Facultad Regional Rosario
Universidad Tecnológica Nacional

Cátedra: Soporte

4º Año Ingeniería en Sistemas de Información
Año: 2022

Trabajo Práctico Integrador - Sistema de detección de columnas de humo

Grupo Nº: 2
Integrantes:

- 46840 D'Ursi Ivo ivodursi211@gmail.com
- 42187 Mendiburu Francisco franmendi.fm@gmail.com
- 47639 Peresin Tomás tomasperesin@gmail.com

Narrativa

Dado los numerosos incendios que ocurren en el día a día en los humedales, se decidió por desarrollar un sistema de detección automática capaz de detectar las columnas de humo. Este sistema tiene como objetivo realizar alertas con la suficiente rapidez para permitir una respuesta en tiempo y forma del equipo de Defensa Civil encargado de tratar con el fuego. Este sistema es capaz de funcionar tanto durante el día como durante la noche, utilizando modelos específicamente construidos para cada escenario. Para facilitar la detección, el sistema trabajará con cámaras instaladas alrededor de los humedales, en puntos claves que permitan un gran ángulo de visión de estos (como, por ejemplo, en el puente Victoria). Una vez detectado el humo en alguna de las cámaras y habiendo realizado la señal de alarma, el sistema almacena la detección en una base de datos para futura referencia y análisis.
En el caso de la detección durante el día, se utiliza un modelo de machine learning entrenado específicamente para la detección de humo. Este modelo analiza la imagen obtenida de las cámaras y, con cierto rango de confianza, es capaz de delimitar las distintas columnas de humo detectadas en esta. La desventaja de este modelo es que requiere un mínimo nivel de procesamiento de GPU, por lo que la máquina en la cual se ejecutará el programa deberá de contar con las especificaciones adecuadas.
Por otro lado, para la detección durante la noche, se optó por un modelo más ligero a nivel de procesamiento. En este caso, el modelo simplemente filtra los colores de la imagen para detectar directamente el fuego de los incendios ya que es casi imposible detectar humo a menos que se utilicen cámaras especializadas las cuales encarecerían demasiado el costo del proyecto.

Requerimientos

Funcionales

- El sistema debe ser capaz de distinguir columnas de humo en un terreno natural.
- El sistema debe almacenar cada una de las detecciones realizadas.
- El sistema debe ser capaz de detectar el humo tanto de día como de noche.
- El sistema debe delimitar en una imagen la detección realizada.
- El sistema debe tener mecanismos de alerta al realizar una detección.
- El sistema debe tener mecanismos para detener las alarmas una vez el incendio haya sido notificado a las autoridades.

No funcionales

- El sistema debe funcionar en PCs con Windows 10 como su sistema operativo.
- El sistema debe ser desarrollado utilizando Python y OpenCV.
- El sistema debe tener una latencia de respuesta con respecto a la imagen obtenida menor a 5 segundos.
- El modelo de reconocimiento de humo debe ser desarrollado utilizando TensorFlow.

Stack tecnológico

Para la realización de este proyecto, se utilizó el lenguaje de programación Python, más específicamente su versión 3.8.10. Entre los módulos más importantes se encuentran OpenCV, la librería de visión computacional más completa disponible para Python y TensorFlow, una librería especializada en machine learning y la creación de modelos de inteligencia artificial en base a este. La lista completa de librerías se presenta a continuación:

absl-py==1.2.0
apache-beam==2.41.0
astunparse==1.6.3
avro-python3==1.10.2
cachetools==5.2.0
certifi==2022.9.24
charset-normalizer==2.1.1
cloudpickle==2.2.0
colorama==0.4.5
contextlib2==21.6.0
contourpy==1.0.5
crcmod==1.7
cycler==0.11.0
Cython==0.29.32
dill==0.3.1.1
dm-tree==0.1.7
docopt==0.6.2
etils==0.8.0
fastavro==1.6.1
flatbuffers==1.12
fonttools==4.37.4
gast==0.4.0
gin-config==0.5.0
google-api-core==2.8.2
google-api-python-client==2.63.0
google-auth==2.12.0
google-auth-httplib2==0.1.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
googleapis-common-protos==1.56.4
grpcio==1.49.1
h5py==3.7.0
hdfs==2.7.0
httplib2==0.20.4
idna==3.4
importlib-resources==5.9.0
joblib==1.2.0
kaggle==1.5.12
keras==2.9.0
Keras-Preprocessing==1.1.2
kiwisolver==1.4.4
libclang==14.0.6
lvis==0.5.3
lxml==4.9.1
Markdown==3.4.1
MarkupSafe==2.1.1
matplotlib==3.6.0
numpy==1.22.4
oauth2client==4.1.3
oauthlib==3.2.1
object-detection @ file:///C:/Users/ivodu/Desktop/humo/models/research
opencv-python==4.6.0.66
opencv-python-headless==4.6.0.66
opt-einsum==3.3.0
orjson==3.8.0
packaging==21.3
pandas==1.5.0
Pillow==9.2.0
portalocker==2.5.1
promise==2.3
proto-plus==1.22.1
protobuf==3.19.6
psutil==5.9.2
py-cpuinfo==8.0.0
pyarrow==7.0.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycocotools==2.0.5
pydot==1.4.2
pymongo==3.12.3
pyparsing==2.4.7
PyQt5==5.15.7
PyQt5-Qt5==5.15.2
PyQt5-sip==12.11.0
python-dateutil==2.8.2
python-slugify==6.1.2
pytz==2022.4
pywin32==304
PyYAML==5.4.1
regex==2022.9.13
requests==2.28.1
requests-oauthlib==1.3.1
rsa==4.9
sacrebleu==2.2.0
scikit-learn==1.1.2
scipy==1.9.1
sentencepiece==0.1.97
seqeval==1.2.2
six==1.16.0
tabulate==0.8.10
tensorboard==2.9.1
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.9.2
tensorflow-addons==0.18.0
tensorflow-datasets==4.6.0
tensorflow-estimator==2.9.0
tensorflow-hub==0.12.0

tensorflow-io==0.27.0
tensorflow-io-gcs-filesystem==0.27.0
tensorflow-metadata==1.10.0
tensorflow-model-optimization==0.7.3
tensorflow-text==2.9.0
termcolor==2.0.1
text-unidecode==1.3
tf-models-official==2.9.2
tf-slim==1.1.0
threadpoolctl==3.1.0
toml==0.10.2
tqdm==4.64.1
typeguard==2.13.3
typing_extensions==4.3.0
uritemplate==4.1.1
urllib3==1.26.12
Werkzeug==2.2.2
wrapt==1.14.1
zipp==3.8.

Como motor de base de datos, se optó por utilizar SQLite3. Esto por su facilidad de uso debido a que el proyecto no depende demasiado en operaciones con la base de datos.

Reglas de negocio

- Si es detectado una columna de humo o fuego este debe ser informado sin importar la duración del mismo
- Si es detectada una columna de humo o fuego debe ser identificada visualmente.
- Las alertas deben ser realizadas en un intervalo de sesenta segundos.
- De cada detección se conoce su fecha y hora y una imagen de lo detectado.
- La detección puede ocurrir tanto por columnas de humo como por el mismo fuego.
- Las alarmas deben estar presentes tanto visual como auditivamente.
- El aviso de detección al operador debe realizarse por medio del mismo sistema y por correo electrónico
- No se almacenan imágenes en donde no se detecta nada.
- Debe haber equipos capaces de utilizar el sistema disponibles como backup en caso de avería del equipo principal.
- Se deben realizar backups de la base de datos de manera periódica.
- Se deben realizar análisis de las imágenes almacenadas en la base de datos de manera periódica para así parametrizar la eficacia del modelo.
- Se deben realizar reentrenamientos periódicos del modelo de detección con nuevas imágenes para así mejorar su eficacia.
- El horario de detección nocturna es de 18:00 - 07:00 en invierno y de 20:00 - 5:00 en verano.

Caso de uso

Código y Nombre del CASO DE USO: CURS01 - Deteccion de incendio y alarma

Dimensiones de clasificación:

Nivel	**Estructura**	**Alcance**	**Caja**	**Instanciación**	**Interacción**
Resumen	Sin estructurar	Sistema	Negra	Real	Semántico

Meta del CASO DE USO: Detectar e informar de los incendios ocurridos en los humedales

Actor Primario: Operador de Defensa Civil

Otros: Defensa Civil, Municipalidad de Rosario, Bomberos

PRECONDICIONES (de sistema): El sistema se encuentra cargado en el equipo, las cámaras se encuentran encendidas.

DISPARADOR: Operador inicia el equipo con el sistema de detecciones

CAMINO BÁSICO:

1. El Operador inicia el equipo con el sistema de detecciones. El Sistema informa a las cámaras que ya se encuentra activo.
2. Las cámaras captan distintas imágenes de los humedales y se las envían al sistema.
3. El sistema recibe las imágenes y analiza si en la imagen se encuentran una o múltiples columnas de humo o focos de fuego.
4. Una vez detectada la columna de fuego o foco, el sistema carga a la base de datos la detección con su fecha e imagen realizada.
5. El Sistema prepara los mensajes de alarma e informa al Operador.
6. El Operador informa a Defensa Civil y a los Bomberos.
7. El Operador ingresa en el sistema que ya se informó del incendio. El sistema detiene las alarmas.

Los pasos 2-7 se repiten hasta que ya no se necesiten los servicios del sistema

8. El operador apaga el sistema. El Sistema informa a las cámaras.

CAMINOS ALTERNATIVOS:

2.a <Posterior> Una o más de las imágenes captadas por las cámaras se encuentran en mal estado:

2.a.1 El sistema muestra las imágenes al Operador**.**

2.a.2 El operador detecta la anomalía e informa al área de TI para que revisen las cámaras.

2.a.3 El operador apaga el sistema.

FIN CU

3.a <Posterior> El sistema no detecta columnas de humo o fuego en la imagen:

3.a.1 Sistema muestra la imagen sin detecciones al Operador.

Vuelve al paso 2

5.a <Posterior> Se detecta un falso positivo:

5.a.1 El operador desactiva las alarmas del sistema

5.a.2 El operador registra la detección errónea en una base de datos separada para futuro análisis.

Vuelve al paso 2

***.a <Durante> Son las 18:00/20:00 hs.**

***.a.1** El sistema informa que es el horario de cambio de modelo.

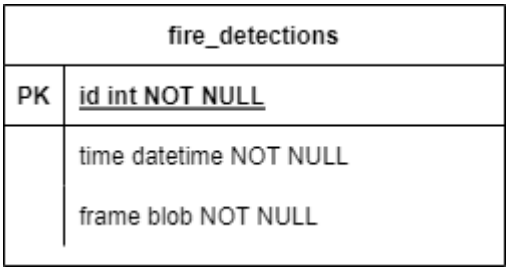
***.a.2** El sistema detiene el modelo de detección diurno

***.a.3** El sistema inicia el modelo de detección nocturna.

***.b <Durante> Son las 07:00/05:00 hs.**

***.b.1** El sistema informa que es el horario de cambio de modelo.
***.b.2** El sistema detiene el modelo de detección nocturna
***.b.3** El sistema inicia el modelo de detección diurno.

Modelo de dominio

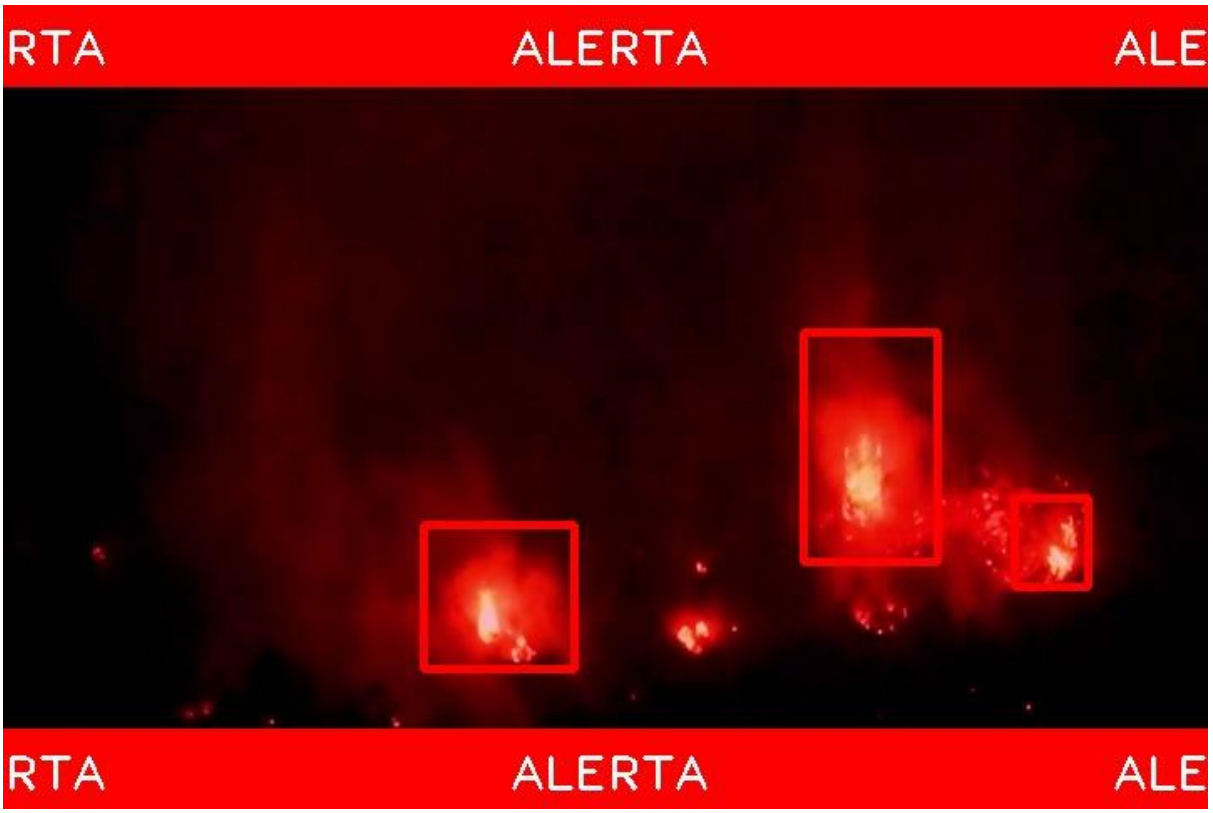


Debido a los objetivos del proyecto, la única entidad almacenada son las detecciones realizadas por el sistema. Estas cuentan con un id único para diferenciar entre distintas detecciones, la fecha y hora en la que fue realizada la detección y una imagen de la detección realizada.

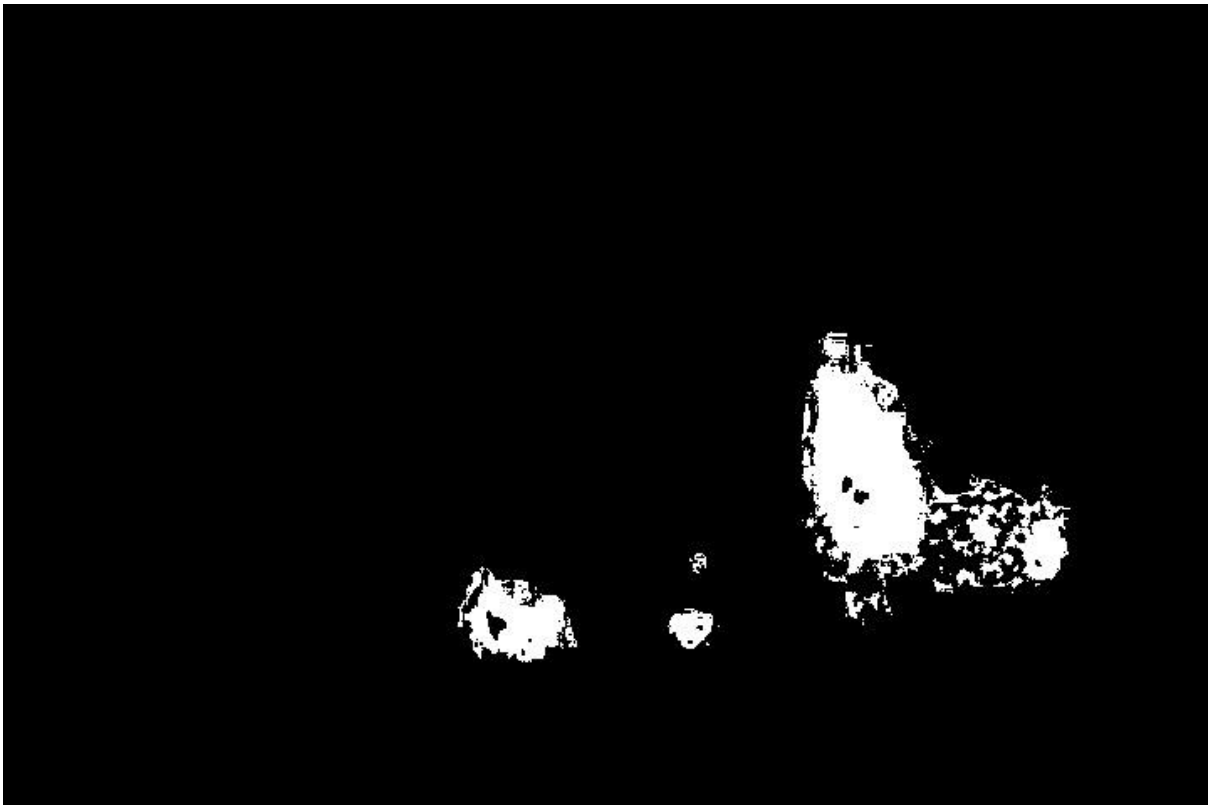
POSTCONDICIONES (de sistema):

Éxito: Se analizó las imágenes obtenidas por cámara y se informó de sus resultados.
Fracaso: Las imágenes no pudieron ser analizadas.
Éxito alternativo: Se analizó las imágenes obtenidas, sin haber realizado detección de humo o fuego.
Se analizaron las imágenes obtenidas luego de intercambiar los modelos de detección.

Imágenes



Detección durante la noche, en donde el sistema detecta y remarca el fuego detectado de las imágenes provistas por la cámara. Se puede apreciar cómo el sistema remarca con rectángulos los focos del fuego y además genera las alertas en la imagen.



En esta imagen se puede apreciar el filtro que realiza el sistema para detectar el fuego durante la noche, los pixeles en blanco son aquellos en donde el sistema encontró los pixeles con el rango de color adecuado.

Detección durante el día, en este caso el sistema usa el modelo de machine learning para detectar las columnas de humo, remarcandolas en la imagen y denotando la confianza en la detección.

Bibliografía

Documentación de Librerías

https://docs.opencv.org/4.x/
https://www.tensorflow.org/api_docs

Modelo de reconocimiento de humo

https://aiformankind.org/lets-stop-wildfires-hackathon-2.0/
https://github.com/abg3/Smoke-Detection-using-Tensorflow-2.2

Instalacion de modulos

https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html

Código fuente

https://github.com/IvoDur/deteccion-humo-rosario

database.py

```
class Database:
    def __init__(self, db):
        self.conn = sqlite3.connect(db)
        self.cur = self.conn.cursor()
        self.cur.execute("""
                CREATE TABLE IF NOT EXISTS  fire_detections (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        time datetime DEFAULT CURRENT_TIMESTAMP,
                        frame blob
                        )""")
        self.conn.commit()

    def can_save(self):
        last = self.get_last()
        if last:
            # convirto str a datetime
            last_time = datetime.datetime.strptime(last[0][1], '%Y-%m-%d %H:%M:%S')
            print( f"Pasaron {(datetime.datetime.now() - last_time).total_seconds()}")
            if (datetime.datetime.now() - last_time).total_seconds() < 60:
                return False
            else:
                print("Se almacenó en la base de datos")
                return True
```

```
        else:
            return True

    def insert(self, frame, time=None):
        frame = sqlite3.Binary(frame)
        if self.can_save():
            if time:
                self.cur.execute("INSERT INTO fire_detections (time,frame) VALUES  (?, ?)",
(time, frame))
            else:
                tiempo = datetime.datetime.now()
                tiempo = f"{tiempo.year}-{tiempo.month}-{tiempo.day}
{tiempo.hour}:{tiempo.minute}:{tiempo.second}"
                print(tiempo)
                self.cur.execute("INSERT INTO fire_detections (time,frame) VALUES  (?, ?)",
(tiempo, frame))
                # self.cur.execute("INSERT INTO fire_detections (frame) VALUES  (?)", (frame,))
            self.conn.commit()

    def view(self):
        self.cur.execute("SELECT * FROM fire_detections")
        rows = self.cur.fetchall()
        return rows

    def get_last(self):
        self.cur.execute("SELECT * FROM fire_detections ORDER BY id DESC LIMIT 1")
        rows = self.cur.fetchall()
        return rows


    def __del__(self):
        self.conn.close()


db = Database("fire_detections.db")
for algo in db.view():
    # if algo[0] == 1:
    print(f"{algo[0]}: T:{algo[1]}")
        # print(algo[2])
        # print(algo.id)
    # print(algo)
    # pass
```

deteccion_noche.py

```
import cv2
import numpy as np
```

```
from database import Database

cap = cv2.VideoCapture("video_noche.mp4")
w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
print(w, h)

ROJO = (0,0,255)
BLANCO = (255,255,255)

FUENTE = cv2.FONT_HERSHEY_PLAIN
ESCALA_FUENTE = 2
GROSOR_FUENTE = 2 #Pixeles
SEPARACION_TEXTO = w//2
SEPARACION_EJEY_TEXTO = ((h//10) // 5) * 4

desplazamiento = 0
VELOCIDAD_DESPLAZAMIENTO = 1

humo_detectado = False
HOLGURA_FRAMES = 5
frames_sin_deteccion = HOLGURA_FRAMES


valor_minimo = np.array([0,125,125])
valor_maximo = np.array([25,255,255])


def alerta(frame,desplazamiento):
    cv2.rectangle(frame, (0,0), (w,h//10), ROJO, -1)
    cv2.rectangle(frame, (0,h), (w,h-h//10), ROJO, -1)

    cv2.putText(frame, "ALERTA", (desplazamiento-w, h-11), FUENTE, ESCALA_FUENTE,
BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento-w+SEPARACION_TEXTO, h-11),
FUENTE, ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento, h-11), FUENTE, ESCALA_FUENTE,
BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento+SEPARACION_TEXTO, h-11), FUENTE,
ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)

    cv2.putText(frame, "ALERTA", (desplazamiento-w, SEPARACION_EJEY_TEXTO),
FUENTE, ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento-w+SEPARACION_TEXTO,
SEPARACION_EJEY_TEXTO), FUENTE, ESCALA_FUENTE, BLANCO,
GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento, SEPARACION_EJEY_TEXTO),
FUENTE, ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)
```

```
    cv2.putText(frame, "ALERTA", (desplazamiento+SEPARACION_TEXTO,
SEPARACION_EJEY_TEXTO), FUENTE, ESCALA_FUENTE, BLANCO,
GROSOR_FUENTE)

while True:
  database = Database("fire_detections.db")
  ref, frame = cap.read()
  frame_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

  valor_minimo = np.array([0,125,125])
  valor_maximo = np.array([25,255,255])

  mask = cv2.inRange(frame_hsv, valor_minimo, valor_maximo)
  res = cv2.bitwise_and(frame, frame, mask=mask)
  contornos, jerarquia = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

  if len(contornos) != 0:
    humo_detectado = True
    for contorno in contornos:
      if cv2.contourArea(contorno) > 500:
        x,y,width,height = cv2.boundingRect(contorno)
        cv2.rectangle(frame, (x,y), (x + width, y+height), (0,0,255), 3)
    database.insert(frame)

  if humo_detectado:
    alerta(frame, desplazamiento)
    frames_sin_deteccion = 0
  elif frames_sin_deteccion < HOLGURA_FRAMES:
    alerta(frame, desplazamiento)
    frames_sin_deteccion += 1

  cv2.imshow("Prueba", mask)
  cv2.imshow("frame", frame)



  tecla = cv2.waitKey(1)
  if tecla & 0xFF == ord("q"):
    break
  elif tecla & 0xFF == ord("h"):
       humo_detectado = not humo_detectado

  desplazamiento += VELOCIDAD_DESPLAZAMIENTO
  if desplazamiento > w:
     desplazamiento = 0

cap.release()
```

```
cv2.destroyAllWindows()
```

ejemplo_video.py

```
import cv2

import numpy as np

import tensorflow as tf

from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

import os
from database import Database

cap = cv2.VideoCapture("humo.mp4")
w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
print(w, h)

ROJO = (0,0,255)
BLANCO = (255,255,255)

FUENTE = cv2.FONT_HERSHEY_PLAIN
ESCALA_FUENTE = 2
GROSOR_FUENTE = 2 #Pixeles
SEPARACION_TEXTO = w//2
SEPARACION_EJEY_TEXTO = ((h//10) // 5) * 4

desplazamiento = 0
VELOCIDAD_DESPLAZAMIENTO = 1

humo_detectado = False
HOLGURA_FRAMES = 5
frames_sin_deteccion = HOLGURA_FRAMES


def alerta(frame,desplazamiento):
    cv2.rectangle(frame, (0,0), (w,h//10), ROJO, -1)
    cv2.rectangle(frame, (0,h), (w,h-h//10), ROJO, -1)

    cv2.putText(frame, "ALERTA", (desplazamiento-w, h-11), FUENTE, ESCALA_FUENTE,
BLANCO, GROSOR_FUENTE)
```

```
    cv2.putText(frame, "ALERTA", (desplazamiento-w+SEPARACION_TEXTO, h-11),
FUENTE, ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento, h-11), FUENTE, ESCALA_FUENTE,
BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento+SEPARACION_TEXTO, h-11), FUENTE,
ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)

    cv2.putText(frame, "ALERTA", (desplazamiento-w, SEPARACION_EJEY_TEXTO),
FUENTE, ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento-w+SEPARACION_TEXTO,
SEPARACION_EJEY_TEXTO), FUENTE, ESCALA_FUENTE, BLANCO,
GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento, SEPARACION_EJEY_TEXTO),
FUENTE, ESCALA_FUENTE, BLANCO, GROSOR_FUENTE)
    cv2.putText(frame, "ALERTA", (desplazamiento+SEPARACION_TEXTO,
SEPARACION_EJEY_TEXTO), FUENTE, ESCALA_FUENTE, BLANCO,
GROSOR_FUENTE)


#recover our saved model

#generally you want to put the last ckpt from training in here
pipeline_config = 'model\\pipeline.config'
model_dir = 'model\\checkpoint\\ckpt-0'
configs = config_util.get_configs_from_pipeline_file(pipeline_config)
model_config = configs['model']
detection_model = model_builder.build(
      model_config=model_config, is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(
      model=detection_model)
ckpt.restore(os.path.join('model\\checkpoint\\ckpt-0'))

def saveInDataBase():
    """
    we save the data in a database
      time: time of the detection
      frame: frame of the detection
      coordinates: coordinates of the detection

    """
   # print("Guardando en base de datos")
    # for dato in datos:
    #query="insert into fire_detections values
(%d,%d,%d,%d,%d,'%s')"%(dato[0],dato[1],dato[2],dato[3],dato[4],dato[5])
    # print(query)
    #cursor.execute(query)
```

```
    #cnxn.commit()
    # print("se agregaron ",len(datos),"registros")
    pass

def get_model_detection_function(model):
  """Get a tf.function for detection."""

  @tf.function
  def detect_fn(image):
    """Detect objects in image."""

    image, shapes = model.preprocess(image)
    prediction_dict = model.predict(image, shapes)
    detections = model.postprocess(prediction_dict, shapes)

    return detections, prediction_dict, tf.reshape(shapes, [-1])

  return detect_fn

detect_fn = get_model_detection_function(detection_model)

#map labels for inference decoding
label_map_path = configs['eval_input_config'].label_map_path
#label_map = label_map_util.load_labelmap(label_map_path)
label_map = label_map_util.load_labelmap("content\\train\\Smoke_label_map.pbtxt")

categories = label_map_util.convert_label_map_to_categories(
    label_map,
    max_num_classes=label_map_util.get_max_label_map_index(label_map),
    use_display_name=True)
category_index = label_map_util.create_category_index(categories)
label_map_dict = label_map_util.get_label_map_dict(label_map, use_display_name=True)



while True:
  database = Database("fire_detections.db")
  ref, frame = cap.read()

  frame_correcto = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

  input_tensor = tf.convert_to_tensor(
      np.expand_dims(frame_correcto, 0), dtype=tf.float32)
  detections, predictions_dict, shapes = detect_fn(input_tensor)


  label_id_offset = 1
  image_np_with_detections = frame_correcto.copy()
```

```
  viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections,
        detections['detection_boxes'][0].numpy(),
        (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
        detections['detection_scores'][0].numpy(),
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=200,
        min_score_thresh=.5,
        agnostic_mode=False,
  )


  frame_mostrar = cv2.cvtColor(image_np_with_detections, cv2.COLOR_RGB2BGR)

  if any(valor > .5 for valor in detections['detection_scores'][0].numpy()):
    alerta(frame_mostrar, desplazamiento)
    database.insert(frame)
    frames_sin_deteccion = 0
  elif frames_sin_deteccion < HOLGURA_FRAMES:
    alerta(frame_mostrar, desplazamiento)
    frames_sin_deteccion += 1

  cv2.imshow("Camara", frame_mostrar)


  tecla = cv2.waitKey(1)
  if tecla & 0xFF == ord("q"):
    break
  elif tecla & 0xFF == ord("h"):
        humo_detectado = not humo_detectado

  desplazamiento += VELOCIDAD_DESPLAZAMIENTO
  if desplazamiento > w:
     desplazamiento = 0

cap.release()
cv2.destroyAllWindows()
```

main.py

```
from time import sleep
import datetime
from subprocess import Popen
from sys import executable
```

```
print(datetime.datetime.now().hour)

noche_ejecutado = False
dia_ejecutado = False
extProc = None
#executable almacena la ubicacion del interprete de python

while True:

    if datetime.datetime.now().hour >= 20 and not noche_ejecutado:
        try:
            Popen.terminate(extProc)
        except:
            pass
        dia_ejecutado = False
        extProc = Popen([executable, "deteccion_noche.py"])
        noche_ejecutado = True
    elif datetime.datetime.now().hour < 20 and not dia_ejecutado:
        try:
            Popen.terminate(extProc)
        except:
            pass
        noche_ejecutado = False
        extProc = Popen([executable, "ejemplo_video.py"])
        dia_ejecutado = True

    sleep(300)
    print("Se intento cambiar el programa")
```