



INSTITUTO TECNOLÓGICO DE ZACATEPEC INGENIERÍA ELECTROMECÁNICA

“CONTROL DE VENTILACIÓN PARA CUBICULO Y AUDITORIO”

PRESENTA:

Giovanni Velazquez Avilez.
No. De Ctrl: 14091162.
Carrera: Ing. Electromecánica
Correo: velazquezgio96@gmail.com
Tel: 777 438 76 67

EMPRESA:

Instituto de Energías Renovables.
Calle priv. Xochicalco S/N.
Municipio de Temixco, Morelos.
C.P - 62580 México.
Tels. (52) 777 362 0090 (ext. 29744) y (52) 555 622 9744.
RFC: UNA2907227Y5

ASESOR INTERNO:

Dr. Antonio Champion Coria

ASESOR EXTERNO:

Dr. Guillermo Barrios del Valle.

PERIODO: 10 de agosto de 2020 al 10 de enero del 2021.

AGRADECIMIENTOS

Quiero aprovechar estas líneas para agradecer en primer lugar a Dios por concederme la vida y salud para llevar a cabo este proyecto, así como a mis amados padres quienes han sido una pieza fundamental de inspiración, ejemplo y apoyo a lo largo de todos mis estudios y preparación en todas las áreas de mi vida, y por su puesto a mi amada esposa por estar siempre a mi lado apoyándome en todo lo que emprendo.

A mis asesores externos el Dr. Guillermo Barrios del Valle y el Dr. Guillermo Ramírez Zúñiga, por su labor en el asesoramiento del área de introducción al lenguaje Python, y al manejo de internet de las cosas (IoT), así como su tiempo dedicado a la evaluación del avance de mi proyecto.

Agradezco al proyecto se agradece al proyecto: “Edificios demostrativos de diseño bioclimático en clima cálido subhúmedo en el Instituto de Energías Renovables UNAM”. Proyecto número 291600 del Fondo de Sustentabilidad Energética.

Y a mi asesor interno el Dr. Antonio Champion Coria quien llevo a cabo la revisión de mi proyecto y reporte final durante el tiempo de mi residencia.

RESÚMEN

La ventilación juega un papel muy importante a la hora de la construcción de un lugar, cuando en un recinto no hay una buena ventilación, debería ser tratado como uno de los factores de mayor prioridad a corregir, ya que este problema afecta no solamente a la comodidad de las personas que lo residen sino peor aún, a su rendimiento físico. Con el fin de evitar ese conflicto, el presente trabajo consiste en el diseño, instrumentación y construcción de un sistema de medición de apertura para una ventana deslizable, de uno de los cubículos del Instituto de Energías Renovables (IER) de la UNAM ubicado en Temixco, Mor. La propuesta de dicho sistema consiste en un dispositivo capaz de medir la magnitud de apertura de la ventana cuando un usuario haga uso de ella y que a su vez pueda mandar la información a una plataforma de código abierto, específicamente ThingsBoard, por medio de una red wifi.

Para ello se utilizaron 3 componentes indispensables que efectúan el trabajo deseado, como componente principal está el ESP8266, este microcontrolador esta encargado de gestionar y controlar las entradas y salidas del resto de los componentes, como sensor de distancia se utilizó un sensor ultrasónico, el modulo HC-SR04 que logra obtener la distancia de apertura mediante pulsos de ultrasonido, puesto que el dispositivo debe estar todo el tiempo encendido para leer en cualquier momento la distancia de apertura, se activó el modo “sueño profundo” o deep-sleep que incorpora el microcontrolador, este modo de trabajo consiste en dormir o desactivar casi todas las terminales del ESP excepto el RTC (reloj en tiempo real), esto logra que el consumo de energía normal se reduzca considerablemente, a fin de tener un mayor ahorro energético, esto da paso a nuestro tercer componente, el sensor de vibración SW-420, que tiene como función detectar el movimiento que se produce cuando una persona abre la ventana, y cuando percibe esta vibración manda una señal digital al pin RST(reset) del ESP8266 lo cual reinicia el arranque del programa principal logrando que se inicie la lectura del sensor ultrasónico y se publiquen los datos en ThingsBoard.

A la par de esto, se desarrolló la implementación de un sistema de control para la apertura y cierre de las ventanas del auditorio Tonatiuh, ubicado dentro de las mismas instalaciones del instituto, este sistema está instalado en un ESP32, el hermano mayor del ESP8266. La interfaz del sistema lo componen una pantalla LCD de 20X4 y una botonera de 4 push-button que funge como el control. Se utilizó también un acelerómetro, el módulo MPU6050 que añade también la función giroscopio, este componente fue utilizado para ser fijado en una de las ventanas del auditorio y lograr conocer los grados de inclinación a la que se

encuentran, el modo de obtener dicha inclinación se logró solo con la aceleración de la gravedad y el uso de 2 ecuaciones basadas en la conversión de coordenadas rectangulares a coordenadas polares. El sistema funciona a grandes rasgos de la siguiente manera: para el manejo del sistema se utilizaron los 4 botones que llevan las etiquetas (SI, NO, ↑, ↓) respectivamente.

Primero el programa muestra al usuario el estado actual de apertura de las ventanas, y pregunta si desea cambiarlo, si el usuario decide hacerlo le mostrara un menú con las opciones disponibles de apertura o el cierre en su debido caso, cuando se haya seleccionado una de ellas entonces del ESP32 de uno de sus pines que admiten DAC(conversor digital-analógico) saldrá una señal eléctrica que entrara a un amplificador de voltaje y seguidamente esa señal amplificada será conectada a la entrada de los servomotores donde se regula el nivel de apertura de las ventanas. Finalmente, cuando acabe el proceso, mandará los datos a ThingsBoard para ser visualizados en tiempo real y volverá una vez más a mostrar el estado de las ventanas, esperando la siguiente modificación. La programación de los 2 sistemas realizados en este proyecto está basada en el lenguaje MicroPython con la versión de Python 3.8, con el sistema operativo Linux como entorno de trabajo.

Tanto el sistema de medición de la ventana deslizable como el sistema de control de las ventanas resultaron funcionar de la manera esperada, puede decirse entonces que los objetivos del proyecto se cumplieron satisfactoriamente, con esta participación se espera que la ventilación tanto en el cubículo como el auditorio sea más favorable para los usuarios y con los datos proporcionados en la plataforma puedan usarse para proyectos de mejora en un futuro.

INDICE

CAPÍTULO 1.....	1
GENERALIDADES DEL PROYECTO	1
1.1 <i>Introducción</i>	1
1.2 <i>Descripción de la empresa y del puesto o área del trabajo del estudiante</i>.....	2
1.3 <i>Planteamiento del problema</i>	4
1.4 <i>Objetivos</i>	4
1.5 <i>Justificación</i>	5
CAPÍTULO 2.....	6
MARCO TEÓRICO	6
2.1 FUNDAMENTOS TEÓRICOS	6
2.1.1 <i>Confort térmico</i>.....	6
2.1.2 <i>Electrónica y sistemas de control</i>.....	10
2.1.3 <i>Internet de las cosas (IoT)</i>.....	24
CAPÍTULO 3.....	35
EDIFICACIÓN	35
3.1 CUBÍCULO Y VENTANA DESLIZABLE	35
3.2 AUDITORIO Y VENTILAS	38
3.2.1 <i>Antecedentes</i>	41
CAPÍTULO 4.....	43
SISTEMA DE MEDICIÓN DE VENTANA DESLIZABLE	43
4.1 <i>ESPECIFICACIONES Y REQUERIMIENTOS</i>.....	43
4.2 <i>INSTRUMENTACIÓN Y CONTROL</i>.....	44
4.2.1 <i>Componentes</i>	44
4.2.1 <i>Diseño electrónico</i>	46
4.2.2 <i>Programación</i>	51
4.2.3 <i>PCB</i>	56
CAPÍTULO 5.....	57
SISTEMA DE CONTROL DE VENTILAS	57
5.1 <i>ESPECIFICACIONES Y REQUERIMIENTOS</i>.....	57
5.2 <i>INSTRUMENTACIÓN Y CONTROL</i>.....	58
5.2.1 <i>Componentes del sistema de control</i>	58
5.2.2 <i>Diseño electrónico</i>	60

<i>5.2.2 Programación</i>	71
<i>5.2.3 PCB</i>	79
CAPÍTULO 6.....	80
RESULTADOS.....	80
<i>6.1 SISTEMA DE MEDICION DE VENTANA DESLIZABLE</i>	80
<i>6.2 SISTEMA DE CONTROL DE VENTILAS</i>	88
CAPÍTULO 7.....	99
CONCLUSIONES Y RECOMENDACIONES	99
CAPÍTULO 8.....	100
COMPETENCIAS DESARROLADAS	100
CAPÍTULO 9.....	101
REFERENCIAS BIBLIOGRAFICAS	101
ANEXOS	103

ÍNDICE DE FIGURAS

Figura 1.1. Logotipo del Instituto (“IER-UNAM”)	3
Tabla 2.1. Parámetros físicos de un recinto.Figura 1.1. Logotipo del Instituto (“IER-UNAM”)	3
Figura 2.1. Sistema de ventilación de doble flujo.	9
Figura 2.1. Sistema de ventilación de doble flujo.	9
Figura 2.2. Sistema de ventilación natural.	9
Figura 2.2. Sistema de ventilación natural.	9
Figura 2.3. Estructura general de un sistema de control.	11
Figura 2.3. Estructura general de un sistema de control.	11
Figura 2.4 Clasificación de los tipos de sensores.	12
Figura 2.5 Sensor Ultrasónico HC-SR04.Figura 2.4 Clasificación de los tipos de sensores.	12
Figura 2.5 Sensor Ultrasónico HC-SR04.....	13
Figura 2.6 Funcionamiento del sensor HC-SR04.Figura 2.5 Sensor Ultrasónico HC-SR04.	13
Figura 2.6 Funcionamiento del sensor HC-SR04.	14
Figura 2.7 Sensor de vibración SW420.Figura 2.6 Funcionamiento del sensor HC-SR04.	14
Figura 2.7 Sensor de vibración SW420.....	15
Figura 2.7 Sensor de vibración SW420.....	15
Figura 2.8 Acelerómetro MPU6050.	16
Figura 2.9 Ejes en el MPU6050.Figura 2.8 Acelerómetro MPU6050.	16
Figura 2.9 Ejes en el MPU6050.....	17
Figura 2.10 Clasificación de los tipos de actuadores.Figura 2.9 Ejes en el MPU6050.	17
Figura 2.10 Clasificación de los tipos de actuadores.....	18
Figura 2.10 Clasificación de los tipos de actuadores.....	18
Figura 2.11 Amplificador inversor.	21
Figura 2.12 Amplificador no inversor.Figura 2.11 Amplificador inversor.	21
Figura 2.12 Amplificador no inversor.	21
Figura 2.13 OPAMP LM324N.Figura 2.12 Amplificador no inversor.	21
Figura 2.13 OPAMP LM324N.....	22
Tabla 2.2 Componentes electrónicos más comunes.Figura 2.13 OPAMP LM324N.....	22
Figura 2.14 Representación gráfica de IoT	24
Figura 2.14 Representación gráfica de IoT	24
Figura 2.15 Diagrama de bloques del ESP8266.....	26
Figura 2.16 Diagrama de bloques – Tensilica L106.Figura 2.15 Diagrama de bloques del ESP8266.26	26
Figura 2.16 Diagrama de bloques – Tensilica L106.	26

Figura 2.17 SoC ESP8266.Figura 2.16 Diagrama de bloques – Tensilica L106.	26
Figura 2.17 SoC ESP8266	27
Figura 2.18 ESP8266 NodeMCU.	28
Figura 2.19 ESP8266 PinOUT.Figura 2.18 ESP8266 NodeMCU.	28
Figura 2.19 ESP8266 PinOUT.....	28
Figura 2.20 Diagrama de bloques del ESP32.Figura 2.19 ESP8266 PinOUT.	28
Figura 2.20 Diagrama de bloques del ESP32.....	29
Figura 2.21 ESP32 NodeMCU.Figura 2.20 Diagrama de bloques del ESP32.	29
Figura 2.21 ESP32 NodeMCU.	31
Figura 2.22 ESP32 PinOUT.Figura 2.21 ESP32 NodeMCU.	31
Figura 2.22 ESP32 PinOUT.....	31
Figura 2.23. Interfaz de ThingsBoard.Figura 2.22 ESP32 PinOUT.	31
Figura 2.23. Interfaz de ThingsBoard.	34
Figura 3.1. Ventana deslizable.Figura 2.23. Interfaz de ThingsBoard.	34
Figura 3.1. Ventana deslizable.	35
Figura 3.2. Fachada de cubículo IER-UNAM.Figura 3.1. Ventana deslizable.....	35
Figura 3.2. Fachada de cubículo IER-UNAM.....	36
Figura 3.3. Foto de ventana de cubículo IER-UNAM.Figura 3.2. Fachada de cubículo IER-UNAM... ..	36
Figura 3.3. Foto de ventana de cubículo IER-UNAM.....	36
Figura 3.4 Dimensiones de la ventana deslizable.Figura 3.3. Foto de ventana de cubículo IER-UNAM.....	36
Figura 3.4 Dimensiones de la ventana deslizable.	37
Figura 3.4 Dimensiones de la ventana deslizable.	37
Figura 3.5 Ubicación del auditorio Tonatiuh.....	38
Figura 3.6 Detalles del auditorio Tonatiuh.Figura 3.5 Ubicación del auditorio Tonatiuh.	38
Figura 3.6 Detalles del auditorio Tonatiuh.....	38
Figura 3.7 Ventila de la pared este.Figura 3.6 Detalles del auditorio Tonatiuh.....	38
Figura 3.7 Ventila de la pared este.....	39
Figura 3.8 Ventilas de la pared oeste.Figura 3.7 Ventila de la pared este.....	39
Figura 3.8 Ventilas de la pared oeste.....	40
Figura 3.9 Vista interior de las ventilasFigura 3.8 Ventilas de la pared oeste.	40
Figura 3.9 Vista interior de las ventilas	40
Figura 3.10 Esquema de control de ventilas por Rodríguez, 2011Figura 3.9 Vista interior de las ventilas	40

Figura 3.10 Esquema de control de ventila por Rodríguez, 2011.....	41
Figura 3.11 Servomotor Belimo NMB24-SR.Figura 3.10 Esquema de control de ventila por Rodríguez, 2011.....	41
Figura 3.11 Servomotor Belimo NMB24-SR.....	42
Tabla 4.1 Componentes del sistema de medición.Figura 3.11 Servomotor Belimo NMB24-SR.....	42
Figura 4.1 Esquema de conexión del sistema de medición.	46
Tabla 4.2 Pines de conexión del sistema de medición.Figura 4.1 Esquema de conexión del sistema de medición.....	46
Figura 4.2 Conexión prothoboard del sistema de medición.	47
Figura 4.3 Funcionamiento del sensor ultrasónico HC-SR04.Figura 4.2 Conexión prothoboard del sistema de medición.	47
Figura 4.3 Funcionamiento del sensor ultrasónico HC-SR04.	48
Figura 4.4 Velocidad de las ondas sonoras.Figura 4.3 Funcionamiento del sensor ultrasónico HC-SR04.....	48
Figura 4.4 Velocidad de las ondas sonoras.	48
Figura 4.4 Velocidad de las ondas sonoras.	48
Figura 4.5 Fuente de alimentación.....	50
Figura 4.6 Algoritmo general del sistema de medición.Figura 4.5 Fuente de alimentación.	50
Figura 4.6 Algoritmo general del sistema de medición.....	51
Figura 4.6 Algoritmo general del sistema de medición.....	51
Figura 4.7 Esquemático del sistema de medición.	56
Figura 4.8 Diseño PCB del sistema de medición.Figura 4.7 Esquemático del sistema de medición.	56
Figura 4.8 Diseño PCB del sistema de medición.	56
Tabla 5.1 Componentes del sistema de control.Figura 4.8 Diseño PCB del sistema de medición. ..	56
Figura 5.1 Esquema de conexión del sistema de control.....	60
Tabla 5.2 Pines de conexión del sistema de control.Figura 5.1 Esquema de conexión del sistema de control.....	60
Figura 5.2 Conexión prothoboard del sistema de control.	61
Figura 5.2 Conexión prothoboard del sistema de control.	61
Figura 5.3 Funcionamiento de un MEMS.	62
Tabla 5.3 Descripción de variables del MPU6050.Figura 5.3 Funcionamiento de un MEMS.	62
Figura 5.4 Rangos de configuración del MPU6050.	63
Figura 5.5 Secuencia de rotación aeroespacial (Yaw-Pitch-Roll) Figura 5.4 Rangos de configuración del MPU6050.....	63
Figura 5.5 Secuencia de rotación aeroespacial (Yaw-Pitch-Roll)	64

Figura 5.6 Formación de ángulos respecto a la gravedad.....	64
Figura 5.7 Relación de voltaje-grados de apertura.Figura 5.6 Formación de ángulos respecto a la gravedad.....	64
Figura 5.7 Relación de voltaje-grados de apertura.	66
Figura 5.8 Diagrama eléctrico del amplificador de voltaje.Figura 5.7 Relación de voltaje-grados de apertura.....	66
Figura 5.8 Diagrama eléctrico del amplificador de voltaje.	68
Figura 5.9 Simulación del amplificador de voltaje.Figura 5.8 Diagrama eléctrico del amplificador de voltaje.....	68
Figura 5.9 Simulación del amplificador de voltaje.	68
Tabla 5.4 Relación de variables del sistema de control.Figura 5.9 Simulación del amplificador de voltaje.....	68
Figura 5.10 Esquemático del sistema de control.	79
Figura 5.11 Diseño PCB del sistema de control.Figura 5.10 Esquemático del sistema de control. ...	79
Figura 5.11 Diseño PCB del sistema de control.....	79
Figura 6.1 Sistema de medición.Figura 5.11 Diseño PCB del sistema de control.	79
Figura 6.1 Sistema de medición.	80
Figura 6.2 Lectura 1 del sensor.Figura 6.1 Sistema de medición.	80
Figura 6.2 Lectura 1 del sensor.	80
Figura 6.4 Lectura 3 del sensor.Figura 6.2 Lectura 1 del sensor.	80
Figura 6.4 Lectura 3 del sensor.	81
Figura 6.3 Lectura 2 del sensor.Figura 6.4 Lectura 3 del sensor.	81
Figura 6.3 Lectura 2 del sensor.	81
Figura 6.5 Lecturas del sensor en la terminal.Figura 6.3 Lectura 2 del sensor.	81
Figura 6.5 Lecturas del sensor en la terminal.	82
Figura 6.6 Datos publicados en ThingsBoard.Figura 6.5 Lecturas del sensor en la terminal.....	82
Figura 6.6 Datos publicados en ThingsBoard.	82
Figura 6.7 Modo Deep Sleep del ESP8266.Figura 6.6 Datos publicados en ThingsBoard.....	82
Figura 6.7 Modo Deep Sleep del ESP8266	83
Figura 6.8 Sensor de vibración en funcionamiento.	83
Figura 6.8 Sensor de vibración en funcionamiento.	83
Figura 6.9 Dispositivo colocado en ventana.....	84
Figura 6.10 Objeto para el rebote de la señal.Figura 6.9 Dispositivo colocado en ventana.....	84
Figura 6.10 Objeto para el rebote de la señal.....	85

Figura 6.11 Dispositivo alimentado.Figura 6.10 Objeto para el rebote de la señal.....	85
Figura 6.11 Dispositivo alimentado.....	85
Figura 6.12 Sistema de medición publicando en ThingsBoard.Figura 6.11 Dispositivo alimentado.	85
Figura 6.12 Sistema de medición publicando en ThingsBoard.	86
Figura 6.13 Bitácora de aperturas de la ventana.Figura 6.12 Sistema de medición publicando en ThingsBoard.....	86
Figura 6.13 Bitácora de aperturas de la ventana.	86
Figura 6.14 Bitácora del día Domingo.Figura 6.13 Bitácora de aperturas de la ventana.....	86
Figura 6.14 Bitácora del día Domingo.	87
Figura 6.15 Bitácora del día Lunes.Figura 6.14 Bitácora del día Domingo.....	87
Figura 6.15 Bitácora del día Lunes.	87
Figura 6.16 Posición del acelerómetroFigura 6.15 Bitácora del día Lunes.	87
Figura 6.16 Posición del acelerómetro.....	88
Figura 6.18 Aceleración de AcZ (MPU de cabeza).Figura 6.16 Posición del acelerómetro.....	88
Figura 6.18 Aceleración de AcZ (MPU de cabeza).....	88
Figura 6.17 Aceleración de AcZ.Figura 6.18 Aceleración de AcZ (MPU de cabeza).	88
Figura 6.17 Aceleración de AcZ.	88
Figura 6.19 Aceleración de AcX.Figura 6.17 Aceleración de AcZ.	88
Figura 6.19 Aceleración de AcX.....	89
Figura 6.20 Aceleración de AcY.Figura 6.19 Aceleración de AcX.	89
Figura 6.20 Aceleración de AcY.	89
Figura 6.21 Grados de inclinación en la terminal.Figura 6.20 Aceleración de AcY.	89
Figura 6.21 Grados de inclinación en la terminal.....	90
Figura 6.22 Inclinación publicada en ThingsBoard.Figura 6.21 Grados de inclinación en la terminal.	90
Figura 6.22 Inclinación publicada en ThingsBoard.....	90
Figura 6.23 Escuadras de 30 y 45º.Figura 6.22 Inclinación publicada en ThingsBoard.	90
Figura 6.23 Escuadras de 30 y 45º.	91
Figura 6.23 Escuadras de 30 y 45º.	91
Figura 6.24 Acelerómetro inclinado a 30, 45, y 60º.....	91
Figura 6.25 inclinación de 30, 45, y 60º en ThingsBoard.Figura 6.24 Acelerómetro inclinado a 30, 45, y 60º.	91
Figura 6.25 inclinación de 30, 45, y 60º en ThingsBoard.	92
Figura 6.26 Amplificador de voltaje.Figura 6.25 inclinación de 30, 45, y 60º en ThingsBoard.	92

Figura 6.26 Amplificador de voltaje.....	92
Figura 6.27 Señal amplificada (2-6V).Figura 6.26 Amplificador de voltaje.....	92
Figura 6.27 Señal amplificada (2-6V).....	93
Figura 6.28 Segunda pantalla del sistema de control.Figura 6.27 Señal amplificada (2-6V).....	93
Figura 6.28 Segunda pantalla del sistema de control.....	94
Figura 6.29 Pantalla de horario programado.Figura 6.28 Segunda pantalla del sistema de control.	
.....	94
Figura 6.29 Pantalla de horario programado.....	94
Figura 6.32 Pantalla “ventilas cerradas”Figura 6.29 Pantalla de horario programado.....	94
Figura 6.32 Pantalla “ventilas cerradas”	95
Figura 6.31 Primer pantalla del sistema de control.Figura 6.32 Pantalla “ventilas cerradas”	95
Figura 6.31 Primer pantalla del sistema de control.....	95
Figura 6.31 Primer pantalla del sistema de control.....	95
Figura 6.30 Control del sistema.....	95
Figura 6.33 Pantallas de instrucción del menúFigura 6.30 Control del sistema	95
Figura 6.33 Pantallas de instrucción del menú	96
Figura 6.34 Pantalla menú seleccionableFigura 6.33 Pantallas de instrucción del menú	96
Figura 6.34 Pantalla menú seleccionable.....	96
Figura 6.34 Pantalla menú seleccionable.....	96
Figura 6.35 Pantalla “abriendo 25%”	97
Figura 6.35 Pantalla “abriendo 25%”	97
Figura 6.36 Pantalla “operación exitosa”	97
Figura 6.36 Pantalla “operación exitosa”	97
Figura 6.37 Segunda pantalla del sistema de control	98
Figura 6.37 Segunda pantalla del sistema de control	98
Figura 6.38 Publicación del sistema de control.....	98

ÍNDICE DE TABLAS

Tabla 2.1. Parámetros físicos de un recinto.....	8
Tabla 2.2 Componentes electrónicos más comunes.....	23
Tabla 4.1 Componentes del sistema de medición.....	44
Tabla 4.2 Pines de conexión del sistema de medición.....	46
Tabla 5.1 Componentes del sistema de control.....	58
Tabla 5.2 Pines de conexión del sistema de control.....	60
Tabla 5.3 Descripción de variables del MPU6050.....	63
Tabla 5.4 Relación de variables del sistema de control.....	69

CAPÍTULO 1

GENERALIDADES DEL PROYECTO

En este capítulo se encuentra la descripción general del proyecto haciendo una pequeña introducción al trabajo que se llevó a cabo, también se definen la problemática a resolver, los objetivos, descripción de la empresa, entre otros asuntos más.

1.1Introducción

Una buena ventilación en los lugares de trabajo tiene mucha importancia para la productividad y salud de las personas que laboran. Esto evita el estrés térmico producido por la acumulación excesiva de calor en nuestro cuerpo. Evita también la transmisión de agentes causantes de enfermedades en el ambiente, y por su puesto provee un ambiente de mayor comodidad mientras el usuario hace uso en su estancia.

Una de las principales premisas que se estiman en el ambiente profesional es que el aire de calidad puede mejorar el rendimiento de los trabajadores. Según los científicos de la Universidad Técnica de Dinamarca, “una buena calidad del aire puede mejorar la productividad en un 10%”. En el Instituto de Energías Renovables de la UNAM se ha propuesto como proyecto la implementación de dos sistemas que colaboren para la mejora de la ventilación tanto de unos de sus cubículos como del auditorio Tonatiuh. El primero es un sistema de medición de apertura de una ventana deslizable que envía la información de la apertura a ThingsBoard, el segundo consiste en un sistema de control de las ventanas del auditorio para abrir a diferentes niveles de apertura e incluso programar horarios y enviar los datos a la plataforma, a fin de poder llevar un control de datos de las veces que se abren tanto la ventana como las ventanas y el nivel de apertura de las mismas así a futuro se podrá calcular el flujo de calor que se mueve dentro de los recintos con el fin de mejorar la calidad del aire para sus trabajadores y colaboradores.

1.2 Descripción de la empresa y del puesto o área del trabajo del estudiante

“Instituto de Energías Renovables - (IER)”

Forma parte del Campus Morelos de la UNAM, está ubicado en el municipio de Temixco, Morelos. Surge como Instituto en enero del 2013 como una transformación del Centro de Investigación en Energía (CIE) y pertenece al Subsistema de la Investigación Científica de la UNAM.

Objetivos Institucionales

- Realizar investigación científica y tecnológica sobre fenómenos, materiales, procesos, dispositivos y sistemas que aprovechen las fuentes renovables de energía y áreas afines, que impulsen el uso racional y eficiente de la energía y la protección al medio ambiente por la sociedad.
- Fomentar y llevar a cabo programas de enseñanza y capacitación de alta calidad en ciencia e ingeniería para impulsar la formación de recursos humanos capaces de generar conocimiento y usar de manera inteligente y sustentable la energía, principalmente en energías renovables y áreas afines, con la visión de conservación y respeto al medio ambiente.
- Fomentar la innovación basada en ciencia y tecnología, la transferencia de tecnologías y la creación de empresas de base tecnológica relacionadas con el aprovechamiento de las fuentes renovables de energía y con el uso racional de los recursos energéticos para impulsar el desarrollo sustentable del país.
- Realizar difusión y divulgación sobre las energías renovables y áreas afines.
- Fomentar actividades de vinculación con organismos públicos, privados y sociales con el fin de incrementar la colaboración.
- Impulsar la colaboración con las entidades académicas de la UNAM para hacer sinergia y generar conocimiento científico y tecnológico para el aprovechamiento integral de las fuentes renovables de energía.

Misión

El Instituto tiene como misión realizar investigación científica básica y aplicada en energía, con énfasis en energías renovables, que coadyuven al desarrollo de tecnologías energéticas sustentables; llevar a cabo estudios, asesorías y capacitación a los distintos sectores de la sociedad; formar recursos humanos especializados, y difundir los conocimientos adquiridos para el beneficio del país.

Visión

Ser un Instituto de investigación con liderazgo académico internacional en investigación en energías renovables y temas afines, que propicie el desarrollo científico y tecnológico y permita su aplicación en la solución de problemas relacionados con los ámbitos de la energía y su impacto al medio ambiente para el desarrollo sustentable del país.

Ubicación

El instituto de Energías Renovables se encuentra ubicado en Priv. Xochicalco S/N, Col. La Azteca, Temixco. Mor. Código Postal 62580

Logotipo de la empresa



Figura 1.1. Logotipo del Instituto (“IER-UNAM”)

1.3 Planteamiento del problema

Actualmente en el IER-UNAM existe la necesidad de mejorar y automatizar el sistema de ventilación de sus cubículos y del auditorio Tonatiuh, a fin de alcanzar en cada uno de ellos el confort térmico. Automatizar un sistema de ventilación tiene como prioridad más que la comodidad de quienes ocupan el espacio la renovación del aire, ya que si el aire no se renueva su calidad disminuye y terminara afectando la salud de las personas. Por ello se requiere la instrumentación necesaria para lograr implementar dos sistemas capaces de medir, obtener y proporcionar datos que ayuden a mejorar el actual sistema de ventilación.

1.4 Objetivos

General

Implementar control del nivel de apertura de una ventana deslizable y del sistema de apertura de las ventanas del auditorio Tonatiuh, usando tecnologías abiertas.

Específicos

- Familiarización con el proyecto
- Diseñar y construir control para la apertura de ventanas
- Diseñar y construir la instrumentación para medir el nivel de apertura de ventanas
- Programar horarios de apertura de ventanas en el sistema de control
- Implementar menú de niveles de apertura
- Publicar datos en ThingsBoard
- Redactar reporte
- Actualizar repositorio

1.5 Justificación

Contar con una buena ventilación es indispensable para cualquier lugar, llámese departamento, casa, oficina, taller, escuela, o cualquier recinto cerrado, ya que esto evitara principalmente agentes causantes de enfermedades, además de ayudar a reducir el estrés térmico causado por el calor acumulado en nuestro cuerpo.

Sin embargo, no es suficiente con solo ventilar un lugar, sino que también es necesario controlar esa ventilación, ya que si se controla se asegura que la renovación del aire sea constante, logrando mejorar su calidad y por lo tanto el cuidado de la salud de las personas.

Es por ello que el presente proyecto tiene como propósito implementar un sistema control de la apertura de las ventanas del auditorio Tonatiuh, así como un sistema de medición de apertura para una ventana deslizable de uno de sus cubículos, con el fin de mejorar los estándares de calidad del aire que hay en ambos lugares, llevando un monitoreo y registro de datos de los factores puedan ayudar a mejorar dicha ventilación.

CAPÍTULO 2

MARCO TEÓRICO

En este capítulo se abordarán los temas principales o bases de este proyecto, proporcionando una breve definición, descripción del tema, y en algunos casos una representación gráfica del mismo.

2.1 FUNDAMENTOS TEÓRICOS

2.1.1 Confort térmico

El ser humano a lo largo de su trayectoria por esta tierra, al experimentar cada uno de los ecosistemas, climas, y estaciones del año, se ha dado a la tarea de descubrir, imaginar y construir nuevos y diferentes tipos de edificaciones, artefactos y parámetros que se adapten mejor al entorno climático donde residen, todo esto, con el fin de poder sentirse lo más cómodo posible y llevar así un mejor estilo de vida.

Dicho lo anterior podemos llamar a esta satisfacción de comodidad que busca el ser humano como “confort térmico”, y no es otra cosa más que la sensación que experimentan las personas cuando no sienten ni frío ni calor con respecto al ambiente en el que viven o desarrollan su actividad, es decir, perciben una sensación de placer con el entorno.

La Norma ISO-7730 define confort térmico como: “*Esa condición de mente en la que se expresa la satisfacción con el ambiente térmico*” [1]. Sin embargo, para lograr llegar a ese estado de confort hay 2 condiciones que se tienen que cumplir primero, los cuales se mencionan a continuación:

- 1.- Neutralidad térmica corporal.
- 2.- Balance de energía corporal.

- *Neutralidad térmica corporal:*

El ser humano tiene un muy eficaz sistema de regulación de temperatura que asegura que la temperatura del centro del cuerpo sea constante en aproximadamente 37° C.

El cuerpo cuenta con 2 reacciones de protección que se activan por los siguientes sensores:

- Hipotálamo: Sensor de calor, este indica cuando la temperatura central del cuerpo ha excedido los 37°C.
- Sensores de la piel: Sensor de frío, indican cuando la temperatura de la piel ha descendido por debajo de los 34°C.

Estos son los encargados de mandar una señal al cuerpo para alertar que ha bajado o subido la temperatura y ha rebasado los estándares normales.

La primera reacción de protección se activa cuando la temperatura interna del cuerpo ha excedido los 37°C, haciendo que el cuerpo comience a sudar, a fin multiplicar la pérdida de calor de cuerpo.

La segunda reacción de protección se activa de manera contraria, cuando la temperatura de la piel ha descendido por debajo de los 34°C, estimulando el movimiento constante de los músculos del cuerpo (tiriteo), logrando una mayor producción de calor corporal.

Por lo tanto, se considera una condición indispensable para la salud, y para la vida, mantener una temperatura corporal dentro de los pequeños límites vitales de la diferencia de 4 o 5° C. Finalmente el hombre califica un ambiente confortable cuando ningún tipo de incomodidad térmica está presente, por lo que la primera condición de confort es la neutralidad térmica, es decir cuando no se siente ni frío ni calor.

- *Balance de energía corporal:*

La segunda condición para el confort térmico es el cumplimiento del balance de energía del cuerpo: “*el calor producido por el metabolismo debe ser igual a la cantidad de calor perdida por el cuerpo*” [2].

Recordemos que al medir el clima interior térmico de un lugar el ser humano no siente el clima de la habitación, sino la perdida de energía de su cuerpo, por lo tanto, los parámetros a medir son todos aquellos que afecten a la perdida de energía del mismo (Tabla 2.1).

Temperatura del aire	°C
Temperatura media radiante	°C
Velocidad del aire	m/s
Humedad	Pa

Tabla 2.1. Parámetros físicos de un recinto.

Claramente la influencia que tienen estos factores respecto a la perdida de energía del cuerpo es de distinta magnitud, sin embargo, para evaluar esta perdida de energía sería insuficiente medir solo uno de ellos. Por lo tanto, es de suma importancia tomar en cuenta los parámetros físicos de un lugar si queremos llegar a tener el confort térmico.

La ventilación de un lugar por su puesto ahora juega un papel muy importante respecto al tema de confort térmico. La británica Florence Nightingale considerada precursora de la enfermería moderna aseguraba que una buena ventilación junto con otros factores como la iluminación, la temperatura, y algunos más, son la base para conseguir un entorno saludable.

Podemos decir que un sistema de ventilación es el método y los elementos que se utilizan para ventilar un lugar cerrado. Actualmente existen 2 tipos de sistemas ventilación y se clasifican de la siguiente manera:

- *Sistemas de ventilación mecánica controlada*

Estos a su vez se subdividen en 2 grupos:

Sistemas de simple flujo: Cuentan con un equipo de ventilación mecánica controlada para extraer el aire contaminado por medio de redes de conductos que lo expulsan al exterior y no controla la entrada de aire.

Sistemas de doble flujo: Estos además de extraer el aire contaminado y expulsarlo al exterior, también se ocupan de tratar el aire que entra en el interior. Son los que más se utilizan en la actualidad, ya que ofrecen una mejor calidad de aire interior. A su vez cuentan con un recuperador de calor que posibilita una reducción en el consumo energético y, en consecuencia, un considerable ahorro económico.

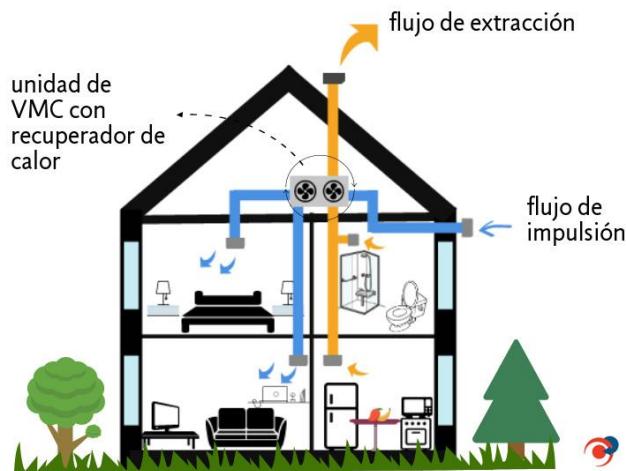


Figura 2.1. Sistema de ventilación de doble flujo.

- *Sistemas de ventilación natural*

Los sistemas de ventilación natural son aquellos cuya técnica es permitir el ingreso de aire exterior dentro de un recinto por medios naturales, es decir no mecánicos.

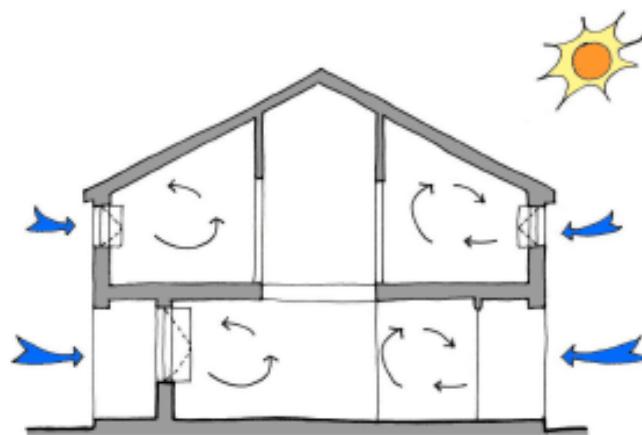


Figura 2.2. Sistema de ventilación natural.

Los sistemas de ventilación mecánica controlada de doble flujo parecieran ser la mejor opción para tener una correcta ventilación en un hogar u oficina sustituyendo a los caros aires acondicionados, sin embargo, para recintos demasiado grandes supondrían un elevado costo para su instalación y mayor aun su consumo energético, es por ello que generalmente a tipo de lugares de gran capacidad se les aplica el sistema de ventilación nocturna.

La ventilación nocturna es aquella que durante las horas de la noche cuando la temperatura ha descendido permite a través de ventanas o ventanas la renovación del aire interno del lugar, ¿Cómo sucede esto?, la masa térmica de un espacio enfriado por la noche absorberá calor durante el día, manteniendo una temperatura fresca a lo largo de su superficie.

En climas secos, con una diferencia de temperatura entre el día y la noche de 11 C° o más, y donde las temperaturas nocturnas de verano caen por lo menos 5.5 C° por debajo de la temperatura interior deseada durante el día, se utiliza el aire fresco de la noche para disipar el calor acumulado en el espacio. De esta manera el recinto se mantendrá fresco durante el día sin el uso de fuentes de energía, contribuyendo a crear condiciones de confort a los usuarios de dicho lugar.

2.1.2 Electrónica y sistemas de control

A lo largo de los años la electrónica se ha vuelto una herramienta indispensable y eficaz para la construcción y el manejo de una inmensa variedad de aparatos electrónicos, que generalmente abundan en cada hogar.

Para el confort térmico no ha sido la excepción, puesto que, a través del diseño, instrumentación y control, es como se han generado diversos aparatos electrónicos que ayudan a una buena ventilación y a mejorar la calidad del aire dentro de un lugar, que van desde un simple ventilador hasta los sistemas de ventilación más modernos como lo puede ser un sofisticado aire acondicionado. La instrumentación electrónica se aplica en la medición y procesamiento de la información proveniente de variables físicas y químicas, a partir de las cuales realiza el monitoreo y control de procesos, empleando dispositivos y tecnologías electrónicas.

Antes de continuar sobre la aplicación de la electrónica en este proyecto definiremos lo que es un sistema de control.

Un sistema de control es aquel que está dedicado a obtener la salida deseada de un sistema o proceso. En la figura 2.3 se muestra la estructura general de un sistema de control.

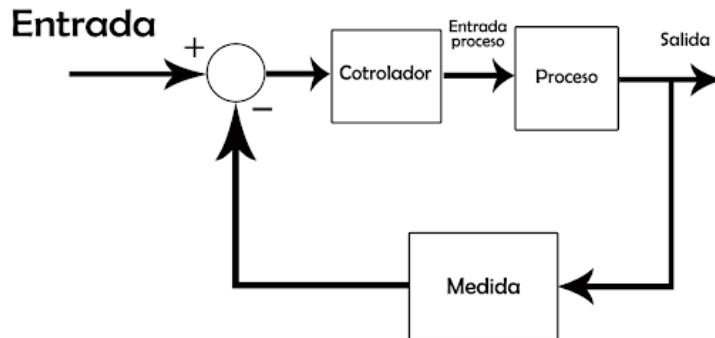


Figura 2.3. Estructura general de un sistema de control.

Existen 2 tipos de sistemas de control y se clasifican de la siguiente manera:

- *Sistema de lazo abierto:*

En este tipo de sistema de control no existe información o retroalimentación sobre la variable a controlar. Es decir, la salida no depende en absoluto de la entrada.

- *Sistema de lazo cerrado:*

Al contrario, en este tipo de sistema de control sí hay información sobre la variable, incluso retroalimentación sobre los estados que va tomando. La información sobre la variable se obtiene mediante el uso de sensores que son colocados de forma estratégica. Los sensores hacen posible que el proceso sea completamente autónomo.

Cuando hablamos que un sistema de control está con miras a ser completamente autónomo, nos introducimos al tema de “automatización” o “sistema de control automatizado”, que en esencia es la operación resultante de la interacción de 2 tipos de componentes en un sistema: los sensores y actuadores.

**Sensores*: son los sentidos del sistema de control, le proporcionan información sobre lo que está ocurriendo.

Un sensor es todo aquello que tiene una propiedad sensible a una magnitud del medio, y al variar esta magnitud también varía con cierta intensidad la propiedad, es decir, manifiesta la presencia de dicha magnitud, y también su medida. En la figura () se muestra la clasificación de los tipos de sensores.

Clasificación de los sensores



Figura 2.4 Clasificación de los tipos de sensores.

Sensor ultrasónico - HC-SR04

Existen por lo menos más de 3 tipos de sensores para medir distancia, entre ellos está el sensor ultrasónico HC-SR04, mismo que por sugerencia del Dr. Guillermo Barrios del IER fue seleccionado para integrar el sistema de medición de apertura de la ventana deslizable.

El sensor HC-SR04 es un sensor de distancia que utiliza ultrasonido (ondas ultrasónicas) para determinar la distancia de un objeto en un rango de 2 cm a 4m. Destaca por su pequeño tamaño, bajo consumo energético, buena precisión y excelente precio.



Figura 2.5 Sensor Ultrasónico HC-SR04.

Especificaciones técnicas:

- Voltaje de operación: 5V DC
- Corriente de reposo: < 2mA
- Corriente de trabajo: 15mA
- Rango de medición: 2cm a 400cm
- Precisión: +- 3mm
- Ángulo de apertura: 15°
- Frecuencia de ultrasonido: 40KHz
- Duración mínima del pulso de disparo TRIG (nivel TTL): 10 μ s
- Duración del pulso ECO de salida (nivel TTL): 100-25000 μ s
- Dimensiones: 45mm x 20mm x 15mm
- Tiempo mínimo de espera entre una medida y el inicio de otra 20ms (recomendable 50ms)

Pines de conexión:

- VCC (5V DC)
- TRIG (Disparo del ultrasonido)
- ECHO (Recepción del ultrasonido)
- GND (0V)

Funcionamiento:

El sensor HC-SR04 posee dos transductores: un emisor y un receptor piezoeléctricos, además de la electrónica necesaria para su operación, el funcionamiento del sensor es el siguiente:

El emisor piezoeléctrico emite 8 pulsos de ultrasonido (40KHz) luego de recibir la orden en el pin TRIG, las ondas ultrasónicas viajan en el aire y rebotan al encontrar un objeto, el sonido de rebote es detectado por el receptor piezoeléctrico y el pin ECHO cambia a alto (5V). Ver figura () para comprender mejor el funcionamiento.

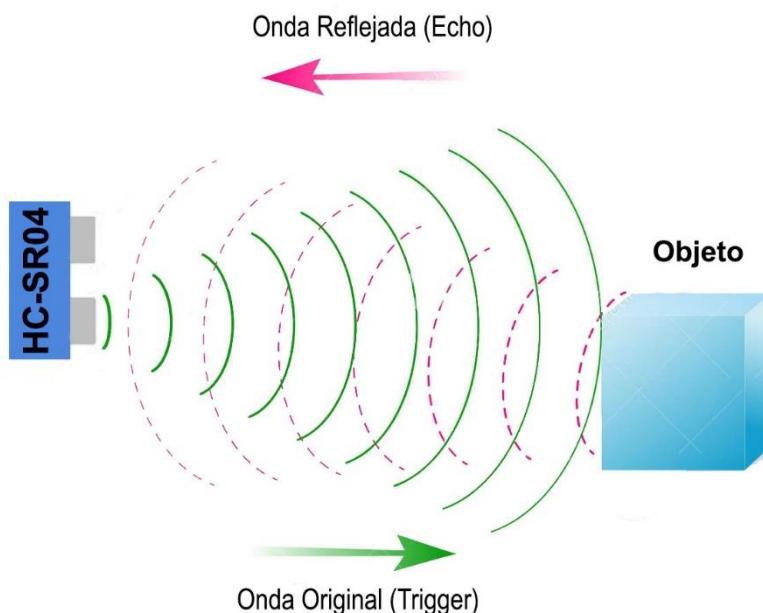


Figura 2.6 Funcionamiento del sensor HC-SR04.

Sensor de vibración - SW420

Este módulo es un circuito que consta de un sensor de vibración SW-420 en conjunto con el OPAMP LM393. Esta constituido interiormente por un resorte y un pequeño poste en su interior, por lo que cada vez que el sensor sea sometido a una vibración o golpe su salida se verá afectada. Al estar encapsulado presenta cierta resistencia al polvo y al agua, por lo que puede ser utilizado en entornos industriales.

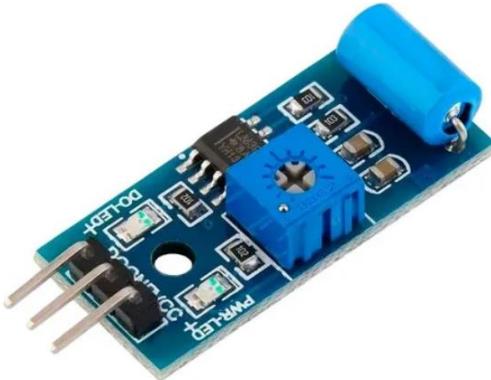


Figura 2.7 Sensor de vibración SW420.

Especificaciones técnicas:

- Consumo: 15mA.
- Voltaje de operación: 3.3V – 5V.
- Salida digital.
- Comparador: LM393.
- Dimensiones: 3.2 x 1.4cm
- Forma de salida: salida digital (0 y 1).

Pines de conexión:

- VCC (3.3V – 5V DC)
- GND (0V)
- DO (Señal de salida)

Funcionamiento:

El sensor de vibración SW- 420 en conjunto con el OPAMP LM393 detectan si hay alguna vibración más allá del umbral fijado. La sensibilidad puede ser ajustada a través del potenciómetro tipo preset que está integrado en el módulo.

Gracias al comparador LM393 integrado, la tarjeta tiene una salida digital, la salida digital se puede usar para leer a través de una tarjeta de desarrollo o microcontrolador.

Acelerómetro - MPU6050

Para el sistema de control de ventilas se ha ocupado un módulo, mismo que está basado en el sensor MPU6050 y contiene todo lo necesario para medir movimiento en 6 grados de libertad, combinando un giroscopio de 3 ejes y un acelerómetro de 3 ejes en un mismo chip.

Integra un DMP (Procesador digital de movimiento) capaz de realizar complejos algoritmos de captura de movimiento de 9 ejes. Se comunica a través de una interfaz I2C y posee una librería muy difundida para su uso inmediato. Este sensor incorpora un regulador de tensión a 3.3V y resistencias pull-up para su uso directo por I2C.



Figura 2.8 Acelerómetro MPU6050.

Especificaciones técnicas:

- Sensor: MPU6050
- Voltaje de operación: 3V/3.3V~5V DC
- Regulador de voltaje en placa
- Grados de libertad (DoF): 6
- Rango Acelerómetro: 2g/4g/8g/16g
- Rango Giroscopio: 250Grad/Seg, 500Grad/Seg, 1000Grad/Seg, 2000Grad/Seg
- Sensibilidad Giroscopio: 131 LSBs/dps
- Interfaz: I2C
- Conversor AD: 16 Bits (salida digital)
- Tamaño: 2.0cm x 1.6cm x 0.3cm

Pines de conexión:

- VCC (3.3V DC)
- GND (0V)
- SCL (Reloj serial)
- SDA (Datos en serie)
- XDA (Datos en serie auxiliar)
- XCL (Reloj serial auxiliar)
- ADO (Puerto de protocolo I2C)
- INT (Interruptor)

Funcionamiento:

I2C es un protocolo de dos cables para comunicarse entre dispositivos. A nivel físico consta de 2 cables: SCL y SDA, el reloj y las líneas de datos respectivamente, mismo que utiliza para comunicarse este módulo.

EL MPU6050 contiene un giroscopio de tres ejes con el que podemos medir velocidad angular y un acelerómetro también de 3 ejes con el que medimos los componentes X, Y, Z de la aceleración, el acelerómetro trabaja sobre el principio piezo eléctrico, posee además de un sensor de temperatura.

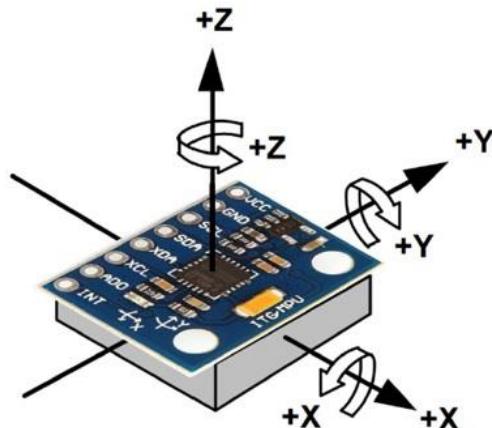


Figura 2.9 Ejes en el MPU6050.

Los acelerómetros internamente tienen un MEMS (Micro Electro Mechanical Systems) que de forma similar a un sistema masa-resorte permite medir la aceleración, también tiene un conversor ADC de 16 bits que convierte los datos a un valor digital.

**Actuadores*: son las manos del sistema de control, le permiten modificar lo que ocurre en la planta.

Un actuador es un dispositivo que convierte la energía en movimiento o que se utiliza para aplicar fuerza. El dispositivo toma energía de una determinada fuente que puede ser energía creada por aire, líquido o electricidad y la convierte en el movimiento deseado. Los dos tipos de movimiento básico deseados son lineal y rotativo, pero también es común el movimiento oscilatorio. En la figura 2.10 se muestra su clasificación.

Clasificación de los actuadores

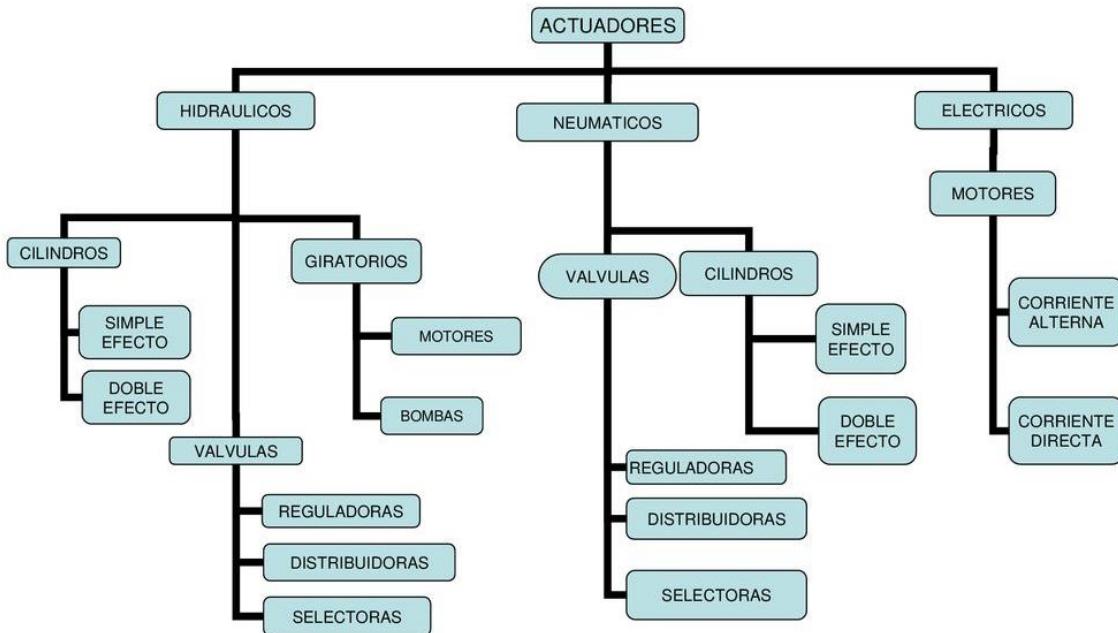


Figura 2.10 Clasificación de los tipos de actuadores.

Los componentes electrónicos están relacionados con las señales eléctricas y sus variaciones, y tienen como finalidad captar información, transmitirla, o utilizarla para gestionar y controlar numerosas aplicaciones, forman parte de cualquier circuito electrónico y se diseñan para ser conectados entre sí, normalmente mediante soldadura, se clasifican según su funcionamiento, estructura física, material base de fabricación y tipo de energía.

Clasificación de los componentes electrónicos

Según su funcionamiento:

- **Pasivos:** Son todos aquellos que no proporcionan ganancia, pero sí consumen energía eléctrica para realizar una determinada función.
- **Activos:** Estos son capaces de generar, modificar o amplificar una señal eléctrica; es decir, aquellos que aportan y proporcionan ganancia, o el control de las señales eléctricas.

Según el tipo energía:

- **Electromagnéticos:** Son aquellos que aprovechan las propiedades electromagnéticas de los materiales, fundamentalmente se relacionan a los transformadores por su tipo de núcleo e inductores.
- **Electroacústicos:** Estos transforman la energía acústica en eléctrica y viceversa como son los micrófonos, altavoces, bocinas y auriculares.
- **Optoelectrónicos:** Transforman la energía lumínica en eléctrica y viceversa como los LED, células fotoeléctricas o fotoceldas.

Según su estructura física:

- **Discretos:** Es un componente electrónico con sólo un elemento eléctrico, puede ser activo o pasivo e incorporar múltiples funciones, se utilizan en las áreas que requieren altos voltajes o potencias.
- **Integrados:** Estos pueden contener desde unos pocos componentes discretos hasta millones de ellos. Son los denominados circuitos integrados, como por ejemplo un amplificador operacional.

Según el material base de fabricación:

- **Semiconductores:** Son aquellos cuyos no son buenos conductores ni buenos aisladores, una de las aplicaciones importantes es la de dirigir o controlar el flujo de la corriente eléctrica.
- **No semiconductores:** Son aquellos componentes que solo cumplen una función, aunque se les agregue sustancias químicas siempre serán solo conductores o solo aisladores.

Amplificador operacional

Los amplificadores operaciones entran en la clasificación de los integrados según su estructura física.

Un amplificador operacional o también llamado OPAMP es un circuito integrado que tiene como principal función amplificar el voltaje con una entrada de tipo diferencial para tener una salida amplificada y con referencia a tierra.

Aspectos básicos de un OPAMP:

- Un amplificador operacional tiene dos entradas, invertida (-) y no invertida (+), y una salida.
- La polaridad de una señal aplicada a la entrada invertida se invierte a la salida. Una señal aplicada a la entrada no invertida mantiene su polaridad a la salida.
- El grado de amplificación o ganancia (G) de un amplificador operacional está determinada por una resistencia de retroalimentación que alimenta parte de la señal amplificada de la salida a la entrada invertida, también es llamada ganancia de lazo cerrado.

La salida es la diferencia de las dos entradas multiplicada por la ganancia:

$$V_{out} = G (V_+ - V_-)$$

La ganancia está determinada por la fórmula:

$$A_v = -V_{out} / V_{in}$$

*El signo negativo indica que la señal en la salida será la opuesta a la entrada.

Configuraciones típicas:

Amplificador inversor

En este circuito, la entrada V_+ (+) está conectada a masa y la señal se aplica a la entrada V_- (-) a través de R_1 , con realimentación desde la salida a través de R_2 . La entrada V_- (-) es un punto de tierra virtual, ya que está a un potencial cero.

El amplificador inversor amplifica e invierte una señal 180° , es decir, el valor de la tensión de salida está en oposición de fase con la de entrada y su valor se obtiene al multiplicar la tensión de la entrada por una ganancia fija constante, establecida por la relación entre R_2 y R_1 , resultando invertida esta señal.

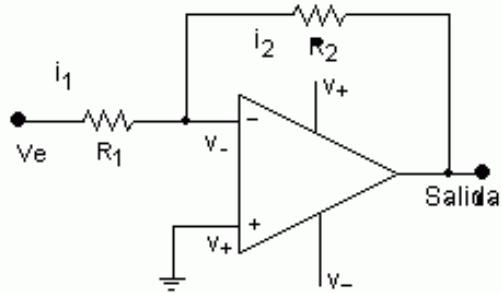


Figura 2.11 Amplificador inversor.

$$\begin{aligned}V_- &= V_+ = 0 \\i_1 &= i_2; i_1 + i_2 = 0 \\(V_e - V_-) / R_1 + (V_0 - V_-) / R_2 &= 0 \\&; \\V_e / R_1 + V_0 &= 0 \\V_0 &= -R_2 / R_1\end{aligned}$$

Amplificador No inversor

Este es el caso en que la tensión de entrada V_e , está en fase con la de salida V_s , esta tensión de salida, genera una corriente a través de R_2 hacia el terminal inversor, a su vez a través de R_1 , se genera una corriente hacia el mismo terminal, pero de signo contrario, por lo que ambas corrientes se anulan, reflejando en la salida la tensión de entrada amplificada.

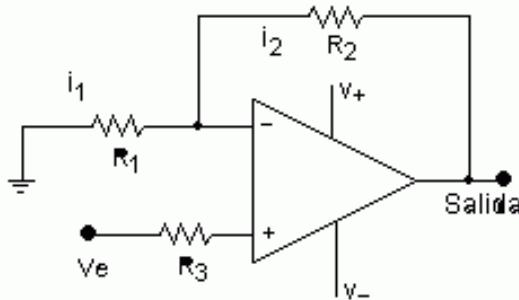


Figura 2.12 Amplificador no inversor.

Según se ha mencionado antes, el valor de $+V_e$ se refleja en la entrada inversora $-V_e$ del amplificador operacional y teniendo en cuenta que se considera un «cortocircuito virtual», podemos establecer que $i_e = V_e/R_1$. Y como la corriente en la entrada inversora $i_- = 0$; $i_1 = i_2$; por lo tanto:

$$V_o = (R_1 + R_2) i_1, \text{ sustituyendo; } V_o / V_e = (1 + R_2 / R_1);$$

y $V_o = (1 + R_2 / R_1) V_e$, finalmente la ganancia en tensión es:

$$A_v = (1 + R_2 / R_1)$$

OPAMP LM324N

Para este proyecto se ha seleccionado el OPAMP LM324N.

El circuito integrado LM324N es un amplificador operacional cuádruple con entradas diferenciales verdaderas. Tiene ventajas sobre los amplificadores operacionales convencionales en aplicaciones de fuente sencilla de alimentación, ya que puede trabajar con voltajes de alimentación desde 3V hasta 32V.

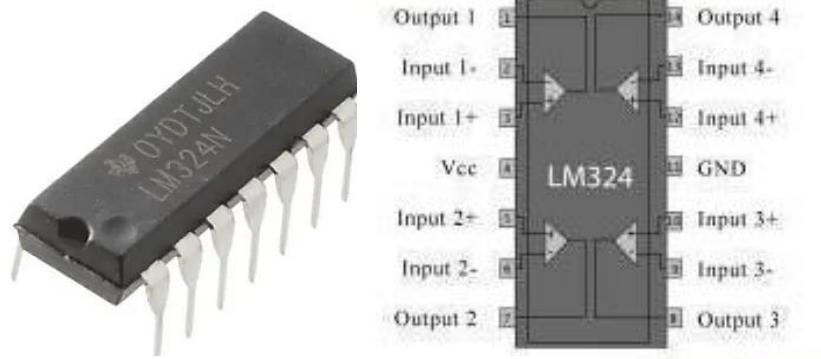


Figura 2.13 OPAMP LM324N.

Características principales:

- Internamente compensado en frecuencia para ganancia unidad
- Alta ganancia en DC (100 dB)
- Gran ancho de banda (ganancia unidad) 1MHz (compensada con la temperatura)
- Alimentación simple: entre 3V y 32V
- Alimentación doble: entre +/- 1,5V y +/- 16V
- Consumo de corriente muy bajo ($700 \mu\text{A}$) independiente de la alimentación
- Muy baja corriente de polarización de entrada (45 nA) (compensado con la temperatura)
- Bajo offset de voltaje de entrada (2mV) y offset de corriente (5 nA)
- El rango de voltaje de entrada en modo común incluye masa.
- El rango de voltaje diferencial en la entrada es igual al voltaje de alimentación.
- Excursión máxima del voltaje de salida: desde 0V hasta $V_+ - 1,5V$

El uso de la electrónica en los sistemas de control es un gran recurso debido a la gran variedad de componentes electrónicos que existen. En la siguiente tabla se mencionan algunos de los componentes electrónicos más comunes junto con una pequeña descripción:

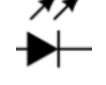
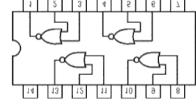
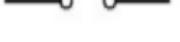
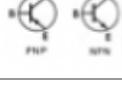
Componente	Como se ve	Función	Simbología esquemática
Resistencia		Limita la cantidad de corriente que fluye por el circuito (Ω)	
Potenciómetro		Limita la cantidad de corriente que fluye por el circuito en un valor variable (Ω)	
Capacitor		Almacena temporalmente corriente eléctrica de un circuito (F)	
Diodo		Se utiliza una válvula de un solo sentido para el flujo de la corriente eléctrica del circuito	
LED		Se utiliza como un recurso de luz y también funciona como indicador en displays alpha numéricos	
Circuito integrado		Combina e integra de forma miniaturizada otros componentes en un pequeño paquete o chip	
Cable		Se utiliza para llevar una corriente eléctrica	
Botón pulsador		Es un operador que, cuando se oprime, permite el paso de la corriente eléctrica y, cuando se deja de oprimir, lo interrumpe.	
Transistor		Se utiliza para amplificar las señales eléctricas	
Pantalla LCD		Muestra en pantalla una cadena de letras o números según su programación	

Tabla 2.2 Componentes electrónicos más comunes.

2.1.3 Internet de las cosas (IoT)

Los avances tecnológicos no se detienen, y el tiempo tampoco, es por ello que todas las generaciones han tenido que adaptarse a la época que les tocó, los de las décadas pasadas se acostumbraron a la introducción de los primeros electrodomésticos y a ver a uno que otro de manera muy esporádica con un teléfono celular, la época actual, a ver de manera muy extraña al que no tiene uno de ellos, inevitablemente el mundo de lo digital ha invadido nuestro mundo, cada vez con mayor fuerza y difícilmente se detendrá.

Internet de las Cosas es una traducción de la expresión en inglés “Internet of Things” (IoT), que describe un escenario en el que diversas cosas están conectadas y se comunican entre sí. Esta innovación tecnológica tiene como objetivo conectar los ítems que usamos diariamente a internet, con el objetivo de aproximar cada vez más el mundo físico al digital.

En pocas palabras “Internet de las cosas” no es más que objetos conectados entre sí por medio de la red (figura 5). Estos intercambian información para facilitar o crear diversas acciones. Para que algo así pueda ocurrir hay un conjunto de tres factores que necesitan ser combinados para que una aplicación funcione dentro del concepto de Internet de las Cosas.

Estos son: dispositivos, la red y un sistema de control.



Figura 2.14 Representación gráfica de IoT.

Historia del internet de las cosas

El término “Internet de las Cosas” fue acuñado por el empresario Kevin Ashton, uno de los fundadores del Centro Auto-ID del MIT. Ashton era parte de un equipo que descubrió cómo vincular objetos a la Internet a través de una etiqueta RFID. Él utilizó por primera vez la frase “Internet de las Cosas” en una presentación en 1999 – y se ha arraigado entre nosotros desde entonces.

Ashton puede haber sido el primero en utilizar el término Internet de las Cosas, pero el concepto de los dispositivos conectados – en particular máquinas conectadas – se ha venido usando por algún tiempo. Por ejemplo, las máquinas se han comunicado entre sí desde que se crearon los primeros telégrafos eléctricos a finales de la década de 1830.

Luego en 1982, una máquina modificada de Coca-Cola en la Universidad Carnegie Mellon se convirtió en el primer aparato inteligente conectado. Utilizando la red Ethernet local de la universidad, o ARPANET – precursor de la Internet actual –, los estudiantes podían saber qué bebidas había en existencia y si éstas estaban frías. [1]

Dispositivos IoT

Como bien se mencionó en la definición de internet de las cosas, el primer elemento para que una aplicación sea llamada con ese nombre son los dispositivos. Un dispositivo IoT consiste en un objeto al que se le ha dotado de conexión a Internet y cierta inteligencia software, sobre el que se pueden medir parámetros físicos o actuar remotamente y que por tanto permite generar un ecosistema de servicios alrededor del mismo.

Debido al gran impacto que IoT ha causado en el sector tecnológico Espressif Systems, una compañía de semiconductores con sede en Shanghái ha lanzado 2 de los microcontroladores Wifi más adorables que se están abriendo paso en los lugares destacados de la moda IoT y tan solo del tamaño de una caja de cerillos, el ESP8266 y su hermano mayor el ESP32, mismos utilizados en este proyecto como dispositivos IoT.

ESP8266

El ESP8266 es un SoC de las siglas (system on chip) fabricado por la compañía china Espressif. Este SoC agrupa distintos componentes en un mismo integrado, siendo los principales un procesador de 32 bits y un chip Wifi con gestión de pila TCP/IP. En resumen, el ESP8266 es un chip que integra en un encapsulado un procesador de propósito general con conectividad Wifi completa.

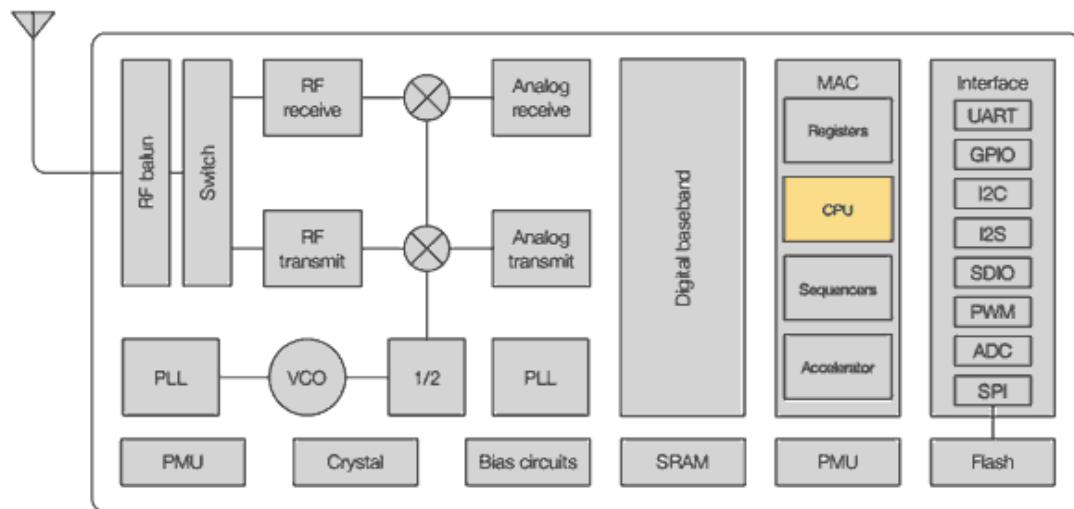


Figura 2.15 Diagrama de bloques del ESP8266.

El procesador integrado en el ESP8266 es un Tensilica L106 de 32-bits con arquitectura RISC que funciona a una velocidad de 80Mhz, con una velocidad máxima de 160Mhz.

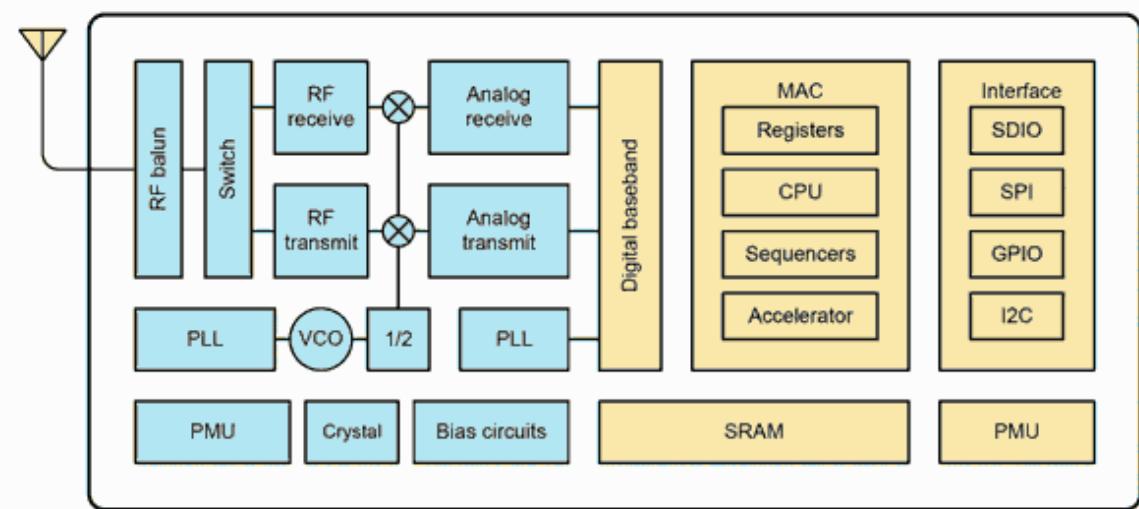


Figura 2.16 Diagrama de bloques – Tensilica L106.

Características del ESP8266

Algunas de las características más sobresalientes de este SoC son:

- Procesador de 32 bit de bajo consumo
- Velocidad de 80MHz (máximo de 160MHz)
- 32 KiB RAM instrucciones, 32 KiB RAM cache
- 80 KiB RAM para datos de usuario
- Memoria flash externa hasta 16MiB
- Pila de TCP/IP integrada
- Wifi 802.11 b/g/n 2.4GHz (soporta WPA/WPA2)
- Certificado por FCC, CE, TELEC, WiFi Alliance y SRRC
- 16 pins GPIO
- PWM en todos los pines (10 bits)
- Conversor analógico digital de 10 bits
- UART (2x TX y 1x RX)
- SPI, I2C, I2S
- Voltaje de operación 3.0 a 3.6V
- Consumo medio 80mA
- Modo consumo stand-by (1mW) y deep sleep (1uA).



Figura 2.17 SoC ESP8266.

En comparación con Arduino donde hay una compañía que pone cierto orden en los diseños y modelos disponibles de su empresa, con el ESP8266 no ha sido así, ya que las placas de desarrollo que integran este SoC han evolucionado de la mano de distintos fabricantes, entre ellos la muy conocida NodeMCU.

NodeMCU es una plataforma IoT de código abierto, que incluye el firmware que se ejecuta en el SoC Wi-Fi ESP8266 de Espressif Systems y el hardware que se basa en el módulo ESP-12. En este proyecto se hará uso de la placa ESP8266 V1.0 /V2 de NodeMCU.

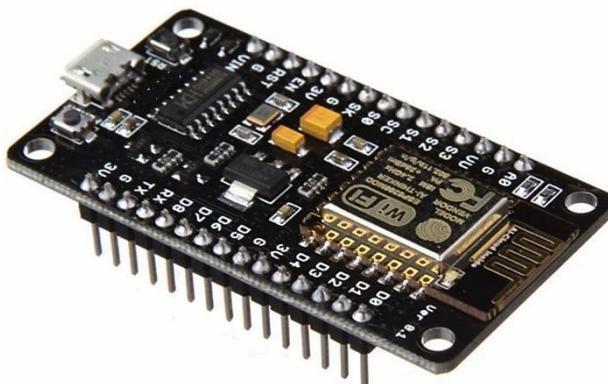


Figura 2.18 ESP8266 NodeMCU.

Pin-OUT del ESP8266 - NodeMCU

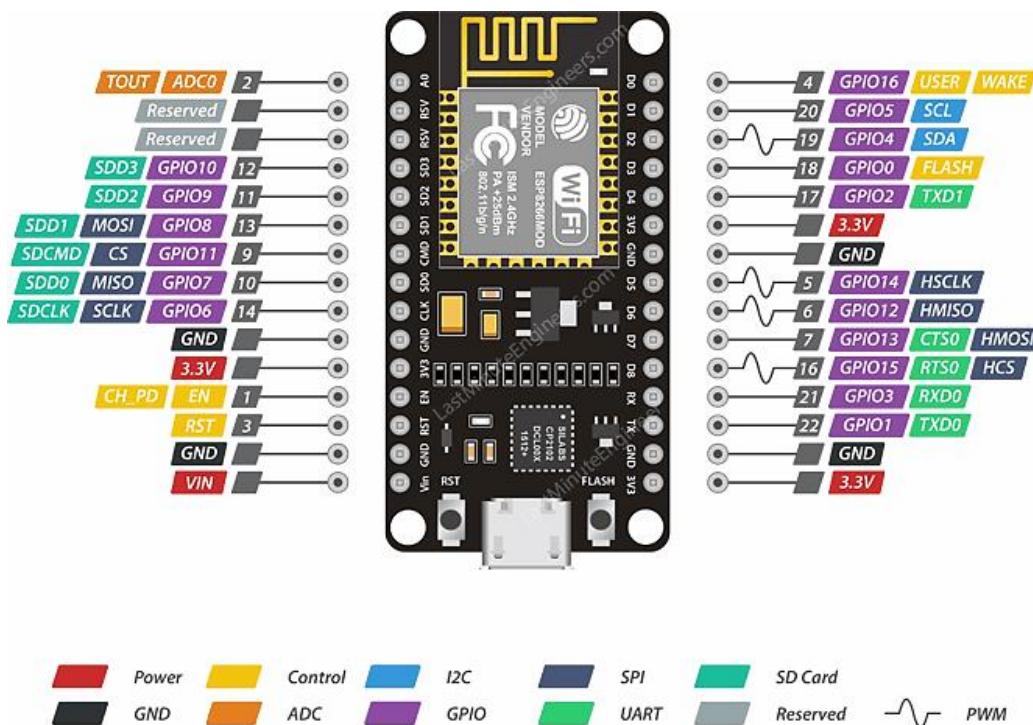


Figura 2.19 ESP8266 PinOUT.

ESP32

Este microcontrolador fabricado por la misma compañía china es llamado el “hermano mayor” del Esp82266, ya que contiene un procesador más potente y de doble núcleo. La siguiente ventaja evidente es que incorpora Bluetooth BLE, además de Wifi. Además, el ESP32 incorpora más memoria, encriptado de la Flash, arranque seguro, encriptación por hardware, generador de números aleatorios y reloj de tiempo real (RTC).

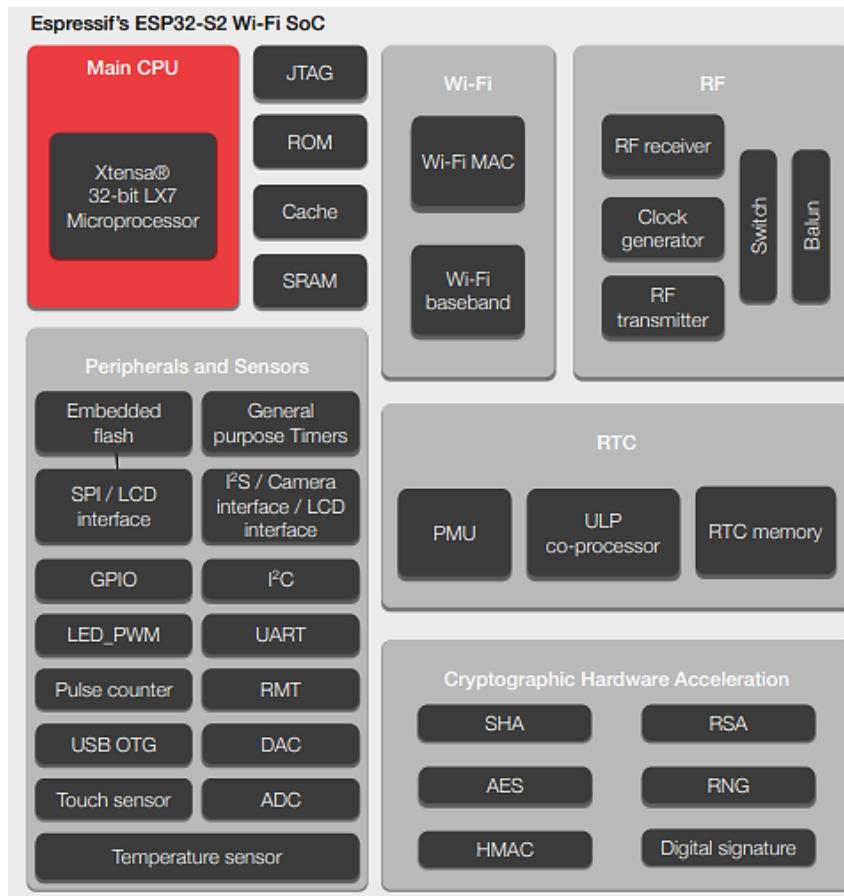


Figura 2.20 Diagrama de bloques del ESP32.

Integra en un único chip un procesador Tensilica Xtensa de doble núcleo de 32bits a 160Mhz (con posibilidad de hasta 240Mhz). En resumen, es una máquina muy interesante y tiene un potencial enorme para elaborar todo tipo de proyectos, sobre todo por su capacidad de comunicación, ocupando un lugar destacado en aplicaciones de IoT.

Características del ESP32

Algunas sus características más sobresalientes son:

- Procesador Xtensa LX6 de 32 bits de doble núcleo
- Velocidad de 160Mhz (máximo 240 MHz)
- Co-procesador de ultra baja energía
- Memoria 520 KiB SRAM
- Memoria flash externa hasta 16MiB
- Encriptación de la Flash
- Arranque seguro
- Pila de TCP/IP integrada
- Wifi 802.11 b/g/n 2.4GHz (soporta WFA/WPA/WPA2/WAPI)
- Bluetooth v4.2 BR/EDR y BLE
- Criptografía acelerada por hardware
- 32 pin GPIO
- Conversor analógico digital (ADC) de 12bits y 18 canales
- 2 conversores digitales analógico (DAC) de 8bits
- 16 salidas PWM (LED PWM)
- 1 salida PWM para motores
- 11 conversor analógico a digital de 10 pin
- 10x sensores capacitivos (en GPIO)
- 3x UART, 4x SPI, 2x I2S, 2x I2C, CAN bus 2.0
- Controladora host SD/SDIO/CE-ATA/MMC/eMMC
- Controladora slave SDIO/SPI
- Sensor de temperatura
- Sensor de efecto Hall
- Generador de números aleatorios
- Reloj tiempo real (RTC)
- Controlador mando a distancia infrarrojos (8 canales)

NodeMCU también ha decidido incluir su firmware junto a este poderoso SoC y añadirlo como nueva placa de desarrollo en sus modelos para IoT. En este proyecto se hará uso de la placa ESP32 DEVKIT V1 de NodeMCU.



Figura 2.21 ESP32 NodeMCU.

Pin-OUT del ESP32 - NodeMCU

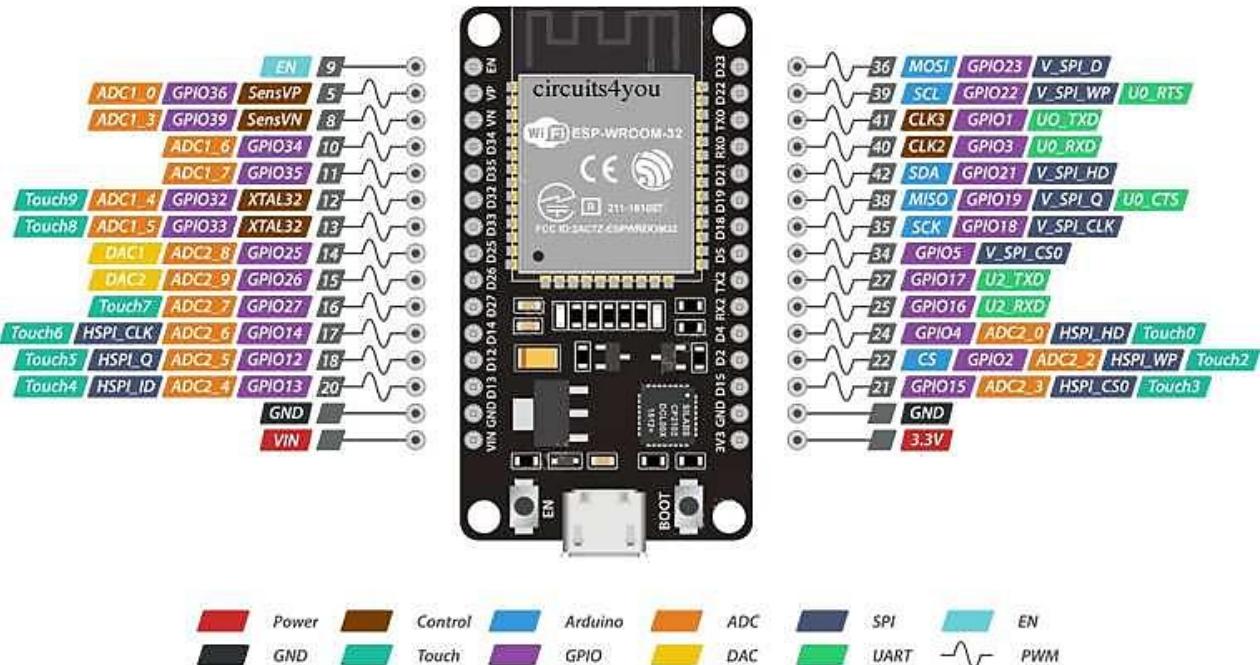


Figura 2.22 ESP32 PinOUT.

En cuanto a lenguajes de programación compatibles tanto en el ESP8266 y ESP32 hay varias opciones, es posible emplear el IDE de Arduino, RTOS, Mongoose OS, Espruino y MicroPython. En este proyecto se hará uso del ultimo mencionado, “MicroPython”.

según el sitio oficial: “*MicroPython es un pequeño pero eficiente interprete del Lenguaje de Programación Python 3 que incluye un subconjunto mínimo de librerías y que además está optimizado para que pueda correr en microcontroladores y ambientes restringidos*” [5].

A continuación, se mencionan algunas de las características que han hecho que este lenguaje de programación sea considerado mejor que otros:

- *RELP Interactiva*

(Read-Eval-Print Loop Por sus siglas en Ingles). Que es un pequeño programa que lee e interpreta los comandos del usuario, los evalúa y después imprime el resultado. Esto permite conectar alguna tarjeta (microcontrolador que soporte Python) y esta tiene que ejecutar el código sin necesidad de compilar ni cargar el programa.

- *Muchas librerías*

Así como Python cuenta con un sin fin de librerías para la ejecución de tareas, MicroPython también viene bien cargado con bastantes paquetes para ahorrar trabajo. Es posible ejecutar análisis de datos JSON desde un servicio web, búsqueda de texto en expresiones regulares o hasta levantar un Socket dentro de una red tan solo con las funciones ya precargadas.

- *Extensibilidad*

Para usuarios avanzados de MicroPython, pueden extender de Python a funciones de más bajo nivel como C o C++, pudiendo mezclar códigos que requieran de ejecución más rápida a bajo nivel con MicroPython que es de más alto nivel.

Sin embargo, ¿qué sería un dispositivo con modulo wifi, excelentes características de funcionamiento y bien programado sino está conectado a la internet?, es por ello que el segundo elemento necesario para una aplicación IoT es la conexión a la red.

Ya que se ha usado un dispositivo y un lenguaje de programación “open source”, es decir de código abierto, la plataforma que se encargará de la interacción con la red en este proyecto no será la excepción, nos referimos a “ThingsBoard”.

Según el sitio oficial: “*ThingsBoard es una plataforma de IoT de código abierto para la recopilación, el procesamiento, la visualización y la gestión de dispositivos de datos*” [5].

La plataforma permite un rápido desarrollo, gestión y escalado de proyectos de IoT, su principal objetivo es permitir la conectividad de dispositivos a través de protocolos de IoT estándar de la industria: MQTT, CoAP y HTTP, y admite implementaciones tanto en la nube como en las instalaciones. ThingsBoard combina escalabilidad, tolerancia a fallas y rendimiento para que nunca pierda los datos.

Características principales:

❖ *Visualización de datos*

Proporciona más de 30 widgets configurables listos para usar y la capacidad de crear sus propios widgets con el editor incorporado. Gráficos de líneas integrados, medidores digitales y analógicos, mapas y mucho más.

❖ *Gestión de dispositivos*

Proporciona la capacidad de registrar y administrar dispositivos. Permite monitorear los atributos del dispositivo del lado del cliente y del lado del servidor. Proporciona API para que las aplicaciones del lado del servidor envíen comandos RPC a los dispositivos y viceversa.

❖ Seguridad

Admite el cifrado de transporte para los protocolos MQTT y HTTP (s). Admite la autenticación de dispositivos y la gestión de credenciales de dispositivos.

❖ Personalización e integración

Es posible ampliar la funcionalidad de la plataforma predeterminada utilizando cadenas de reglas, widgets e implementaciones de transporte personalizables. Además de la compatibilidad con MQTT, CoAP y HTTP, los usuarios de ThingsBoard pueden utilizar sus propias implementaciones de transporte o personalizar el comportamiento de los protocolos existentes.

❖ 100% de código abierto

ThingsBoard tiene la licencia Apache License 2.0, por lo que puede usarlo en sus productos comerciales de forma gratuita. Incluso puede alojarlo como una solución SaaS o PaaS.

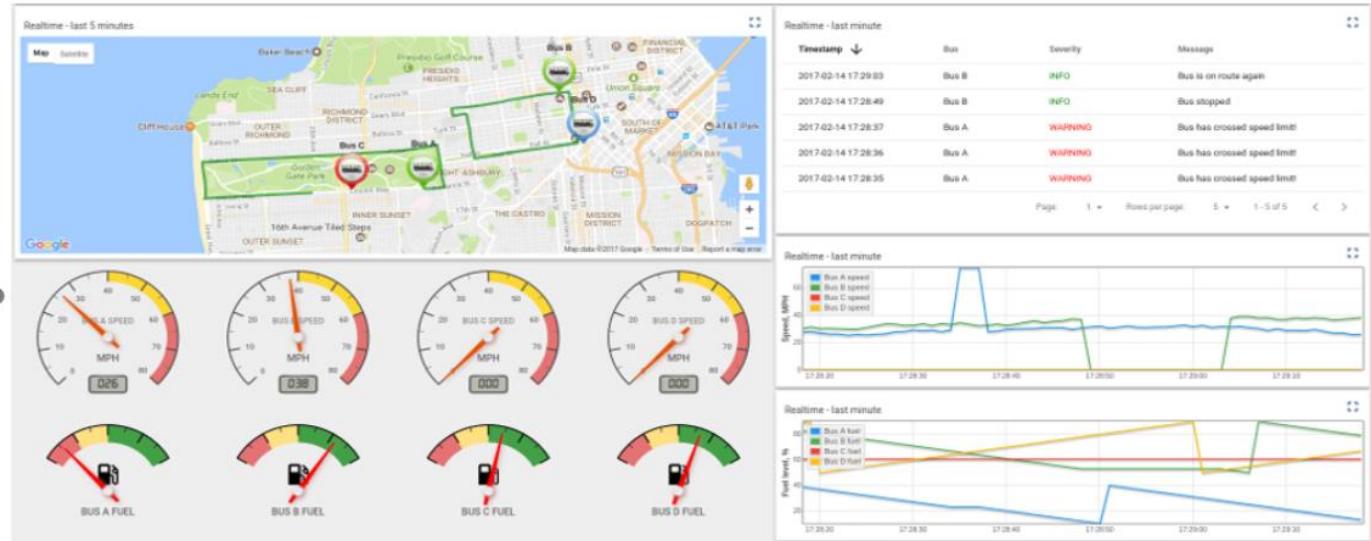


Figura 2.23. Interfaz de ThingsBoard.

CAPÍTULO 3

EDIFICACIÓN

En este capítulo se presentan los dos espacios destinados para la aplicación del proyecto, así como la instrumentación electrónica que se utilizó en cada uno de ellos para llevar a cabo el sistema de control de los mismos.

3.1 CUBÍCULO Y VENTANA DESLIZABLE

El Instituto de Energías Renovables de la UNAM es un espacio dedicado a la investigación científica básica y dedicada a la energía, por lo que la construcción y uso de cubículos como espacios de estudio y trabajo ha sido una excelente opción para este ramo y se ha vuelto un modelo replicado en muchas instituciones educativas y de gobierno.

Es por eso que uno de los cubículos ha sido el primer lugar asignado para el desarrollo del presente proyecto, estos cuentan con ventanas de tipo deslizable (figura 33) y su mecanismo de funcionamiento es el tradicional, una mitad de la ventana se mantiene fija y la restante es la que se abre o cierra deslizándose de manera paralela a la otra.

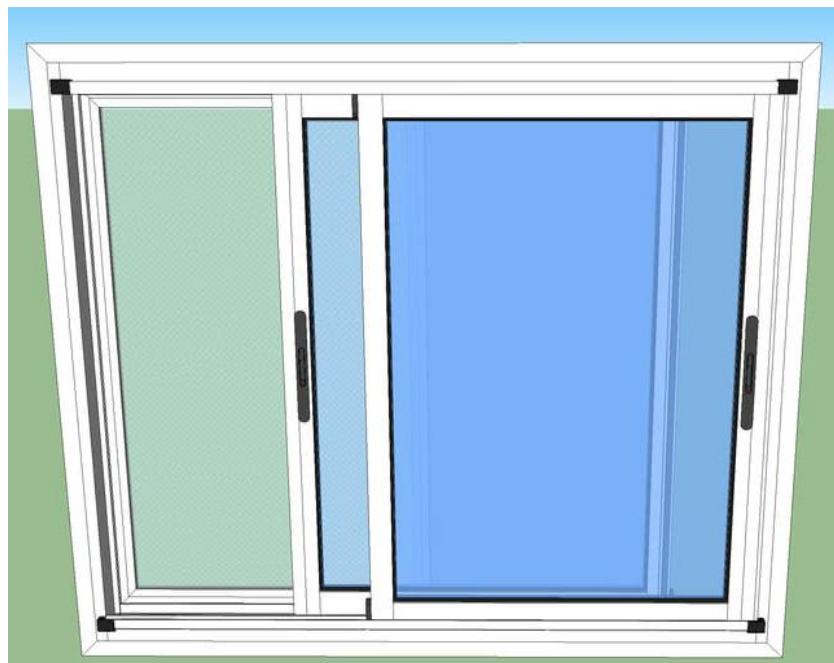


Figura 3.1. Ventana deslizable.

A continuación, se muestra una fotografía tomada a uno de los cubículos del instituto a manera de apreciar las ventanas que tienen instaladas y se ha colocado de la misma manera sus medidas correspondientes.



Figura 3.2. Fachada de cubículo IER-UNAM.



Figura 3.3. Foto de ventana de cubículo IER-UNAM.

Dimensiones:

La ventana del cubículo tiene un tamaño de 1.40 m de largo por 1 m de alto como se muestra en la siguiente figura () .

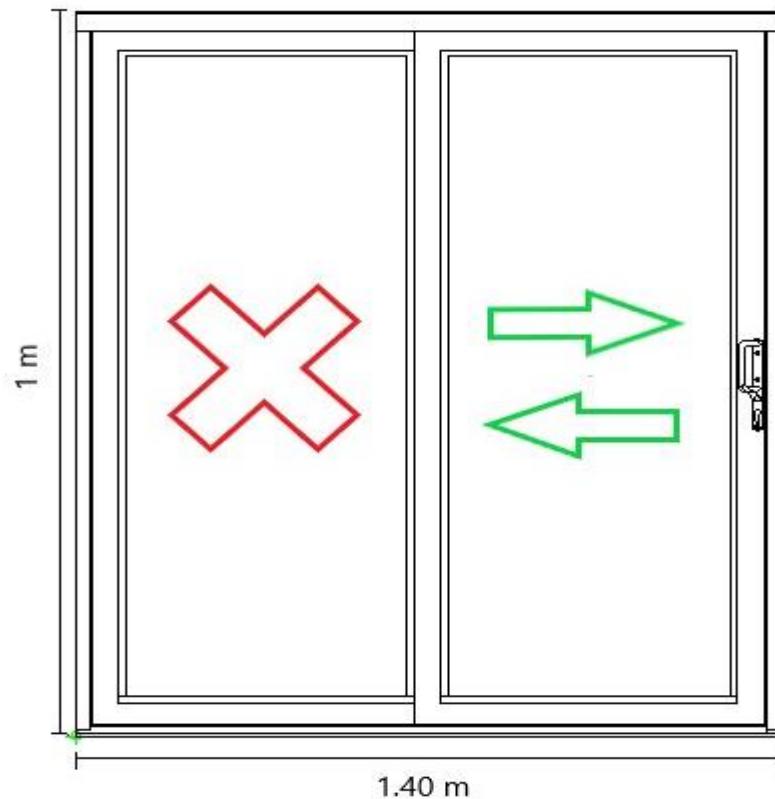


Figura 3.4 Dimensiones de la ventana deslizable.

Nota:

Cabe mencionar que no todas las ventanas de los cubículos restantes del instituto son del mismo tamaño, sin embargo, uno de los retos del proyecto es que la propuesta del sistema de control de la ventana deslizable que se aplique a esta en particular también sea funcional para cualquiera de ellas.

3.2 AUDITORIO Y VENTILAS

El auditorio Tonatiuh está ubicado dentro de las instalaciones del Instituto de Energías Renovables de la UNAM y es el recinto asignado para las conferencias, presentaciones y reuniones de alta importancia. En este lugar se desarrollará el segundo sistema diseñado en este proyecto: El “Sistema de control de ventilas”.

A continuación, se muestra su localización dentro de las instalaciones del instituto:

- 1 Of. Administrativas
2. Of. Administrativas
3. Cubículos
4. Lab. Refrigeración
5. Lab. Termo ciencias
6. Lab. Fotovoltaico 1
7. Lab. Fotovoltaico 2
8. Biblioteca
9. Microscopio Electrónico
10. Auditorio Tonatiuh
11. Cómputo.

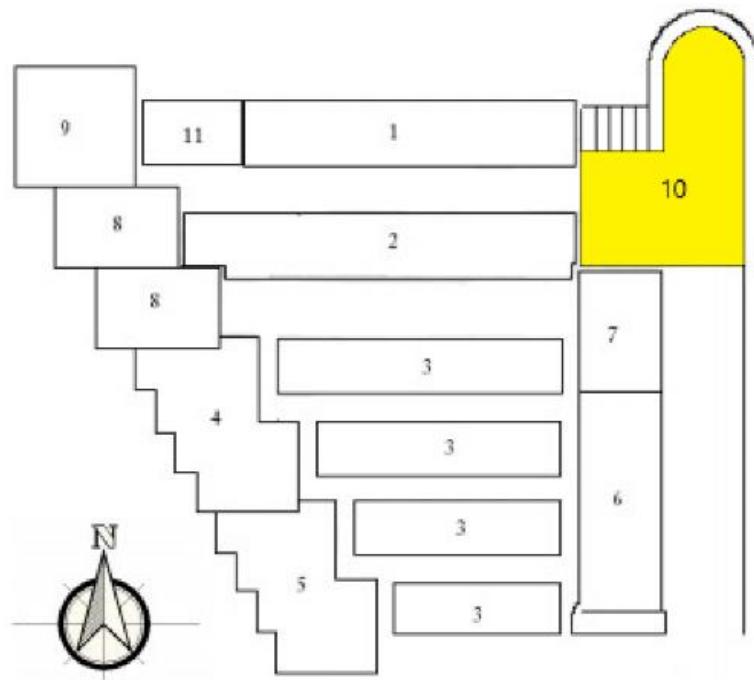


Figura 3.5 Ubicación del auditorio Tonatiuh.

En la figura () se muestran algunos detalles más del auditorio:

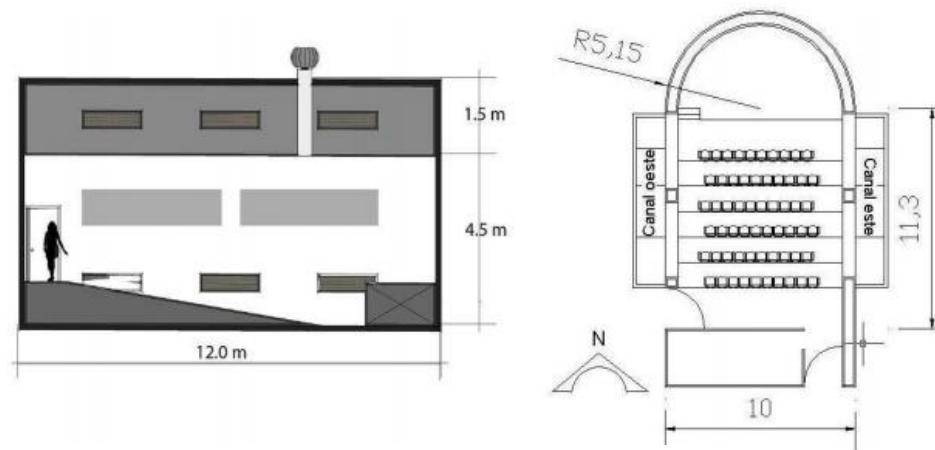


Figura 3.6 Detalles del auditorio Tonatiuh.

En un principio el auditorio contaba con unas ventanas fijas, que permanecían abiertas todo el día, localizadas en la parte inferior de las paredes este y oeste y unas turbinas eólicas en la parte superior con lo que se tenía ventilación todo el tiempo.

En ese tiempo se hicieron mediciones de temperatura en el interior y exterior del auditorio y del análisis de datos se concluyó que ese tipo de ventilación no era una buena estrategia para lograr el confort térmico, debido a que en días donde el calor era intenso fuera del auditorio, también la temperatura en el interior se disparaba y como resultado, un ambiente poco confortable.

Las ventanas fijas se cambiaron por ventanas que se pueden abrir y cerrar usando motores con control automático. En las figuras () se muestran las ventanas actuales del auditorio Tonatiuh.



Figura 3.7 Ventila de la pared este.



Figura 3.8 Ventilas de la pared oeste.

Fotografía de las ventilas vista desde adentro del auditorio (Figura)



Figura 3.9 Vista interior de las ventilas

3.2.1 Antecedentes

“ESTRATEGIAS DE ENFRIAMIENTO DE BAJO CONSUMO ENERGÉTICO PARA ZONAS DEL ESTADO DE MOLEROS CON CLIMA CÁLIDO-SUBHÚMEDO”

En 2011, Noé Rodríguez llevo a cabo en el auditorio Tonatiuh del IER-UNAM en ese entonces el Centro de Investigación en Energía (CIE) el diseño, instrumentación y evaluación de un sistema automatizado para las ventilas de del auditorio como implementación de una estrategia de ventilación nocturna de enfriamiento pasivo.

Su sistema consistió en la implementación de un control automático en las ventilas, con el cual se abrían y cerraban de manera automática con la ayuda de servomotores según los horarios establecidos en los que el autor consideró que eran óptimos para propiciar una temperatura dentro del rango de confort térmico.

En la siguiente figura se muestra el esquema del sistema de control que implementó Rodríguez para las ventilas del auditorio.

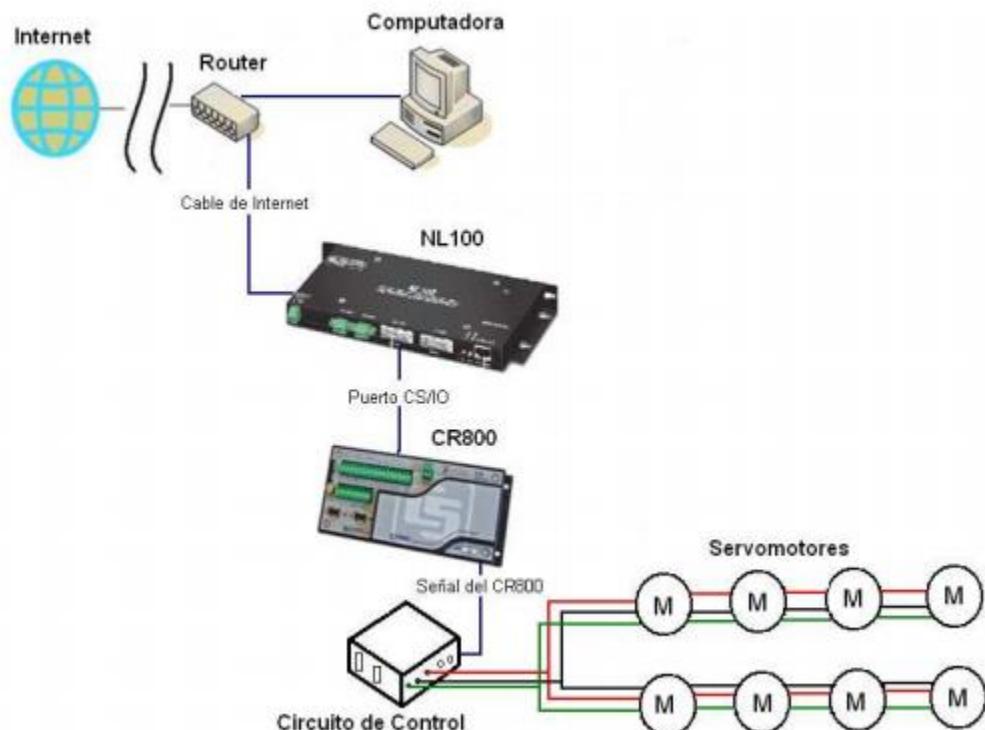


Figura 3.10 Esquema de control de ventilas por Rodríguez, 2011

Funcionamiento:

Por medio de un router y un adaptador ethernet (NL100) efectuó la conexión de un adquisidor de datos de medición y control (CR800) a internet para tener acceso a los datos de manera remota a través de una computadora, el CR800 enviaba la señal de control, la cual llegaba a un circuito de control, dicho que adaptaba el nivel de tensión de la señal enviada por el CR800 a una señal capaz de activar los servomotores instalados para poder realizar la apertura y cierre de las ventanas. El CR800 se programó para realizar la apertura de las ventanas a las 21: 00 hrs y el cierre de estas mismas a las 8: 00 hrs del día siguiente.

Los servomotores que controlan a las ventanas son de la marca Belimo modelo NMB24- SR que funciona con corriente alterna o corriente directa a 24 volts, este voltaje es para energizar el servomotor, la señal para que abran o cierren las ventanas debe de ser de 2 a 10v dc [].

A continuación, se muestra en la figura () el modelo de los servomotores utilizados para abrir las ventanas.



Figura 3.11 Servomotor Belimo NMB24-SR.

CAPÍTULO 4

SISTEMA DE MEDICIÓN DE VENTANA DESLIZABLE

En este capítulo se muestra por secciones el desarrollo completo del sistema de medición de la ventana deslizable elaborado en este proyecto, desde el diseño electrónico, cálculos, programación, diseño de placa PCB, etc.

4.1 ESPECIFICACIONES Y REQUERIMIENTOS

Uno de los proyectos más ambiciosos del Instituto de Energías Renovables de la UNAM es la construcción de un “edificio inteligente”, esto es, un edificio donde prácticamente el 100% de sus instalaciones y departamentos estén con sensores y sistemas de control con el fin de tener un monitoreo de los datos recogidos acerca de fenómenos como el comportamiento de temperatura, humedad, flujo de calor, etc.

Es por ello que, en vista a eso, se propuso la implementación de un sistema de medición de una ventana deslizable en un cubículo.

A continuación, se muestra una lista con los requerimientos del sistema de medición:

- ❖ Hacer un programa que permita conocer el nivel de apertura (cm) a la que se abre la ventana deslizable.
- ❖ Verificar con un instrumento de medición, que la lectura sea correcta.
- ❖ Usar el modo “sueño profundo” del ESP8266 para el ahorro de energía.
- ❖ Añadir un sensor de vibración y su programación que detecte cuando un usuario ha abierto la ventana y entonces despierte del sueño profundo y ejecute el programa.
- ❖ Mandar los datos de la lectura cada vez que se abra la ventana vía internet a la plataforma ThingsBoard.

4.2 INSTRUMENTACIÓN Y CONTROL

4.2.1 Componentes

En la siguiente tabla se muestra la lista de los componentes que integran el sistema de medición de apertura de la ventana deslizable.

Nombre	Componente	Cantidad
ESP8266		1
HC-SR04		1
SW-420		1
R-220		1
LED		1
DYMORE 16340		1
BATERIA-NL166		1

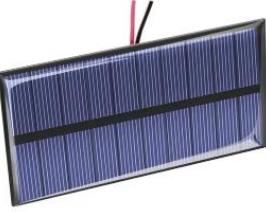
CELDA SOLAR		1 (5V)
CABLE		Necesario

Tabla 4.1 Componentes del sistema de medición.

4.2.1 Diseño electrónico

En la figura () se muestra el esquema general de funcionamiento del sistema de medición de ventana deslizable, y en la tabla () los pines de conexión.

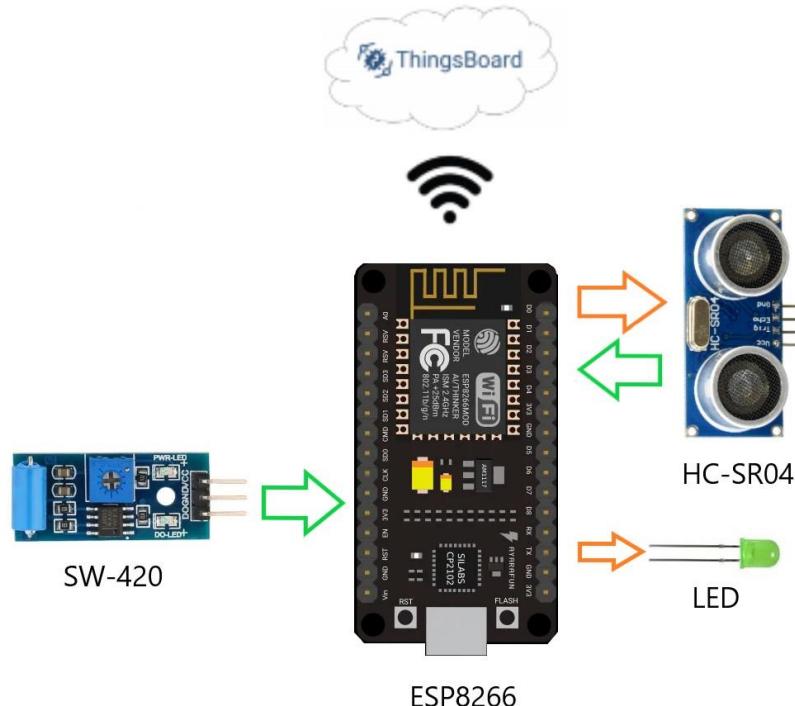


Figura 4.1 Esquema de conexión del sistema de medición.

Pines de conexión:

HC-SR04	ESP8266
VCC	3.3V
TRIG	GPIO14
ECHO	GPIO12
GND	GND
SW-420	
VCC	3.3V
GND	GND
DO	RST
LED	
VCC	GPIO2
GND	GND

Tabla 4.2 Pines de conexión del sistema de medición.

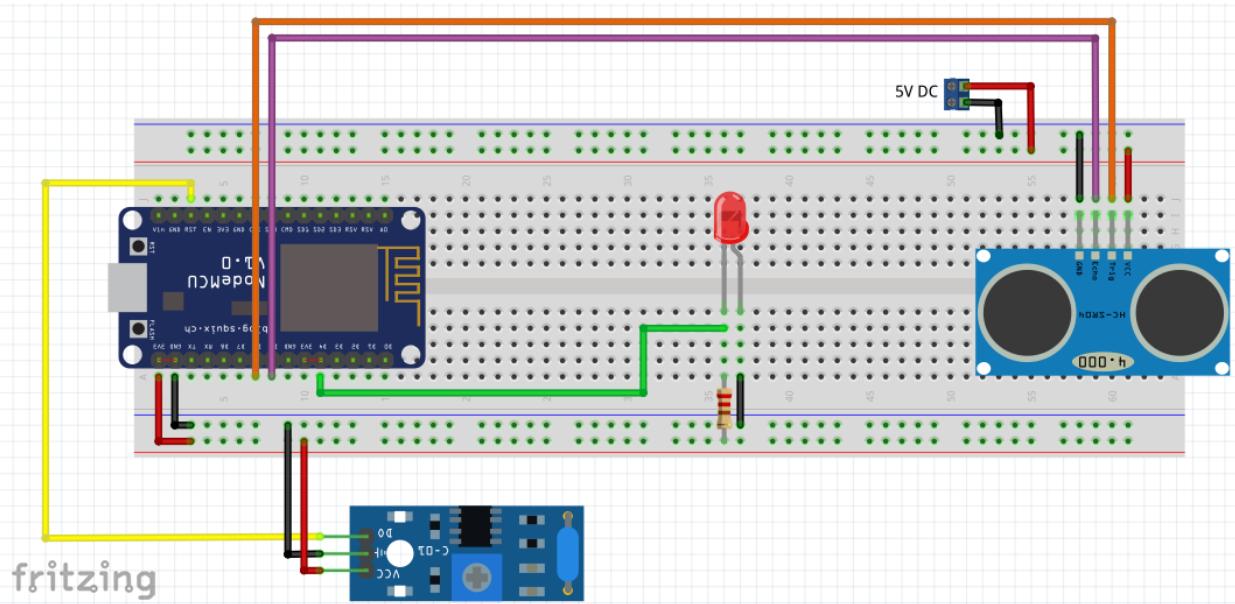


Figura 4.2 Conexión prothoboard del sistema de medición.

El microcontrolador ESP8266 es el componente principal de este sistema, éste se encarga de gestionar las señales de entrada y salida de los 3 componentes restantes, además mediante su modulo wifi que tiene integrado es como se logra el envío de los datos a la plataforma ThingsBoard y visualizarlos en tiempo real. Según las especificaciones técnicas del fabricante debe ser alimentado con 3.3V y GND (0V), aunque hay algunos usuarios que aseguran haber trabajado con este componente a 5V, no se recomienda hacerlo para cuidar la vida útil del mismo. La otra opción de trabajo es mediante la conexión USB que va directo al ordenador, ya que cuenta con un regulador de voltaje AMS1117 de 3.3V como sistema de protección cuando se usa este recurso.

Una vez definida la parte que controla el sistema, pasaremos al componente clave de este proyecto, el sensor ultrasónico HC-SR04. El sensor ultrasónico es alimentado con 5V DC ya que es el voltaje especificado para su correcto funcionamiento, de hecho, con un voltaje menor a ese no funcionara.

Recapitularemos un poco el funcionamiento del HC-SR04: El sensor envía la señal de ultrasonido mediante el emisor (TRIG), la señal viaja por el aire y al topar con el objeto rebota hasta llegar al receptor del sensor (ECHO), como se muestra en la siguiente figura.

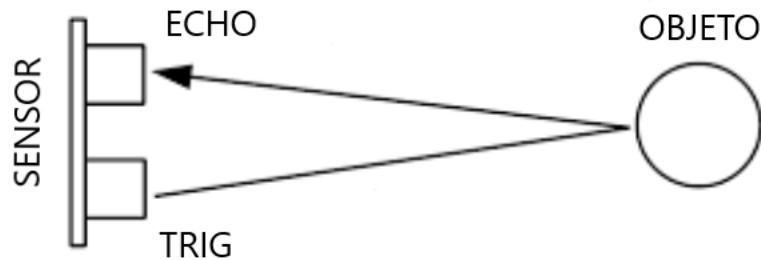


Figura 4.3 Funcionamiento del sensor ultrasónico HC-SR04.

Conociendo el funcionamiento del sensor y que es de tipo ultrasónico, la primera cosa que debemos tener en cuenta si queremos obtener la distancia en unidades de “cm” es conocer a qué velocidad viaja el sonido.

“La velocidad del sonido es la distancia que viajan las ondas sonoras en un período de tiempo determinado, en la atmósfera terrestre a una temperatura de 20° C, ésta es de 343 m/s. Sin embargo, bajo otras condiciones de temperatura u otro medio este valor puede cambiar” [10].

Medio (20 ° C)	Velocidad de las ondas sonoras (m / s)
Aire seco	343
Agua	1437
Madera	3850
Vaso	4540
Aluminio	6320

Figura 4.4 Velocidad de las ondas sonoras.

Para calcular la distancia entonces tomaremos en cuenta los siguientes puntos:

- El sensor mide el tiempo en microsegundos (μs)
- La velocidad del sonido es de 343 m/s.
- El recorrido que hace la señal es de ida y vuelta.

Recordando las fórmulas de física que integran la velocidad, distancia y tiempo tenemos las siguientes 3:

- Velocidad = distancia / tiempo
- Tiempo = distancia / velocidad
- Distancia = velocidad * tiempo (esta es la que nos sirve)

La variable tiempo será igual a la duración que tarde la señal de ultrasonido desde que sale del emisor hasta que llega al receptor, eso lo podemos obtener con un diferencial de tiempo, por lo tanto, duración = $(T_2 - T_1)$.

La velocidad ya la tenemos es 343 m/s, sin embargo, el sensor trabaja el tiempo en microsegundos y la distancia la necesitamos en “cm” por lo tanto haremos la conversión para pasar de (m/s) a (cm/ μs).

$$343 \frac{m}{s} \times 100 \frac{cm}{m} \times \frac{1}{1,000,000} \frac{s}{\mu s} = 0.0343 \frac{cm}{\mu s}$$

Una vez obtenidas las unidades correctas tendríamos nuestra formula de la siguiente manera:

$$\text{Distancia} = \text{duración } \mu s * 0.0343 \text{ m}/\mu s$$

Pero recordemos que la señal viaja una distancia de ida y vuelta, y a nosotros nos interesa solo una de ellas, es decir solo la distancia que hay del sensor al objeto, solo bastara con dividir lo que ya tenemos de la formula entre 2, de manera que nuestra formula queda finalmente así:

$$\boxed{\text{Distancia} = \frac{(T_2 - T_1) * 0.0343}{2}}$$

Limitaciones: La instalación del dispositivo o sistema de medición está diseñado para colocarse en uno de los cubículos del auditorio del IER-UNAM, sin embargo, debido a la crisis de salud que estamos pasando por la llegada del SARS-COVID19 y todas las medidas de prevención que el gobierno ha establecido, no fue posible el acceso al instituto durante el periodo que se desarrolló el proyecto de residencia. Afortunadamente en casa había una ventana del mismo tipo, por lo que ahí se instaló el sistema para las pruebas de funcionamiento.

Descripción general de funcionamiento: En su primer arranque el sensor ultrasónico comienza a realizar la medición de la apertura de la ventana (para la prueba de funcionamiento fue alimentado con un cargador de celular de 5v), en el programa se toman 10 lecturas de esta medición y se saca un valor medio para devolver una medición más estable o con menos variaciones, una vez que ha hecho esto, el programa manda este valor resultante a la plataforma ThingsBoard para su visualización en tiempo real a través de la red, ahora el ESP entra en modo sueño profundo (modo de ahorro de energía), cuando un usuario modifique la apertura de la ventana el movimiento hará que el sensor de vibración active su funcionamiento y mandara una señal digital al Pin RST (reset), es decir reiniciara al ESP y por consiguiente al programa, logrando que se repita todo el proceso.

Como parte del diseño final del sistema de medición y aprovechando los recursos naturales que tenemos a nuestro alcance se propuso la idea de establecer como fuente de alimentación de los componentes del sistema una batería recargable conectada a un pequeño panel solar, a fin de ahorrar significativamente el consumo de energía eléctrica. A continuación, se muestra el esquema de conexión de la fuente.

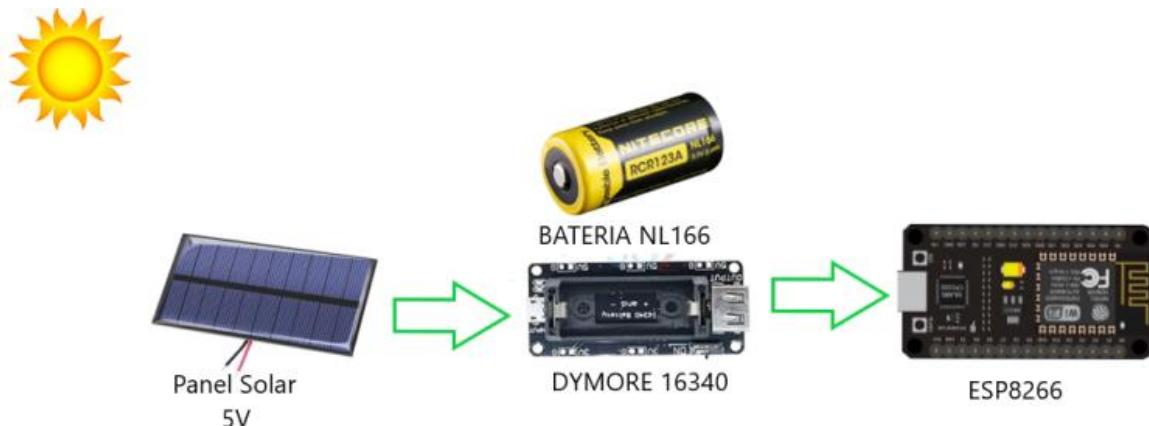


Figura 4.5 Fuente de alimentación.

4.2.2 Programación

La programación se desarrolló en el lenguaje “MicroPython” en la versión de Python 3.8, se utilizó el Shell que viene integrado para conectarnos al ESP8266 y se hizo uso del REPL para la prueba y visualización de los programas. El entorno de trabajo se realizó dentro del sistema operativo Linux.

A continuación, se muestra el algoritmo general de funcionamiento del sistema de medición:

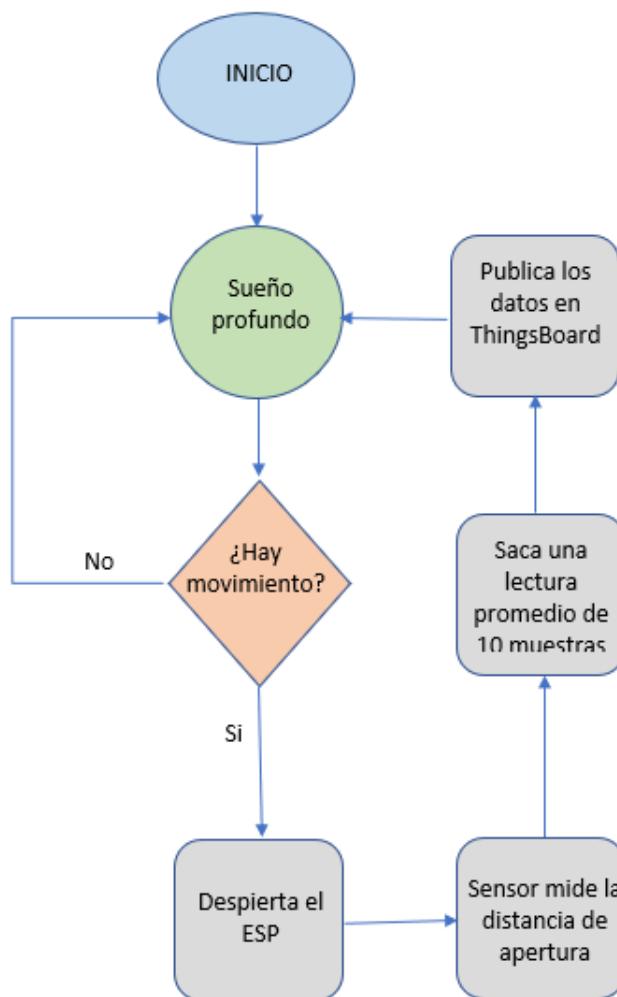


Figura 4.6 Algoritmo general del sistema de medición.

El primer programa realizado fue obtener la distancia en “cm” entre el sensor y un objeto, tomar 10 muestras y devolver un valor medio. A continuación, se muestra el código escrito:

funciones.py

```
1 # - SISTEMA DE MEDICION DE VENTANA DESLIZABLE -
2
3 # POR: GIOVANNI VELAZQUEZ AVILEZ
4 # ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA
5 # RESIDENTE EN EL IER-UNAM, AGO-DIC 2020
6
7 import machine
8 from machine import Pin
9 import utime
10 import math
11
12 def distancia_normal():
13
14     trig=machine.Pin(14, machine.Pin.OUT)
15     trig.off()
16     utime.sleep_us(2)
17     trig.on()
18     utime.sleep_us(10)
19     trig.off()
20     echo=machine.Pin(12, machine.Pin.IN)
21     while echo.value() == 0:
22         pass
23     t1 = utime.ticks_us()
24     while echo.value() == 1:
25         pass
26     t2 = utime.ticks_us()
27     cm = ((t2 - t1)*0.03432)/2
28     utime.sleep_ms(20)
29     return cm
30
31
32 def distancia_media():
33
34     distancia_muestras = []
35
36     for count in range(10):
37         distancia_muestras.append((distancia_normal()))
38
39     distancia_muestras = sorted(distancia_muestras)
40
41     distancia_media = distancia_muestras[int(len(distancia_muestras)/2)]
42
43     return float(distancia_media)
44
```

Se importan los módulos y se define la función y su nombre, después se asignan los pines de entrada y salida, en este caso el Pin 14 que corresponde al TRIG será OUT y el Pin 12 que corresponde al ECHO será IN.

Se usa un “mientras” el estado de ECHO sea 0; T1 será igual al tiempo de inicio. Y cuando ECHO sea igual a 1 entonces T2 será igual al tiempo en microsegundos que tardo en llegar la señal ósea el tiempo final. Luego se escribe la fórmula que obtuvimos para tener la distancia, se da un retardo de 20 milisegundos y por último “return” hará que la función devuelva el valor de la distancia en “cm”.

Se añade una nueva función, esta nos ayudara a recoger 10 lecturas y obtener el valor medio para lograr devolver al usuario un valor mas estable. Lo que hace es añadir una lista con las lecturas que hace la primera función (“distancia normal”) y con un ciclo “for” se define el rango en este caso son 10, después ordena la lista, se asegura de integrar los valores y con la función “len” se obtiene el numero de lecturas que están en la lista y por ultimo se divide entre 2. Y devuelve la distancia tipo flotante.

main.py

```
1 # - SISTEMA DE MEDICION DE VENTANA DESLIZABLE -
2
3 # POR: GIOVANNI VELAZQUEZ AVILEZ
4 # ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA
5 # RESIDENTE EN EL IER-UNAM, AGO-DIC 2020
6
7 import machine
8 from machine import Pin
9 from funciones import *
10 import time , utime
11 import math
12
13 while True:
14
15     distancia = distancia_media()
16     print ("Distancia:", distancia, "cm")
17     time.sleep(1)
18
```

Se importan los módulos y el archivo donde está las funciones principales, luego se inicia un loop infinito, se le da un nombre a la función que devuelve la distancia y por último se imprime en pantalla la distancia en cm y se agrega un retardo de 1 segundo.

Importante:

- Cuando se suben los archivos.py al ESP y se da la orden de ejecutar estos (REPL), de manera predeterminada se correrá primero el archivo que lleve por nombre “boot.py” o “main.py”, es por ello que en el archivo main.py que se muestra en la imagen () se manda a traer el otro archivo y sus funciones que contenga, ya que este será el que inicie primero.
- La razón por la cual los códigos se encuentran en 2 archivos diferentes es para evitar el exceso de script en un solo archivo y que, de una mala presentación, o incluso que resulte confuso lo que se está realizando.

El segundo programa realizado fue lograr que la distancia de apertura obtenida se enviase a la plataforma ThingsBoard justo después de obtener la medición para su visualización en tiempo real. A continuación, se muestra el código del programa.

main.py

```
1 # - SISTEMA DE MEDICION DE VENTANA DESLIZABLE -
2
3 # POR: GIOVANNI VELAZQUEZ AVILEZ
4 # ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA
5 # RESIDENTE EN EL IER-UNAM, AGO-DIC 2020
6
7 import machine
8 from machine import Pin
9 import time
10 from time import sleep
11 import math
12 from funcion import *
13 from funciones import *
14 from wifi import activate_wifi
15 import utime
16
17 red = 'TOTALPLAY_897E2E'
18 clave = 'EZ04022ST1'
19
20 unique_id = '70099f70-10a1-11eb-9c3f-d1ead9980bc3'
21 token = 'Mdihxoakm0rTUqo9GlRM'
22
23 label = 'distancia'
24
25 activate_wifi(red,clave)
26
27 while True:
28
29     print ('DISTANCIA')
30     for count in range(2):
31         result=distancia_media()
32         print (result, 'cm')
33         valor = result
34         data = {label: valor}
35         publish_thingsboard(red,clave,token, unique_id,data)
36         sleep(1)
37
```

Se importan los módulos y se llaman a traer los archivos “función, wifi, y funciones” junto con sus funciones.

Después se proporcionan nombre y clave del internet para conectar a la red local.

Luego se definen el “unique id” y “token”, estos son datos que nos provee nuestro asesor para conectar al dispositivo en ThingsBoard.

Después se agrega una etiqueta con el nombre de “distancia”, misma que aparecerá en el Dashboard de ThingsBoard con los valores que le envíemos.

Por último, se inicia un loop infinito y con el ciclo “for” hará que se obtengan 2 veces la medición, y finalmente se da la instrucción de publicar los datos.

Cabe mencionar que al inicio de la residencia de parte del IER-UNAM recibimos un taller de introducción al IoT, donde nos enseñaron a realizar una conexión exitosa vía internet con la plataforma ThingsBoard, a si mismo se nos proporcionaron los archivos que logran esa conexión, mismos que llevan los nombres de “función.py” y “wifi.py”, los cuales se encuentran en el apartado anexo.

Puesto que la actividad principal del sistema de medición es obtener la distancia cada que un usuario modifica el estado de apertura de la ventana, se consideró que sería mucho gasto de energía que el ESP se mantuviera todo el tiempo activo, por lo que se propusieron las siguientes 2 cosas:

- Activar el modo sueño profundo o “Deep sleep” que consiste básicamente en la suspensión de la mayoría de las terminales y funciones del ESP, excepto el RTC (reloj en tiempo real), que logran reducir significativamente el consumo energético.
- Añadir un sensor de vibración que detecte cuando alguien ha movido la ventana y que envíe una señal para despertar del sueño profundo al ESP.

El código sigue siendo el mismo del programa 2 solo se han añadido algunas pequeñas líneas:

```
17 red = 'TOTALPLAY_897E2E'
18 clave = 'EZ04022ST1'
19
20 unique_id = '70099f70-10a1-11eb-9c3f-d1ead9980bc3'
21 token = 'Mdihxoakm0rTUqo9GiRm'
22
23 label = 'distancia'
24
25 activate_wifi(red,clave)
26
27 led = Pin (2, Pin.OUT)
28 led.value(0)
29 sleep(1)
30 led.value(1)
31 sleep(1)
32
33 while True:
34
35     print ('DISTANCIA')
36     for count in range(2):
37         result=distancia_media()
38         print (result, 'cm')
39         valor = result
40         data = {label: valor}
41         publish_thingsboard(red,clave,token, unique_id,data)
42         sleep(1)
43
44
45     print('IRE A DORMIR')
46     sleep(3)
47
48     machine.deepsleep()
```

Aquí se agrega el código para el LED, lo que hará es parpadear cuando se inicie el programa y se mantendrá encendido

Para activar el modo “deep sleep” solo basta ejecutar la línea “machine.deepsleep()”.

4.2.3 PCB

Se realizó también el diseño PCB del sistema de medición con sus componentes que lo integran, a continuación, se muestra el esquemático del diagrama de conexión y el diseño PCB.

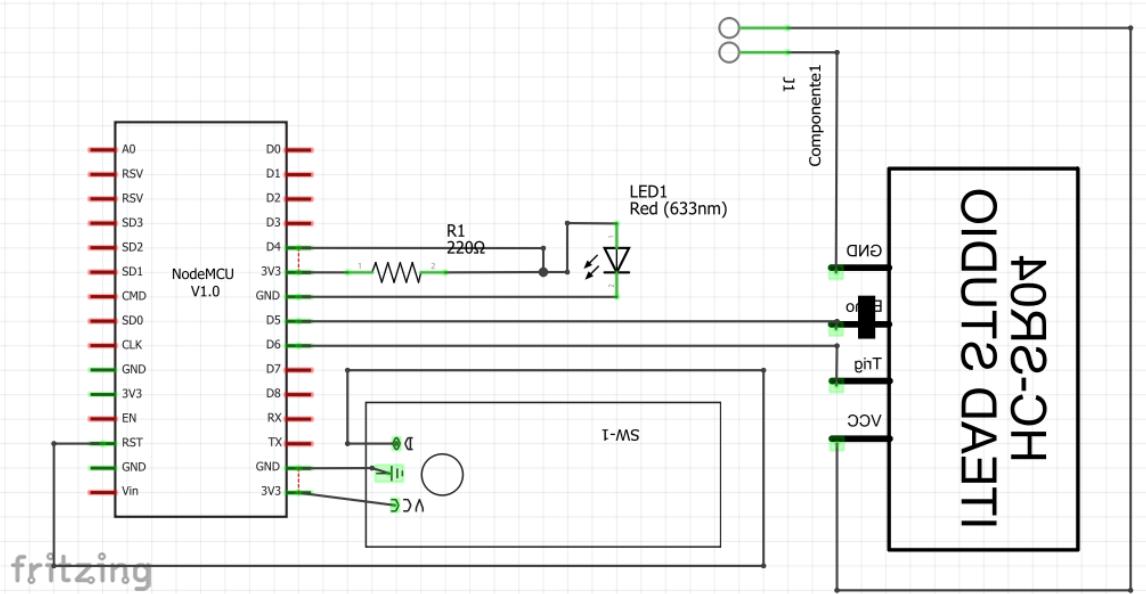


Figura 4.7 Esquemático del sistema de medición.

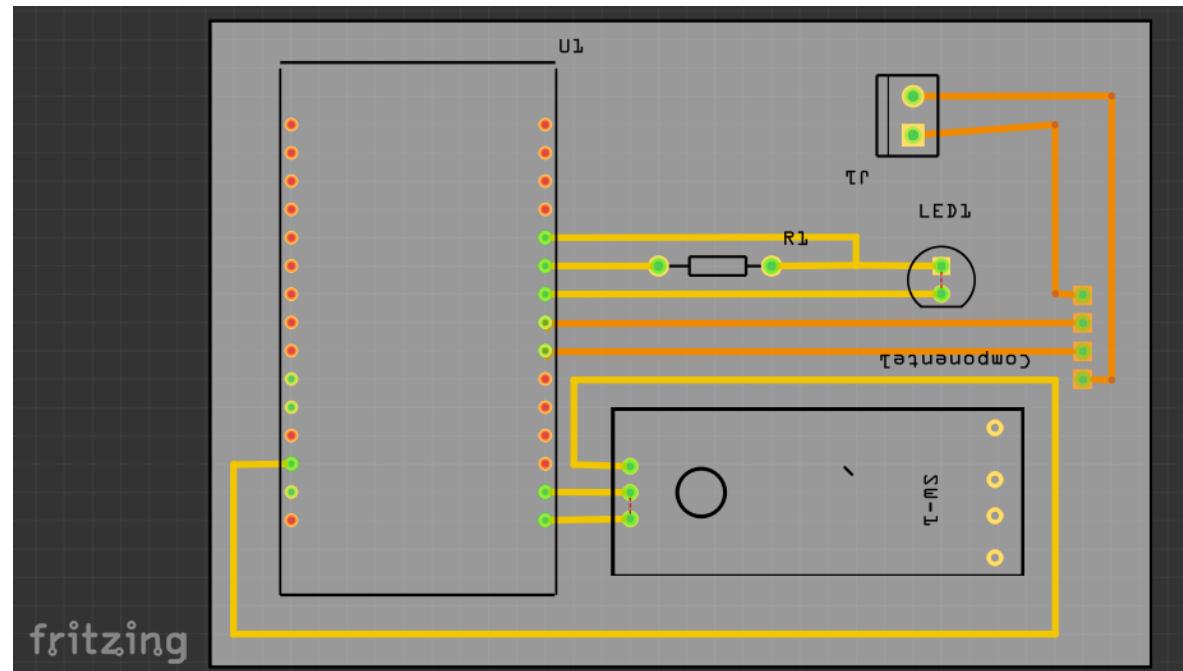


Figura 4.8 Diseño PCB del sistema de medición.

CAPÍTULO 5

SISTEMA DE CONTROL DE VENTILAS

En este capítulo se muestra el desarrollo completo del sistema de control de ventilas para auditorio elaborado en este proyecto, desde el diseño electrónico, cálculos, programación, diseño de placa PCB, etc.

5.1 ESPECIFICACIONES Y REQUERIMIENTOS

El sistema de control de ventilas ha sido asignado para su uso en el auditorio Tonatiuh ubicado en las instalaciones del IER-UNAM, como se mencionó en el capítulo 3 en el apartado “antecedentes” en 2011, Rodríguez implemento un sistema de automatización de las ventilas del auditorio, sin embargo, aunque el funcionamiento fue exitoso los componentes que se utilizaron como el adaptador ethernet NL100 o el adquisidor de datos/control CR800 además de ser muy caros también resulta ser que al día de hoy son componentes un tanto obsoletos, debido a eso se pensó en una actualización de dicho sistema, ahora haciendo uso de tecnologías abiertas.

A continuación, se muestra una lista con los requerimientos del nuevo sistema de control:

- ❖ El componente principal y de control sea un ESP32
- ❖ Interfaz del sistema en pantalla LCD
- ❖ Menú seleccionable de niveles de apertura
- ❖ Programación de horarios
- ❖ Enviar datos a ThingsBoard

5.2 INSTRUMENTACIÓN Y CONTROL

5.2.1 Componentes del sistema de control

En la siguiente tabla se muestra la lista de los componentes que integran el sistema de control de ventilas que se desarrolló en este proyecto.

Nombre	Cantidad	Componente
ESP32	1	
MPU6050	1	
LM324N	1	
R1 - (10K Ω)	1	
R2, R4 - (100K Ω)	2	
R3 - (75K Ω)	1	
R5 - (200K Ω)	1	
R6 - (220Ω)	1	
R7, R8, R9, R10 - (1KΩ)	4	
LED	1	
C1 - 1(μF)	1	

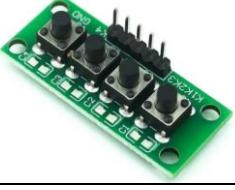
MODULO 4 PB	1	
POWER SOURCE 12V DC	1	
LCD – 20X4 + MODULO I2C	1	
CABLE	Necesario	

Tabla 5.1 Componentes del sistema de control.

5.2.2 Diseño electrónico

En la figura () se muestra el esquema general de funcionamiento del sistema de control de ventanas, y en la tabla () los pines de conexión.

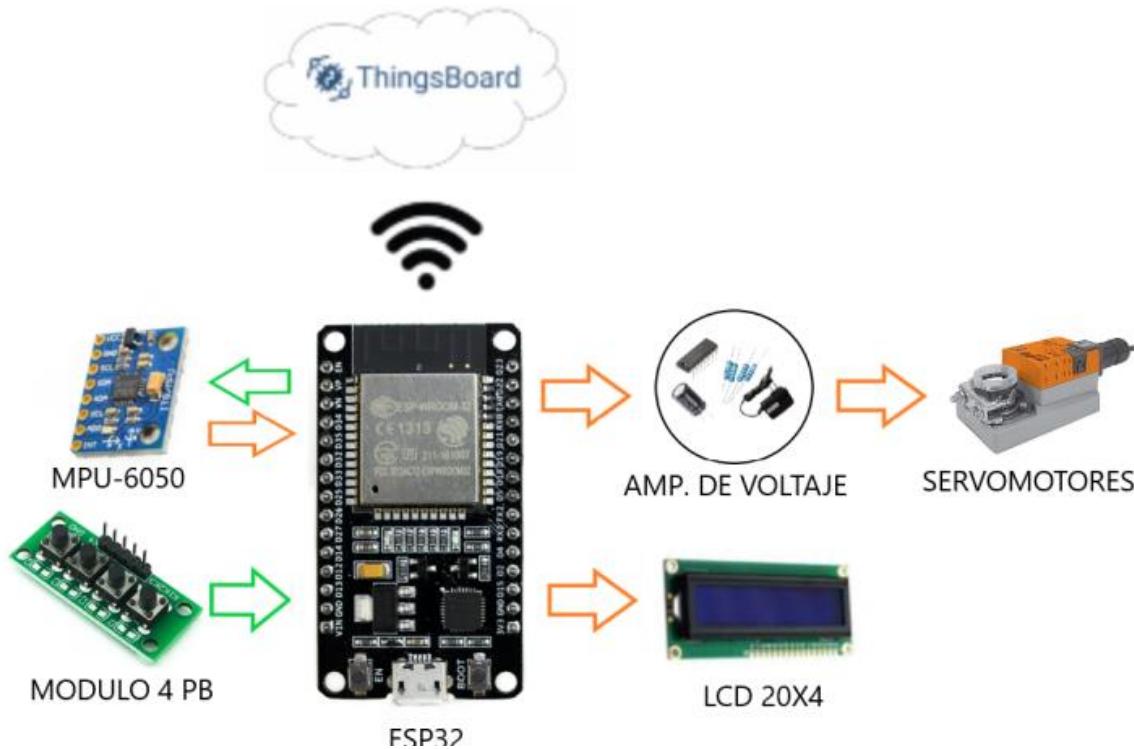


Figura 5.1 Esquema de conexión del sistema de control.

MPU-6050	ESP8266
VCC	3.3V
GND	GND
SCL	GPIO12
SDA	GPIO14
MODULO 4 PB	
1	GPIO18
2	GPIO4
3	GPIO17
4	GPIO15
LCD + MODULO I2C	
VCC	5V
GND	GND
SCL	GPIO26
SDA	GPIO13

Tabla 5.2 Pines de conexión del sistema de control.

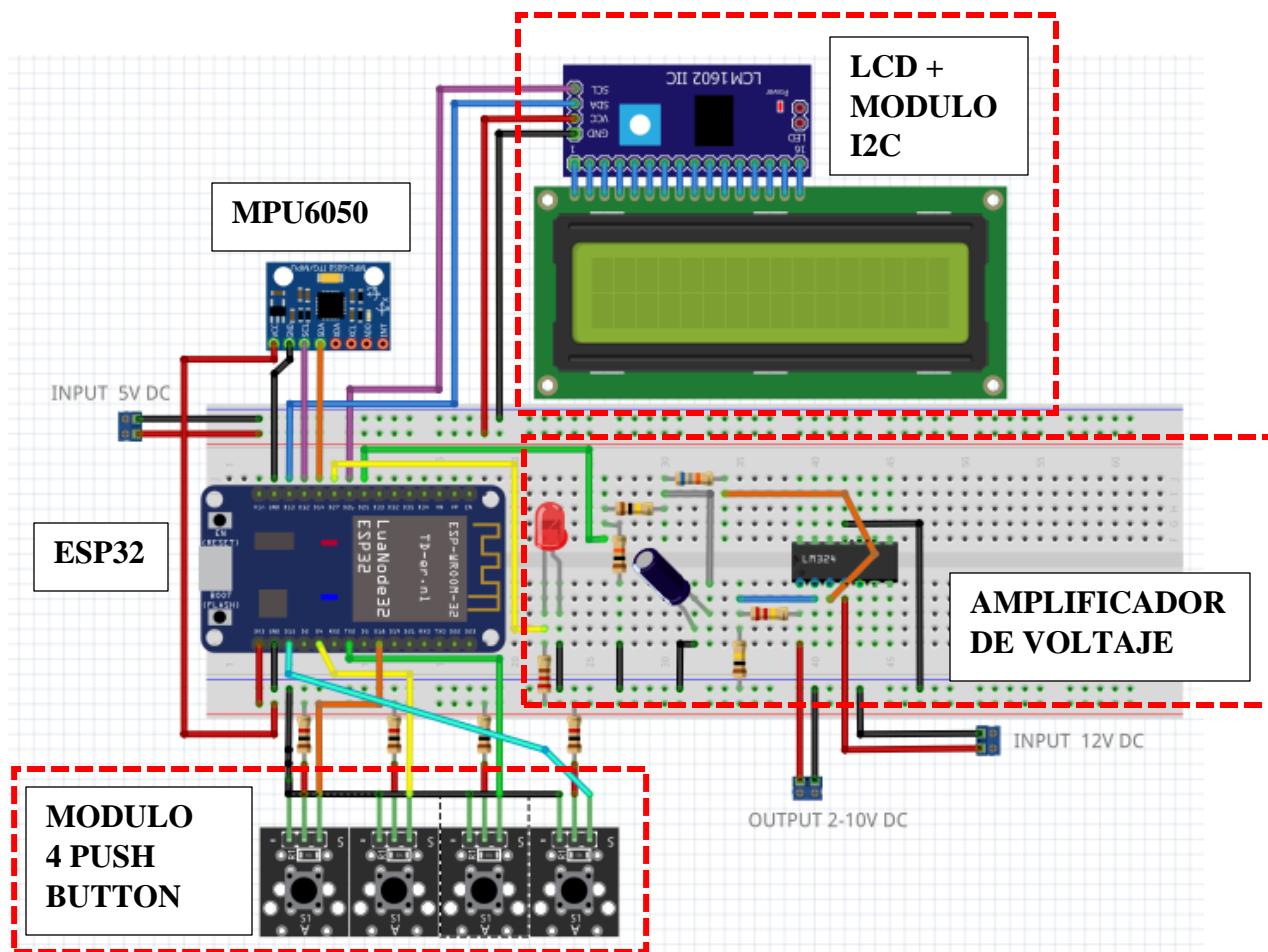


Figura 5.2 Conexión prothoboard del sistema de control.

El microcontrolador ESP32 es el componente principal de este sistema, éste se encarga de gestionar las señales de entrada y salida de los 3 componentes restantes, además mediante su modulo wifi que tiene integrado es como se logra el envío de los datos a la plataforma ThingsBoard y visualizarlos en tiempo real.

Según las especificaciones técnicas del fabricante debe ser alimentado con 5V y GND (0V). La otra opción de alimentación es mediante la conexión USB que va directo al ordenador, ya que cuenta con un regulador de voltaje como sistema de protección cuando se usa este recurso.

El módulo MPU6050 es sin duda un componente clave en este sistema de control, dado que éste irá instalado en una de las ventanas del auditorio nos permitirá conocer el nivel en grados ($^{\circ}$) de apertura a la que se encuentren.

Sin embargo, al usar este componente por primera vez no hará nada de eso, ya que se tiene que configurar e implementar algunas fórmulas en su programación.

Cabe mencionar que al inicio de la residencia cuando se eligió el MPU6050 para este proyecto y supimos que integraba también la función del giroscopio para medir grados de rotación, el Dr. Guillermo Ramírez del IER-UNAM me pidió que lograra obtener los grados de inclinación no con la función del giroscopio sino a partir de la aceleración de la gravedad. Es por ello que solo nos enfocaremos y utilizaremos la función de acelerómetro. A continuación, se muestran las actividades realizadas para su correcta configuración:

La aceleración es la variación de la velocidad por unidad de tiempo es decir razón de cambio en la velocidad respecto al tiempo: $a=dV/dt$

Así mismo la segunda ley de Newton indica que en un cuerpo con masa constante, la aceleración del cuerpo es proporcional a la fuerza que actúa sobre él mismo: $a=F/m$

Este segundo concepto es utilizado por los acelerómetros para medir la aceleración. Los acelerómetros internamente tienen un MEMS (Micro Electro Mechanical Systems) que de forma similar a un sistema masa resorte permite medir la aceleración.

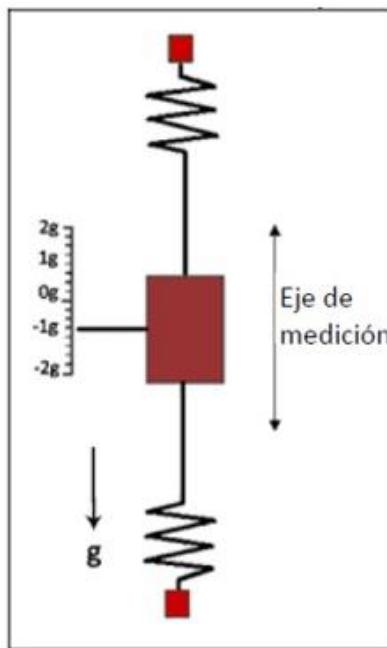


Figura 5.3 Funcionamiento de un MEMS.

Teniendo en cuenta que a pesar que no exista movimiento, siempre el acelerómetro estará censando la aceleración de la gravedad:

$$9.81 \text{ m/s} = 1g$$

En la siguiente tabla se muestran el nombre de las variables que mide nuestro MPU y su descripción:

Medición	Descripción
AcX	Aceleración en el eje X
AcY	Aceleración en el eje Y
AcZ	Aceleración en el eje Z
GyX	Rotación alrededor del eje X
GyY	Rotación alrededor del eje Y
GyZ	Rotación alrededor del eje Z
Tmp	Temperatura

Tabla 5.3 Descripción de variables del MPU6050.

Como ya lo mencionamos anteriormente cuando se conecta el MPU6050 al ESP32 por primera vez y se le piden los valores de la aceleración en los 3 ejes los datos que el módulo arrojara se les conoce como “datos en bruto”, lo que quiere decir que están sin procesar, por lo que realmente de nada nos servirán.

Para configurar entonces el acelerómetro lo primero que hay que establecer es la sensibilidad en la que queremos que trabaje el módulo, de acuerdo al apartado ACCEL_CONFIG del documento datasheet estas son las opciones de configuración de la sensibilidad:

AFS_SEL	Full Scale Range	LSB Sensitivity
0	$\pm 2g$	16384 LSB/g
1	$\pm 4g$	8192 LSB/g
2	$\pm 8g$	4096 LSB/g
3	$\pm 16g$	2048 LSB/g

Figura 5.4 Rangos de configuración del MPU6050.

Puesto que nosotros queremos trabajar solo con el valor de la aceleración de la gravedad que es de 1g, bastara con ajustar la sensibilidad a $\pm 2g$ que es la primera opción. Por último, para seleccionar este rango solo basta con dividir los valores en bruto de AcX, AcY, AcZ entre el LSB (Bit menos significativo) seleccionado, en nuestro caso: 16384 LSB/g, esto se verá más a fondo en el apartado “programación”.

Para obtener la inclinación a partir de la aceleración de la gravedad se hizo una investigación respecto al tema y se obtuvo la siguiente información:

- En ausencia de aceleración lineal, la salida del acelerómetro es una medida del vector del campo gravitacional girado y se puede usar para determinar los ángulos de inclinación y orientación del balanceo del acelerómetro.
- Los ángulos de orientación dependen del orden en que se apliquen las rotaciones. El orden más común es la secuencia aeroespacial de “yaw”, luego “pitch” y finalmente una rotación de “roll”.

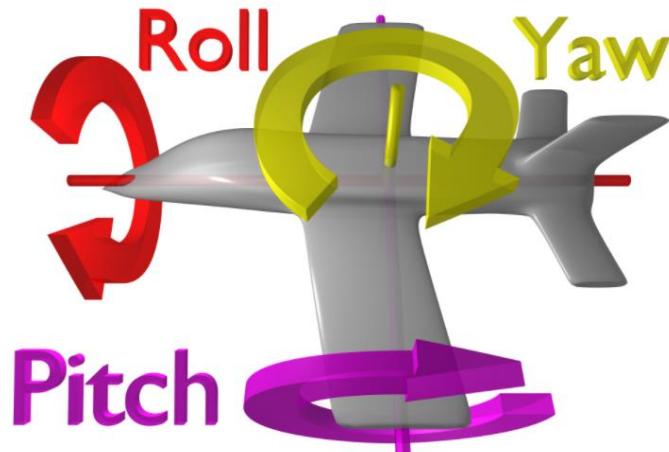


Figura 5.5 Secuencia de rotación aeroespacial (Yaw-Pitch-Roll)

Puesto que solo nos interesa saber la inclinación respecto al eje X o eje Y, solo nos quedaremos con los movimientos de rotación: “roll = θ ” y “pitch = ϕ ”. En la siguiente figura se muestra como es que se forma los 2 ángulos de inclinación respecto a la aceleración de la gravedad cuando se rota el acelerómetro.

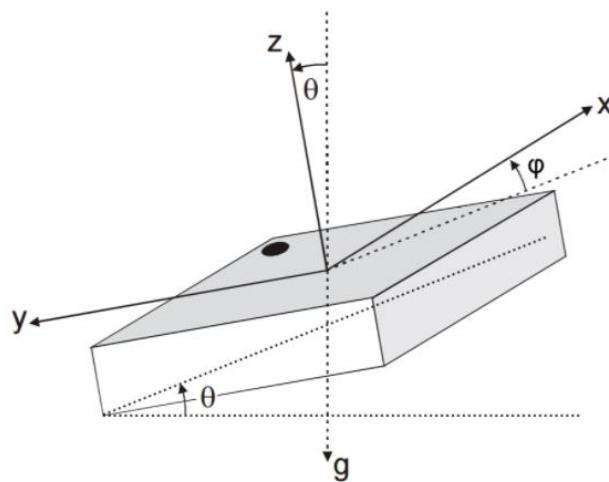


Figura 5.6 Formación de ángulos respecto a la gravedad.

La explicación de la figura 5.6 se basa en encontrar dichos ángulos haciendo uso de las coordenadas rectangulares y las coordenadas polares básicamente.

La estimación de los ángulos se obtuvo en el documento datasheet del acelerómetro triaxial BMA220 de “Bosch Sensortec”, una compañía que desarrolla y fabrica sensores MEMS (Sistemas Microelectromecánicos).

- $AcX = 1g * \sin \theta * \cos \varphi$
- $AcY = -1g * \sin \theta * \sin \varphi$
- $AcZ = 1g * \cos \theta$

Por lo tanto:

$$AcX/AcY = -\tan \varphi$$

De modo que para obtener la inclinación del eje X o del eje Y con respecto a la aceleración de la gravedad finalmente se obtienen con las siguientes 2 ecuaciones.

$$Angulo\ X = \text{atan} \left(\frac{X}{\sqrt{Y^2 + Z^2}} \right)$$

$$Angulo\ Y = \text{atan} \left(\frac{Y}{\sqrt{X^2 + Z^2}} \right)$$

Continuando con la parte del control nos encontramos que los servomotores que abren y cierran las ventanas de la marca “Belimo” trabajan con una tensión de corriente directa o corriente alterna a 24V. Para modular el nivel al que abrirá las ventanas tiene que recibir una señal de voltaje de entre 2 a 10V DC, en la siguiente figura se muestra la relación “señal-grados de apertura” como trabajan los servomotores.

Señal (volts)	Apertura de las ventanas (grados)
2	0
3	22.5
4	45
5	67.5
6	90
7	112.3
8	135
9	157.5
10	180

Figura 5.7 Relación de voltaje-grados de apertura.

Considerando esta información nos encontramos con nuestro siguiente reto: la señal de voltaje. Puesto que nuestro componente principal de control es el ESP32 y según las especificaciones técnicas del mismo mencionan que la señal máxima de entrada/salida de sus pines es de 3.3V apenas podríamos abrir las ventanas a 22.5°, lo que resulta un gran problema.

Para resolver esto se propuso la elaboración de un amplificador de voltaje con un circuito integrado OPAMP (amplificador operacional) como componente principal y algunas resistencias, capacitores y por su puesto una fuente de voltaje de donde se obtengan mínimo los 10V. Enseguida se muestran las actividades realizadas para la creación de este amplificador.

1.- SELECCIÓN DE LA FUENTE DE VOLTAJE

Para la fuente de voltaje se eligió un eliminador de 12V DC ya que uno de mayor voltaje podría estar muy sobrado.

2.- SELECCIÓN DEL OPAMP

Se eligió el amplificador operacional LM324N ya que es un componente que puede recibir en su segunda entrada de voltaje hasta 32V, y sin problemas soportaría los 12V de nuestro eliminador, un plus de este componente es su precio tan accesible con un valor de \$20 pesos.

3.- CONFIGURACIÓN DEL OPAMP

Como queremos controlar la ganancia de voltaje de nuestro amplificador operacional usaremos el diseño de lazo cerrado ya que con la realimentación negativa en la entrada inversora hará que el circuito sea mucho más estable.

Puesto que no queremos invertir nuestra señal de salida utilizaremos la configuración de amplificador no inversor.

4.- CALCULO DE LA GANANCIA

Para calcular la ganancia que debe tener nuestro amplificador operacional tomando en cuenta la configuración de nuestro amplificador, está dado por la siguiente formula:

$$Av = \frac{-Vo}{Vin}$$

Donde:

Av = ganancia

Vo= Voltaje de salida

Vin = Voltaje de entrada

(-) = El signo menos significa que la señal obtenida en la salida será la opuesta a la de la entrada inversora.

Por lo tanto, si tenemos 3.3V que nos da nuestro ESP32 y queremos que nuestro amplificador alcance un voltaje de 10V para modular totalmente los servomotores sustituimos en la ecuación: Vin= 3.3v, Vo=10v

$$Av = \frac{10}{3.3} = 3$$

Por lo tanto, la ganancia que necesitamos es igual a 3.

*Puesto que estamos metiendo una señal negativa a la entrada inversora del amplificador operacional, nuestra salida de voltaje es una señal positiva.

5.- DISEÑO DEL CIRCUITO

En la siguiente figura se muestra la propuesta del circuito.

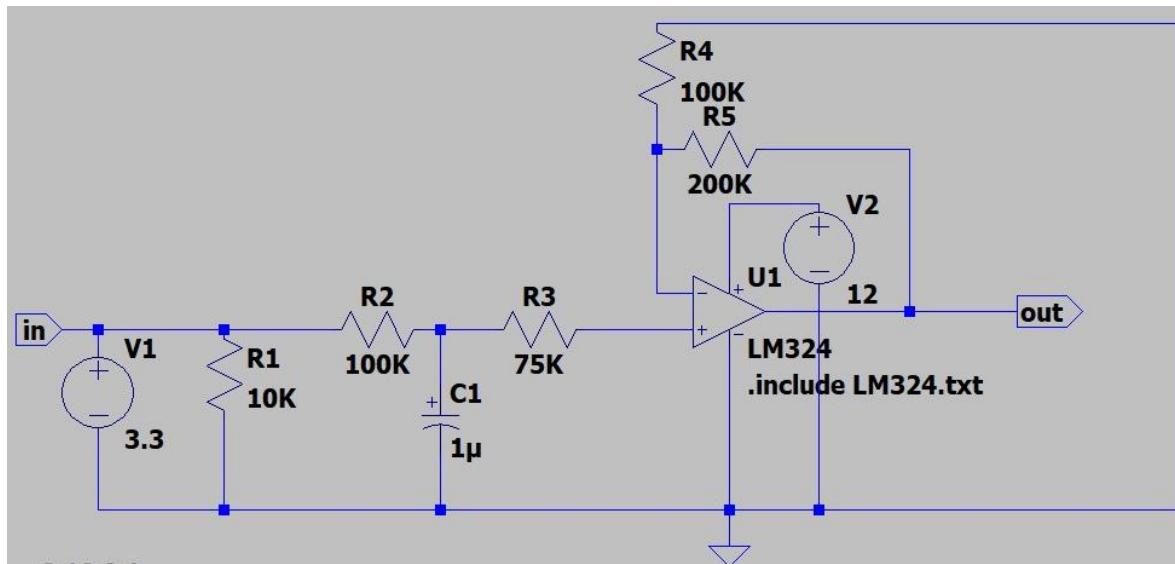


Figura 5.8 Diagrama eléctrico del amplificador de voltaje.

Una herramienta que resulta muy útil a la hora de construir algún proyecto eléctrico o electrónico y saber si realmente esta correcta la propuesta de conexión o en pocas palabras si el diseño propuesto funcionara, es la simulación. Es por ello que se usó un programa gratuito de simulación llamado “LTspice” para comprobar el funcionamiento del amplificador.

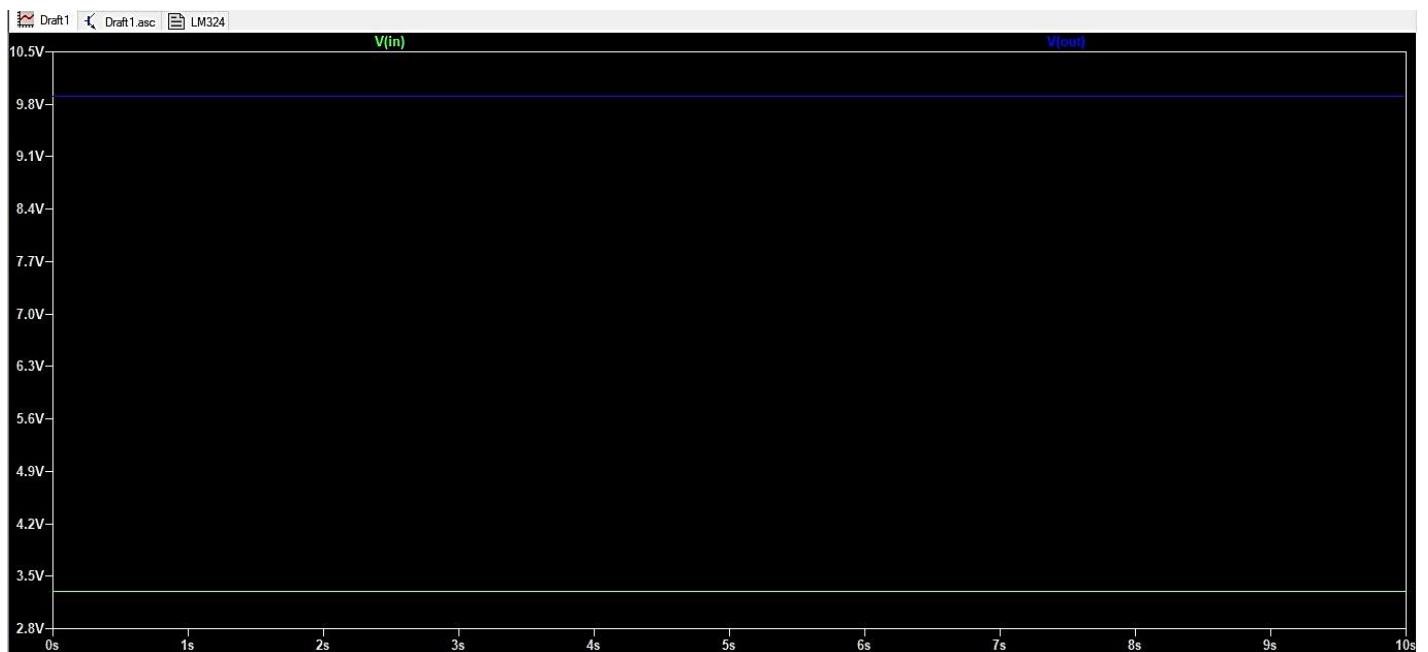


Figura 5.9 Simulación del amplificador de voltaje.

En los resultados de la simulación podemos observar que cuando ponemos 3.3v en la entrada que es el máximo voltaje que pueden dar los pines de nuestro ESP, obtenemos en la salida un voltaje de 10V, lo que nos dice que efectivamente si funcionara nuestro amplificador de voltaje propuesto.

Para enviar el voltaje del ESP32 a la entrada del amplificador se había pensado en un principio usar una señal PWM de uno de los pines del microcontrolador ya que en varios proyectos de modulación de micro servomotores o regulación de intensidad del brillo de leds ocupan este método.

Sin embargo, se tomó en cuenta que hay muchos otros actuadores cuya señal requerida para su funcionamiento debe ser estrictamente de tipo analógica y no una simulación de ella como lo es el PWM, por lo tanto, se descartó esa opción, afortunadamente gracias a que en el ESP32 un par de sus pines (GPIO26 y GPIO25) admiten DAC (convertidor digital-analógico) se optó por esa opción, ya que era exactamente lo que necesitaba el sistema.

Para el uso del DAC se consultó la página oficial de MicroPython en el apartado “modulo-ESP32”, donde se explica que el DAC que admiten ambos pines son de 8 bits cada uno, y por lo tanto tienen un rango de parámetros de 0-255 siendo 0-3.3V respectivamente. Conociendo esto y recordando que 3.3V son 10V amplificados, solo restó hacer una relación de los 4 parámetros relacionados, la cual se muestra en la siguiente tabla.

Nota: Aunque los servomotores tienen la capacidad de abrir hasta 180°, las ventanas instaladas en el auditorio solo pueden ser abiertas a 90° por lo tanto hasta ese valor se consideró para el proyecto.

VALOR DAC	VOLTAJE EN ESP32	VOLTAJE AMPLIFICADO	GRADOS - % DE APERTURA
44	0.66V	2V	0° = 0%
70	0.97V	3V	22.5° = 25%
96	1.32V	4V	45° = 50%
123	1.65V	5V	65.5° = 75%
150	1.98V	6V	90° = 100%

Tabla 5.4 Relación de variables del sistema de control.

Limitaciones: El sistema de control está diseñado para colocarse en el auditorio Tonatiuh del IER-UNAM, sin embargo, debido a la crisis de salud que estamos pasando por la llegada del SARS-COVID19 y todas las medidas de prevención que el gobierno ha establecido, no fue posible el acceso al instituto durante el periodo que se desarrolló el proyecto de residencia. Por lo que el dispositivo fue probado de manera provisional en casa utilizando ambas manos, una usando el control del sistema, seleccionando los niveles de apertura y la otra mano usándola como simulación del movimiento del servomotor abriendo las ventanas produciendo la inclinación del acelerómetro.

Descripción general de funcionamiento: El sistema de control de ventanas consiste en un dispositivo montado en una placa protoboard que cuenta con una interfaz de interacción con el sistema, la cual es conformada por una pantalla LCD de 20x4 y una botonera de 4 push-button con las etiquetas (SI, NO, ↑, ↓) respectivamente. Al ejecutar por primera vez el sistema, la pantalla muestra el siguiente mensaje “BIENVENIDO, SISTEMA DE CONTROL DE VENTILAS”, luego el programa manda a traer al MPU6050 la inclinación a la que se encuentran las ventanas, si la inclinación = 0 entonces se mostrará en pantalla el siguiente mensaje “VENTILAS CERRADAS, ¿DESEA ABRIR?”, con la ayuda de los botones el usuario seleccionará entre si o no según el deseo, si oprime el botón NO entonces regresará a la pantalla anterior que muestra el estado de las ventanas, si el usuario elige SI entonces se mostrará el siguiente mensaje “ELIJA EL NIVEL DE APERTURA DESEADO, USE LA FLECHA ARRIBA Y ABAJO PARA DESPLAZARSE”, entonces se abrirá otra pantalla con un menú de los diferentes niveles de apertura que puede realizar “25%, 50%, 75%, 100%” y una flecha indicadora “←” que se moverá según bajen o suban de opción, suponiendo que elige abrir las ventanas al 25% se mostrará en pantalla “ABIENDO A 25%...” y justo en ese momento del GPIO25 que es nuestro Pin DAC mandará el voltaje requerido a la entrada de nuestro amplificador de voltaje, el cual amplificará al nivel de señal requerido por los servomotores para abrir a 22.5° en este caso, los servomotores abrirán las ventanas y con ellas se moverá el acelerómetro instalado en su estructura, el programa entonces verificará la inclinación pidiendo una vez más al MPU su estado, una vez que coincida con el nivel de apertura ejecutado entonces aparece el siguiente mensaje “OPERACIÓN EXITOSA” y regresa a la pantalla que muestra el estado actual de las ventanas esperando a que se realice la siguiente modificación. Todas las demás opciones funcionan de la misma manera.

5.2.2 Programación

El primer programa realizado con el acelerómetro MPU6050 fue que en pantalla devolviera la aceleración de la gravedad en los 3 ejes: X, Y, Z.

Para ello lo que se hizo fue definir el nivel de sensibilidad como lo mencionamos anteriormente en $+2g$, y una vez definido solo se hace una división del valores sin procesar de los ejes X,Y,X entre el valor del rango de sensibilidad

funcionesMPU.py

```
1 # - SISTEMA DE CONTROL DE VENTILAS -
2
3 # POR: GIOVANNI VELAZQUEZ AVILEZ
4 # ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA
5 # RESIDENTE EN EL IER-UNAM, AGO-DIC 2020
6
7 from machine import I2C, Pin
8 from time import sleep
9 import time
10 import math
11 import machine
12 import utime
13
14
15 MPU6050_ADDR = 0x68
16
17 MPU6050_ACCEL_XOUT_H = 0x3B
18 MPU6050_ACCEL_XOUT_L = 0x3C
19 MPU6050_ACCEL_YOUT_H = 0x3D
20 MPU6050_ACCEL_YOUT_L = 0x3E
21 MPU6050_ACCEL_ZOUT_H = 0x3F
22 MPU6050_ACCEL_ZOUT_L = 0x40
23 MPU6050_TEMP_OUT_H = 0x41
24 MPU6050_TEMP_OUT_L = 0x42
25 MPU6050_GYRO_XOUT_H = 0x43
26 MPU6050_GYRO_XOUT_L = 0x44
27 MPU6050_GYRO_YOUT_H = 0x45
28 MPU6050_GYRO_YOUT_L = 0x46
29 MPU6050_GYRO_ZOUT_H = 0x47
30 MPU6050_GYRO_ZOUT_L = 0x48
31 MPU6050_PWR_MGMT_1 = 0x6B
32
33 MPU6050_LSBG = 340.0
34 MPU6050_TEMP_OFFSET = 36.53
35 MPU6050_LSBG = 16384.0
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52     return [combine_register_values(accel_x_h, accel_x_l) / MPU6050_LSBG,
53             combine_register_values(accel_y_h, accel_y_l) / MPU6050_LSBG,
54             combine_register_values(accel_z_h, accel_z_l) / MPU6050_LSBG]
```

Se importan los módulos, después se escribe en la memoria del MPU las variables que mide en sus 2 posiciones HIGH y LOW.

La línea 35 que está marcada con rojo indica que ese es el valor de la sensibilidad que escogimos, como lo vimos en la tabla del datasheet.

Por ultimo se combinan los valores que están registrados en los ejes X,Y y Z de la aceleración y se divide entre el valor de la sensibilidad establecida, el cual aparece en un recuadro rojo.

Una vez programado el acelerómetro para que entregue correctamente los valores de la aceleración de la gravedad, el siguiente programa fue determinar los grados de inclinación a partir de la aceleración de la gravedad.

funcionesMPU.py

```
38 RestrictPitch = True
39 radToDeg = 57.2957786

87 def mpu6050_get_accel_angle_zx(i2c):
88     accel_x_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_H, 1)
89     accel_x_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_L, 1)
90     x= combine_register_values(accel_x_h, accel_x_l) / MPU6050 LSBG
91
92     accel_y_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_H, 1)
93     accel_y_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_L, 1)
94     y= combine_register_values(accel_y_h, accel_y_l) / MPU6050 LSBG
95
96     accel_z_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_H, 1)
97     accel_z_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_L, 1)
98     z= combine_register_values(accel_z_h, accel_z_l) / MPU6050 LSBG
99
100    if (RestrictPitch):
101        roll = math.atan2(x, z) * radToDeg
102        pitch = math.atan(-y/math.sqrt((x**2)+(z**2))) * radToDeg
103    else:
104
105        roll = math.atan(x/math.sqrt((y**2)+(z**2))) * radToDeg
106        pitch = math.atan2(-y,z) * radToDeg
107
108    return int(roll)
```

Lo primero que se hace es leer los valores de la aceleración escritos en la memoria del MPU de los 3 ejes, luego se integran a una variable: x y z según corresponda, esto se hace para sintetizar lo más posible el código, una vez hecho esto lo único que resta es aplicar las ecuaciones desarrolladas en el subcapítulo anterior, donde se explico como obtener los ángulos de inclinación del eje x o y , respecto a la gravedad con las rotaciones “roll” y “pitch”, por otro lado recordemos que los resultados de esas funciones están en radianes, y nosotros necesitamos grados, para resolver eso solo nos aseguramos de multiplicar por 57.2957 que son los grados que equivalen a un radian.

Lo siguiente que se programó fue la función que maneja los parámetros del DAC (GPIO25) que permite él envío del voltaje modulado a la entrada de nuestro amplificador de voltaje.

funciones.py

```
36 dac1=DAC(Pin(25))
37 led = Pin(27, Pin.OUT)
38
39
52 def nivel1():
53     print ('-----> CERRANDO VENTILAS <-----')
54     lcd_show("CERRANDO VENTILAS",1)
55     #Parpadea el LED
56
57     led.value(1)
58     utime.sleep(0.1)
59     led.value(0)
60     utime.sleep(0.1)
61     led.value(1)
62     utime.sleep(0.1)
63     led.value(0)
64     volts = 2
65     dac1.write(44)
66     sleep(15)
67     dac1.write(0)
68     sleep(1)
69     return volts
70
71 def nivel2():
72     print ('-----> ABRIENDO 25% <-----')
73     lcd_show("ABRIENDO 25%",1)
74     #Parpadea el LED
75     led.value(1)
76     utime.sleep(0.1)
77     led.value(0)
78     utime.sleep(0.1)
79     led.value(1)
80     utime.sleep(0.1)
81     led.value(0)
82     utime.sleep(0.1)
83     led.value(0)
84     volts = 3
85     dac1.write(70)
86     sleep(15)
87     dac1.write(0)
88     sleep(1)
89     return volts
```

Lo primero que se hace es la asignación de Pines, en este caso el GPIO25 trabajara en DAC y el GPIO27 es el led que indicara cuando la señal de voltaje ha empezado a ser enviada.

Las funciones han sido llamadas “nivel1, nivel2,hasta nivel5”, ya que son los 5 diferentes tipos de voltaje los que mueven los servomotores. la estructura es la misma para las funciones restantes.

Lo que hace la función de inicio es hacer que el led parpadee como indicador que será enviada la señal, después con el dac1.write() se asigna el valor del parámetro DAC y durante 15 segundos se mantiene el envío de la señal mientras abren los servomotores, finalmente para detenerlo solo ponemos el parámetro a 0.

El siguiente programa que se realizó para el sistema de control fue la programación de horarios, dado que se piensa utilizar un sistema de ventilación nocturna se programó la hora de apertura a las 10 pm y la hora de cierre de ventilas a las 7 am. Para ello se utilizó el protocolo NTP que consiste en la consulta de la hora actual mediante la conexión a la red wifi.

funciones.py

```

7 import machine
8 import time
9 import gc
10 import network, utime, ntptime
11 from machine import Pin, DAC, I2C
12 import funcionesMPU
13 from funcionesMPU import *
14 from time import sleep
15 from wifi import activate wifi
16
22
23 red = 'TOTALPLAY_897E2E'
24 clave = 'EZ04022ST1'
29 web_query_delay = 600000
30 timezone_hour = 6 # ZONA HORARIA (hours)
31
32 alarm = [11, 30] # ALARMA[HORA, MINUTO]
33 alarm2 = [11, 56] # ALARMA[HORA, MINUTO]
819 def horario_programado():
820     local_time_sec = utime.time() + timezone_hour * 3600
821     local_time = utime.localtime(local_time_sec)
822     update_time = utime.ticks_ms() - web_query_delay
823
824     ntptime.settime()
825     print("HORA DEL SISTEMA ACTUALIZADO:", utime.localtime())
826     update_time = utime.ticks_ms()
827
828
829     if alarm[0] == local_time[3] and alarm[1] == local_time[4]:
830
831         print("!!! HORA PROGRAMADA !!!")
832         lcd_clear()
833         lcd_show("    10:00 PM    ",0)
834         lcd_show("HORARIO PROGRAMADO",1)
835         lcd_show(" DE APERTURA",2)
836         sleep(5)
837         lcd_clear()
838         niveles()
839         lcd_clear()
840         lcd_show("ESPERE UN MOMENTO...",2)
841         sleep(30)
842         lcd_clear()
843         i2c = I2C(scl=Pin(12), sda=Pin(14))
844         mpu6050_init(i2c)
845         grados=mpu6050_get_accel_angle_zx(i2c)
846         sleep(1)
847         if grados >= 88 and grados <=91:
848             lcd_show(" OPERACION EXITOSA ",1)
849             sleep(5)
850             lcd_clear()

```

Se importan los módulos, después se establece un delay de la consulta web, y también se define el desfase de la zona horaria, en nuestro caso en México es GMT-6 , para la activación de la hora programada se utilizaron 2 alarmas donde se definen la hora y el minuto en el que iniciara la apertura y el cierre.

En cuanto a la función lo que hace al inicio es actualizar la hora en tiempo real mediante el protocolo NTP y después con un “if” se verifica si la hora y minuto programado coincide con la hora y minuto del tiempo real entonces se inicia el proceso de la apertura de las ventilas , mostrando en pantalla “10:00 pm horario programado de apertura”, y lo que se hace es ejecutar la función que abre al 100% las ventilas, una vez ejecutada corrobora con el MPU la inclinación actual y mostrara “operación exitosa”

```

859 elif alarm2[0] == local_time[3] and alarm2[1] == local_time[4]:
860     print("!!! HORA PROGRAMADA !!!")
861     lcd_clear()
862     lcd_show("    07:00 AM    ",0)
863     lcd_show("HORARIO PROGRAMADO",1)
864     lcd_show("    DE CIERRE",2)
865     sleep(5)
866     lcd_clear()
867     nivel1()
868     lcd_clear()
869     lcd_show("ESPERE UN MOMENTO...",2)
870     sleep(30)
871     lcd_clear()
872     i2c = I2C(scl=Pin(12), sda=Pin(14))
873     mpu6050_init(i2c)
874     grados=mpu6050_get_accel_angle_zx(i2c)
875     sleep(1)
876     if grados == 0:
877         lcd_show(" OPERACION EXITOSA ",1)
878         sleep(5)
879         lcd_clear()

```

El proceso para el cierre programado es el mismo que para el de apertura solo con la variación de que hora la alarm2 se ejecutara cuando la hora en tiempo real coincida con el horario programado de cierre de ventanas que es a las 7:00 AM.

Por último, se realizó la programación del archivo main.py, ya que este será el programa principal del sistema y hará que se manden a traer las otras funciones descritas arriba para cumplir con su parte asignada.

main.py

```
1 # - SISTEMA DE MEDICION DE VENTANA DESLIZABLE -
2
3 # POR: GIOVANNI VELAZQUEZ AVILEZ
4 # ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA
5 # RESIDENTE EN EL IER-UNAM, AGO-DIC 2020
6
7 import machine
8 import gc
9 import network, utime, ntptime, time
10 from wifi import activate_wifi
11 from funcion import *
12 from funciones import *
13 import funcionesMPU
14 from funcionesMPU import *
15 from machine import I2C, Pin, DAC
16 from esp8266_i2c_lcd import I2cLcd
17 import math
18 gc.enable()
19
20 red = 'TOTALPLAY_897E2E'
21 clave = 'EZ04022ST1'
22 unique_id = '70099f70-10a1-11eb-9c3f-d1ead9980bc3'
23 token = 'MdihxoaKm0rTUqo9G1RM'
24 activate_wifi(red, clave)
25
26 label = 'modulacion'
27 label2='inclinacion'
28
29 lcd = I2cLcd(I2C(scl=Pin(26), sda=Pin(13)), 0x27, 4, 20)
30 dac1=DAC(Pin(25))
31 boton1 = Pin(15, Pin.IN)
32 boton2 = Pin(4, Pin.IN)
33 boton3 = Pin(17, Pin.IN)
34 boton4 = Pin(18, Pin.IN)
35 # LIMPIAR LCD
36 def lcd_clear():
37     lcd.move_to(0, 0)
38     lcd.putstr(" " * 80)
39
40 # MOSTRAR TEXTO EN LCD
41 def lcd_show(text, line):
42     lcd.move_to(0, line)
43     lcd.putstr(text)
44
45
46 web_query_delay = 600000
47 timezone_hour = 6 #ZONA HORARIA
48
49 local_time_sec = utime.time() + timezone_hour * 3600
50 local_time = utime.localtime(local_time_sec)
51 update_time = utime.ticks_ms() - web_query_delay
52
53 lcd_show("BIENVENIDO",0)
54 lcd_show("SISTEMA DE CONTROL",1)
55 lcd_show("DE VENTILAS",2)
56 sleep(3)
57 lcd_clear()
```

De inicio se importan todos los módulos necesarios para el funcionamiento de estas funciones y de las funciones que serán llamadas a traer, luego se define la parte de la información de la red wifi, el ID y token proporcionados por el asesor externo, se definen las etiquetas que se publicaran, se definen los pines ocupados para el control del sistema, luego se escriben las funciones que limpiaran el LCD y permitirá mostrar el texto en la pantalla.

Luego se establecen los parámetros de la zona horaria requeridos para activar la programación de horarios.

Después se escribe el texto que será mostrado en pantalla una vez que se inicie el sistema de control.

```

while True:
    led.value(0)
    if __name__ == "__main__":
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)

while True:
    if (grados > 0) and (grados <= 23):
        print ('-----> NIVEL DE APERTURA <-----')
        lcd_show("ABIERTAS: 25%",0)
        lcd_show(" CAMBIAR EL ",1)
        lcd_show("NIVEL DE APERTURA?",2)
        lcd_show("SI NO",3)

        if boton4.value() is 0:
            horario_programado()
            lcd_clear()
            lcd_show("ELIJA EL NIVEL DE",0)
            lcd_show("APERTURA DESEADO",1)
            sleep(3)
            lcd_clear()
            lcd_show("USE EL BOTON ARRIBA",0)
            lcd_show(" Y ABAJO PARA",1)
            lcd_show(" DESPLAZARSE",2)
            sleep(3)
            lcd_clear()
            result2=todo25()

            i2c = I2C(scl=Pin(12), sda=Pin(14))
            mpu6050_init(i2c)
            grados=mpu6050_get_accel_angle_zx(i2c)

            valor = result2
            valor2 = grados
            data = [{label: valor},
            {label2: valor2}]
            publish_thingsboard(red,clave,token, unique_id,data)
            sleep(3)
            horario_programado()

        elif boton2.value() is 0:
            lcd_clear()
            sleep(2)

    else:
        continue
break

```

Aquí se ejecuta la función que devuelve los grados de inclinación de las ventanas ubicada en el archivo funcionesMPU.py, con el fin de conocer el estado de las ventanas y de acuerdo a eso mostrar el menú de opciones al usuario

La parte esencial de este sistema está en esta sección, aquí de acuerdo a la inclinación que midió el MPU mencionado arriba, pondrá una condición de que si el valor de la apertura entra dentro del rango de 1 a 23° entonces se dirá que las ventanas están abiertas a un 25% y después el usuario tendrá que seleccionar si quiere o no cambiar la apertura.

Si dijere que si escogerá en el menú de las opciones a qué nivel quiere cambiarlo y de acuerdo al que elija se ejecutará la función que aplique el voltaje correcto a la entrada del amplificador de voltaje y posteriormente a entrada de los servomotores y ejecute la modificación.

Por último, se manda a ThingsBoard los datos del cambio de apertura, tanto la inclinación como el voltaje.

```

294     elif (grados < 0):
295         horario_programado()
296         lcd_clear()
297         lcd_show(" APERTURA FUERA DE",0)
298         lcd_show("      RANGO!",1)
299         sleep(3)
300         lcd_clear()
301         lcd_show(" AJUSTE LA POSICION",1)
302         lcd_show(" DEL ACELEROMETRO",2)
303         sleep(5)
304         lcd_clear()
305         break
306
307     elif (grados > 91):
308         horario_programado()
309         lcd_clear()
310         lcd_show(" APERTURA FUERA DE",0)
311         lcd_show("      RANGO!",1)
312         sleep(3)
313         lcd_clear()
314         lcd_show(" AJUSTE LA POSICION",1)
315         lcd_show(" DEL ACELEROMETRO",2)
316         sleep(5)
317         lcd_clear()
318         break

```

Como última parte del código del archivo principal se implemento una alerta de fallo, consiste principalmente en que el programa verifique que el MPU este dentro del rango de inclinación normal de trabajo que es de 0° a 90°, si el programa detecta que está a una inclinación menor de 0° o mayor de 91° (dejando un grado como margen) entonces mandara un texto en pantalla con las palabras “Apertura fuera de rango!, ajuste la posición del acelerómetro”, ya que muy posiblemente la causa de esto es que el acelerómetro se haya ácido o desacomodado de las ventillas.

5.2.3 PCB

Se realizó también el diseño PCB del sistema de control con sus componentes que lo integran, a continuación, se muestra el esquemático del diagrama de conexión y el diseño PCB.

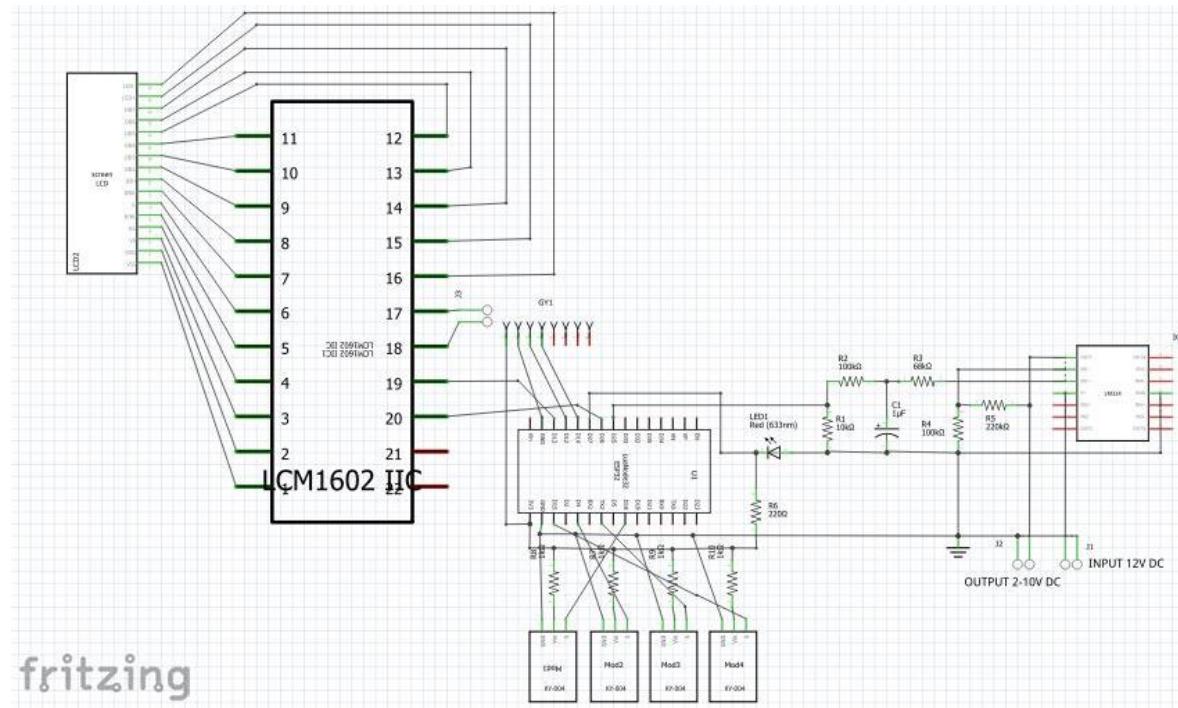


Figura 5.10 Esquemático del sistema de control.

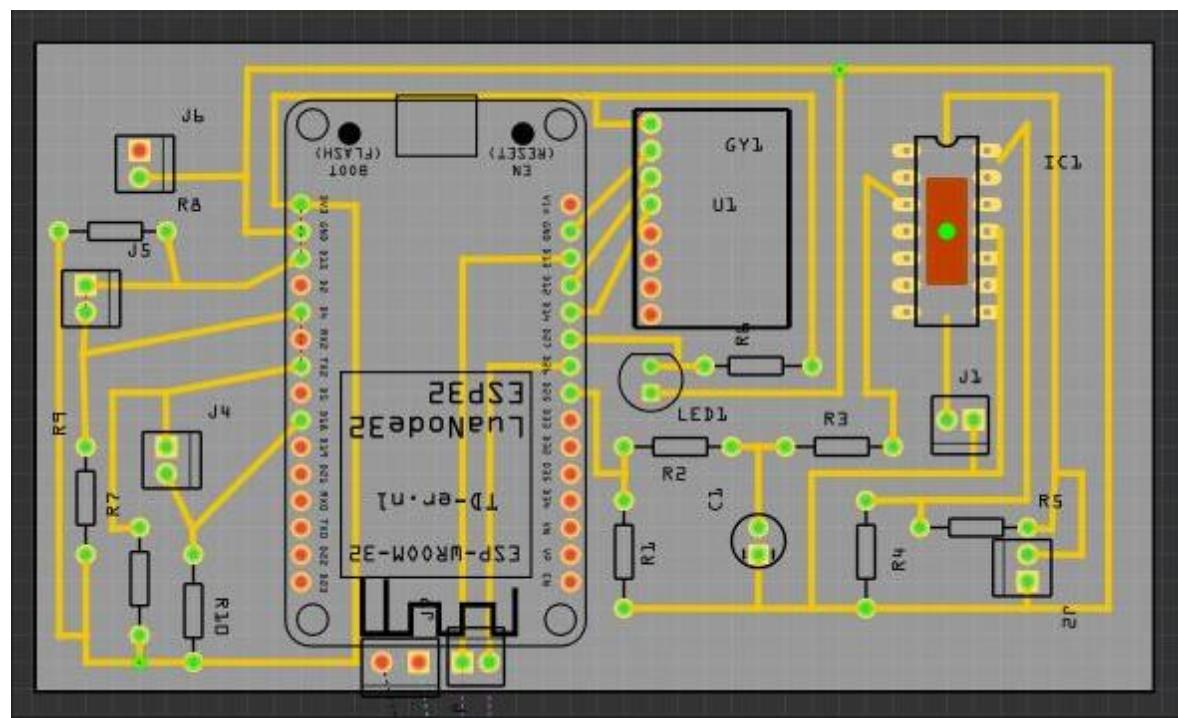


Figura 5.11 Diseño PCB del sistema de control.

CAPÍTULO 6

RESULTADOS

En este capítulo se muestran los resultados obtenidos del sistema de medición de ventana deslizable y del sistema de control de ventanas desarrollados en este proyecto.

6.1 SISTEMA DE MEDICIÓN DE VENTANA DESLIZABLE

A continuación, se muestran los resultados del primer programa:

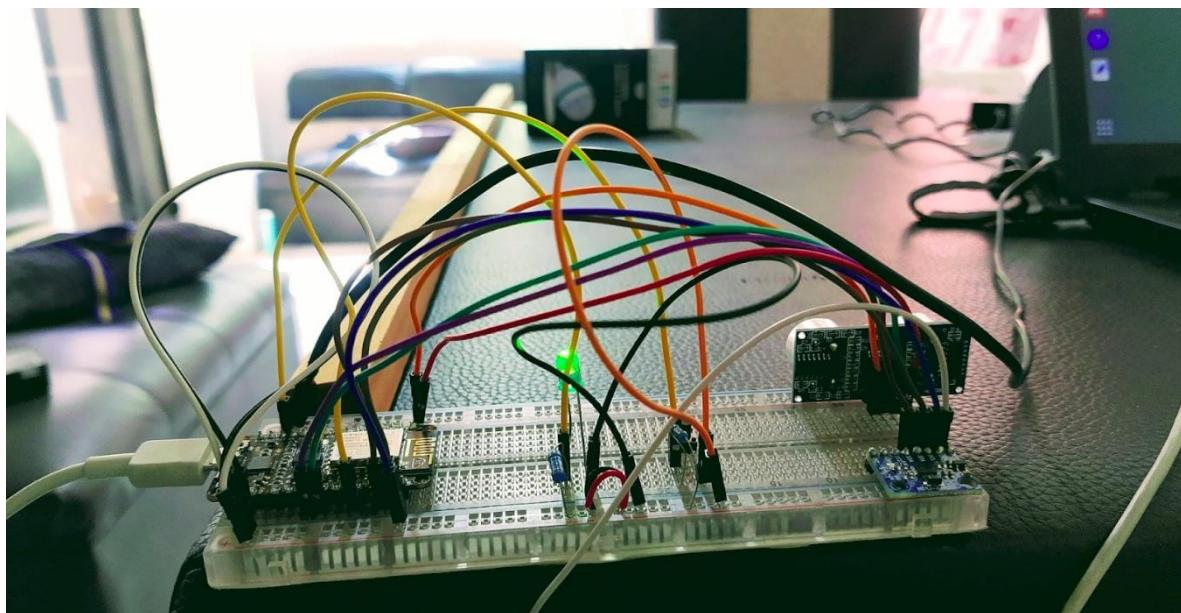


Figura 6.1 Sistema de medición.

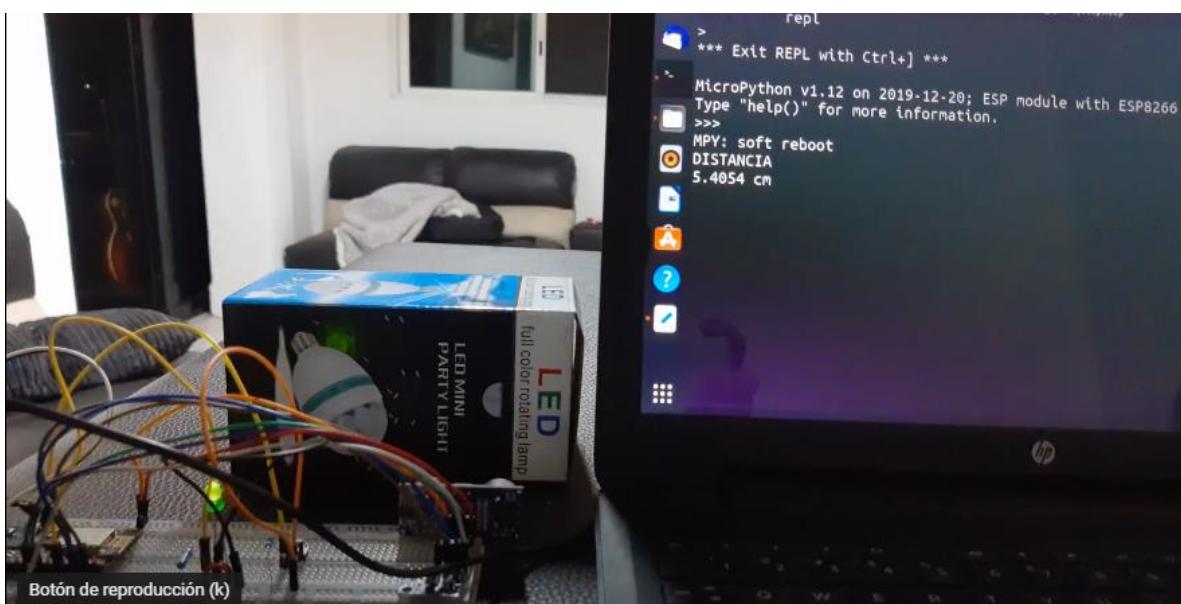


Figura 6.2 Lectura 1 del sensor.



Figura 6.3 Lectura 2 del sensor.

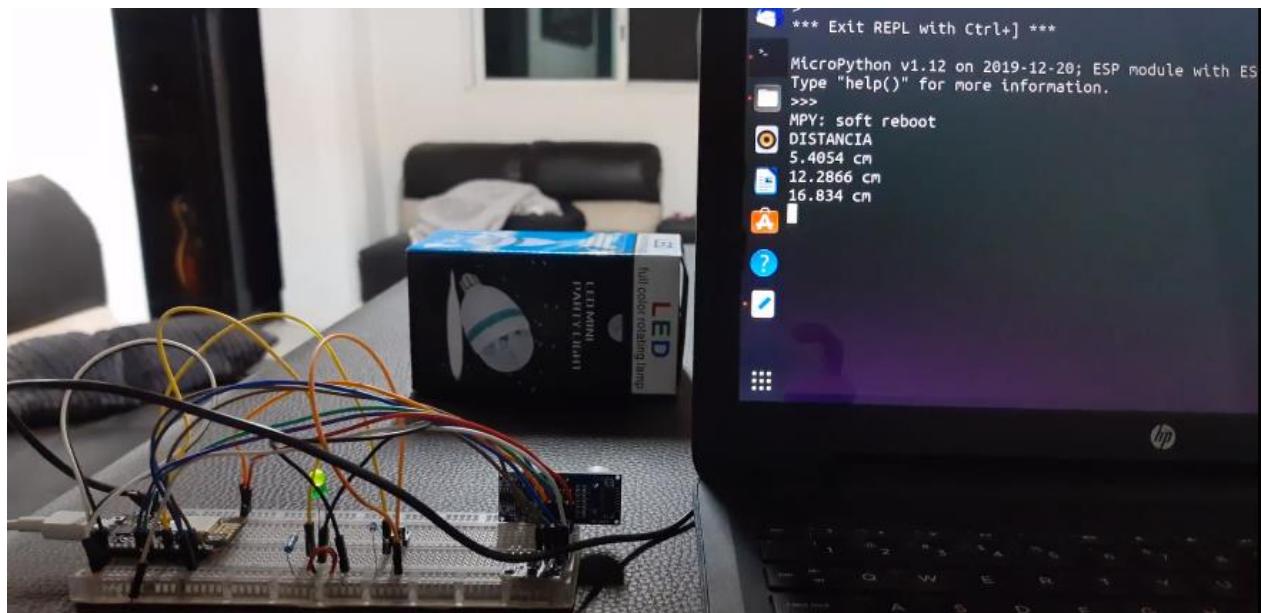
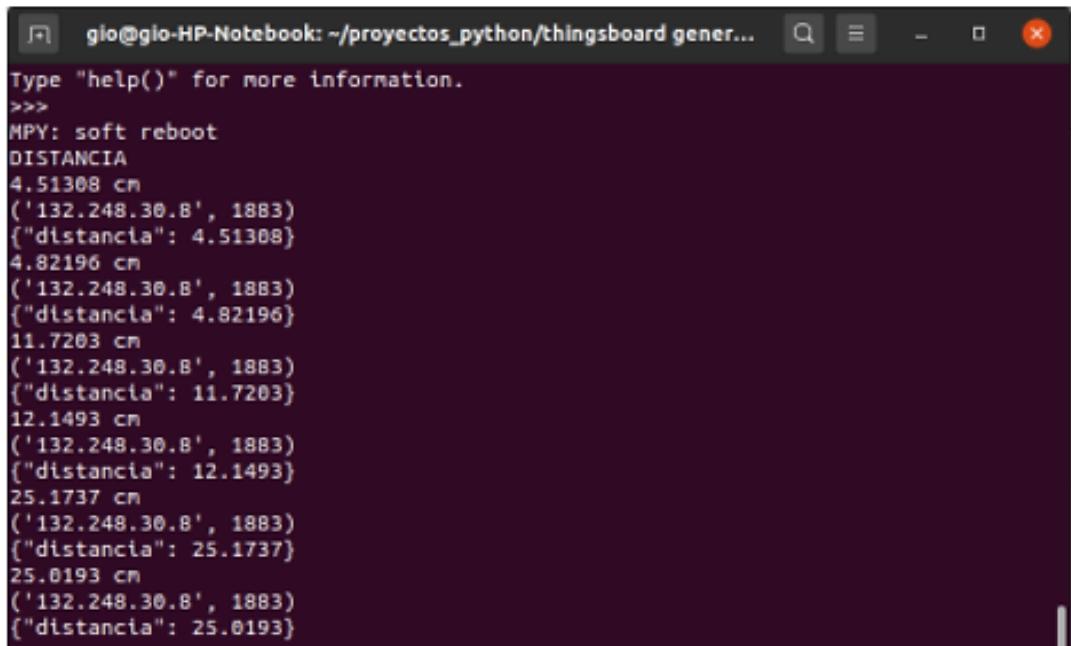


Figura 6.4 Lectura 3 del sensor.

Segundo programa (publicación en ThingsBoard):



```
gio@gio-HP-Notebook: ~/proyectos_python/thingsboard genera...
Type "help()" for more information.
>>>
MPY: soft reboot
DISTANCIA
4.51308 cm
('132.248.30.8', 1883)
{"distancia": 4.51308}
4.82196 cm
('132.248.30.8', 1883)
{"distancia": 4.82196}
11.7203 cm
('132.248.30.8', 1883)
{"distancia": 11.7203}
12.1493 cm
('132.248.30.8', 1883)
{"distancia": 12.1493}
25.1737 cm
('132.248.30.8', 1883)
 {"distancia": 25.1737}
25.0193 cm
('132.248.30.8', 1883)
 {"distancia": 25.0193}
```

Figura 6.5 Lecturas del sensor en la terminal.

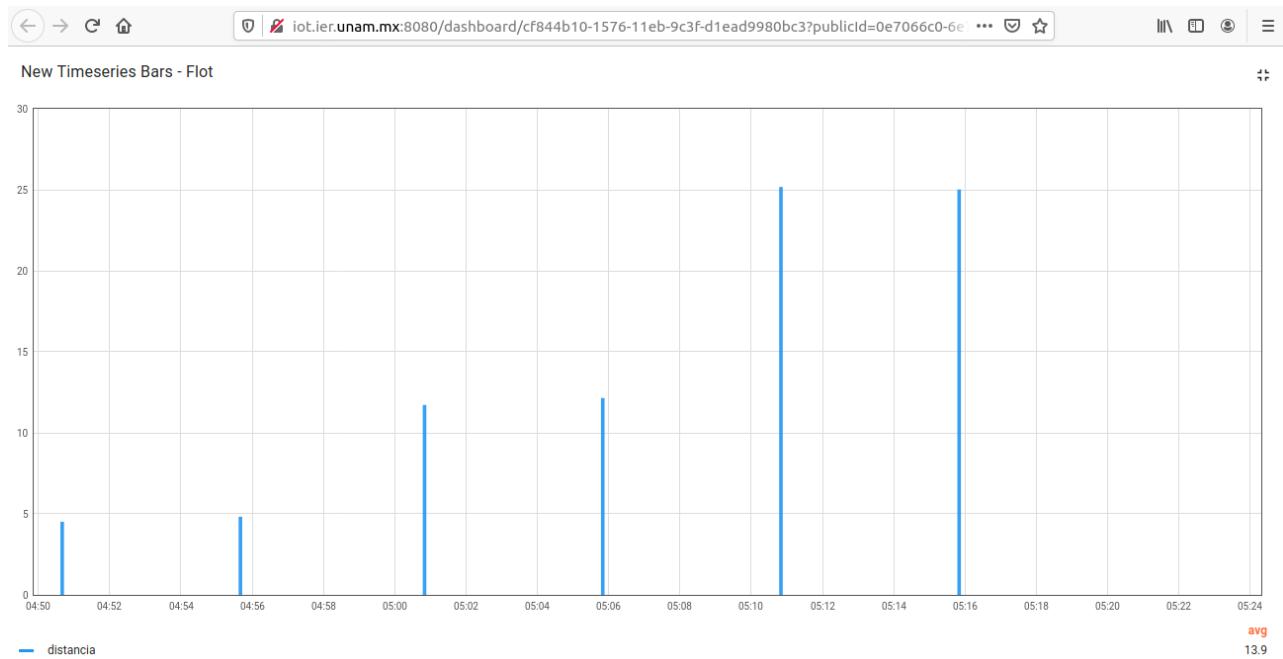


Figura 6.6 Datos publicados en ThingsBoard.

Uso del “deepsleep” y del sensor de vibración:



Figura 6.7 Modo Deep Sleep del ESP8266.

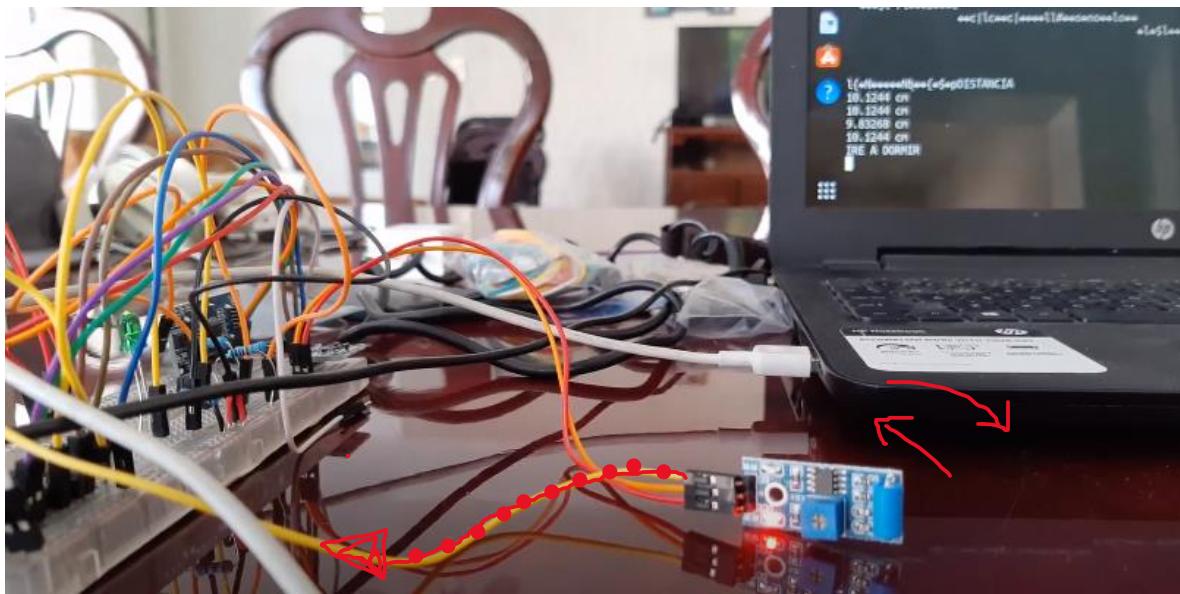


Figura 6.8 Sensor de vibración en funcionamiento.

Nota: Puesto que para la demostración del modo “deepsleep” y del sensor de vibración no puede apreciarse en una imagen aquí se deja el enlace para ver un video realizado y visualizar mejor su funcionamiento:

<https://drive.google.com/file/d/1h4wpB4hpo-rWfWRJp1RCSXv6juMEL1Z1/view?usp=sharing>

https://drive.google.com/file/d/1myFEn-yOMij_-DZfEMccyFSA8YDVwizt/view?usp=sharing

Dispositivo en ventana:

El dispositivo se ha colocado en la ventana que se desliza, a una altura aproximadamente a la mitad respecto a la posición vertical y al costado derecho respecto a la posición horizontal de la ventana (ver figura), justo frente al sensor se ha colocado una especie de tabla rectangular de material MDF de 3mm de espesor fija en el marco de la ventana para que sea ahí donde rebote la señal de ultrasonido (ver figura), unos centímetros más arriba del dispositivo se colocó el sensor de vibración, ambos se fijaron con un poco de silicon caliente, en la figura () se muestra la publicación de datos con el dispositivo ya instalado.



Figura 6.9 Dispositivo colocado en ventana.

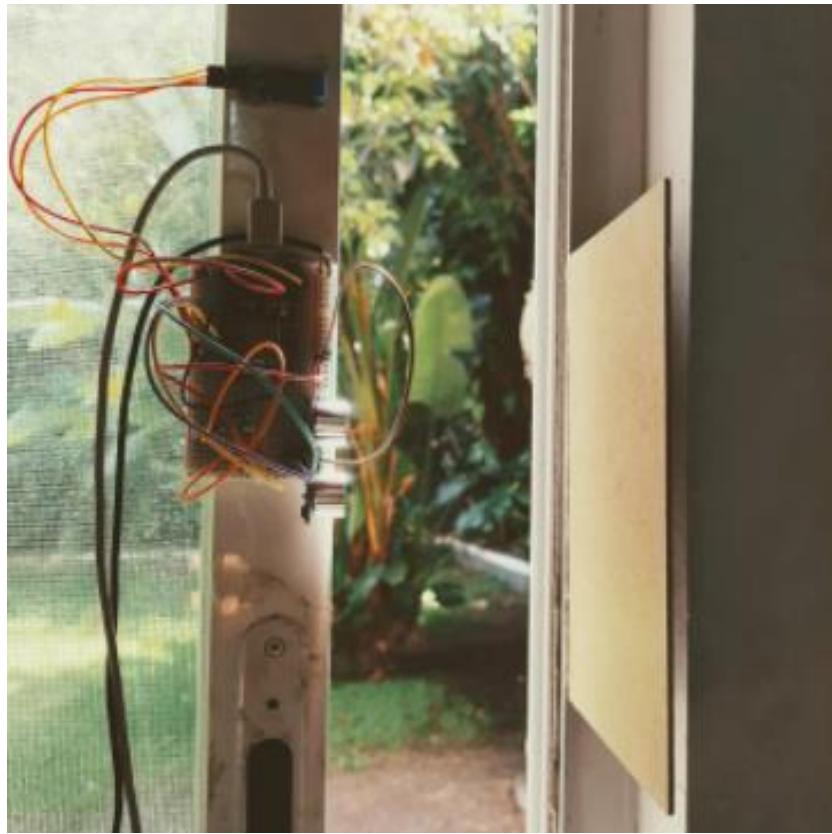


Figura 6.10 Objeto para el rebote de la señal.

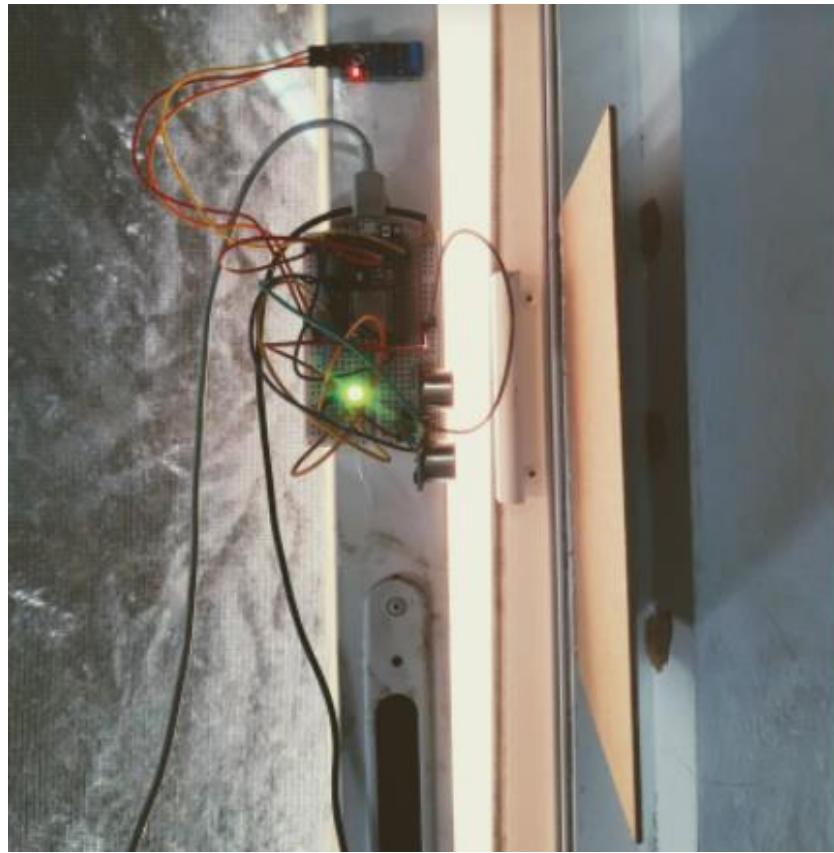


Figura 6.11 Dispositivo alimentado.

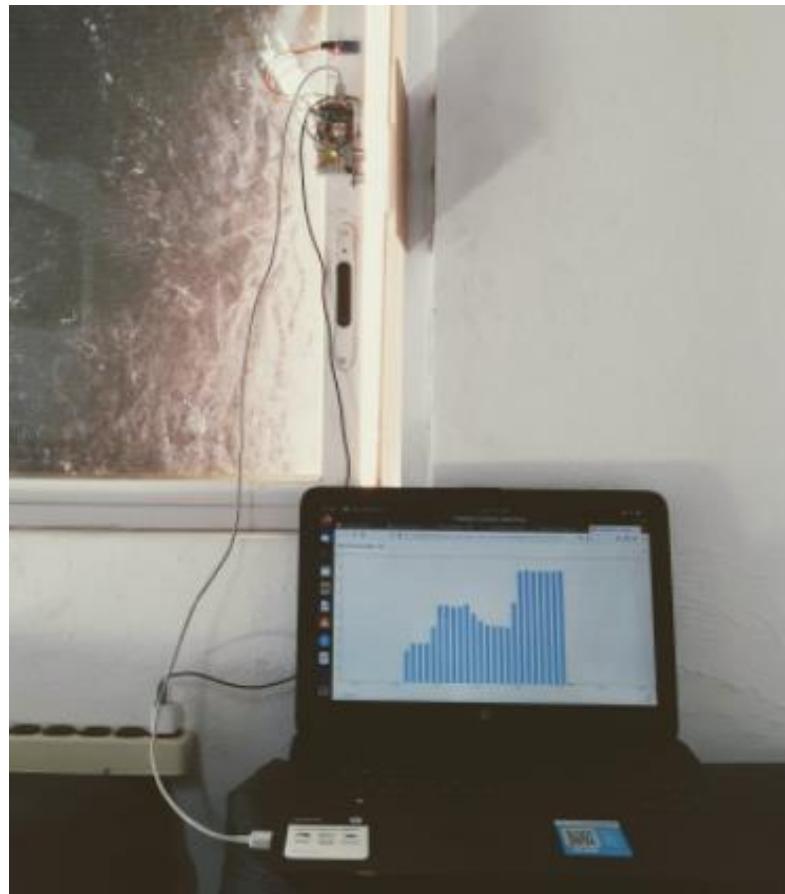


Figura 6.12 Sistema de medición publicando en ThingsBoard.

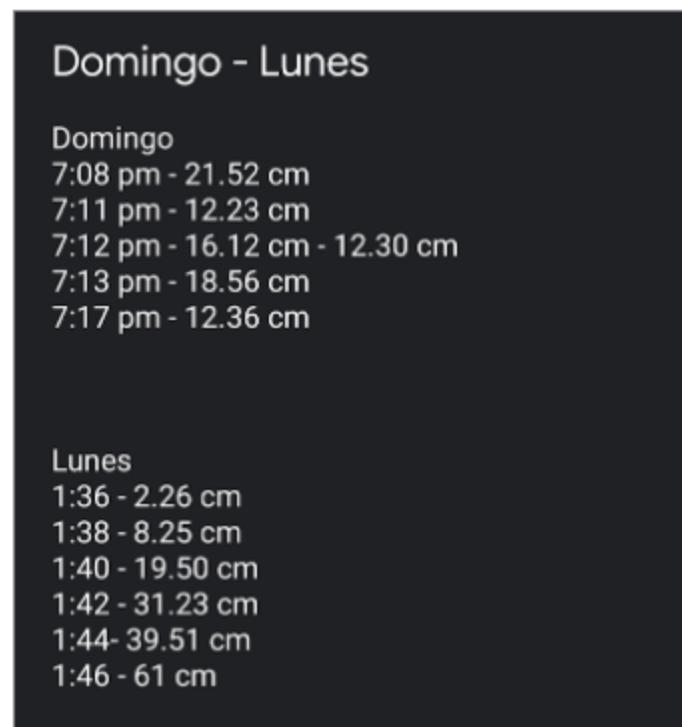


Figura 6.13 Bitácora de aperturas de la ventana.

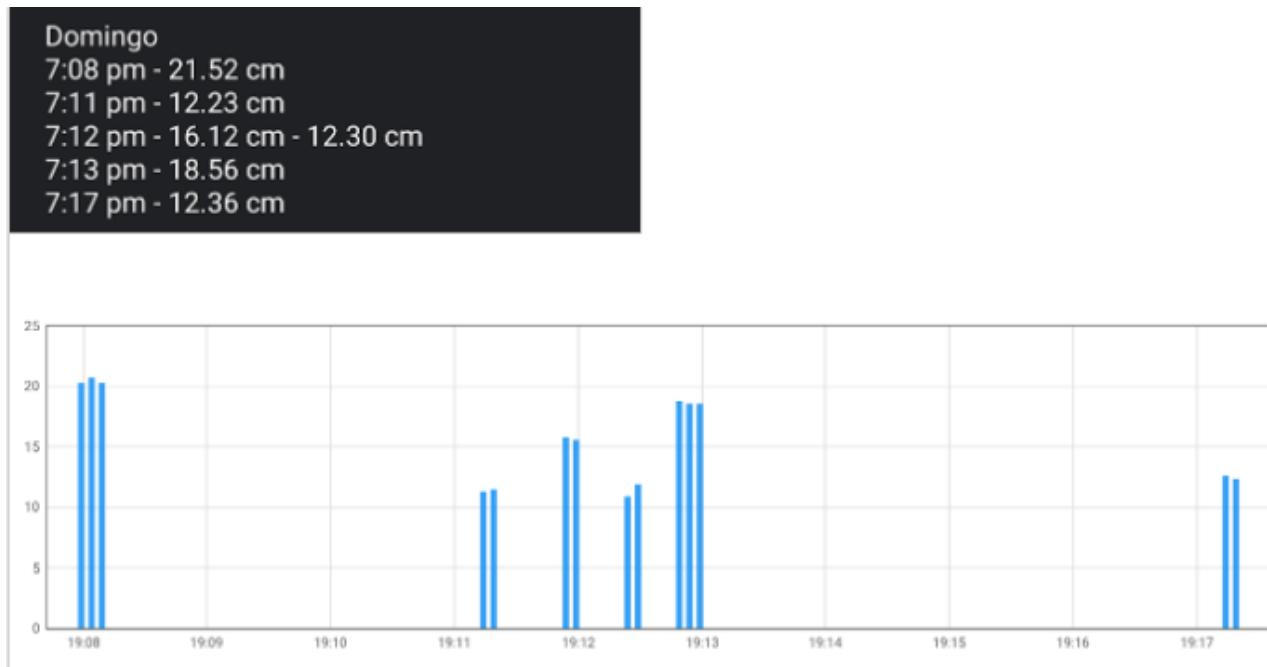


Figura 6.14 Bitácora del día Domingo.

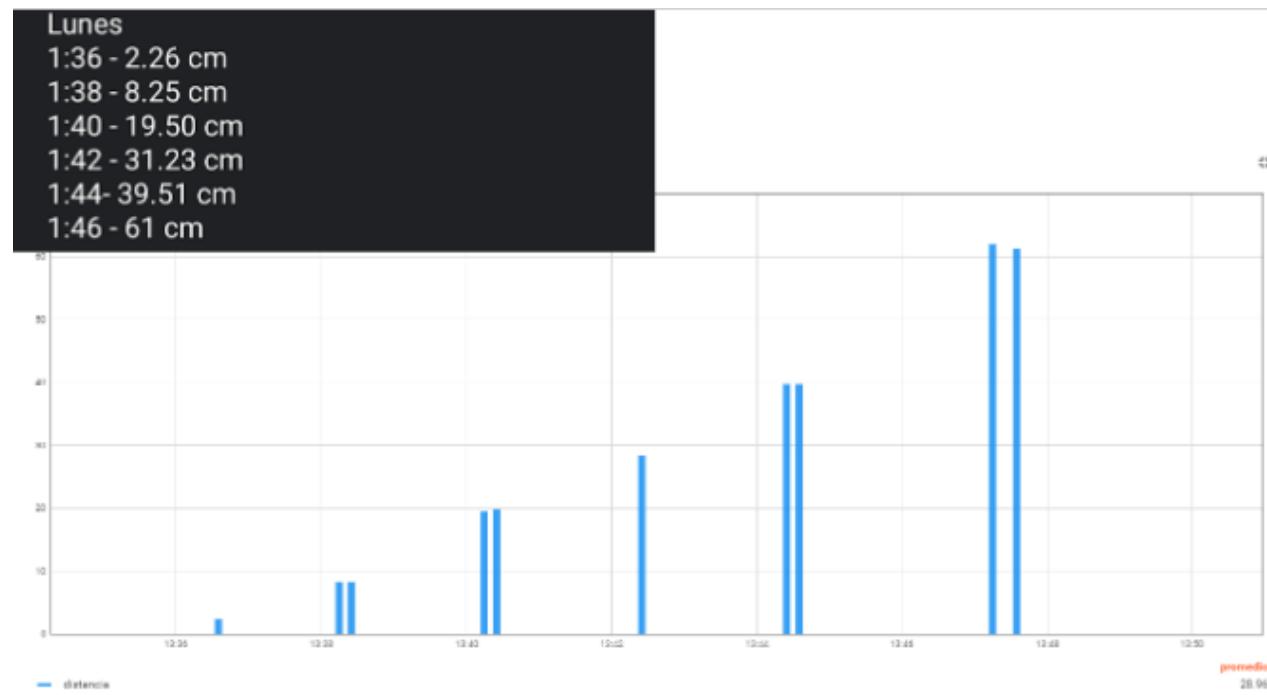


Figura 6.15 Bitácora del día Lunes.

6.2 SISTEMA DE CONTROL DE VENTILAS

A continuación, se muestra el primer programa realizado:

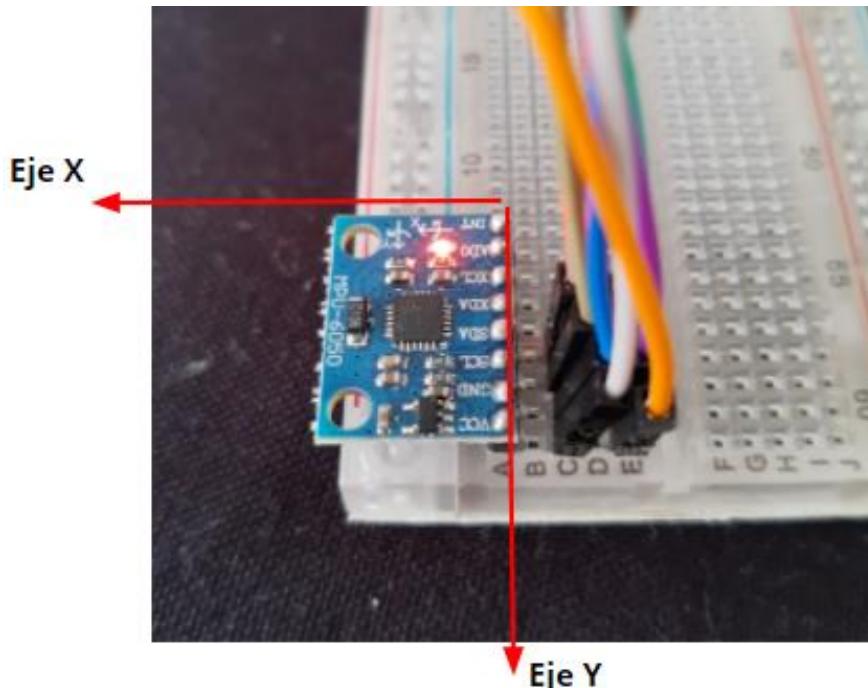


Figura 6.16 Posición del acelerómetro

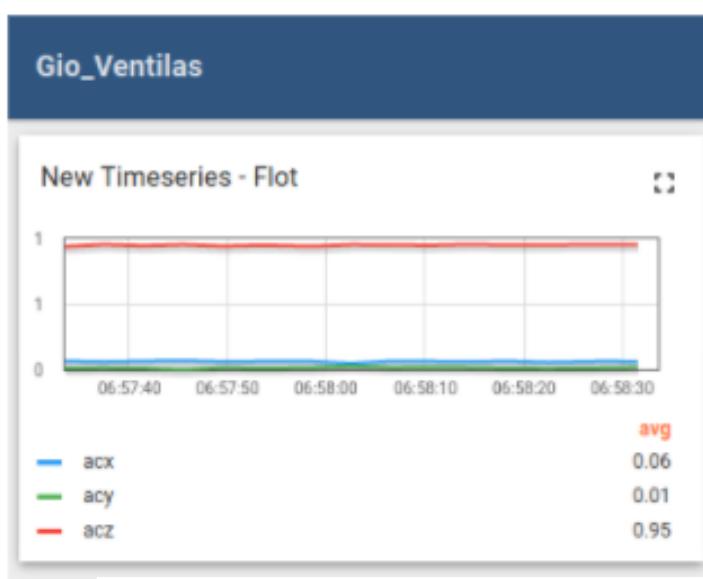


Figura 6.17 Aceleración de AcZ.

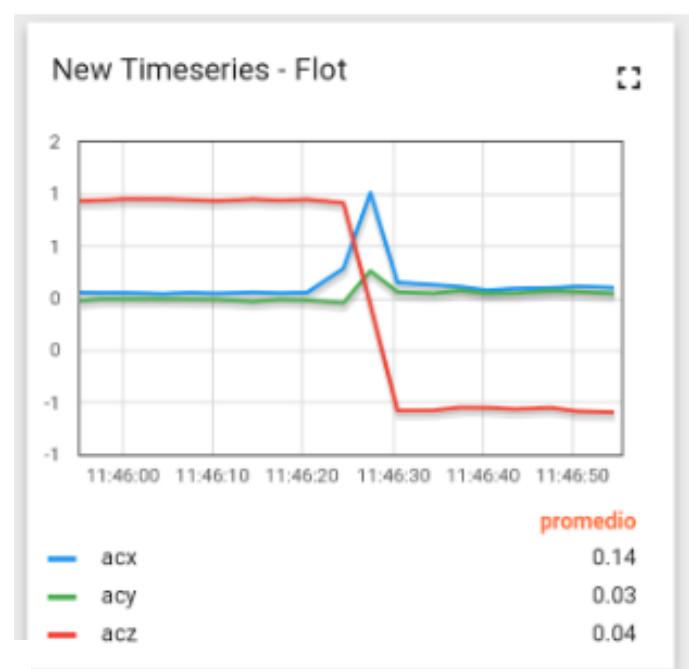


Figura 6.18 Aceleración de AcZ (MPU de cabeza).

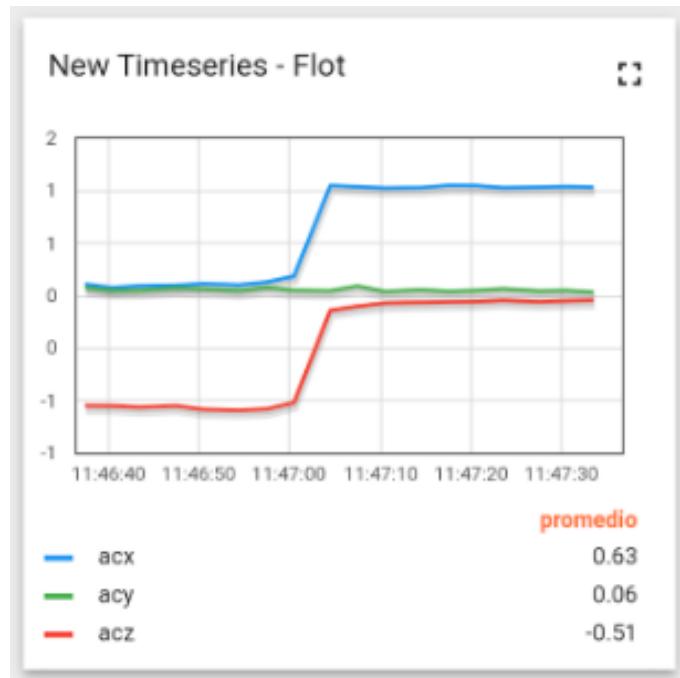
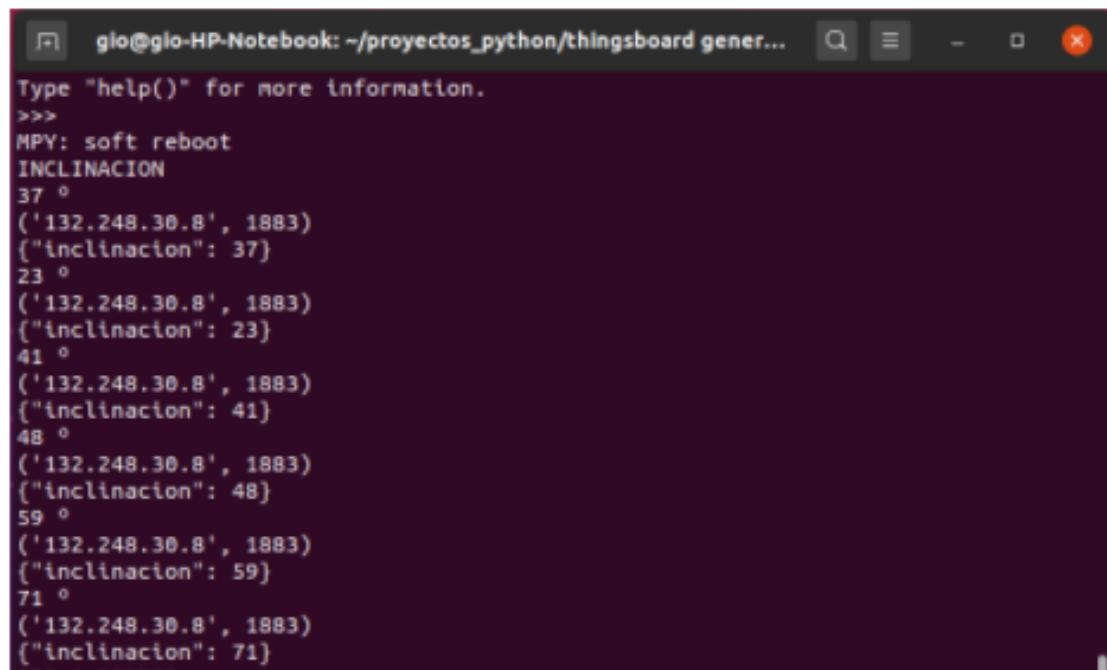


Figura 6.19 Aceleración de AcX.



Figura 6.20 Aceleración de AcY.

Resultados del segundo programa realizado (grados de inclinación a partir de la aceleración):



```
glo@glo-HP-Notebook: ~/proyectos_python/thingsboard genera... 
Type "help()" for more information.
>>>
MPY: soft reboot
INCLINACION
37 °
('132.248.30.8', 1883)
{"inclinacion": 37}
23 °
('132.248.30.8', 1883)
{"inclinacion": 23}
41 °
('132.248.30.8', 1883)
{"inclinacion": 41}
48 °
('132.248.30.8', 1883)
{"inclinacion": 48}
59 °
('132.248.30.8', 1883)
 {"inclinacion": 59}
71 °
('132.248.30.8', 1883)
 {"inclinacion": 71}
```

Figura 6.21 Grados de inclinación en la terminal.

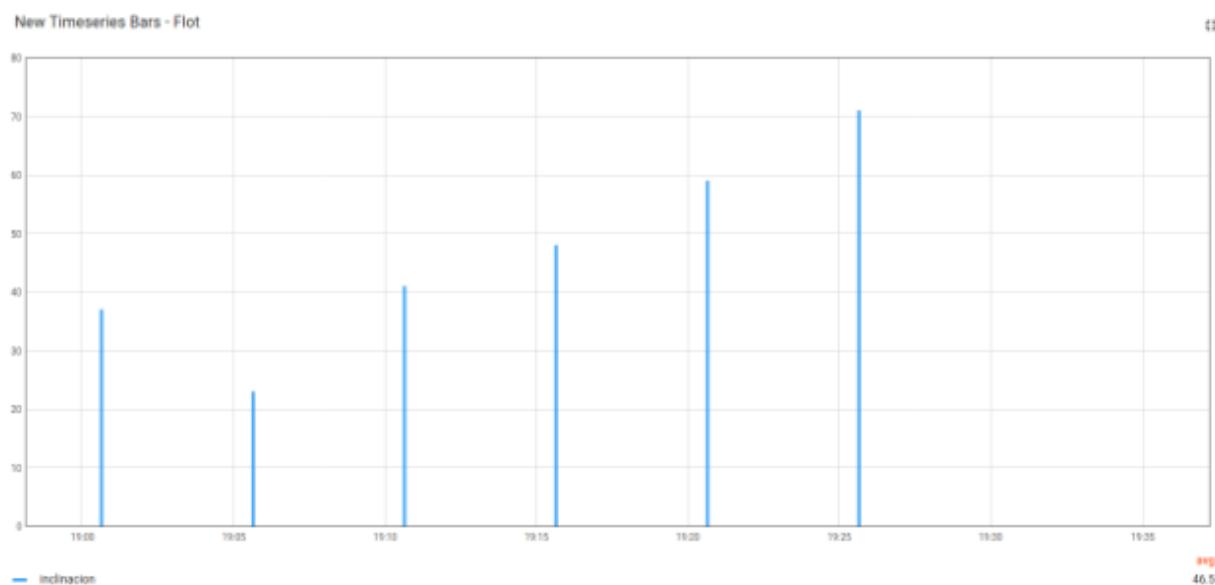


Figura 6.22 Inclinación publicada en ThingsBoard.

Para corroborar que estuviera midiendo bien los grados de inclinación de usaron 2 escuadras a fin de medir 30, 45 y 60° y publicar en ThingsBoard

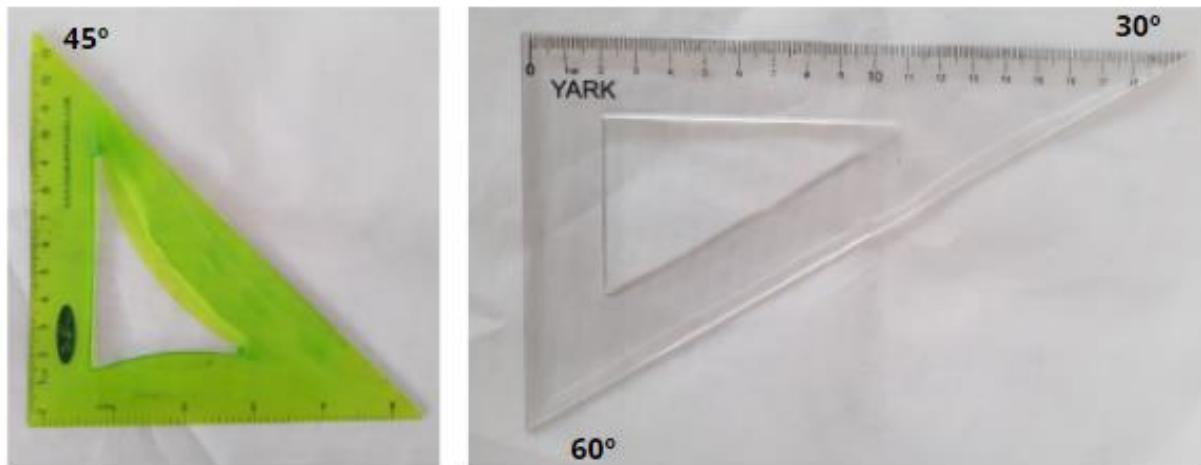


Figura 6.23 Escuadras de 30 y 45°.

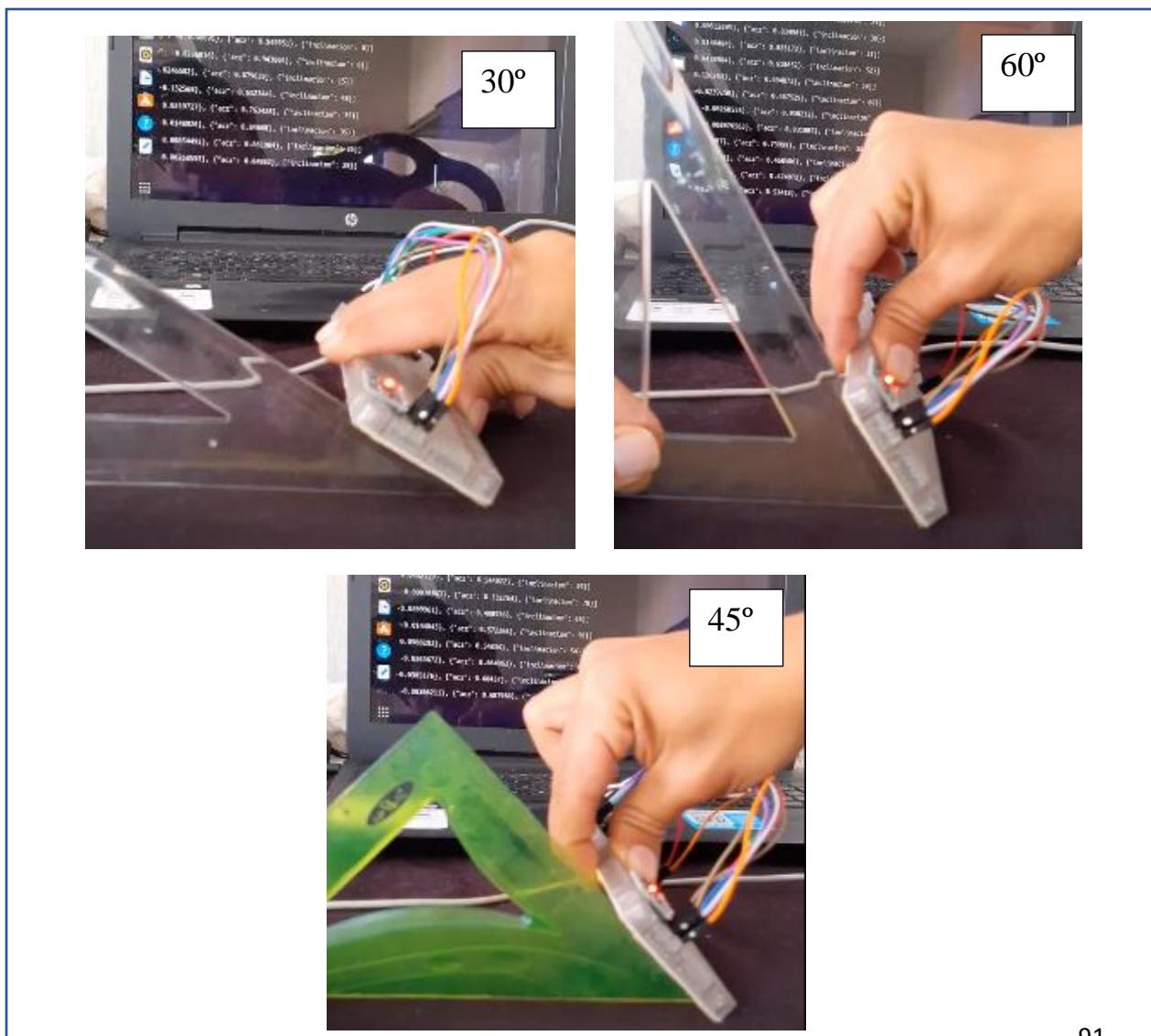


Figura 6.24 Acelerómetro inclinado a 30, 45, y 60°.

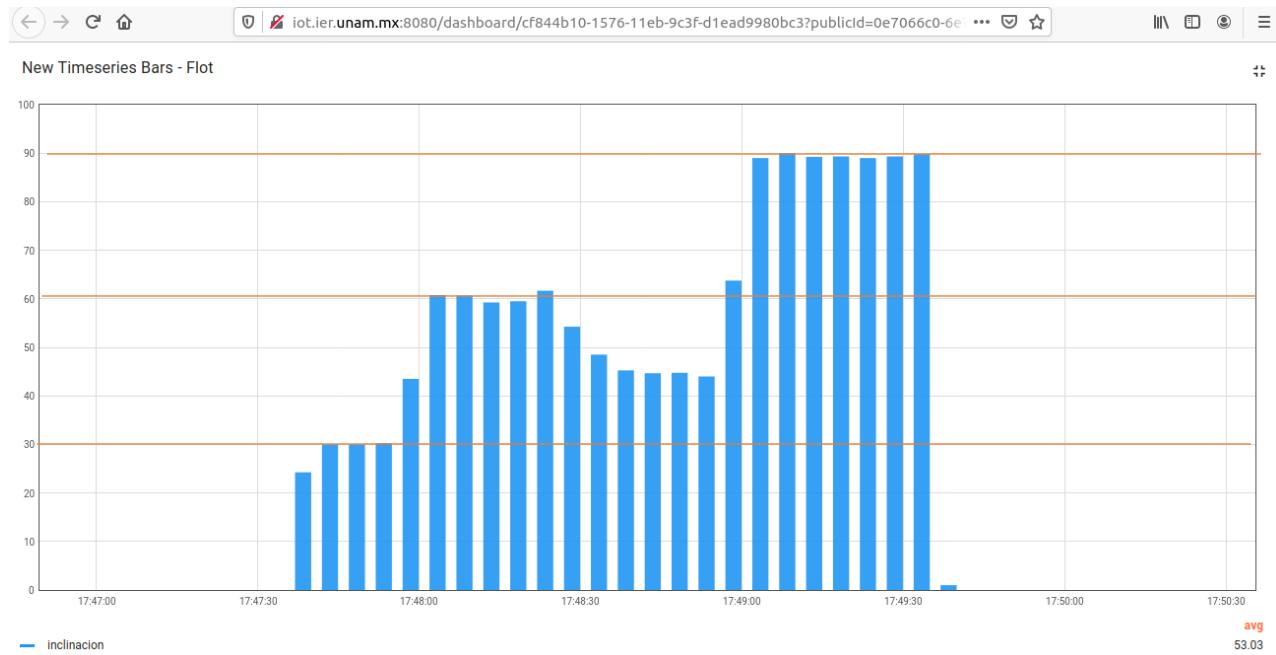


Figura 6.25 inclinación de 30, 45, y 60° en ThingsBoard.

A continuación, se muestran los resultados del tercer programa realizado (uso del DAC para enviar voltaje, pasa por el amplificador de voltaje y finalmente con un multímetro corroboramos que llegue el voltaje amplificado):

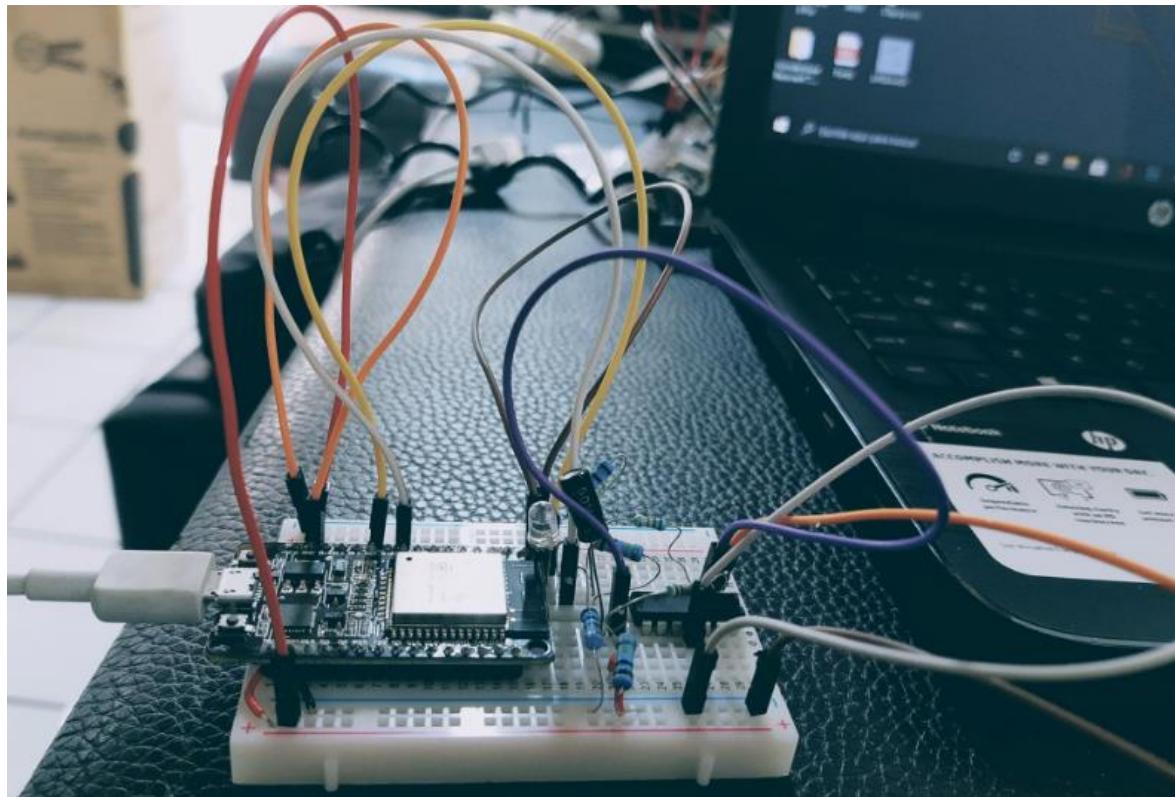


Figura 6.26 Amplificador de voltaje.

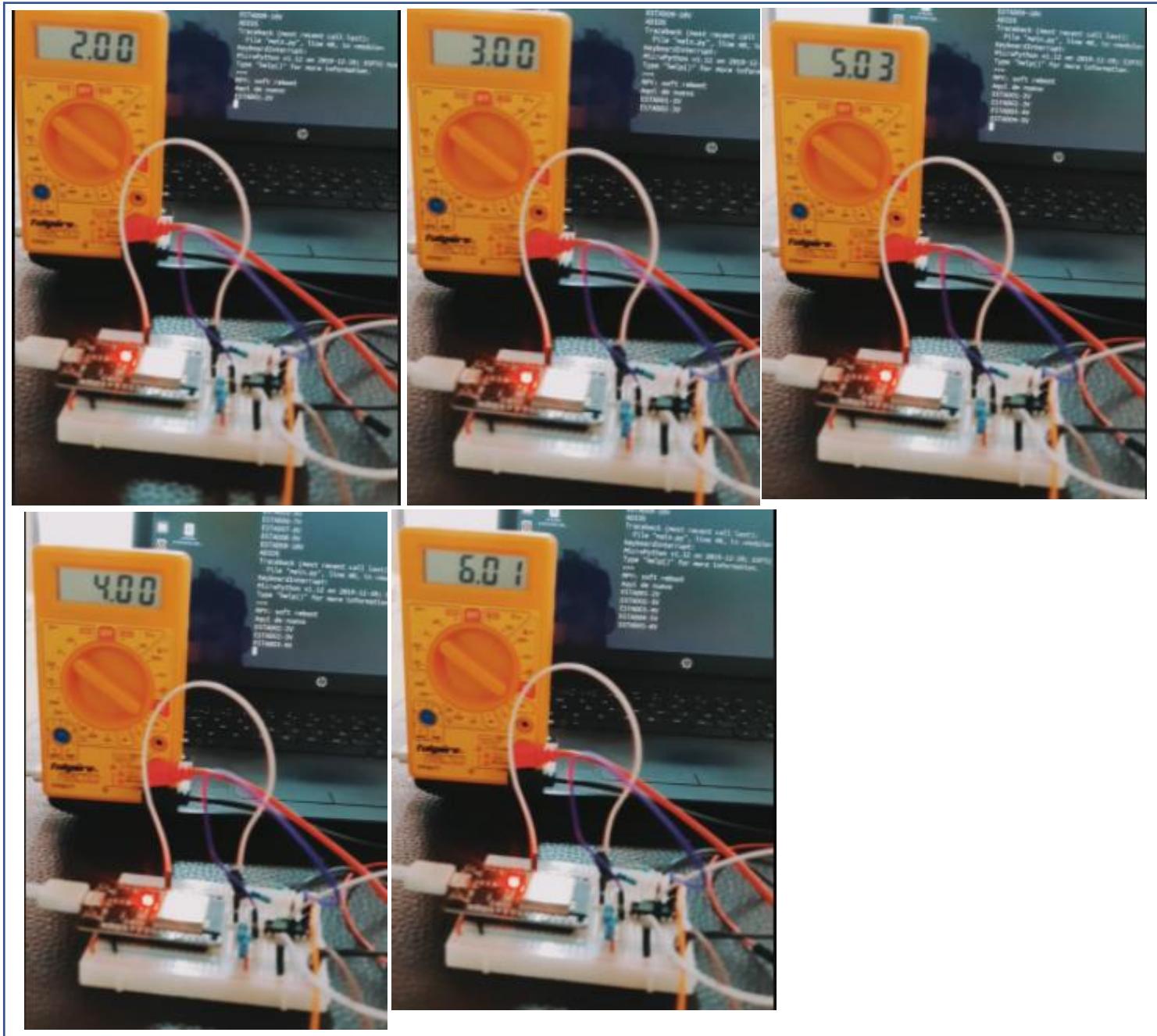


Figura 6.27 Señal amplificada (2-6V).

Los resultados del tercer programa realizado (horario programado), para este ejemplo se programo a las 11:56 pm simulando que fueran las 7:00 am, es decir el horario de cierre de ventanas, esta hora solo se ocupó para ver su funcionamiento.



Figura 6.28 Segunda pantalla del sistema de control.

Como podemos ver a las 11:55:77 pm todavía no sucede nada, sino es hasta que llega el minuto 56 como se muestra a continuación.



Figura 6.29 Pantalla de horario programado.

Finalmente se muestra el ultimo programa que a su vez integra todos los demás, es decir el “sistema de control de ventanas” terminado.

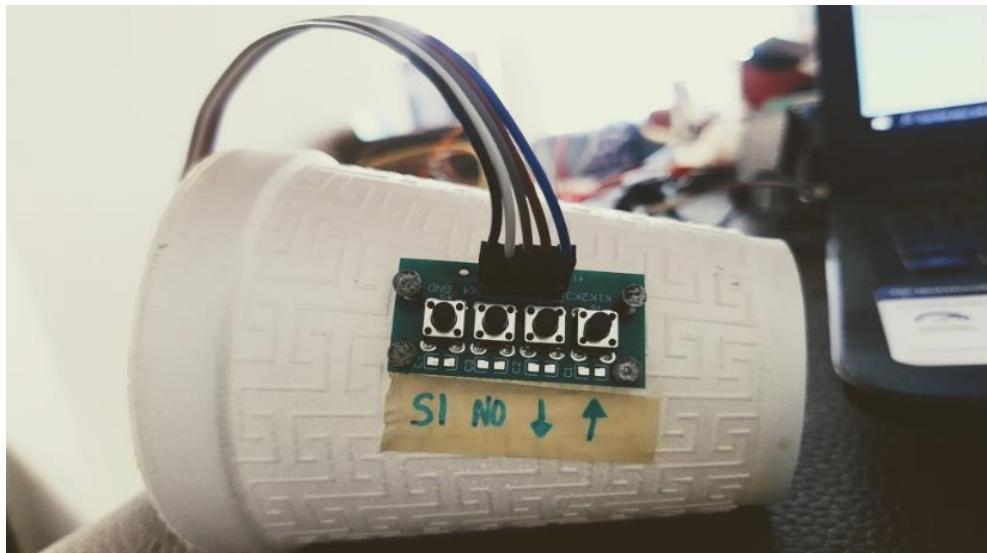


Figura 6.30 Control del sistema.



Figura 6.31 Primer pantalla del sistema de control.

Nota: Para este ejemplo el acelerómetro está en la mesa, no hay ninguna inclinación y por lo tanto la medición será 0° que equivale a ventanas cerradas.



Figura 6.32 Pantalla “ventilas cerradas”

Si el usuario presiona NO entonces regresará a la misma pantalla, pero si presiona SI aparecerán las siguientes pantallas:



Figura 6.33 Pantallas de instrucción del menú

Aquí el usuario deberá desplazarse con las flechas del control y seleccionar el nuevo nivel de apertura, para este ejemplo presionamos 25%.



Figura 6.34 Pantalla menú seleccionable

Puesto que hemos elegido un nuevo nivel de apertura el led parpadeará indicando que ha salido la señal de voltaje a la entrada del amplificador de voltaje, misma que irá a la entrada de la señal de los servomotores causando que se abran las ventanas.

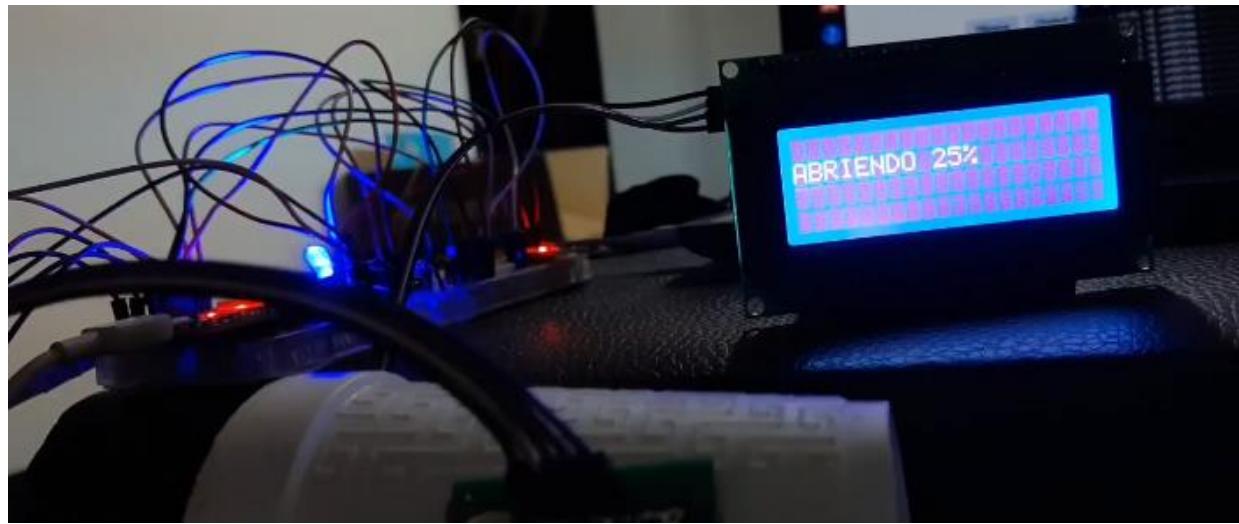


Figura 6.35 Pantalla “abriendo 25%”

Dado que el acelerómetro va fijado a una de las ventanas para ir midiendo los grados de apertura, se hizo la simulación del movimiento del servomotor con la mano, el programa entonces mide la inclinación y al verificar que ahora está a 22.5° entonces se mostrara el texto “operación exitosa” y volverá a la pantalla que muestra el estado actual de apertura.

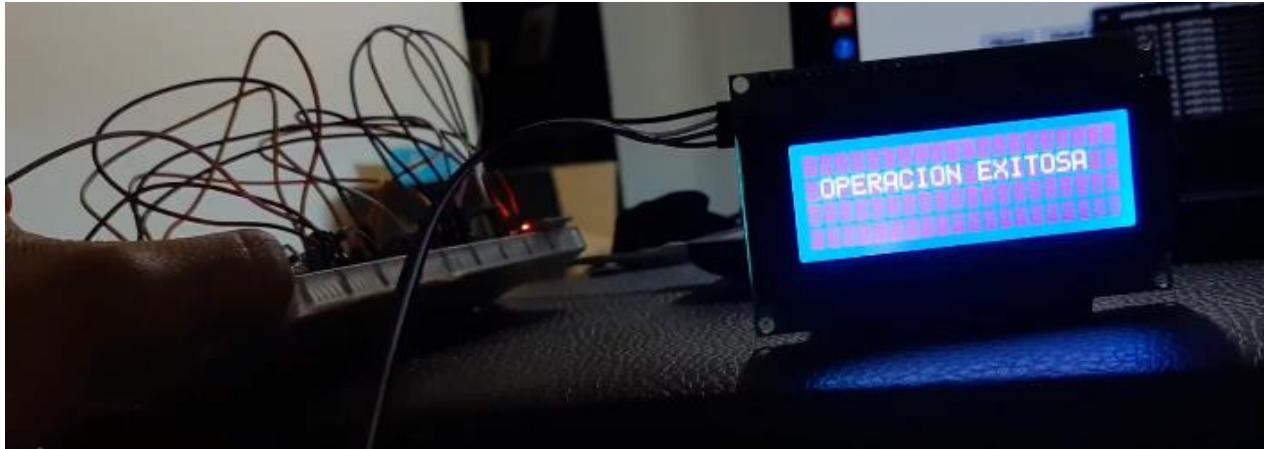


Figura 6.36 Pantalla “operación exitosa”

Ahora mostrara que las ventilas están actualmente abiertas al 25%, puesto que esa fue la última decisión que se tomó.



Figura 6.37 Segunda pantalla del sistema de control

La dinámica de funcionamiento es la misma para cualquier estado de apertura a la que se encuentren las ventilas.

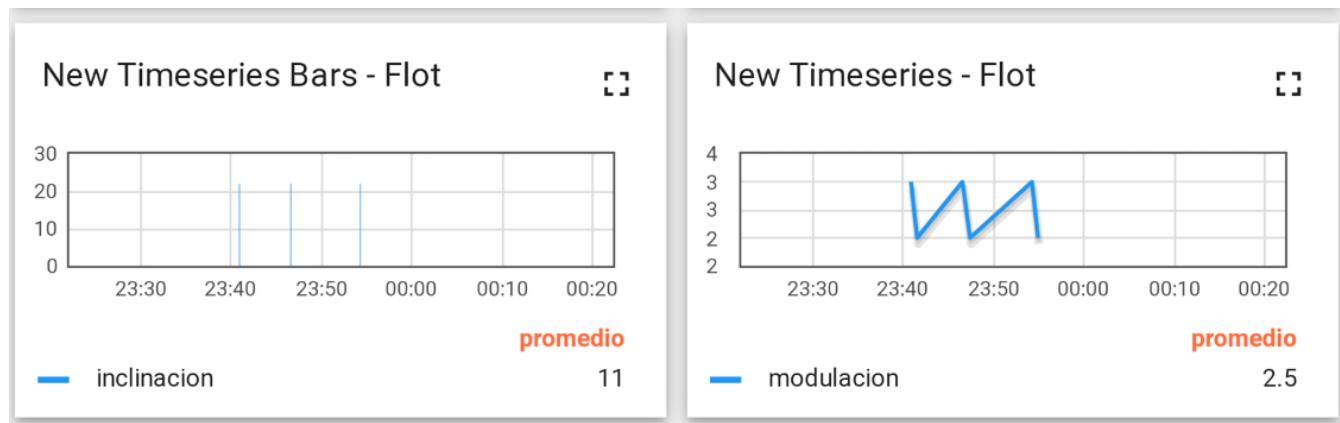


Figura 6.38 Publicación del sistema de control

CAPÍTULO 7

CONCLUSIONES Y RECOMENDACIONES

El sistema de medición de la ventana deslizable está funcionando correctamente y considero que es una muy buena estrategia implementada para llevar un registro de control de las veces al día que se abre y la magnitud de la apertura de la misma, esto ayudara para conocer el flujo de calor que se mueve dentro de un espacio de trabajo tan común como lo es el cubículo, además recomendaría que este sistema fuera adaptado a un futuro pensando en que se instalare algún aire acondicionado en el cubículo, adaptarlos de tal manera que si estuviera encendido el aire acondicionado y alguien abriera la ventana que automáticamente el sistema de aire acondicionado se desactivara para evitar que el aire fresco se escape y evitar también el gasto energía eléctrica.

Por otro lado, aunque lamentablemente por motivos de la pandemia no fue posible la instalación del sistema de control en las ventanas del auditorio, el sistema cumple satisfactoriamente con los objetivos marcados, y se espera que en un futuro cercano pueda ayudar a la mejora de ventilación de ese recinto.

CAPÍTULO 8

COMPETENCIAS DESARROLADAS

- Programación básica en MicroPython
- Manejo del ESP8266
- Manejo del ESP32
- Ahorro energético con el modo “deepsleep”
- Principios de funcionamiento de un acelerómetro triaxial
- Manejo del DAC
- Principios de funcionamiento de un sensor ultrasónico
- Obtener grados de inclinación a partir de la aceleración de la gravedad
- Configuración de OPAMP
- Calculo de ganancia de un OPAMP
- Simular en LTspice
- Publicar en ThingsBoard
- Función del protocolo NTP
- Función del protocolo I2C
- Publicar en un LCD
- Principios de funcionamiento de un sensor de vibración

CAPÍTULO 9

REFERENCIAS BIBLIOGRAFICAS

1. <https://www.tesisenred.net/bitstream/handle/10803/6104/07CAPITULO2.pdf?sequence=7&isAllowed=y>
2. <https://www.diarioelectronicohoy.com/blog/el-amplificador-operacional>
3. <https://www.luisllamas.es/esp8266-nodemcu/>
4. <https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-14-dispositivos-i-internet-de-las-cosas-iot/>
5. <https://unicrom.com/amplificador-operacional-ganancia-lazo-cerrado/>
6. <https://www.siberzone.es/blog-sistemas-ventilacion/breve-historia-de-los-sistemas-de-ventilacion/>
7. <https://passivhaus-paee.com/la-ventilacion-nocturna/>
8. <https://www.espaciohonduras.net/componentes-electronicos-clasificacion-que-son-y-para-que-sirven>
9. https://bookdown.org/alberto_brunete/introAutomatica/sensoractuador.html
10. <https://flexbooks.ck12.org/cbook/ck-12-middle-school-physical-science-flexbook-2.0/section/17.2/primary/lesson/speed-of-sound-ms-ps>

11.<http://image.dfrobot.com/image/data/SEN0168/BMA220%20datasheet.pdf>

pdf

12.https://wiki.dfrobot.com/How_to_Use_a_Three-Axis_Accelerometer_for_Tilt_Sensing?fbclid=IwAR2sh3gctfkM55NsbgZKvSzF26OOKVWo-ATz6SFbPDO6-PxK_fZ4S7NGFmk

13.https://www.sas.com/es_mx/insights/big-data/internet-of-things.html

14.https://www.ecured.cu/Circuito_integrado_LM324

ANEXOS

CODIGOS:

Sistema de medición de ventana deslizable

MAIN.PY

```
# - SISTEMA DE MEDICION DE VENTANA DESLIZABLE -  
# POR: GIOVANNI VELAZQUEZ AVILEZ  
# ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA  
# RESIDENTE EN EL IER-UNAM, AGO-DIC 2020
```

```
import machine  
from machine import Pin  
import time  
from time import sleep  
import math  
from funcion import *  
from funciones import *  
from wifi import activate_wifi  
import utime
```

```
red = 'TOTALPLAY_897E2E'  
clave = 'EZ04022ST1'
```

```
unique_id = '70099f70-10a1-11eb-9c3f-d1ead9980bc3'  
token = 'Mdihxoakm0rTUqo9GiRm'
```

```
label ='distancia'
```

```
activate_wifi(red,clave)
```

```
led = Pin (2, Pin.OUT)  
led.value(0)  
sleep(1)  
led.value(1)  
sleep(1)
```

```
while True:
```

```
    print ('DISTANCIA')  
    for count in range(2):  
        result=distancia_media()  
        print (result, 'cm')  
        valor = result  
        data = {label: valor}  
        publish_thingsboard(red,clave,token, unique_id,data)  
        sleep(1)
```

```

print('IRE A DORMIR')
sleep(3)

FUNCIONES.PY
import machine
import time
from machine import Pin
import utime

def distancia_normal():
    trig=machine.Pin(14, machine.Pin.OUT)
    trig.off()
    utime.sleep_us(2)
    trig.on()
    utime.sleep_us(10)
    trig.off()
    echo=machine.Pin(12, machine.Pin.IN)
    while echo.value() == 0:
        pass
    t1 = utime.ticks_us()
    while echo.value() == 1:
        pass
    t2 = utime.ticks_us()
    cm = ((t2 - t1)*0.03432)/2 #58.0
    utime.sleep_ms(20)
    return cm
    #print (cm , 'cm')

def distancia_media():
    # inicia la lista
    distancia_muestras = []
    # toma 10 muestras y las agrega a la lista
    for count in range(10):
        distancia_muestras.append((distancia_normal()))
        utime.sleep_ms(20)
    # abre la lista
    distancia_muestras = sorted(distancia_muestras)
    # toma el valor medio de la lista de valores
    distancia_media = distancia_muestras[int(len(distancia_muestras)/2)]

    #print(distancia_muestras) #imprime los valores de la lista (imprimir es opcional)
    return float(distancia_media)

```

Sistema de control de ventanas

```

# - SISTEMA DE CONTROL DE VENTILAS
# POR: GIOVANNI VELÁZQUEZ AVILEZ
# ESTUDIANTE DEL ITZ DE ING ELECTROMECANICA
# RESIDENTE EN EL IER-UNAM, AGO-DIC 2020

import machine
import gc
import network, utime, ntptime, time
from wifi import activate_wifi
from funcion import *
from funciones import *
import funcionesMPU
from funcionesMPU import *
from machine import I2C, Pin, DAC
from esp8266_i2c_lcd import I2cLcd
import math
gc.enable()

red = 'TOTALPLAY_897E2E'
clave = 'EZ04022ST1'
unique_id = '70099f70-10a1-11eb-9c3f-d1ead9980bc3'
token = 'MdihxoakM0rTUqo9GiRm'
activate_wifi(red, clave)

label ='modulacion'
label2='inclinacion'

lcd = I2cLcd(I2C(scl=Pin(26), sda=Pin(13)), 0x27, 4, 20)
dac1=DAC(Pin(25))
buton1 = Pin(15, Pin.IN)
buton2 = Pin(4, Pin.IN)
buton3 = Pin(17, Pin.IN)
buton4 = Pin(18, Pin.IN)

# LIMPIAR LCD
def lcd_clear():
    lcd.move_to(0, 0)
    lcd.putstr(" " * 80)

# MOSTRAR TEXTO EN LCD
def lcd_show(text, line):
    lcd.move_to(0, line)
    lcd.putstr(text)
    web_query_delay = 600000
    timezone_hour = 6 #ZONA HORARIA

    local_time_sec = utime.time() + timezone_hour * 3600
    local_time = utime.localtime(local_time_sec)
    update_time = utime.ticks_ms() - web_query_delay

```

```

lcd_show("BIENVENIDO",0)
lcd_show("SISTEMA DE CONTROL",1)
lcd_show("DE VENTILAS",2)
sleep(3)
lcd_clear()

```

while True:

```

led.value(0)
if __name__ == "__main__":
    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)
    while True:
        if (grados > 0) and (grados <= 23):
            print ('-----> NIVEL DE APERTURA <-----')
            lcd_show("ABIERTAS: 25%",0)
            lcd_show("CAMBIAR EL",1)
            lcd_show("NIVEL DE APERTURA?",2)
            lcd_show("SI NO",3)

            if boton4.value() is 0:
                horario_programado()
                lcd_clear()
                lcd_show("ELIJA EL NIVEL DE",0)
                lcd_show("APERTURA DESEADO",1)
                sleep(3)
                lcd_clear()
                lcd_show("USE EL BOTON ARRIBA",0)
                lcd_show(" Y ABAJO PARA",1)
                lcd_show("DESPLAZARSE",2)
                sleep(3)
                lcd_clear()
                result2=todo25()

                i2c = I2C(scl=Pin(12), sda=Pin(14))
                mpu6050_init(i2c)
                grados=mpu6050_get_accel_angle_zx(i2c)

                valor = result2
                valor2 = grados
                data = [{label: valor},
                        {label2: valor2}]
                publish_thingsboard(red,clave,token, unique_id,data)
                sleep(3)
                horario_programado()

```

```

        elif boton2.value() is 0:
            lcd_clear()
            sleep(2)

            horario_programado()
            break
        else:
            continue
        break

    elif (grados > 23) and (grados <= 45):
        horario_programado()
        lcd_show("ABIERTAS: 50%",0)
        lcd_show("DESEA CAMBIAR EL",1)
        lcd_show("NIVEL DE APERTURA?",2)
        lcd_show("SI NO",3)

    if boton4.value() is 0:
        horario_programado()
        lcd_clear()
        lcd_show("ELIJA EL NIVEL DE",0)
        lcd_show("APERTURA DESEADO",1)
        sleep(3)
        lcd_clear()
        lcd_show("USE EL BOTON ARRIBA",0)
        lcd_show(" Y ABAJO PARA",1)
        lcd_show(" DESPLAZARSE",2)
        sleep(3)
        lcd_clear()

        result2=todo50()

        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)

        valor = result2
        valor2 = grados
        data = [{label: valor},
        {label2: valor2}]
        publish_thingsboard(red,clave,token, unique_id,data)
        sleep(3)
        horario_programado()

    elif boton2.value() is 0:
        lcd_clear()
        sleep(2)
        horario_programado()
        break

```

```

else:
    continue
break

elif (grados > 45) and (grados <= 68):
    horario_programado()
    lcd_show("ABIERTAS: 75%",0)
    lcd_show("DESEA CAMBIAR EL",1)
    lcd_show("NIVEL DE APERTURA?",2)
    lcd_show("SI NO",3)

if boton4.value() is 0:
    horario_programado()
    lcd_clear()
    lcd_show("ELIJA EL NIVEL DE",0)
    lcd_show("APERTURA DESEADO",1)
    sleep(3)
    lcd_clear()
    lcd_show("USE EL BOTON ARRIBA",0)
    lcd_show(" Y ABAJO PARA",1)
    lcd_show(" DESPLAZARSE",2)
    sleep(3)
    lcd_clear()

result2=todo75()

i2c = I2C(scl=Pin(12), sda=Pin(14))
mpu6050_init(i2c)
grados=mpu6050_get_accel_angle_zx(i2c)

valor = result2
valor2 = grados
data = [{label: valor},
         {label2: valor2}]
publish_thingsboard(red,clave,token, unique_id,data)
sleep(3)

elif boton2.value() is 0:
    lcd_clear()
    sleep(2)
    horario_programado()
    break

else:
    continue
break

```

```

elif (grados > 88) and (grados <= 91):
    horario_programado()
    lcd_show("ABIERTAS: 100%",0)
    lcd_show("DESEA CAMBIAR EL",1)
    lcd_show("NIVEL DE APERTURA?",2)
    lcd_show("SI NO",3)

if boton4.value() is 0:
    lcd_clear()
    lcd_show("ELIJA EL NIVEL DE",0)
    lcd_show("APERTURA DESEADO",1)
    sleep(3)
    lcd_clear()
    lcd_show("USE EL BOTON ARRIBA",0)
    lcd_show(" Y ABAJO PARA",1)
    lcd_show(" DESPLAZARSE",2)
    sleep(3)
    lcd_clear()

result2=todo100()

i2c = I2C(scl=Pin(12), sda=Pin(14))
mpu6050_init(i2c)
grados=mpu6050_get_accel_angle_zx(i2c)

valor = result2
valor2 = grados
data = [{label: valor},
        {label2: valor2}]
publish_thingsboard(red,clave,token, unique_id,data)
sleep(3)

elif boton2.value() is 0:
    lcd_clear()
    sleep(2)
    horario_programado()
    break

else:
    continue
break

elif grados == 0:
    lcd_show("VENTILAS CERRADAS",0)
    lcd_show(" ABRIR?",1)
    lcd_show("SI NO",3)

```

```

if boton4.value() is 0:
    horario_programado()
    lcd_clear()
    lcd_show("ELIJA EL NIVEL DE",0)
    lcd_show("APERTURA DESEADO",1)
    sleep(3)
    lcd_clear()
    lcd_show("USE EL BOTON ARRIBA",0)
    lcd_show(" Y ABAJO PARA",1)
    lcd_show(" DESPLAZARSE",2)
    sleep(3)
    lcd_clear()

    result2=todocero()

    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)

    valor = result2
    valor2 = grados
    data = [{label: valor},
    {label2: valor2}]
    publish_thingsboard(red,clave,token, unique_id,data)
    sleep(3)

    horario_programado()

elif boton2.value() is 0:
    lcd_clear()
    sleep(2)

    horario_programado()
    break

else:
    continue
break

elif (grados < 0):
    horario_programado()
    lcd_clear()
    lcd_show(" APERTURA FUERA DE",0)
    lcd_show(" RANGO!",1)
    sleep(3)
    lcd_clear()
    lcd_show(" AJUSTE LA POSICION",1)
    lcd_show(" DEL ACELEROMETRO",2)
    sleep(5)

```

```

        lcd_clear()
        break

    elif (grados > 91):
        horario_programado()
        lcd_clear()
        lcd_show(" APERTURA FUERA DE",0)
        lcd_show(" RANGO!",1)
        sleep(3)
        lcd_clear()
        lcd_show(" AJUSTE LA POSICION",1)
        lcd_show(" DEL ACELEROMETRO",2)
        sleep(5)
        lcd_clear()
        break

```

FUNCIONES.PY

FUNCIONES.PY

```

import machine
import time
import gc
import network, utime, ntptime
from machine import Pin, DAC, I2C
import funcionesMPU
from funcionesMPU import *
from time import sleep
from wifi import activate_wifi
import esp8266_i2c_lcd
from esp8266_i2c_lcd import I2cLcd
from lcd_api import LcdApi

# LCD 2004 I2C: Vcc -> 5V, Gnd -> G, SCL -> D1, SDA -> D2
lcd = I2cLcd(I2C(scl=Pin(26), sda=Pin(13)), 0x27, 4, 20)

red = 'TOTALPLAY_897E2E'
clave = 'EZ04022ST1'

label ='modulacion'
label2='inclinacion'

web_query_delay = 600000
timezone_hour = 6 # ZONA HORARIA (hours)

alarm = [11, 30] # ALARMA[HORA, MINUTO]
alarm2 = [11, 56] # ALARMA[HORA, MINUTO]

dac1=DAC(Pin(25))

```

```

led = Pin(27, Pin.OUT)
buton1 = Pin(15, Pin.IN)
buton2 = Pin(4, Pin.IN)
buton3 = Pin(17, Pin.IN)
buton4 = Pin(18, Pin.IN)

# LIMPIAR LCD
def lcd_clear():
    lcd.move_to(0, 0)
    lcd.putstr(" " * 80)

# MOSTRAR TEXTO EN LCD
def lcd_show(text, line):
    lcd.move_to(0, line)
    lcd.putstr(text)

def nivel1():
    print ('-----> CERRANDO VENTILAS <-----')
    lcd_show("CERRANDO VENTILAS",1)
#Parpadea el LED

    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    utime.sleep(0.1)
    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    volts = 2
    dac1.write(44)
    sleep(10)
    dac1.write(0)
    sleep(1)
    return volts

def nivel2():
    print ('-----> ABRIENDO 25% <-----')
    lcd_show("ABRIENDO 25%",1)
#Parpadea el LED

    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    utime.sleep(0.1)
    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    volts = 3
    dac1.write(70)
    sleep(10)
    dac1.write(0)

```

```

sleep(1)
return volts

def nivel3():
    print ('-----> ABRIENDO 50% <-----')
    lcd_show("ABRIENDO 50%",1)
#Parpadea el LED
    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    utime.sleep(0.1)
    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    volts = 4
    dac1.write(96)
    sleep(10)
    dac1.write(0)
    sleep(1)
    return volts

def nivel4():

    print ('-----> ABRIENDO 75% <-----')
    lcd_show("ABRIENDO 75%",1)
#Parpadea el LED

    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    utime.sleep(0.1)
    led.value(1)
    utime.sleep(0.1)
    led.value(0)

    volts = 5.03
    dac1.write(123)
    sleep(10)
    dac1.write(0)
    sleep(1)
    return volts

def nivel5():
    print ('-----> ABRIENDO 100% <-----')
    lcd_show("ABRIENDO 100%",1)
#Parpadea el LED
    led.value(1)
    utime.sleep(0.1)
    led.value(0)
    utime.sleep(0.1)

```

```

led.value(1)
utime.sleep(0.1)
led.value(0)

volts = 6.01
dac1.write(150)
sleep(10)
dac1.write(0)
sleep(1)
return volts

def todo25():
    A = int
    A = 1
    while True:
        if buton3.value() is 0:
            A +=1
        if buton1.value() is 0:
            A -=1
        if A > 4:
            A = 1
        if A == 1 :
            lcd_clear()
            lcd_show("CERRAR <----- ",0)
            lcd_show("50%",1)
            lcd_show("75%",2)
            lcd_show("100%",3)
            if buton4.value() is 0:
                lcd_clear()
                result=nivel1()
                lcd_clear()
                i2c = I2C(scl=Pin(12), sda=Pin(14))
                mpu6050_init(i2c)
                grados=mpu6050_get_accel_angle_zx(i2c)
                sleep(1)
                if grados == 0:
                    lcd_show(" OPERACION EXITOSA ",1)
                    sleep(2)
                    lcd_clear()
                else:
                    lcd_clear()
                    lcd_show(" {} GRADOS ".format(grados) , 1)
                    sleep(3)
                    continue
                break
            else:
                continue
            break

        if A == 2 :

```

```

lcd_clear()
lcd_show("CERRAR",0)
lcd_show("50% <----- ",1)
lcd_show("75%",2)
lcd_show("100%",3)
#utime.sleep_ms(200)
if boton4.value() is 0:
    lcd_clear()

    result=nivel3()
    lcd_clear()
    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)
    sleep(1)
    if grados >= 44 and grados <=46:
        lcd_show(" OPERACION EXITOSA ",1)
        sleep(2)
        lcd_clear()
    else:
        continue

    break
else:
    continue
break

if A == 3 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("50%",1)
    lcd_show("75% <-----",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)

    if boton4.value() is 0:
        lcd_clear()

        result=nivel4()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 66 and grados <=68:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()

```

```

        else:
            continue

            break
        else:
            continue
        break

if A == 4 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("50%",1)
    lcd_show("75%",2)
    lcd_show("100% <-----",3)
    #utime.sleep_ms(200)

    if buton4.value() is 0:
        lcd_clear()

        result=nivel5()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 89 and grados <=91:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

            break
        else:
            continue
        break

return result

def todo50():
    A = int
    A = 1

    while True:
        if buton3.value() is 0:
            A +=1

        if buton1.value() is 0:

```

```

A -=1

if A > 4:
    A = 1

if A == 1 :

    lcd_clear()
    lcd_show("CERRAR <----- ",0)
    lcd_show("25%",1)
    lcd_show("75%",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)
    if boton4.value() is 0:
        lcd_clear()
        result=nivel1()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados == 0:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

        break
    else:
        continue
    break

if A == 2 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25% <----- ",1)
    lcd_show("75%",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)
    if boton4.value() is 0:
        lcd_clear()
        result=nivel2()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)

```

```

        sleep(1)
        if grados >= 21 and grados <=23:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

            break
        else:
            continue
        break

if A == 3 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25%",1)
    lcd_show("75% -----",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)

    if boton4.value() is 0:
        lcd_clear()
        result=nivel4()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 66 and grados <=68:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

            break
    else:
        continue
    break

if A == 4 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25%",1)
    lcd_show("75%",2)

```

```

lcd_show("100% <-----",3)
#utime.sleep_ms(200)

if buton4.value() is 0:
    lcd_clear()
    result=nivel5()
    lcd_clear()
    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)
    sleep(1)
    if grados >= 89 and grados <=91:
        lcd_show(" OPERACION EXITOSA ",1)
        sleep(2)
        lcd_clear()
    else:
        continue

    break
else:
    continue
break

return result

def todo75():
    A = int
    A = 1

    while True:
        if buton3.value() is 0:
            A +=1

        if buton1.value() is 0:
            A -=1

        if A > 4:
            A = 1

        if A == 1 :
            lcd_clear()
            lcd_show("CERRAR <----- ",0)
            lcd_show("25%",1)
            lcd_show("50%",2)
            lcd_show("100%",3)
            #utime.sleep_ms(200)
            if buton4.value() is 0:
                lcd_clear()

```

```

        result=nivel1()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados == 0:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

        break
    else:
        continue
break

```

if A == 2 :

```

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25% <----- ",1)
    lcd_show("50%",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)
    if boton4.value() is 0:
        lcd_clear()
        result=nivel2()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 21 and grados <=23:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

        break
    else:
        continue
break

```

if A == 3 :

```

lcd_clear()
lcd_show("CERRAR",0)
lcd_show("25%",1)
lcd_show("50% <-----",2)
lcd_show("100%",3)
#utime.sleep_ms(200)

if boton4.value() is 0:
    lcd_clear()
    result=nivel3()
    lcd_clear()
    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)
    sleep(1)
    if grados >= 44 and grados <=46:
        lcd_show(" OPERACION EXITOSA ",1)
        sleep(2)
        lcd_clear()
    else:
        continue

    break
else:
    continue
break

if A == 4 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25%",1)
    lcd_show("50%",2)
    lcd_show("100% <-----",3)
    #utime.sleep_ms(200)

    if boton4.value() is 0:
        lcd_clear()
        result=nivel5()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 89 and grados <=91:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:

```

```

        continue

        break
    else:
        continue
    break

return result

def todo100():
    A = int
    A = 1

    while True:
        if boton3.value() is 0:
            A +=1

        if boton1.value() is 0:
            A -=1

        if A > 4:
            A = 1

        if A == 1 :

            lcd_clear()
            lcd_show("CERRAR <----- ",0)
            lcd_show("25%",1)
            lcd_show("50%",2)
            lcd_show("75%",3)
            #utime.sleep_ms(200)
            if boton4.value() is 0:
                lcd_clear()
                result=nivel1()
                lcd_clear()
                i2c = I2C(scl=Pin(12), sda=Pin(14))
                mpu6050_init(i2c)
                grados=mpu6050_get_accel_angle_zx(i2c)
                sleep(1)
                if grados == 0:
                    lcd_show(" OPERACION EXITOSA ",1)
                    sleep(2)
                    lcd_clear()
                else:
                    continue

            break
        else:
            continue

```

```

        break

if A == 2 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25% <----- ",1)
    lcd_show("50%",2)
    lcd_show("75%",3)
    #utime.sleep_ms(200)
    if buton4.value() is 0:
        lcd_clear()
        result=nivel2()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 21 and grados <=23:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

        break
    else:
        continue
    break

if A == 3 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25%",1)
    lcd_show("50% <-----",2)
    lcd_show("75%",3)
    #utime.sleep_ms(200)

    if buton4.value() is 0:
        lcd_clear()
        result=nivel3()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 44 and grados <=46:

```

```

        lcd_show(" OPERACION EXITOSA ",1)
        sleep(2)
        lcd_clear()
    else:
        continue

        break
    else:
        continue
    break

if A == 4 :

    lcd_clear()
    lcd_show("CERRAR",0)
    lcd_show("25%",1)
    lcd_show("50%",2)
    lcd_show("75% <-----",3)
    #utime.sleep_ms(200)

    if buton4.value() is 0:
        lcd_clear()
        result=nivel4()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 66 and grados <=68:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

        break
    else:
        continue
    break

return result

def todocero():
    A = int
    A = 1

    while True:
        if buton3.value() is 0:
            A +=1

```

```

if boton1.value() is 0:
    A -=1

if A > 4:
    A = 1

if A == 1 :

    lcd_clear()
    lcd_show("25% <----- ",0)
    lcd_show("50%",1)
    lcd_show("75%",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)
    if boton4.value() is 0:
        lcd_clear()
        result=nivel2()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >=21 and grados <=23:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            lcd_clear()
            lcd_show(" {} GRADOS ".format(grados) , 1)
            sleep(3)
            continue

        break
    else:
        continue
    break

if A == 2 :

    lcd_clear()
    lcd_show("25%",0)
    lcd_show("50% <----- ",1)
    lcd_show("75%",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)
    if boton4.value() is 0:
        lcd_clear()

```

```

        result=nivel3()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 44 and grados <=46:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

            break
    else:
        continue
    break

if A == 3 :

    lcd_clear()
    lcd_show("25%",0)
    lcd_show("50%",1)
    lcd_show("75% <-----",2)
    lcd_show("100%",3)
    #utime.sleep_ms(200)

    if boton4.value() is 0:
        lcd_clear()
        result=nivel4()
        lcd_clear()
        i2c = I2C(scl=Pin(12), sda=Pin(14))
        mpu6050_init(i2c)
        grados=mpu6050_get_accel_angle_zx(i2c)
        sleep(1)
        if grados >= 66 and grados <=68:
            lcd_show(" OPERACION EXITOSA ",1)
            sleep(2)
            lcd_clear()
        else:
            continue

            break
    else:
        continue
    break

if A == 4 :

    lcd_clear()

```

```

lcd_show("25%",0)
lcd_show("50%",1)
lcd_show("75%",2)
lcd_show("100% <-----",3)
#utime.sleep_ms(200)

if boton4.value() is 0:
    lcd_clear()
    result=nivel5()
    lcd_clear()
    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)
    sleep(1)
    if grados >= 89 and grados <=91:
        lcd_show(" OPERACION EXITOSA ",1)
        sleep(2)
        lcd_clear()
    else:
        continue

    break
else:
    continue
break

return result

def horario_programado():
    local_time_sec = utime.time() + timezone_hour * 3600
    local_time = utime.localtime(local_time_sec)
    update_time = utime.ticks_ms() - web_query_delay

    ntptime.settime()
    print("HORA DEL SISTEMA ACTUALIZADO:", utime.localtime())
    update_time = utime.ticks_ms()

if alarm[0] == local_time[3] and alarm[1] == local_time[4]:
    print("!!! HORA PROGRAMADA !!!")
    lcd_clear()
    lcd_show(" 10:00 PM ",0)
    lcd_show("HORARIO PROGRAMADO",1)
    lcd_show(" DE APERTURA",2)
    sleep(5)
    lcd_clear()
    nivel5()
    lcd_clear()
    lcd_show("ESPERE UN MOMENTO...",2)
    sleep(30)

```

```

lcd_clear()
i2c = I2C(scl=Pin(12), sda=Pin(14))
mpu6050_init(i2c)
grados=mpu6050_get_accel_angle_zx(i2c)
sleep(1)
if grados >= 88 and grados <=91:
    lcd_show(" OPERACION EXITOSA ",1)
    sleep(5)
    lcd_clear()

else:
    lcd_clear()
    lcd_show(" {} GRADOS ".format(grados) , 1)
    sleep(5)
    brea

elif alarm2[0] == local_time[3] and alarm2[1] == local_time[4]:
    print("!!! HORA PROGRAMADA !!!")
    lcd_clear()
    lcd_show(" 07:00 AM ",0)
    lcd_show("HORARIO PROGRAMADO",1)
    lcd_show(" DE CIERRE",2)
    sleep(5)
    lcd_clear()
    nivel1()
    lcd_clear()
    lcd_show("ESPERE UN MOMENTO...",2)
    sleep(30)
    lcd_clear()
    i2c = I2C(scl=Pin(12), sda=Pin(14))
    mpu6050_init(i2c)
    grados=mpu6050_get_accel_angle_zx(i2c)
    sleep(1)
    if grados == 0:
        lcd_show(" OPERACION EXITOSA ",1)
        sleep(5)
        lcd_clear()

    else:
        lcd_clear()
        lcd_show(" {} GRADOS ".format(grados) , 1)
        sleep(5)

```

```

from machine import I2C, Pin
from time import sleep
import time
import math
import machine
import utime

MPU6050_ADDR = 0x68

MPU6050_ACCEL_XOUT_H = 0x3B
MPU6050_ACCEL_XOUT_L = 0x3C
MPU6050_ACCEL_YOUT_H = 0x3D
MPU6050_ACCEL_YOUT_L = 0x3E
MPU6050_ACCEL_ZOUT_H = 0x3F
MPU6050_ACCEL_ZOUT_L = 0x40
MPU6050_TEMP_OUT_H = 0x41
MPU6050_TEMP_OUT_L = 0x42
MPU6050_GYRO_XOUT_H = 0x43
MPU6050_GYRO_XOUT_L = 0x44
MPU6050_GYRO_YOUT_H = 0x45
MPU6050_GYRO_YOUT_L = 0x46
MPU6050_GYRO_ZOUT_H = 0x47
MPU6050_GYRO_ZOUT_L = 0x48
MPU6050_PWR_MGMT_1 = 0x6B

MPU6050_LSBC = 340.0
MPU6050_TEMP_OFFSET = 36.53
MPU6050 LSBG = 16384.0
MPU6050 LSBDS = 131.0

RestrictPitch = True
radToDeg = 57.2957786

def mpu6050_init(i2c):
    i2c.writeto_mem(MPU6050_ADDR, MPU6050_PWR_MGMT_1, bytes([0]))

def combine_register_values(h, l):
    if not h[0] & 0x80:
        return h[0] << 8 | l[0]
    return -((h[0] ^ 255) << 8) | (l[0] ^ 255) + 1

def mpu6050_get_accel(i2c):
    accel_x_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_H, 1)
    accel_x_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_L, 1)
    accel_y_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_H, 1)
    accel_y_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_L, 1)
    accel_z_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_H, 1)
    accel_z_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_L, 1)
    return [combine_register_values(accel_x_h, accel_x_l) / MPU6050 LSBG,
            combine_register_values(accel_y_h, accel_y_l) / MPU6050 LSBG,
            combine_register_values(accel_z_h, accel_z_l) / MPU6050 LSBG]

```

```

combine_register_values(accel_z_h, accel_z_l) / MPU6050_LSBG]

def mpu6050_get_accel_angle_zy(i2c):
    accel_x_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_H,
1)
    accel_x_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_L, 1)
    a= combine_register_values(accel_x_h, accel_x_l) / MPU6050_LSBG

    accel_y_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_H,
1)
    accel_y_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_L, 1)
    b= combine_register_values(accel_y_h, accel_y_l) / MPU6050_LSBG

    accel_z_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_H,
1)
    accel_z_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_L, 1)
    c= combine_register_values(accel_z_h, accel_z_l) / MPU6050_LSBG

    if (RestrictPitch):
        roll = math.atan2(b, c) * radToDeg
        pitch = math.atan(-a/math.sqrt((b**2)+(c**2))) * radToDeg
    else:
        roll = math.atan(b/math.sqrt((a**2)+(c**2))) * radToDeg
        pitch = math.atan2(-a,c) * radToDeg

    return int(roll)

def mpu6050_get_accel_angle_zx(i2c):
    accel_x_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_H,
1)
    accel_x_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_XOUT_L, 1)
    x= combine_register_values(accel_x_h, accel_x_l) / MPU6050_LSBG

    accel_y_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_H,
1)
    accel_y_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_YOUT_L, 1)
    y= combine_register_values(accel_y_h, accel_y_l) / MPU6050_LSBG

    accel_z_h = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_H,
1)
    accel_z_l = i2c.readfrom_mem(MPU6050_ADDR, MPU6050_ACCEL_ZOUT_L, 1)
    z= combine_register_values(accel_z_h, accel_z_l) / MPU6050_LSBG

    if (RestrictPitch):
        roll = math.atan2(x, z) * radToDeg
        pitch = math.atan(-y/math.sqrt((x**2)+(z**2))) * radToDeg
    else:

```

```

roll = math.atan(x/math.sqrt((y**2)+(z**2))) * radToDeg
pitch = math.atan2(-y,z) * radToDeg

return int(roll)

if __name__ == "__main__":
i2c = I2C(scl=Pin(12), sda=Pin(14))
mpu6050_init(i2c)
while True:

#print("Acelerometro:\t", mpu6050_get_accel(i2c), "g")
#print("Angulo zy:\t", mpu6050_get_accel_angle_zy(i2c),"°")
print("Angulo zx:\t", mpu6050_get_accel_angle_zx(i2c),"°")
time.sleep(1)

```

CODIGOS DE APOYO

FUNCION.PY

```

import math
from umqtt.simple import MQTTClient
import gc
import json
import machine, time
from wifi import do_connect

def settimeout(duration):
pass
def t3_publication(topic, msg):
print (topic, ';', msg)
pycom.rgbled(0xff00)

def publish_thingsboard(red, clave,token,UNIQUE_ID,data):

contador = 0
condicion = True
while condicion:
try:
client = MQTTClient(UNIQUE_ID, "iot.ier.unam.mx", port = 1883, user=token, password="")
client.settimeout = settimeout
client.connect()
print(json.dumps(data))
client.publish('v1/devices/me/telemetry', json.dumps(data) )
client.disconnect()
condicion = False
except Exception as inst:
do_connect(red,clave);
time.sleep(10)
contador += 1
print("Falla ",contador)
if contador >= 10:

```

```

machine.reset()

NTP.TIME.PY
try:
import usocket as socket
except:
import socket
try:
import ustruct as struct
except:
import struct

# (date(2000, 1, 1) - date(1900, 1, 1)).days * 24*60*60
NTP_DELTA = 3155673600

# The NTP host can be configured at runtime by doing: ntptime.host = 'myhost.org'
host = "pool.ntp.org"

def time():
NTP_QUERY = bytearray(48)
NTP_QUERY[0] = 0x1B
addr = socket.getaddrinfo(host, 123)[0][-1]
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
try:
s.settimeout(1)
res = s.sendto(NTP_QUERY, addr)
msg = s.recv(48)
finally:
s.close()
val = struct.unpack("!!", msg[40:44])[0]
return val - NTP_DELTA

```

```

LCD_API.PY
"""Provides an API for talking to HD44780 compatible character LCDs."""

import time

class LcdApi:
"""Implements the API for talking with HD44780 compatible character LCDs.
This class only knows what commands to send to the LCD, and not how to get
them to the LCD.

It is expected that a derived class will implement the hal_xxx functions.
"""

# The following constant names were lifted from the avrlib lcd.h

```

```

# header file, however, I changed the definitions from bit numbers
# to bit masks.
#
# HD44780 LCD controller command set

LCD_CLR = 0x01 # DB0: clear display
LCD_HOME = 0x02 # DB1: return to home position

LCD_ENTRY_MODE = 0x04 # DB2: set entry mode
LCD_ENTRY_INC = 0x02 # --DB1: increment
LCD_ENTRY_SHIFT = 0x01 # --DB0: shift

LCD_ON_CTRL = 0x08 # DB3: turn lcd/cursor on
LCD_ON_DISPLAY = 0x04 # --DB2: turn display on
LCD_ON_CURSOR = 0x02 # --DB1: turn cursor on
LCD_ON_BLINK = 0x01 # --DB0: blinking cursor

LCD_MOVE = 0x10 # DB4: move cursor/display
LCD_MOVE_DISP = 0x08 # --DB3: move display (0-> move cursor)
LCD_MOVE_RIGHT = 0x04 # --DB2: move right (0-> left)

LCD_FUNCTION = 0x20 # DB5: function set
LCD_FUNCTION_8BIT = 0x10 # --DB4: set 8BIT mode (0->4BIT mode)
LCD_FUNCTION_2LINES = 0x08 # --DB3: two lines (0->one line)
LCD_FUNCTION_10DOTS = 0x04 # --DB2: 5x10 font (0->5x7 font)
LCD_FUNCTION_RESET = 0x30 # See "Initializing by Instruction" section

LCD_CGRAM = 0x40 # DB6: set CG RAM address
LCD_DDRAM = 0x80 # DB7: set DD RAM address

LCD_RS_CMD = 0
LCD_RS_DATA = 1

LCD_RW_WRITE = 0
LCD_RW_READ = 1

def __init__(self, num_lines, num_columns):
    self.num_lines = num_lines
    if self.num_lines > 4:
        self.num_lines = 4
    self.num_columns = num_columns
    if self.num_columns > 40:
        self.num_columns = 40
    self.cursor_x = 0
    self.cursor_y = 0
    self.implied_newline = False
    self.backlight = True
    self.display_off()
    self.backlight_on()
    self.clear()

```

```

self.hal_write_command(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)
self.hide_cursor()
self.display_on()

def clear(self):
    """Clears the LCD display and moves the cursor to the top left
    corner.
    """
    self.hal_write_command(self.LCD_CLR)
    self.hal_write_command(self.LCD_HOME)
    self.cursor_x = 0
    self.cursor_y = 0

def show_cursor(self):
    """Causes the cursor to be made visible."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
        self.LCD_ON_CURSOR)

def hide_cursor(self):
    """Causes the cursor to be hidden."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def blink_cursor_on(self):
    """Turns on the cursor, and makes it blink."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
        self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

def blink_cursor_off(self):
    """Turns on the cursor, and makes it no blink (i.e. be solid)."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
        self.LCD_ON_CURSOR)

def display_on(self):
    """Turns on (i.e. unblanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def display_off(self):
    """Turns off (i.e. blanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL)

def backlight_on(self):
    """Turns the backlight on.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = True
    self.hal_backlight_on()

def backlight_off(self):

```

```

"""Turns the backlight off.

This isn't really an LCD command, but some modules have backlight
controls, so this allows the hal to pass through the command.
"""

self.backlight = False
self.hal_backlight_off()

def move_to(self, cursor_x, cursor_y):
    """Moves the cursor position to the indicated position. The cursor
    position is zero based (i.e. cursor_x == 0 indicates first column).
    """

    self.cursor_x = cursor_x
    self.cursor_y = cursor_y
    addr = cursor_x & 0x3f
    if cursor_y & 1:
        addr += 0x40 # Lines 1 & 3 add 0x40
    if cursor_y & 2: # Lines 2 & 3 add number of columns
        addr += self.num_columns
    self.hal_write_command(self.LCD_DDRAM | addr)

def putchar(self, char):
    """Writes the indicated character to the LCD at the current cursor
    position, and advances the cursor by one position.
    """

    if char == '\n':
        if self.implied_newline:
            # self.implied_newline means we advanced due to a wraparound,
            # so if we get a newline right after that we ignore it.
            pass
        else:
            self.cursor_x = self.num_columns
    else:
        self.hal_write_data(ord(char))
        self.cursor_x += 1
    if self.cursor_x >= self.num_columns:
        self.cursor_x = 0
        self.cursor_y += 1
    self.implied_newline = (char != '\n')
    if self.cursor_y >= self.num_lines:
        self.cursor_y = 0
    self.move_to(self.cursor_x, self.cursor_y)

def putstr(self, string):
    """Write the indicated string to the LCD at the current cursor
    position and advances the cursor position appropriately.
    """

    for char in string:
        self.putchar(char)

```

```

def custom_char(self, location, charmap):
    """Write a character to one of the 8 CGRAM locations, available
    as chr(0) through chr(7).
    """
    location &= 0x7
    self.hal_write_command(self.LCD_CGRAM | (location << 3))
    self.hal_sleep_us(40)
    for i in range(8):
        self.hal_write_data(charmap[i])
        self.hal_sleep_us(40)
        self.move_to(self.cursor_x, self.cursor_y)

def hal_backlight_on(self):
    """Allows the hal layer to turn the backlight on.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_backlight_off(self):
    """Allows the hal layer to turn the backlight off.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_write_command(self, cmd):
    """Write a command to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

def hal_write_data(self, data):
    """Write data to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

def hal_sleep_us(self, usecs):
    """Sleep for some time (given in microseconds)."""
    time.sleep_us(usecs)

```

ESP8266_12C_LCD.PY
 """Implements a HD44780 character LCD connected via PCF8574 on I2C.
 This was tested with: <https://www.wemos.cc/product/d1-mini.html>"""

```

from lcd_api import LcdApi
from machine import I2C
from time import sleep_ms

# The PCF8574 has a jumper selectable address: 0x20 - 0x27
DEFAULT_I2C_ADDR = 0x27

# Defines shifts or masks for the various LCD line attached to the PCF8574

MASK_RS = 0x01
MASK_RW = 0x02
MASK_E = 0x04
SHIFT_BACKLIGHT = 3
SHIFT_DATA = 4

class I2cLcd(LcdApi):
    """Implements a HD44780 character LCD connected via PCF8574 on I2C."""

    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.i2c.writeto(self.i2c_addr, bytarray([0]))
        sleep_ms(20) # Allow LCD time to powerup
        # Send reset 3 times
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(5) # need to delay at least 4.1 msec
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        sleep_ms(1)
        # Put LCD into 4 bit mode
        self.hal_write_init_nibble(self.LCD_FUNCTION)
        sleep_ms(1)
        LcdApi.__init__(self, num_lines, num_columns)
        cmd = self.LCD_FUNCTION
        if num_lines > 1:
            cmd |= self.LCD_FUNCTION_2LINES
        self.hal_write_command(cmd)

    def hal_write_init_nibble(self, nibble):
        """Writes an initialization nibble to the LCD.

This particular function is only used during initialization.
"""

        byte = ((nibble >> 4) & 0x0f) << SHIFT_DATA
        self.i2c.writeto(self.i2c_addr, bytarray([byte | MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytarray([byte]))


def hal_backlight_on(self):

```

```

"""Allows the hal layer to turn the backlight on."""
self.i2c.writeto(self.i2c_addr, bytearray([1 << SHIFT_BACKLIGHT]))

def hal_backlight_off(self):
    """Allows the hal layer to turn the backlight off."""
    self.i2c.writeto(self.i2c_addr, bytearray([0]))

def hal_write_command(self, cmd):
    """Writes a command to the LCD.

Data is latched on the falling edge of E.
"""

byte = ((self.backlight << SHIFT_BACKLIGHT) | (((cmd >> 4) & 0x0f) << SHIFT_DATA))
self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
self.i2c.writeto(self.i2c_addr, bytearray([byte]))
byte = ((self.backlight << SHIFT_BACKLIGHT) | ((cmd & 0x0f) << SHIFT_DATA))
self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
self.i2c.writeto(self.i2c_addr, bytearray([byte]))
if cmd <= 3:
    # The home and clear commands require a worst case delay of 4.1 msec
    sleep_ms(5)

def hal_write_data(self, data):
    """Write data to the LCD."""
    byte = (MASK_RS | (self.backlight << SHIFT_BACKLIGHT) | (((data >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytearray([byte]))
    byte = (MASK_RS | (self.backlight << SHIFT_BACKLIGHT) | ((data & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytearray([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytearray([byte]))

```

WIFI.PY
import network
import time

```

def do_connect(red,clave):

sta_if = network.WLAN(network.STA_IF)
if not sta_if.isconnected():
    print('connecting to network...')
    sta_if.active(True)
        sta_if.connect(red,clave)

```

```
sta_if.connect(red, clave)
while not sta_if.isconnected():
    pass
else:
    print("Already connected")
    print('network config:', sta_if.ifconfig())
    time.sleep(5)

def activate_wifi(red,clave):
    sta_if = network.WLAN(network.STA_IF)
    sta_if.active(True)
    ap_if = network.WLAN(network.AP_IF)
    sta_if.connect(red,clave)
```