FLOOD MONITORING AND EARLY WARNING

# PHASE -3

# DEVELOPING PART 1

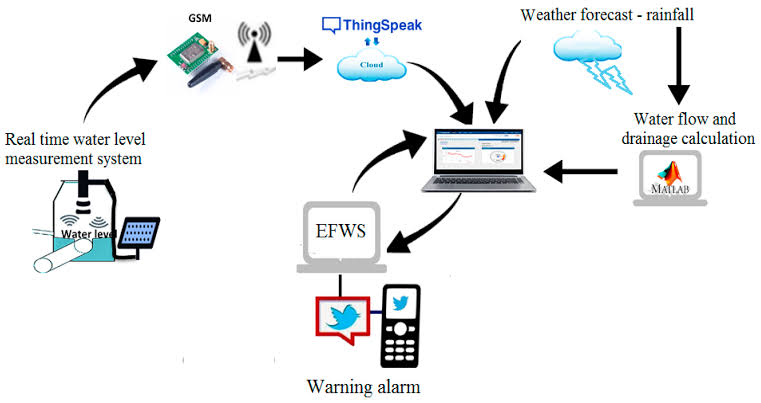
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

## **ABSTACT:**

This paper discusses different Internet of Things (IoT) based techniques and applications implemented for efficient flood monitoring and an early warning system and it is observed that in future, the combination of IoT and Synthetic Aperture Radar (SAR) data may be helpful to develop robust and secure flood monitoring and early warning system that provides effective and efficient mapping during natural disasters. The emerging technology in the discipline of computing is IoT, an embedded system that enables devices to gather real-time data to further store it in the computational devices using Wireless Sensor Networks (WSN) for further processing. The IoT based projects that can help collect data from sensors are an added advantage for researchers to explore in providing better services to people. These systems can be integrated with cloud computing and analyzing platforms. Researchers recently have focused on mathematical modeling based flood prediction schemes rather than physical parametric based flood prediction. The new methodologies explore the algorithmic approaches. There have been many systems proposed based on analog technology to web-based and now using mobile applications. Further, alert systems have been designed using web-based applications that gather processed data by Arduino Uno Microcontroller which is received from ultrasonic and rain sensors. Additionally, the machine learning based embedded systems can measure different atmospheric conditions such as temperature, moisture, and rains to forecast floods by analyzing varying trends in climatic changes.

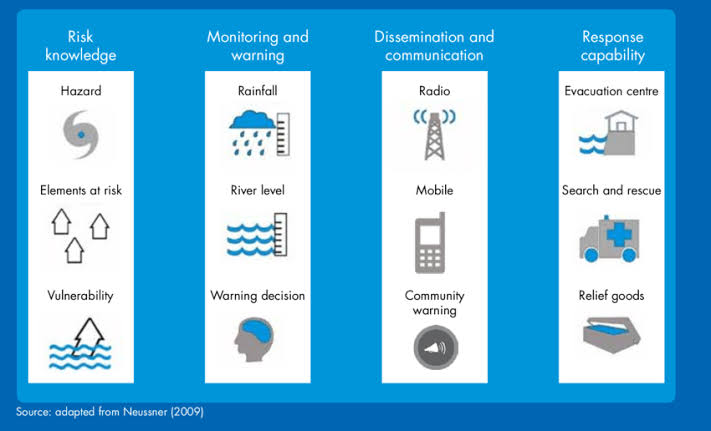
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# WORKING PRINCIPLE DIAGRAM:



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**KEY ELEMENT:**



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Creating a flood monitoring and early warning Python program is a complex task that involves various components, including data acquisition, analysis, and notification systems. Here's a simplified outline of the steps you can follow to get started:**

**Data Acquisition:**

Obtain real-time or historical data related to rainfall, water levels, weather forecasts, and river flow rates from reliable sources or APIs.

**Data Processing:**

Process and clean the acquired data to ensure accuracy and consistency.

**Threshold Setting:**

Define critical thresholds for rainfall and river levels that could trigger flood warnings.

**Early Warning Logic:**

Implement the logic to evaluate the current data against the predefined thresholds.If thresholds are exceeded, trigger a warning.

**Notification System:**

Integrate a notification system to alert relevant authorities and the public. You can use email, SMS, or other communication channels.

**Visualization:**

Create visualizations such as charts and maps to display real-time data and warnings.

**Database:**

Store historical data for analysis and future reference.

**User Interface (Optional):**

. Develop a user interface for users to view data and warnings.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Here's a simple example of Python code for flood monitoring and early warning:**

import random

def acquire\_data():

# Simulate data acquisition, replace with actual data retrieval code

return {

"rainfall": random.uniform(0, 50),

"river\_level": random.uniform(0, 10)

}

def check\_for\_flood(data, thresholds):

if data["rainfall"] > thresholds["rainfall"] or data["river\_level"] > thresholds["river\_level"]:

return True

return False

def send\_notification():

# Implement your notification logic here

print("Flood Warning: Take necessary precautions!")

if \_\_name\_\_ == "\_\_main\_\_":

thresholds = {

"rainfall": 30, # Example rainfall threshold in mm

"river\_level": 7 # Example river level threshold in meters

}

while True:

data = acquire\_data()

if check\_for\_flood(data, thresholds):

send\_notification()

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Remember that a real flood monitoring system should be much more sophisticated, including reliable data sources, extensive error handling, and a more comprehensive notification system. Additionally, ensure you have the necessary permissions and consider legal and ethical considerations when implementing such a system.

**Writing the code in Python IDE.**

# 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()

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**THE END**