**PREDICTING THE OUTPUT ENERGY OF WIND TURBINE BASED ON WEATHER CONDITION**

**General Description:** Renewable energy resources have gained an important place in human life. It is a fact that the amount of non-renewable energy resources is decreasing rapidly, at the same time, the need for renewable energy resources increases. The history of humanity teaches us to use the resources prudently and efficiently. This can happen only with well planning and devising ways to realize the same. Even in the case of efficient use of renewable energy, the data analysis has greater value, since better the analysis, better the efficient and effective use. This paper is an outcome of making the data analysis of a wind turbine by collecting and communicating the data to the cloud server. These details can be accessed locally with the use of an LCD module, and remotely with the web browser or through a mobile app.

**INTRODUCTION:** There are different sources of energy and they could be classified as renewable and non-renewable sources of energy. The limited availability of the non-renewable sources of energy has made the world to think and move towards renewable energy resources.Now in this document we concentrate on wind. The wind is a clean source of energy as it does not pollute the air and the environment. It has a significant role among other renewable energy sources. Wind energy is not a new form of energy for humans. Wind energy has its source in solar energy, and so as long as Sun will be there that long wind energy will also remain. The better utilization of wind energy comes with enhanced forecasting and for that, it is important to have data comes from the wind turbine.In this it collects the data and sends to the cloud server for the purpose of remote monitoring and better analysis. We will be using even many sensors so they will be helpful in testing the temperature or humidity analysis.

**PROPOSED METHODOLOGY:** There are five parts to this model. First is the collection of data. There are various sensors in the wind turbine. . The sensors are used for collecting various information and sending them to a controller. The sensor used to measure temperature is LM35 temperature sensor, current sensor, voltage sensor, vibration sensor and humidity sensor. By using Universal asynchronous receiver/transmitter interfacing connecting both PIC Microcontroller and Internet of things. IoT consists of the one microcontroller which is used to control the relay.

These sensors send the data in every microsecond and these values reach the data collector. The purpose of this model is to collect these values that reach the data collector and send them to the cloud server. The second part is data acquisition. It has two components. They are RS485 and Arduino Mega 2560. The data that reach the collector needs to be taken into the Arduino. If only it reaches the Arduino, they can be sent to the cloud server, and perform analysis on it. In order to do that RS485 interface is used. RS845 acts as an interface between the data controller and Arduino Mega. The third part is data storage. The components that come in this module are SD card and the cloud server thinger.io where the data will be stored. The data that arrive in the Arduino will be attached with a timestamp first. This will enable people to understand the time when the data recorded. The data which are acquired from the wind turbine are stored on the SD Card with a timestamp and at the same time, they are sent to the cloud server database thinger.io. Forth part is communication. The data need to be sent to a cloud server which is an online database that is used for this project. GSM module is used to send the data to the cloud server. This makes the remote access of the data possible. The fifth part is the LCD display. The data stored in the cloud server could be seen with the web browser as well as the mobile app. There may be situations when someone wants to see the values that flow from the wind turbine and may not have access to the cloud server. There comes the use of the LCD screen. It is used to display some specific values that flow from the turbine. These values will be updated every 2 seconds.

**TECHNOLOGY STACK:** These technologies were identified as originating primarily from the academic sector, some start-up companies and a few larger industrial entities. The following areas were considered: airborne wind energy, offshore floating concepts, smart rotors, wind-induced energy harvesting devices, blade tip-mounted rotors, unconventional power transmission systems, multi-rotor turbines, alternative support structures, modular high voltage direct current generators, innovative blade manufacturing techniques, diffuser-augmented turbines and small turbine technologies. The future role of advanced multiscale modelling and data availability is also considered.

**SOCIAL IMPACTS:** Renewable energy is a key contributor to sustainable development. The renewable energy industry is core to the implementation of SDG 7, which focuses on access to affordable, reliable, and sustainable energy and SDG 13, which centers on urgent action to combat climate change. Renewables could be key in facilitating universal access to electricity. To date, hydro and bioenergy have been the greatest contributors to clean energy consumption. In the future, wind and solar deployment will play a major role. The benefits of wind energy to our grandchildren could reach up to $386 billion. The negative impacts of wind energy on biodiversity are limited. Renewables could enhance economic growth. Wind energy could have positive impacts on the economic development in developing countries.

Scope of Work:

Hybrid Models for Wind Farm Power Forecasting :Recent research on wind power forecasting has involved the combination of the physical and statistical modeling.

Models for Wind Power Ramp Forecasting: The variability in wind power can present a substantial challenge to the grid, when the penetration of wind power is high.

Estimation of Forecast Uncertainty :Uncertainty analysis of the wind forecasts made plays a key role in grid integration and other power system operations.

Use of Wind Power Forecasting in Power System Operations: Accurate models for wind power forecasting aids in several power system operations like operating reserve requirements, unit commitment, dispatch formulations etc.

Wind Forecasting and Offshore Wind Farms: Due to limited land area in several regions, many countries are already into offshore wind farms. These offshore wind farms have opened up huge prospects of research in operation, maintenance, control of wind farms and wind resource assessment to identify potential sites etc

Link:

<https://forms.zohopublic.in/info172/form/IBMHackChallenge2020/publicrecord/gyfO5Zv4_WUtNtJvf34AZp1B75tvGZlq8j9IUJWp2dg/save>

<https://forms.zohopublic.in/info172/form/IBMHackChallenge2020/publicrecord/gyfO5Zv4_WUtNtJvf34AZp1B75tvGZlq8j9IUJWp2dg/save>

<https://forms.zohopublic.in/info172/form/IBMHackChallenge2020/publicrecord/gyfO5Zv4_WUtNtJvf34AZp1B75tvGZlq8j9IUJWp2dg/save>

)FwkoGrKVFOhLGnx)y

|  |
| --- |
| ## |
|  | # Ultrasonic library for MicroPython's pyboard. |
|  | # Compatible with HC-SR04 and SRF04. |
|  | # |
|  | # Copyright 2018 - Sergio Conde Gómez <skgsergio@gmail.com> |
|  | # Copyright 2014 - Mithru Vigneshwara |
|  | # |
|  | # This program is free software: you can redistribute it and/or modify |
|  | # it under the terms of the GNU General Public License as published by |
|  | # the Free Software Foundation, either version 3 of the License, or |
|  | # (at your option) any later version. |
|  | # |
|  | # This program is distributed in the hope that it will be useful, |
|  | # but WITHOUT ANY WARRANTY; without even the implied warranty of |
|  | # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the |
|  | # GNU General Public License for more details. |
|  | # |
|  | # You should have received a copy of the GNU General Public License |
|  | # along with this program. If not, see <http://www.gnu.org/licenses/>. |
|  | ## |
|  |  |
|  | from time import sleep\_us |
|  | from machine import Pin, time\_pulse\_us |
|  |  |
|  |  |
|  | class MeasurementTimeout(Exception): |
|  | def \_\_init\_\_(self, timeout): |
|  | super().\_\_init\_\_("Measurement timeout, exceeded {} us".format(timeout)) |
|  |  |
|  |  |
|  | class Ultrasonic(object): |
|  | def \_\_init\_\_(self, trigger\_pin, echo\_pin, timeout\_us=30000): |
|  | # WARNING: Don't use PA4-X5 or PA5-X6 as echo pin without a 1k resistor |
|  |  |
|  | # Default timeout is a bit more than the HC-SR04 max distance (400 cm): |
|  | # 400 cm \* 29 us/cm (speed of sound ~340 m/s) \* 2 (round-trip) |
|  |  |
|  | self.timeout = timeout\_us |
|  |  |
|  | # Init trigger pin (out) |
|  | self.trigger = Pin(trigger\_pin, mode=Pin.OUT, pull=None) |
|  | self.trigger.off() |
|  |  |
|  | # Init echo pin (in) |
|  | self.echo = Pin(echo\_pin, mode=Pin.IN, pull=None) |
|  |  |
|  | def distance\_in\_inches(self): |
|  | return (self.distance\_in\_cm() \* 0.3937) |
|  |  |
|  | def distance\_in\_cm(self): |
|  | # Send a 10us pulse |
|  | self.trigger.on() |
|  | sleep\_us(10) |
|  | self.trigger.off() |
|  |  |
|  | # Wait for the pulse and calc its duration |
|  | time\_pulse = time\_pulse\_us(self.echo, 1, self.timeout) |
|  |  |
|  | if time\_pulse < 0: |
|  | raise MeasurementTimeout(self.timeout) |
|  |  |
|  | # Divide the duration of the pulse by 2 (round-trip) and then divide it |
|  | # by 29 us/cm (speed of sound = ~340 m/s) |
|  | return (time\_pulse / 2) / 29 |

|  |
| --- |
| import pyb |
|  | import time |
|  | import ultrasonic |
|  |  |
|  |  |
|  | switch = pyb.Switch() |
|  | sensor = ultrasonic.Ultrasonic(pyb.Pin.board.X3, pyb.Pin.board.X4) |
|  |  |
|  |  |
|  | while not switch.value(): |
|  | try: |
|  | dist = sensor.distance\_in\_cm() |
|  | print("Dist = {}".format(dist)) |
|  |  |
|  | except ultrasonic.MeasurementTimeout as e: |
|  | print("{}".format(e)) |
|  |  |
|  | time.sleep(1) |

Code 2:

|  |
| --- |
| import time |
|  | from uos import uname |
|  | from machine import Pin |
|  |  |
|  | try: |
|  | from machine import time\_pulse\_us |
|  | except: |
|  | from pycom import pulses\_get |
|  |  |
|  | #\_\_author\_\_ = 'Roberto Sánchez' |
|  | #\_\_license\_\_= "Apache License 2.0. https://www.apache.org/licenses/LICENSE-2.0" |
|  | #\_\_author-update\_\_ = "Mauro Riva" |
|  |  |
|  | class HCSR04: |
|  | """ |
|  | Driver to use the untrasonic sensor HC-SR04. |
|  | The sensor range is between 2cm and 4m. |
|  |  |
|  | The timeouts received listening to echo pin are converted to OSError('Out of range') |
|  |  |
|  | """ |
|  | # echo\_timeout\_us is based in chip range limit (400cm) |
|  | def \_\_init\_\_(self, trigger\_pin, echo\_pin, echo\_timeout\_us=500\*2\*30): |
|  | """ |
|  | trigger\_pin: Output pin to send pulses |
|  | echo\_pin: Readonly pin to measure the distance. The pin should be protected with 1k resistor |
|  | echo\_timeout\_us: Timeout in microseconds to listen to echo pin. |
|  | By default is based in sensor limit range (4m) |
|  | """ |
|  | self.echo\_timeout\_us = echo\_timeout\_us |
|  | # Init trigger pin (out) |
|  | self.trigger = Pin(trigger\_pin, mode=Pin.OUT) |
|  | self.trigger.value(0) |
|  | # Init echo pin (in) |
|  | if (uname().sysname == 'WiPy'): |
|  | self.echo = Pin(echo\_pin, mode=Pin.OPEN\_DRAIN) |
|  | else: |
|  | self.echo = Pin(echo\_pin, mode=Pin.IN) |
|  |  |
|  | def \_send\_pulse\_and\_wait(self): |
|  | """ |
|  | Send the pulse to trigger and listen on echo pin. |
|  | We use the method `machine.time\_pulse\_us()` to get the microseconds until the echo is received. |
|  | """ |
|  | self.trigger.value(0) # Stabilize the sensor |
|  | time.sleep\_us(5) |
|  | self.trigger.value(1) |
|  | # Send a 10us pulse. |
|  | time.sleep\_us(10) |
|  | self.trigger.value(0) |
|  | try: |
|  | if (uname().sysname == 'WiPy'): |
|  | pulse\_list = pulses\_get(self.echo, self.echo\_timeout\_us) |
|  | if(len(pulse\_list) == 0): |
|  | pulse\_time = -1 |
|  | else: |
|  | pulse\_time = pulse\_list[0][1] |
|  | else: |
|  | pulse\_time = time\_pulse\_us(self.echo, 1, self.echo\_timeout\_us) |
|  |  |
|  | return pulse\_time |
|  | except OSError as ex: |
|  | if ex.args[0] == 110: # 110 = ETIMEDOUT |
|  | raise OSError('Out of range') |
|  | raise ex |
|  |  |
|  | def distance\_mm(self): |
|  | """ |
|  | Get the distance in milimeters without floating point operations. |
|  | """ |
|  | pulse\_time = self.\_send\_pulse\_and\_wait() |
|  |  |
|  | # To calculate the distance we get the pulse\_time and divide it by 2 |
|  | # (the pulse walk the distance twice) and by 29.1 becasue |
|  | # the sound speed on air (343.2 m/s), that It's equivalent to |
|  | # 0.34320 mm/us that is 1mm each 2.91us |
|  | # pulse\_time // 2 // 2.91 -> pulse\_time // 5.82 -> pulse\_time \* 100 // 582 |
|  | mm = pulse\_time \* 100 // 582 |
|  | return mm |
|  |  |
|  | def distance\_cm(self): |
|  | """ |
|  | Get the distance in centimeters with floating point operations. |
|  | It returns a float |
|  | """ |
|  | pulse\_time = self.\_send\_pulse\_and\_wait() |
|  |  |
|  | # To calculate the distance we get the pulse\_time and divide it by 2 |
|  | # (the pulse walk the distance twice) and by 29.1 becasue |
|  | # the sound speed on air (343.2 m/s), that It's equivalent to |
|  | # 0.034320 cm/us that is 1cm each 29.1us |
|  | cms = (pulse\_time / 2) / 29.1 |
|  | return cms |

Nodemcu : token : AVO!If\_dvUBWeYhBeO

Arduino: 1.8.12 (Windows Store 1.8.33.0) (Windows 10), Board: "Generic ESP8266 Module, 80 MHz, Flash, Legacy (new can return nullptr), All SSL ciphers (most compatible), dtr (aka nodemcu), 26 MHz, 40MHz, DOUT (compatible), 4MB (FS:3MB OTA:~512KB), 1, nonos-sdk 2.2.1+100 (190703), v2 Lower Memory, Disabled, None, Only Sketch, 512000"

sketch\_may26a:24:35: error: 'callback' was not declared in this scope

PubSubClient client(server, 1883, callback, wifiClient);

^

C:\Users\SHRAVANI\Documents\Arduino\sketch\_may26a\sketch\_may26a.ino: In function 'void setup()':

sketch\_may26a:31:15: error: 'mqttConnect' was not declared in this scope

mqttConnect();

^

C:\Users\SHRAVANI\Documents\Arduino\sketch\_may26a\sketch\_may26a.ino: In function 'void loop()':

sketch\_may26a:36:17: error: 'publishData' was not declared in this scope

publishData();

^

sketch\_may26a:41:17: error: 'mqttConnect' was not declared in this scope

mqttConnect();

^

C:\Users\SHRAVANI\Documents\Arduino\sketch\_may26a\sketch\_may26a.ino: At global scope:

sketch\_may26a:59:1: error: expected declaration before '}' token

}

^

Multiple libraries were found for "ArduinoJson.h"

Used: C:\Users\SHRAVANI\Documents\Arduino\libraries\ArduinoJson

Not used: C:\Users\SHRAVANI\Documents\Arduino\libraries\ArduinoJson-5.0.7

exit status 1

'callback' was not declared in this scope

This report would have more information with

"Show verbose output during compilation"

option enabled in File -> Preferences.

API key: lf1qu7JnpidQ-S9UI191eomTFh2RvMuzLMsMbtYga-kK

int ledPin = 12; // choose pin for the LED  
int inputPin = 13; // choose input pin (for Infrared sensor)

int val = 0; // variable for reading the pin status

void setup()   
{

pinMode(ledPin, OUTPUT); // declare LED as output

pinMode(inputPin, INPUT); // declare Infrared sensor as input

}

void loop()  
{

val = digitalRead(inputPin); // read input value

if (val == HIGH)

{ // check if the input is HIGH

digitalWrite(ledPin, LOW); // turn LED OFF

}

else

{

digitalWrite(ledPin, HIGH); // turn LED ON

}

}

Download the**"IR.ino"** file and open it up in the Arduino IDE.  
Then Create a new sketch and paste the code below in the Arduino IDE and hit Upload You can tinker with the code to make it more useful or just use it as it is.

const byte blinkPin = 12; // change this to fit your circuit

byte oldBlinkState = HIGH; // assume eye open (or supply init/calibrate routine)

const unsigned long debounceTime = 10; // milliseconds

unsigned long blinkStartAt; // when the blink started

unsigned long oldBlinkStartAt; // when the previous blink started

unsigned long unBlinkStartAt; // when blink ended

unsigned long tweenTime; // time since last blink

unsigned long blinkTime;

unsigned long blinkChange;

void setup ()

{

Serial.begin (9600);

pinMode (blinkPin, INPUT\_PULLUP);

} // end of setup

void loop ()

{

byte blinkState = digitalRead (blinkPin);

if (blinkState != oldBlinkState)

{

if (millis () - blinkChange &gt;= debounceTime)

{

blinkChange = millis (); // when blink state changed

oldBlinkState = blinkState; // remember for next time

if (blinkState == LOW) { // change to LOW if your sensor is LOW when blinking

blinkStartAt = blinkChange;

tweenTime = blinkChange - oldBlinkStartAt;

oldBlinkStartAt = blinkStartAt;

Serial.print (&quot;Blink at: &quot;);

Serial.print (blinkStartAt);

Serial.print (&quot; Time since last blink: &quot;);

Serial.println (tweenTime) ;

} // end if switchState is LOW

else {

unBlinkStartAt = blinkChange;

Serial.print (&quot;UnBlink at: &quot;);

Serial.print (unBlinkStartAt);

Serial.print (&quot; Blink length: &quot;);

Serial.println (unBlinkStartAt - blinkStartAt);

} // end if blinkState is HIGH

} // end if debounce time up

} // end of state change

// other code here ...

} // end of loop