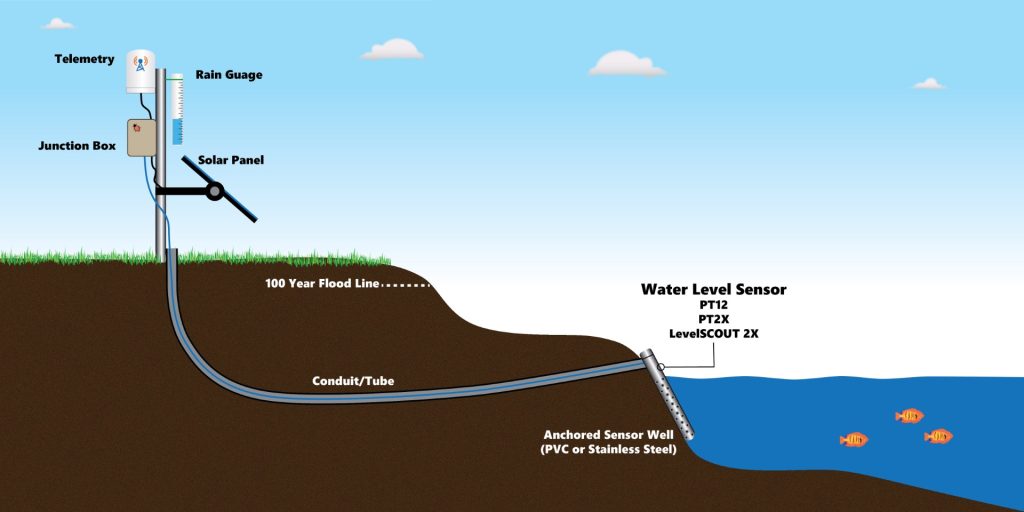
**FLOOD MONITORING AND EARLY WARNING**

****

**The establishment of flood warning systems near any major waterway or body of water provides critical information that can protect property and save lives**.

1. **Introduction :**

In most countries in the world, flood had caused damages to properties and it involved a large amount of loss to individuals and governments. During flood, it is important to have efficient flood response operation system to manage all activities among different related agencies. These last decades, lots of flooding risk technologies has been developed to minimize the danger of flood in inhabited areas. Currently, the Philippine government funded the Project NOAH of the Department of Science and Technology (DOST). They installed Automated Rain Gauges (ARG) and Water Level Monitoring Stations (WLMS) along the country’s major river basins (RBs) [1]. However, project NOAH is still under development in which some essential information are not yet available to view in their website. Most of these technologies being developed commonly apply in weather forecasting, flood detection and monitoring system using sensing devices, modeling software, Internet and mobile technology [2].systems are usually for one-way communication only. In order to get an update or latest information, local communities need to access the website. And in accessing this website, it requires

Program for flood and early warning:

**# OpenCV packages for Python**

**import cv2**

**# Python plotting package**

**import matplotlib.pyplot as plt**

**# Fork of argparse to add features and simplify its code**

**import argparse**

**# functions to make basic image processing functions**

**import imutils**

**# this for add math function**

**import math**

**import time**

**# package for array computing with Python**

**import pandas as pd**

**from numpy import asarray as pn**

**from sklearn.linear\_model import LinearRegression**

**from imutils.perspective import four\_point\_transform**

**from imutils import paths**

**from sklearn.metrics import mean\_squared\_error**

**# capture frames from a camera**

**cap = cv2.VideoCapture(0)**

**cap.set(3, 640)**

**cap.set(4, 480)**

**count = 0**

**height = []**

**flag = 0**

**# reads frames from a camera**

**ret, frame = cap.read()**

**cv2.imwrite("testimage.jpg", frame)**

**im = cv2.imread("testimage.jpg")**

**r = cv2.selectROI(img=im, windowName="test")**

**t = time.localtime()**

**current\_time = time.strftime("%H:%M:%S", t)**

**# loop runs if capturing has been initialized**

**while (1):**

**ret, frame = cap.read()**

**if frame is None:**

**break**

**# Crop image**

**frame = frame[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]**

**# Convert the img to grayscale**

**gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)**

**# Apply edge detection method on the image**

**edges = cv2.Canny(gray, 100, 120)**

**# Run Hough on edge detected image**

**lines = cv2.HoughLinesP(edges, 1, math.pi/180, 20, None, 20, 480)**

**dot1 = (lines[0][0][0], lines[0][0][1])**

**dot2 = (lines[0][0][2], lines[0][0][3])**

**slope = ((lines[0][0][3] - lines[0][0][1])/(lines[0][0][2] - lines[0][0][0]))**

**#cv2.line draws a line in img from dot1 to dot2**

**# (255,0,0) denotes the colour of the line to be drawn**

**if 0 <= slope <= 0.15:**

**cv2.line(frame, dot1, dot2, (255, 0, 0), 3)**

**length = 150 - lines[0][0][3]**

**print(length)**

**height.append(length)**

**cv2.imshow("Detected Line", frame)**

**# finds edges in the input video and**

**# marks them in the output map edges**

**edged\_frame = cv2.Canny(frame, 1, 100)**

**cv2.imshow('Edged Frame', edged\_frame)**

**if cv2.waitKey(1) & 0xFF == ord('q'):**

**break**

**x = []**

**y = []**

**file = open("Saved.txt","a")**

**for i in range(len(height)):**

**x.append(i)**

**y.append(height[i])**

**file.write(str(x[i-1])+","+str(y[i-1])+"\n")**

**X=np(x)**

**Y=np(y)**

**X = X.reshape(len(X),1)**

**Y = Y.reshape(len(Y),1)**

**model = LinearRegression()**

**model.fit(X,Y)**

**model = LinearRegression().fit(X,Y)**

**r\_sq = model.score(X,Y)**

**y\_pred = model.predict(X)**

**y\_pred = model.intercept\_+ model.coef\_\*X**

**print('Predicted Response:', y\_pred, sep='\n')**

**print('Start :', current\_time)**

**print('Coefficient of Determination:', r\_sq)**

**print('Intercept:', model.intercept\_)**

**accuracy = mean\_squared\_error(y, y\_pred)**

**print('Accuracy :', accuracy)**

**t = time.localtime()**

**current\_time2 = time.strftime("%H:%M:%S", t)**

**print('Stop :', current\_time2)**

**plt.plot(X,Y,'.',color='black')**

**cap.release()**

**cv2.destroyAllWindows()**

**plt.plot(X,y\_pred)**

**plt.title('Test Data')**

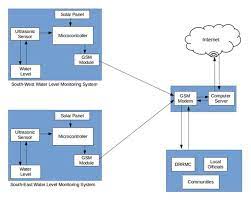
**plt.xlabel('Time')**

**plt.ylabel('Height')**

**plt.show()**

Prototype Monitoring System Testing:

The researchers tested the developed prototype through a temporary basin to test the level of water. The inputs have several sub-parameters to obtain accurate data. In the designed prototype, water level is measured in inches. The input has four options to consider.

****

IOT-based Flood Monitoring:

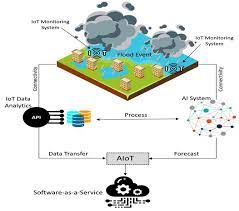
Techniques There are different models from some of the existing research that is based on different flood predicting methods which highlight the importance of implementing different approaches in tackling floods. These models use WSNs to build energy efficient monitoring and early alert systems. These models can support in designing of an efficient system to predict and prevent damages caused by floods [6].

Monitoring of Air Quality using Smart Sensors:

A smart sensors network for monitoring indoor and outdoor air quality was designed by Postolache et al. in 2009 [7]. They installed nodes of some of the sensors inside rooms which consisted of sensors such as tin dioxide connected to the central unit through hardwires or wirelessly [8]. For the accuracy of the result, the concentration of gas in the temperature and humidity is measured. In order to compensate for the influence of the above measurements, they applied MISO neural network (NN) which is based on multiple inputs single output. IEEE 802.11 (Wi-Fi) technology was used for communication between sensors.

Monitoring Environment using Controller Area Network:

Controller Area Network (CAN) based environmental monitoring system was proposed by Rao et al. in 2012 [9]. The CAN and ZigBee technology was utilized for effective communication among the sensors [10]. The sensors are connected to the microcontroller, ATMEL-89S52 through an interface of CAN to further share this data to the server using ZigBee Communication. This is used since the CAN protocol provides a higher data rate. For any specific area, the benefit of this system is that it uses a precise and dependable method for data broadcast. Communication is inexpensive and there is no loss in terms of data. 2.3. Flood Forecasting A flood forecasting model that uses Wireless Sensor Networks was created by Seal et al. [11]. This design used simple and fast calculations using multiple variable robust linear regression methods for flood forecasting. Its implementation is very cost-effective and also simple and easy to understand. It used very low-cost hardware resources. It has all the features desired by any real-world algorithm such as real-time predictions and reliable

****

**program**

#!/usr/bin/env python3

# -- coding: utf-8 --

import h5py

from osgeo import gdal

import numpy as np

import os, glob

import richdem as rd

import pyproj

from floodpy.utils.Reproject import reproject\_image\_to\_master

def nparray\_to\_tiff(nparray, reference\_gdal\_dataset, target\_gdal\_dataset):

'''

Functionality that saves information numpy array to geotiff given a reference

geotiff.

Args:

nparray (np.array): information we want to save to geotiff.

reference\_gdal\_dataset (string): path of the reference geotiff file.

target\_gdal\_dataset (string): path of the output geotiff file.

Returns:

None.

'''

# open the reference gdal layer and get its relevant properties

raster\_ds = gdal.Open(reference\_gdal\_dataset, gdal.GA\_ReadOnly)

xSize = raster\_ds.RasterXSize

ySize = raster\_ds.RasterYSize

geotransform = raster\_ds.GetGeoTransform()

projection = raster\_ds.GetProjection()

# create the target layer 1 (band)

driver = gdal.GetDriverByName('GTIFF')

target\_ds = driver.Create(target\_gdal\_dataset, xSize, ySize, bands = 1, eType = gdal.GDT\_Float32)

target\_ds.SetGeoTransform(geotransform)

target\_ds.SetProjection(projection)

target\_ds.GetRasterBand(1).WriteArray(nparray)

target\_ds = None

def reproject(outname, infilename, UTM\_CRS\_EPSG ):

'''

Reproject funcionality

'''

ds = gdal.Warp(outname, infilename, dstSRS='EPSG:{}'.format(UTM\_CRS\_EPSG),

srcNodata = -32768, dstNodata = -32768)

#outputType=gdal.GDT\_Int16, xRes=0.00892857142857143, yRes=0.00892857142857143)

ds = None

return 0

def generate\_slope\_aspect(dem\_file, slope\_outname, aspect\_outname):

'''

Calculates aspect and slope of given DEM.

Args:

dem\_file (string): path to DEM geotiff file .

slope\_outname (string): path to DEM-slope generated geotiff file.

aspect\_outname (string): path to DEM-aspect generated geotiff file.

'''

dem\_temp = rd.LoadGDAL(dem\_file, no\_data=-32768)

slope = rd.TerrainAttribute(dem\_temp, attrib='slope\_degrees')

rd.SaveGDAL(slope\_outname, slope)

aspect = rd.TerrainAttribute(dem\_temp, attrib='aspect')

rd.SaveGDAL(aspect\_outname, aspect)

def WGS84\_to\_UTM(lon\_list, lat\_list):

'''

Finds the best WGS84 UTM projection given a list of lats/lons

Args:

lon\_list (list): list of longitudes.

lat\_list (list): list of latitudes.

Returns:

utm\_crs\_epsg (string): the UTM code projection.

'''

representative\_longitude = round(np.mean(lon\_list), 10)

utm\_zone = int(np.floor((representative\_longitude + 180) / 6) + 1)

representative\_latitude = round(np.mean(lat\_list), 10)

if representative\_latitude>0:

hemisphere='north'

else:

hemisphere='south'

utm\_crs\_str = '+proj=utm +zone={} +{} +ellps=WGS84 +datum=WGS84 +units=m +no\_defs'.format(utm\_zone,hemisphere)

utm\_crs\_epsg = pyproj.CRS(utm\_crs\_str).to\_epsg()

return utm\_crs\_epsg

def get\_S1\_aux (Preprocessed\_dir):

'''

Funcionality that calculates and reprojects to UTM the auxiliary data

(DEM-slope & aspect) that are required for latest steps.

'''

SAR\_stack\_file=os.path.join(Preprocessed\_dir,'Stack/SAR\_Stack.h5')

SAR\_stack=h5py.File(SAR\_stack\_file,'r')

tiff\_files=glob.glob(os.path.join(Preprocessed\_dir,'\*.tif'))

master\_tiff\_wgs84=None

for tiff\_file in tiff\_files:

if os.path.exists(tiff\_file.split('.')[0]+'.h5'):

master\_tiff\_wgs84 = tiff\_file

# asserts that we found the master tiff file!

assert (master\_tiff\_wgs84)

# Calculate UTM projection

lon\_nparray=SAR\_stack['longitude'][:].flatten()

lon\_list = list(lon\_nparray[np.nonzero(lon\_nparray)])

lat\_nparray=SAR\_stack['latitude'][:].flatten()

lat\_list = list(lat\_nparray[np.nonzero(lat\_nparray)])

UTM\_CRS\_EPSG = WGS84\_to\_UTM(lon\_list, lat\_list)

del lon\_nparray, lon\_list, lat\_nparray, lat\_list

#############################

# B. Calculate slopes from DEM and threshold (<12 degrees) to get low slopes [slope\_mask]

#############################

# reproject master\_tiff (sigma\_VV, sigma\_VH, elevation, lat, lon, localIncidenceAngle) from wgs84 to utm

master\_tiff\_utm=master\_tiff\_wgs84[:-4]+'\_utm.tif'

reproject(master\_tiff\_utm, master\_tiff\_wgs84, UTM\_CRS\_EPSG)

# write dem\_utm

dem\_nparray=gdal.Open(master\_tiff\_utm).ReadAsArray()[1,:,:] # order of writing

dem\_nparray[dem\_nparray==-32768]=np.nan # nan values

dem\_utm\_dataset=os.path.join(os.path.dirname(master\_tiff\_utm), 'dem\_utm.tif')

nparray\_to\_tiff(dem\_nparray, master\_tiff\_utm, dem\_utm\_dataset)

del dem\_nparray

# calculate aspect and slope at UTM projection

slope\_outname=os.path.join(os.path.dirname(master\_tiff\_utm),'dem\_slope\_utm.tif')

aspect\_outname=os.path.join(os.path.dirname(master\_tiff\_utm),'dem\_aspect\_utm.tif')

generate\_slope\_aspect(dem\_utm\_dataset, slope\_outname, aspect\_outname)

# reproject dem,slope,aspect UTM to WGS84

reproject\_image\_to\_master(master\_tiff\_wgs84, dem\_utm\_dataset, dem\_utm\_dataset[:-7]+'wgs84.tif')

reproject\_image\_to\_master(master\_tiff\_wgs84, slope\_outname, slope\_outname[:-7]+'wgs84.tif')

utname, aspect\_outname[:-7]+'wgs84.tif')

**What are the advantages of flood monitoring and warning system?**

Advantages. Timely detection of possible flood risks and floods. Highly reliable and available real-time data. Tailored solution that can be integrated with external developments at any level (device, connectivity, cloud or user application).

What are the disadvantages of flood warning system?

Disadvantages. Some people may not be able to access the warnings. Flash floods may happen too quickly for a warning to be effective. They do not stop land from flooding they just warn people that a flood is likely