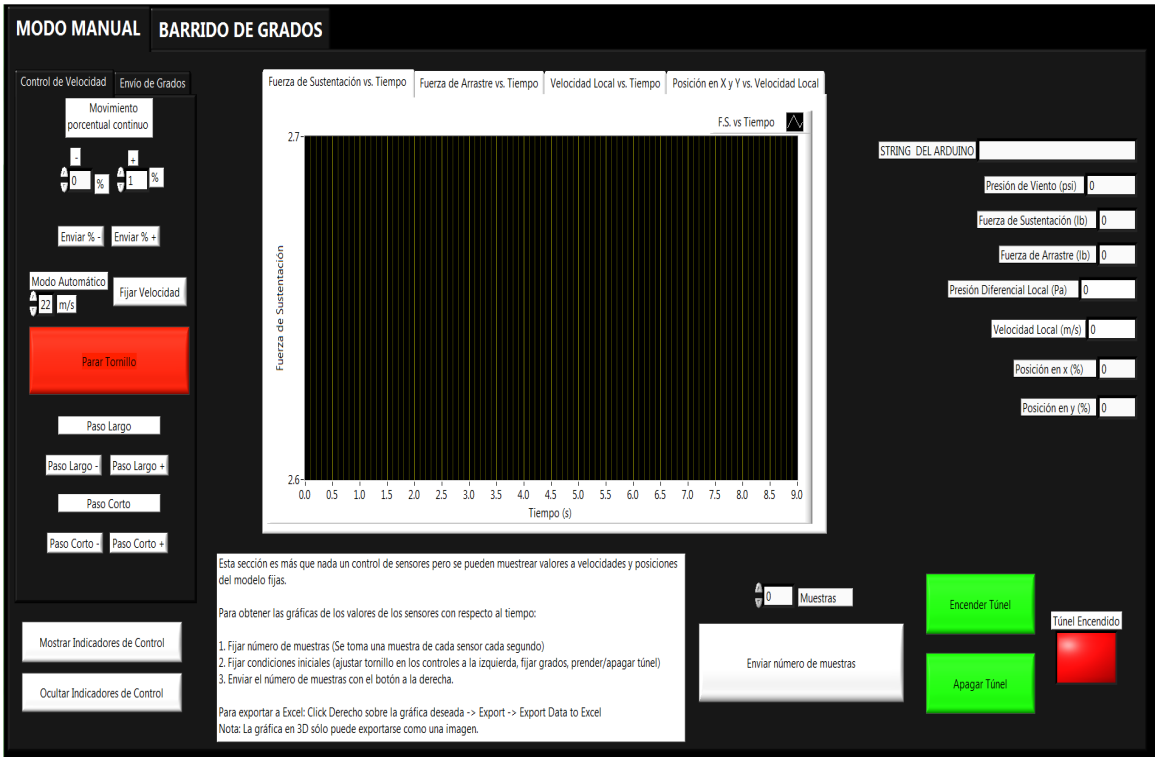


Figura 69. Cambios posteriores a la distribución del modo de barrido

Como parte de la revisión continua a la interfaz gráfica, se realizaron algunos cambios posteriores para facilitar la visibilidad de los controles y comandos en la interfaz. Se agregaron en la esquina superior izquierda de ambas secciones del programa botones para mostrar y desplegar los indicadores de control desde el Arduino, organizando estos en la esquina derecha del programa. Para esto se utilizaron las propiedades de visualización de Labview, que permiten cambiar el estado visible o no visible de cualquier objeto de acuerdo a señales booleanas. Adicional a esto, se agregaron descripciones de los procesos anteriormente programados. Es importante destacar que se realizó un cambio significativo en el modo de muestreo: en vez de tomar una serie de 60 muestras cada minuto, como anteriormente se programó, se cambió el loop en el programa para ingresar un número de muestras, las cuales se tomarían cada segundo. Esto se decidió hacer ya que no era considerado necesario por motivos de tiempo que el usuario muestreara por hasta 10 minutos a la vez – probablemente se requerirían solo un par de muestras por prueba al realizar experimentos.

Figura 70. Interfaz final de muestreo

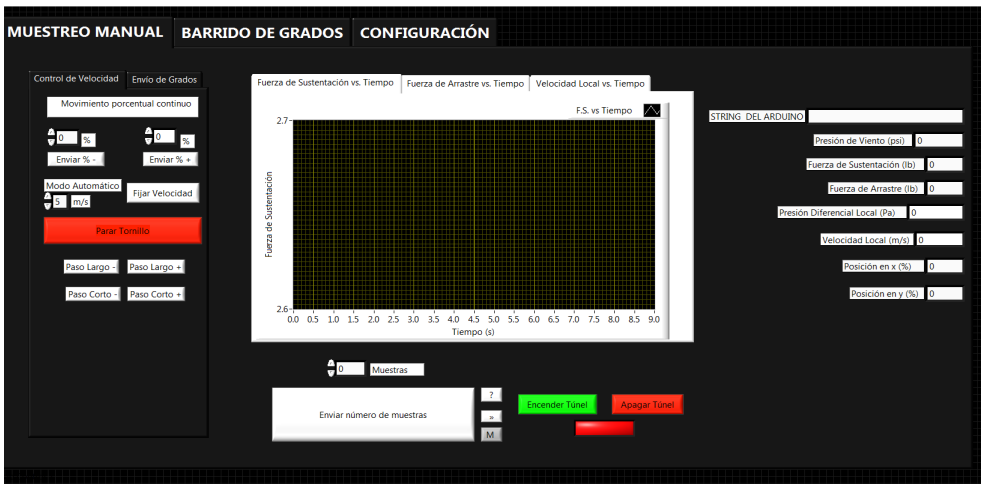


Figura 71. Interfaz final de barrido de grados

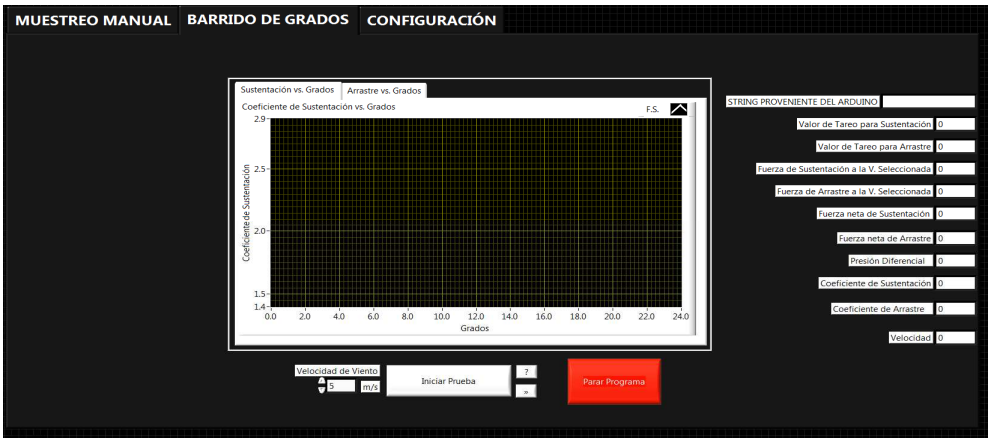


Figura 72. Interfaz final de configuración

Figura 73. Despliegue de ayuda y uso de botones de visualización para el muestreo manual

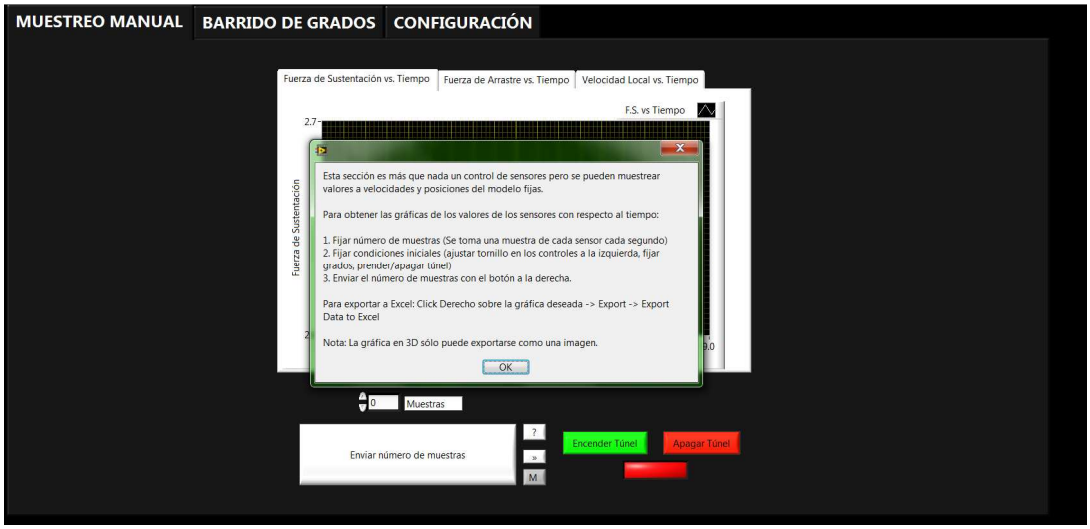
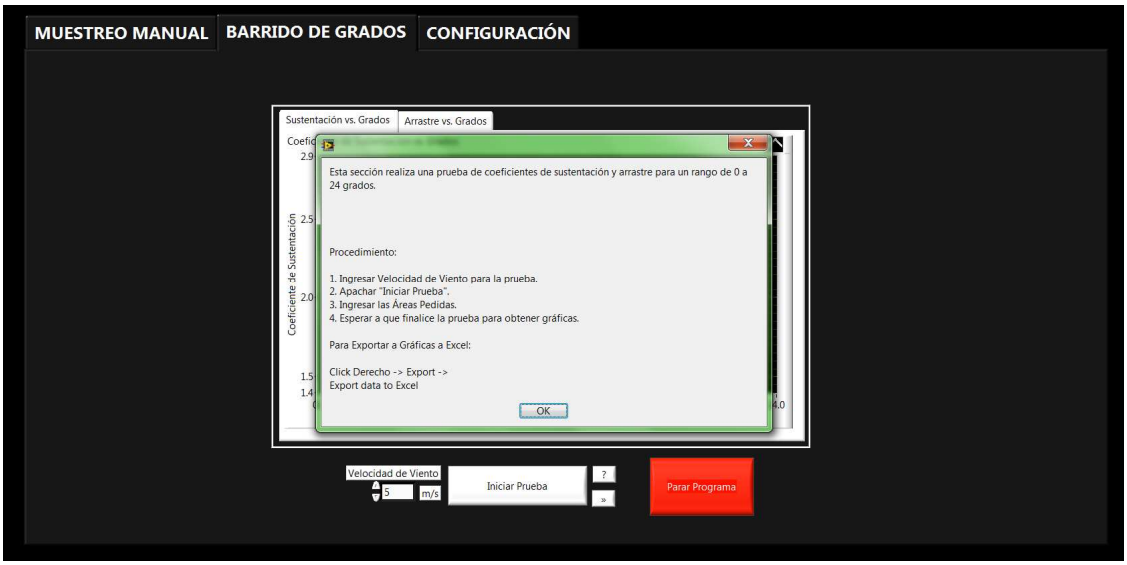


Figura 74. Despliegue de ayuda y uso de botones de visualización para la prueba de barrido



Los cambios finales a la estructura de la interfaz para el túnel de viento pueden observarse en las Figuras 70 a 74. Para reducir la cantidad de elementos en la pantalla y obtener el diseño más simple y eficiente posible, se implementó un solo botón de visualización, en vez de los dos colocados anteriormente. El uso anterior de dos botones se prestaba a confusión y era mal diseño, por lo cual esto se cambió al estado final que puede observarse en todas las figuras anteriormente mencionadas como el botón con el símbolo "»” directamente debajo del botón de ayuda designado por un símbolo de interrogación. El botón de ayuda en particular se agregó para eliminar el texto descriptivo de la pantalla, y para tener un botón con simbología clara desde la perspectiva del usuario, siendo el signo de interrogación el símbolo típicamente usado para la ayuda en interfaces con el usuario. Los mensajes anteriormente presentes en las Figuras 68 y 69 como descripciones de ambos tipos de pruebas se pueden ver desplegados en las Figuras 73 y 74, donde se muestran en el programa después de apachar el botón de ayuda. Asimismo, al texto descriptivo de las pruebas también se le agregó una breve descripción de sus botones de visualización relevantes. En el caso de la prueba manual, donde se optó por poner un segundo botón de visualización para el control manual de velocidad y posicionamiento en el túnel con un botón marcado por la letra “M”, se agregó una descripción para la visualización de datos de entrada al igual que para este botón anteriormente descrito, para asegurarse que en cualquier caso de confusión el usuario pueda identificar el propósito de los botones no críticos como estos. Finalmente cabe destacar que para la interfaz final se le agregó una sección de configuración mostrada en la Figura 72. Esta sección fue implementada para permitir que si se modifica cualquiera de las dos pruebas por separado, se pueda configurar el puerto del Arduino del máster en la computadora sin necesidad de tener que programar en Labview. Anteriormente, el puerto de comunicaciones fue establecido como un parámetro fijo, pero se notó que esto podría presentar problemas para cambios futuros en el túnel por lo cual se agregó esta sección.

VI. CONCLUSIONES

1. El uso de ciclos de ejecución como for loops en Labview deshabilita el panel frontal, imposibilitando el envío de grados, velocidades, u otros comandos. Esto es útil porque elimina la posibilidad que el usuario realice cambios abruptos y rápidos en el túnel que generen resultados no representativos, pero puede presentar problemas ya que imposibilita el movimiento del modelo en el túnel o que se generen cambios de velocidad mientras se ejecuta un período de muestreo. Es preferible, en último caso, sacrificar un poco de versatilidad para asegurar resultados confiables e útiles.

2. Es posible cuando se programa una interfaz en Labview, con el uso de microcontroladores, que cualquiera de los dos se encargue de realizar cálculos con los valores suministrados o a suministrar, respectivamente. Lo único que es importante tomar en cuenta es que las conversiones de tipo de dato presentan mayor facilidad de manipulación en C que en el software de Labview. Si se manejan datos con precisión o complejidad y se requiere su conversión para que se desplieguen en una manera útil en una VI en Labview, es muy importante saber qué conversiones se deben utilizar y dónde en Labview están ubicadas.

3. Se necesitan enviar alrededor de 200 bytes por segundo en el túnel, valor muy bajo comparado a la Baud rate de 9600 y la velocidad de I2C de 400kbps en el túnel. El envío de valores flotantes usando Serial.print() en el Arduino fue posible gracias a el gran margen de seguridad con las velocidades de envío.

4. La programación de interfaces es un proceso continuo que requiere la posibilidad de efectuar muchos cambios en las interfaces programadas, aunque ya estén terminadas y funcionales. Siempre es posible ajustar los parámetros visuales y los datos de una interfaz para mejorarla, de acuerdo a la retroalimentación del usuario. Se notó que la interfaz podía ser confusa para otras personas en múltiples ocasiones, por lo cual se fue modificando su diseño de acuerdo a los comentarios recibidos al utilizar la interfaz.

5. El envío de grados y velocidades fue implementado exitosamente. Se realizaron muchas pruebas para asegurarse de la funcionalidad de estos envíos.

6. La implementación de las funciones en el túnel involucró poco código y fue bien documentada. Se cumplió con el deseo de mantener el diseño de la interfaz simple y accesible.

7. El único gasto involucrado en este proyecto fue el diseño de la caja para contener los microcontroladores, el gasto en la placa, y el precio del Arduino, que no superó los Q300. Se cumplió con el objetivo que la interfaz fuera diseñada en una manera económica.

VII. RECOMENDACIONES

1. Se sugiere la implementación de un módulo de tareo para permitir calibrar los sensores en el túnel. Existen fuerzas cuando el túnel no está prendido que afectan los verdaderos valores siendo analizados en el túnel.

2. Es posible programar modos de prueba para modelos experimentales ya mucho más específicos, si se desea. Esto sería interesante si se quieren hacer pruebas con mayor exactitud.

3. Es recomendable realizar pruebas para determinar las capas límites exactas de todos los modelos que se desean utilizar en el túnel, para poder obtener datos que no sean afectados por turbulencia.

4. Si se desean implementar líneas de flujo en el túnel, es necesario modificar el módulo de posicionamiento para permitir movimiento automático o en curvas, y definir la resolución mínima para poder determinar qué puntos de posición se encuentran en la misma línea de flujo.

VIII. BIBLIOGRAFÍA

1. Arduino. *Begin().*
http://arduino.cc/es/Serial/Begin. 15/10/2013.

2. Arduino. *End().*
http://arduino.cc/es/Serial/End. 15/10/2013.

3. Arduino. *Librería Wire.*
http://arduino.cc/es/Reference/Wire. 15/10/2013.

4. Arduino. *Master Writer/Slave Receiver.*
http://arduino.cc/en/Tutorial/MasterWriter. 15/10/2013.

5. Arduino. *Print().*
http://arduino.cc/es/Serial/Print. 15/10/2013.

6. Arduino. *Read().*
http://arduino.cc/es/Serial/Read. 15/10/2013.

7. Arduino. *Serial*.
http://arduino.cc/es/Reference/Serial. 15/10/2013.

8. Arduino. *What is Arduino?*
http://arduino.cc/en/Guide/Introduction. 15/10/2013.

9. Arduino. *Wire.begin().*
http://arduino.cc/es/Reference/WireBegin. 15/10/2013.

10. Arduino. *Wire.beginTransmission(address).*
http://arduino.cc/es/Reference/WireBeginTransmission. 15/10/2013.

11. Arduino. Wire.endTransmission().
http://arduino.cc/es/Reference/WireEndTransmission. 15/10/2013.

12. Arduino. *Wire.requestFrom(address,quantity).*
http://arduino.cc/es/Reference/WireRequestFrom. 15/10/2013.

13. Arduino. *Wire.send(value).*
http://arduino.cc/es/Reference/WireSend. 15/10/2013.

14. Arduino. *Write().*
http://arduino.cc/es/Serial/Write. 15/10/2013.

15. Byte Paradigm. *I2C vs. SPI*.
http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/. 15/10/2013.

16. Cornell University Ergonomics Web. *Ergonomic Guidelines for User Interface Design.*
http://ergo.human.cornell.edu/ahtutorials/interface.html.15/10/2013

17. Marai, Liz. *User Interface Design.*
http://vis.cs.pitt.edu/teaching/cs2620/lectures/L11_UI.pdf. 15/10/2013.

18. Martin, Suzanne. Effective Visual Communication for Graphical User Interfaces.
http://web.cs.wpi.edu/~matt/courses/cs563/talks/smartin/int_design.html. 15/10/2013.

19. Munson, Bruce R., *et al. Fundamentals of Fluid Mechanics*. 2013. Estados Unidos: John Wiley & Sons. Págs. 480-542.

20. National Instruments. *Advantages of Using LabView in Academic Research.*
http://www.ni.com/white-paper/8534/en/. 15/10/2013.

21. National Instruments. *A Quick Comparison of RS-232, RS-422, and RS-485 Serial Communication Interfaces.* 2013.
http://digital.ni.com/public.nsf/allkb/2CABB3FD5CAF2F8686256F1D005AD0CD. 15/10/2013.

22. Patrick, John. *Serial Protocols Compared*. 31/5/2002. http://www.embedded.com/design/connectivity/4023975/Serial-Protocols-Compared. 15/10/2013.

23. VI High 7: How to program events using the Event Structure in Labview. http://www.youtube.com/watch?v=8eO64fo3Pho. 15/10/2013.

IX. ANEXOS

Prueba Primer Código:

```
// Wire Master Writer
// by Nicholas Zambetti <http://www.zambetti.com>

// Demonstrates use of the Wire library
// Writes data to an I2C/TWI slave device
// Refer to the "Wire Slave Receiver" example for use with this

// Created 29 March 2006

// This example code is in the public domain.

// Header original mantenido para mostrar el uso del ejemplo de master/writer


#include <Wire.h>

int direccionMarco=1;

void setup()
{
  Wire.begin();
    // join i2c bus (address optional for master)
}

byte x = 100; // byte para el envio de la primera prueba

void loop()
{
  Wire.beginTransmission(direccionMarco); // transmit to device #4
  Wire.write(x) ;

  Wire.endTransmission();    // stop transmitting

  delay(500);
}
```

Segundo Código de Prueba:

```
#include <Wire.h>

byte prueba=80;
int direccionMarco=1;
char comando;
byte valorenvio;
int i=0;
int i1=0;
byte r1marco;
byte r2marco;
byte pm1=200;
byte pm2=100;



void setup ( )
{
  // Creacion de parametros Seriales
  Serial.begin (9600);
  Wire.begin(); // Esta sera la Baud Rate para todo el sistema

}
void loop ( )
{
  // Recepcion de Comando Inicial de envio desde Labview
  while(Serial.available()){
    if (i==0){
      comando=Serial.read();
    }
    if (i==1){
      valorenvio=Serial.read();
    }
    i++;
  }

    i=0;

// Comando de envio para trasmicion de grados
    if(comando=='2'){
      Wire.beginTransmission(direccionMarco);
      Wire.write(valorenvio);
      Wire.endTransmission();

    }
}
```

Programa Integrado:

```
#include <Wire.h>

byte direccionVelocidad = 4;
int direccionFuerzas = 2;
int direccionPosicionamiento= 3;
int direccionGrados = 1;
int fuerzas[2];
int fuerzas2[2];
float sustentacion;
float arrastre;
char comando;
byte valorenvio;
int i=0;
int ic=0;
int r1Grados;
int r2Grados;
float rGrados;
char pm1='c';
char pm2='d';
byte posicionamiento[3];
byte posicionamiento2;
byte start1=255;
float presiondiferencial;
float presionnegativa;
float posicionenvio[3];

void setup ( )
{
  // Creacion de parametros Seriales
  Serial.begin (9600);
  Wire.begin(); // Esta sera la Baud Rate para todo el sistema

}
void loop ( )
{
  // Recepcion de Comando desde Labview
  while(Serial.available()){
    if (i==0){
      comando=Serial.read();
    }
    if (i==1){
      valorenvio=Serial.read();
    }
```

```
        i++;
      }
        delay(50);
        i=0;

        if(comando=='1'){

        Wire.requestFrom(direccionGrados, 2);     // pedir los dos bytes que constituyen la presion diferencial al modulo de grados
        while(Wire.available()){
        if(i == 0){
        r1Grados = Wire.read();
        }
        if (i == 1){
        r2Grados = Wire.read();
        }

        i++;
        }
        rGrados=((r1Grados*256)+r2Grados)/10000.0; // formula de conversion para obtener el valor de presion
        i=0;

        Wire.requestFrom(direccionFuerzas, 2);
        while(Wire.available())
        {
         fuerzas[i]=Wire.read();
         i++;
         }

       //mapeo de fuerzas para obtener los valores
       fuerzas2[0]=map(fuerzas[0],0,256,0,1024);
       fuerzas2[1]=map(fuerzas[1],0,256,0,1024);
       // Formulas para obtener los valores a enviar
       sustentacion=(0.009132*fuerzas2[0])-0.094155;
       arrastre=(0.009381*fuerzas2[1])-.081717;
       i=0;

       Wire.requestFrom(direccionPosicionamiento, 3);
       while(Wire.available())
       {
         posicionamiento[i]=Wire.read(); //y,x, despues presion
         i++;


       }


       posicionamiento2=map(posicionamiento[2],0,256,0,1024);
       presiondiferencial=((5.6304*posicionamiento2)-2907.1);
```

```
i=0;
posicionenvio[0]=posicionamiento[0]/100.0;
posicionenvio[1]=posicionamiento[1]/100.0;
posicionenvio[2]=presiondiferencial/1000.0;


// 1000.0000 ---- 1.0000000

Serial.write(start1); //1
Serial.print(rGrados,4); //7
Serial.print(","); // 8
Serial.print(sustentacion,4); //14
Serial.print(","); // 15
Serial.print(arrastre,4);// 21
Serial.print(","); // 22
Serial.print(posicionenvio[0],4); // y , 28
Serial.print(","); // 29
Serial.print(posicionenvio[1],4); // x , 35
Serial.print(","); // 36
Serial.print(posicionenvio[2],7); // 45
// 36



 }

 // Envio de Grados

 if(comando=='2'){
   Wire.beginTransmission(direccionGrados);
   Wire.write(valorenvio);
   Wire.endTransmission();
 }
 // Paso corto abrir tunel
 if(comando=='3'){
   Wire.beginTransmission(direccionVelocidad);
   Wire.write('+');
   Wire.write('x');
   Wire.endTransmission();
 }
// Paso corto cerrar tunel
   if(comando=='4'){
   Wire.beginTransmission(direccionVelocidad);
   Wire.write('-');
   Wire.write('x');
   Wire.endTransmission();
 }
 // Paso largo abrir tunel
   if(comando=='5'){
   Wire.beginTransmission(direccionVelocidad);
```

```
  Wire.write('m');
  Wire.write('x');
  Wire.endTransmission();
}
// Paso largo cerrar tunel
  if(comando=='6'){
    Wire.beginTransmission(direccionVelocidad);
    Wire.write('n');
    Wire.write('x');
    Wire.endTransmission();
}
// Enviar continuo -
if(comando=='7'){
  Wire.beginTransmission(direccionVelocidad);
  Wire.write('a');
  Wire.write(valorenvio);
  Wire.endTransmission();
}
// Enviar continuo +
if(comando=='8'){
  Wire.beginTransmission(direccionVelocidad);
  Wire.write('d');
  Wire.write(valorenvio);
  Wire.endTransmission();
}
// Enviado automatico
if(comando=='9'){
  Wire.beginTransmission(direccionVelocidad);
  Wire.write('p');
  Wire.write(valorenvio);
  Wire.endTransmission();
}
// Parar tornillo
if(comando=='t'){
  Wire.beginTransmission(direccionVelocidad);
  Wire.write('f');
  Wire.write(valorenvio);
  Wire.endTransmission();
}
// Encender Tunel
if(comando=='e'){
  Wire.beginTransmission(direccionVelocidad);
  Wire.write('e');
  Wire.write('x');
  Wire.endTransmission();
}
// Apagar Tunel
if(comando=='a'){
  Wire.beginTransmission(direccionVelocidad);
  Wire.write('o');
```

```
    Wire.write('x');
    Wire.endTransmission();
  }
  //Prueba de Coeficientes de Arrastre y Sustentacion
  if(comando=='c'){


    Wire.beginTransmission(direccionGrados);
    Wire.write(valorenvio);
    Wire.endTransmission();
    delay(7500);

    //Pedir fuerzas de arrastre y sustentacion
    Wire.requestFrom(direccionFuerzas, 2);
    while(Wire.available())
    {
      fuerzas[i]=Wire.read();
     i++;
    }
    fuerzas2[0]=map(fuerzas[0],0,256,0,1024);
    fuerzas2[1]=map(fuerzas[1],0,256,0,1024);

    sustentacion=(0.009132*fuerzas2[0])-0.094155;
    arrastre=(0.009381*fuerzas2[1])-.081717;
    i=0;



    Serial.write(start1); //1
    Serial.print(sustentacion,4); //7
    Serial.print(","); //8
    Serial.print(arrastre,4); // 14

  }
  comando=0;
  valorenvio=0;
}
```

X. GLOSARIO

Bus Master/Slave: Un bus master/slave es un bus que contiene un master que controla una serie de otros componentes, conocidos como “slaves”. Esta configuración de buses es casi siempre síncrona ya que es conveniente para estas configuraciones que el master maneje tiempos para el envío de datos, y use un reloj.

Bus Multi-Master: Un bus multi-master es un bus master/slave que permite múltiples masters en control del mismo bus. Para saber qué máster rige el proceso en cada momento se realizan pruebas de arbitración entre los masters, que determinan cuál controlará el proceso y por lo tanto los comandos a ejecutarse.

Bus Multi-punto: Esta configuración de buses involucra a más de dos peers, y se diferencia de la interfaz Multi-Drop por el hecho que permite la comunicación en ambas direcciones (enviar o recibir). Estos buses son half dúplex.

Bus síncrono/asíncrono: Un bus síncrono es un bus que envía información de acuerdo a señales de reloj, y un bus asíncrono no depende de un reloj para el envío de datos.

Byte: Valor decimal ASCII entre 0 y 255, sin signo.

Char: Valor ASCII entre -128 y 127. Los números se traducen directamente a sus valores utilizando una tabla ASCII.

Drag: Término en inglés sinónimo con arrastre.

Float: Valores que pueden contener valores hasta 3.4028235E+38, no menores a -3.4028235E+38. Se guardan como 4 bytes de información.

Full Duplex y Half Duplex: Un protocol de comunicación full-duplex permite enviar y recibir información por el puerto serial simultáneamente. La comunicación half-duplex permite enviar o recibir, pero sólo se puede realizar una de las dos en cualquier momento.

GUI: Siglas en inglés para “Graphical User Interface”, o interfaz de usuario gráfica.

Interfaz Multi-Drop: Este término se refiere a una interfaz en la que existen múltiples receptores y sólo un transmisor de información.

Interfaces punto a punto o “peer”: Esta configuración de comunicación involucra a dos componentes de hardware que no son ni masters ni slaves, y tienen la misma precedencia. Este tipo de interfaces son típicamente asíncronas por la poca importancia de manejar tiempos.

Lift: Término en ingles sinónimo con sustentación.

“On the Wire”: Este término representa el hecho que en un programa gráfico no existen variables de estado fijas, y los datos se quedan fijos en los cables que conectan los componentes gráficos.

String: Un string en el contexto de este proyecto es un array de chars.

Upstream: Término en ingles sinónimo con “ascendente”: