Contents

List of figures

1 Introduction :

Technology has brought an unprecedented explosion in unstructured data. Sources like mobile devices, websites, social media, scientific apparatus, satellites, IoT devices, and surveillance cameras are generating a vast number of images and videos every second.

Managing and efficiently analyzing this data is a challenge. Consider a city's network of video-surveillance cameras. It is impractical and inefficient to monitor every camera's video stream to discover any objects or events of interest. Instead, computer vision (CV) libraries process these video streams and provide intelligent video analytics and object detection.

Many tasks drive the use of big-data technologies in video stream analytics: parallel and on-demand processing of large-scale video streams, extracting a different set of information from a video frame, analyzing the data with different machine learning libraries, piping the analyzed data to different components of application for further processing, and outputting the processed data in different formats.

To process large-scale video stream data reliably and efficiently, a scalable, fault-07cing and loosely coupled distributed system is needed

The sample applications in this mini project use open-source technologies to build such systems, including OpenCV, Kafka, and Spark. You can also use Amazon S3 or HDFS for storage.

The system consists of three main components: video Stream Collector, Stream Data Buffer, and Video Stream Processor.

The video stream collector needs to work with a cluster of webcams (IP cameras) that provide real-time streaming data for video content and convert the video stream to frames using the OpenCV video processing library, passing the data to Kafka Broker in JSON format for use by streaming data buffer components.

The video streaming component is built on Apache Spark and uses OpenCV for video streaming data processing.

In this mini project we will try to Develop a motion detection application that can:

- Record a video only when motion is being detected ,saving you a ton in megabytes.
- Be able to watch the video online.
- Be able to adjust your parameters according to what you want to detect: small or only big events.
- Background SubtractorMog

A proof-of-concept application using apache Kafka using image processing.

- Use Flask to be able to watch our video online Realtime.

2 Problem in the context of Big data :

2.1 The core points.

In the field of unstructured data, technology has brought unprecedented and explosive changes. Data sources such as mobile devices, Web sites, social media, scientific instruments, satellites, IoT devices, and surveillance cameras produce large amounts of pictures and videos every second.

Managing and effectively analyzing this data is a big challenge, and we can consider a city's video surveillance network. Trying to monitor the video stream for each camera to find objects and events of interest is unrealistic and inefficient. Instead, the Computer Vision (CV) library can handle these video streams and provide intelligent video analysis and object detection results.

However, traditional CV systems have some limitations. In traditional video analytics systems, servers with CV libraries collect and process data at the same time, so server failures can result in the loss of video streaming data. Detecting node failures and transferring processing processes to other nodes can result in fragmented data.

There are many practical tasks that push big data-related technologies into video streaming analytics: parallel and on-demand processing of large-scale video streams,

extracting different sets of information from video frames, analyzing data using different machine learning libraries, streaming the data from the analysis to different components of the app for subsequent processing, outputting the processed data in different formats, and so on.

2.2 Video Streaming Analysis - Motion detection

To process large-scale video streaming data reliably and efficiently, a scalable, fault-07cing, loosely coupled distributed system is required. We will discuss these principles in the video streaming analysis discussed in this article.

Video streaming analysis includes the following types:

- Object tracking
- Motion detection
- Facial recognition (face recognition)
- Gesture recognition (gesture recognition)
- Augmented reality
- Image segmentation

The use scenario for the sample app in this article will be motion detection in the video stream.

Motion sensing refers to the process of finding an object (usually a human) relative to the position of its surroundings. Most of it is used to continuously monitor video surveillance systems in specific areas. The algorithm provided by the CV library analyses the real-time video provided by the camera and looks for the action that is occurring. If the action is sensed, an event is triggered that sends a message to the app or prompts the user.

In this article, the application for video streaming analysis consists of three main components:

- Video Stream Collector
- Stream data buffer
- Video Stream processor

The video stream collector accepts video streaming data from the network camera cluster. This component serializes video frames into streaming data buffers, a fault-0uffable data queue for video streaming data. The video streaming processor consumes and processes the streaming data in the buffer. This component will use a video processing algorithm to detect actions in video streaming data.

3 Data and Benchmarks

We have addressed the moving object detection in video sequences captured by a fixed camera in the presence or absence of undesired changes in the scene. To do that, the principal idea is to create a stable background modelling and then to apply a background subtraction technique, namely, to subtract current frame from background to detect moving objects. So, our mainly data are coming from the captured video of the camera. The video that we used as a data will be with the project attachment files.

4 Algorithm Description

4.1 What is Background Subtraction:

Background subtraction is a technique that is commonly used to identify moving objects in a video stream. A real-world use case would be video surveillance or in a factory to detect moving objects on a conveyor belt using a stationary video camera.

The idea behind background subtraction is that once you have a model of the background, you can detect objects by examining the difference between the current video frame and the background frame.

The idea behind this algorithm is that we first take a snapshot of the background. We then identify changes by taking the absolute difference between the current video frame and that original snapshot of the background.

This algorithm runs fast, but it is sensitive to noise, like shadows and even the smallest changes in lighting.

4.2 Conventional Approaches:

A robust background subtraction algorithm should be able to handle lighting changes, repetitive motions from clutter and long-term scene changes. The following analyses make use of the function of V(x,y,t) as a video sequence where t is the time dimension, x and y are the pixel location variables. e.g., V(1,2,3) is the pixel intensity at (1,2) pixel location of the image at t = 3 in the video sequence.

4.2.1. Using Frame Differencing :

A motion detection algorithm begins with the segmentation part where foreground or moving objects are segmented from the background. The simplest way to implement this is to take an image as background and take the frames obtained at the time t, denoted by I(t) to compare with the background image denoted by B. Here using simple arithmetic calculations, we can segment out the objects simply by using image subtraction technique of computer vision meaning for each pixel in I(t), take the pixel value denoted by P[I(t)] and subtract it with the corresponding pixels at the same position on the background image denoted as P[B].

In mathematical equation, it is written as:

(1) $$|P|F(t)] - P[f(t+1)]| > Thresshol$$

This means that the difference image's pixels' intensities are thresholder or filtered based on value of Threshold. [3] The accuracy of this approach is dependent on speed of movement in the scene. Faster movements may require higher thresholds.

(2) $$B(x,y,t) = \frac{1}{N}\sum_{i=1}^{\mathbb{N}} V(x,y,t-i)$$

4.2.2. Mean filter :

For calculating the image containing only the background, a series of preceding images are averaged. For calculating the background image at the instant t,

(3) $$|V(x_1 y,t) - B(x,y,t)| > Th$$

Where Th is threshold similarly, we can also use median instead in the above calculation of B(x,y,z) Usage of global and time-independent thresholds(same Th value for all pixels in the image) may limit the accuracy of the above two approaches.

4.2.3.Running Gaussian average :

For this method, fitting a Gaussian probabilistic density function (pdf) on the most recent n frames. To avoid fitting the pdf from scratch at each new frame time t, a running (or on-line cumulative) average is computed.

The pdf of every pixel is characterized by mean μ_t and variance σ_t^2. The following is a possible initial condition (assuming that initially every pixel is background):

(4) $\boldsymbol{\mu_0 = I_0}$

(5) $\boldsymbol{\sigma_0^2 = some\ default\ value}$

Where I_t is the value of the pixel's intensity at time t. To initialize variance, we can, for example, use the variance in x and y from a small window around each pixel. Note that background may change over time (e.g., due to illumination changes or non-static background objects). To accommodate for that change, at every frame t, every pixel's mean and variance must be updated, as follows:

(6) $\boldsymbol{\mu_t = pI_t \vdash (1-p)\mu_t - 1}$

(7) $\boldsymbol{\sigma_C = d^2p + (1-p)\sigma_{t-1}^2}$

(8) $\boldsymbol{d = |(I_t - \mu_{t-1})|}$

Where p determines the size of the temporal window that is used to fit the pdf (usually p =0.01) and d is the Euclidean distance between the mean and the value of the pixel. Gaussian distribution for each pixel. We can now classify a pixel as background if its current intensity lies within some confidence interval of its distribution's mean:

(9) $\boldsymbol{\frac{(I_t-\mu_t)}{\sigma_t} > k \rightarrow foreground}$

(10) $\boldsymbol{\frac{|(I_t-\mu_t)|}{\sigma_t} \leq k \rightarrow background}$

where the parameter k is a free threshold (usually k=2.5). A larger value for k allows for more dynamic background, while a smaller k increases the probability of a transition from background to foreground due to more subtle changes.

In a variant of the method, a pixel's distribution is only updated if it is classified as background. This is to prevent newly introduced foreground objects from fading into the background. The update formula for the mean is changed accordingly:

$$\mu_t = M\mu_{t-1} + (1 - M)(I_t p + (1 - p)\mu t - 1) \quad (11)$$

where M=1 when I_t is considered foreground and M=0 otherwise. So when M=1, that is, when the pixel is detected as foreground, the mean will stay the same. As a result, a pixel, once it has become foreground, can only become background again when the intensity value gets close to what it was before turning foreground. This method, however, has several issues: It only works if all pixels are initially background pixels (or foreground pixels are annotated as such). Also, it cannot cope with gradual background changes: If a pixel is categorized as foreground for a too long period of time, the background intensity in that location might have changed (because illumination has changed etc.). As a result, once the foreground object is gone, the new background intensity might not be recognized as such anymore.

4.2.4. Background mixture models

Mixture of Gaussians method approaches by modelling each pixel as a mixture of Gaussians and uses an on-line approximation to update the model. In this technique, it is assumed that every pixel's intensity values in the video can be modelled using a Gaussian mixture model. A simple heuristic determines which intensities are most probably of the background. Then the pixels which do not match to these are called the foreground pixels. Foreground pixels are grouped using 2D connected component analysis.

4.2.5. OpenCV :

4.2.5.1. Background subtraction

Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of the scene or, more in general, everything that can be considered as background given the characteristics of the observed scene.

OpenCV provides a class called BackgroundSubtractor, which is a handy way to operate foreground and background segmentation. BackgroundSubtractor is a fully fledged class with a plethora of methods that not only perform background subtraction, but also improve background detection in time through machine learning and lets you save the classifier to a file.

Background modelling consists of two main steps:

1. Background Initialisation.
2. Background Update.

In the first step, an initial model of the background is computed, while in the second step that model is updated to adapt to possible changes in the scene.

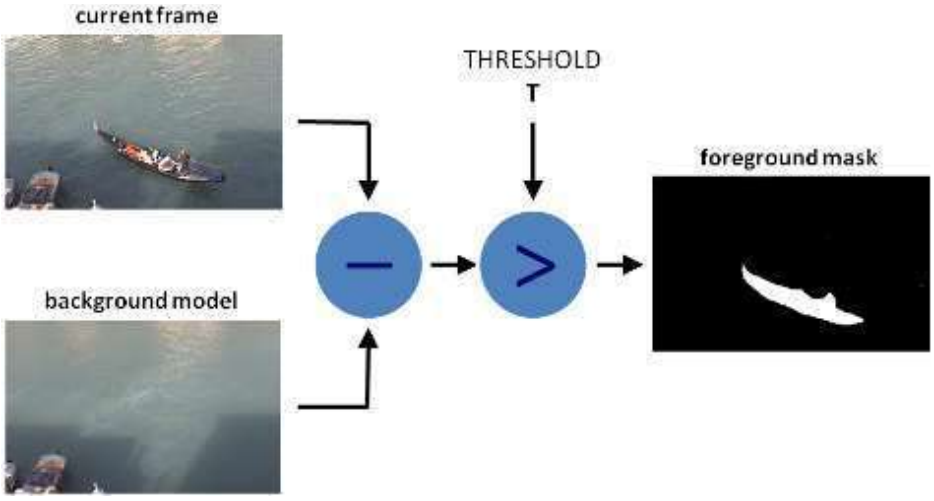

Figure 4-1 : Background modelling

4.2.5.2. Reference Image Subtraction:

is modality consisting of having a reference image where there is no moving object. Moving elements are obtained from this image by subtracting each frame from the reference image. Typically, the first frame of a video sequence is taken.

It is very sensitive to changes in light. Imagine that you take the reference image in a room with natural light. At 10:00 in the morning there will be light conditions. But at 18:00 in the evening there will be others. It is also very sensitive to camera movements. A very small movement can cause false positives to be detected at the scene. On the contrary, this method works very well in environments with controlled lighting and perfectly detects the silhouette of moving objects.

4.2.5.3. Subtraction with previous frames :

In this mode, the background is obtained from the previous frames. The technique consists of taking a reference image, letting some time pass applying a delay and begin to compare with the frames that we are obtaining. This delay will depend on factors such as the speed of the objects.

One of the biggest disadvantages is that if the moving object or person stays still, it is not detected. It is not able to detect silhouettes. However, it is a robust method to changes in lighting and camera movements and it gets stabilized after a while.

4.2.5.4. Phases of the motion detection process:

Now we will see what are the phases that we must follow to create an algorithm that allows us to detect movement with OpenCV. The process will perform various tasks.

- Grayscale conversion and noise removal
- Subtraction operation between the background and the foreground.
- Apply a threshold to the image resulting from the subtraction.
- Detection of contours or blobs

Something very common in computer science, particularly in computer vision, are parameters. Each parameter can have a range of values. The correct value will depend on many factors. It is in our hands to adapt each value to a specific situation.

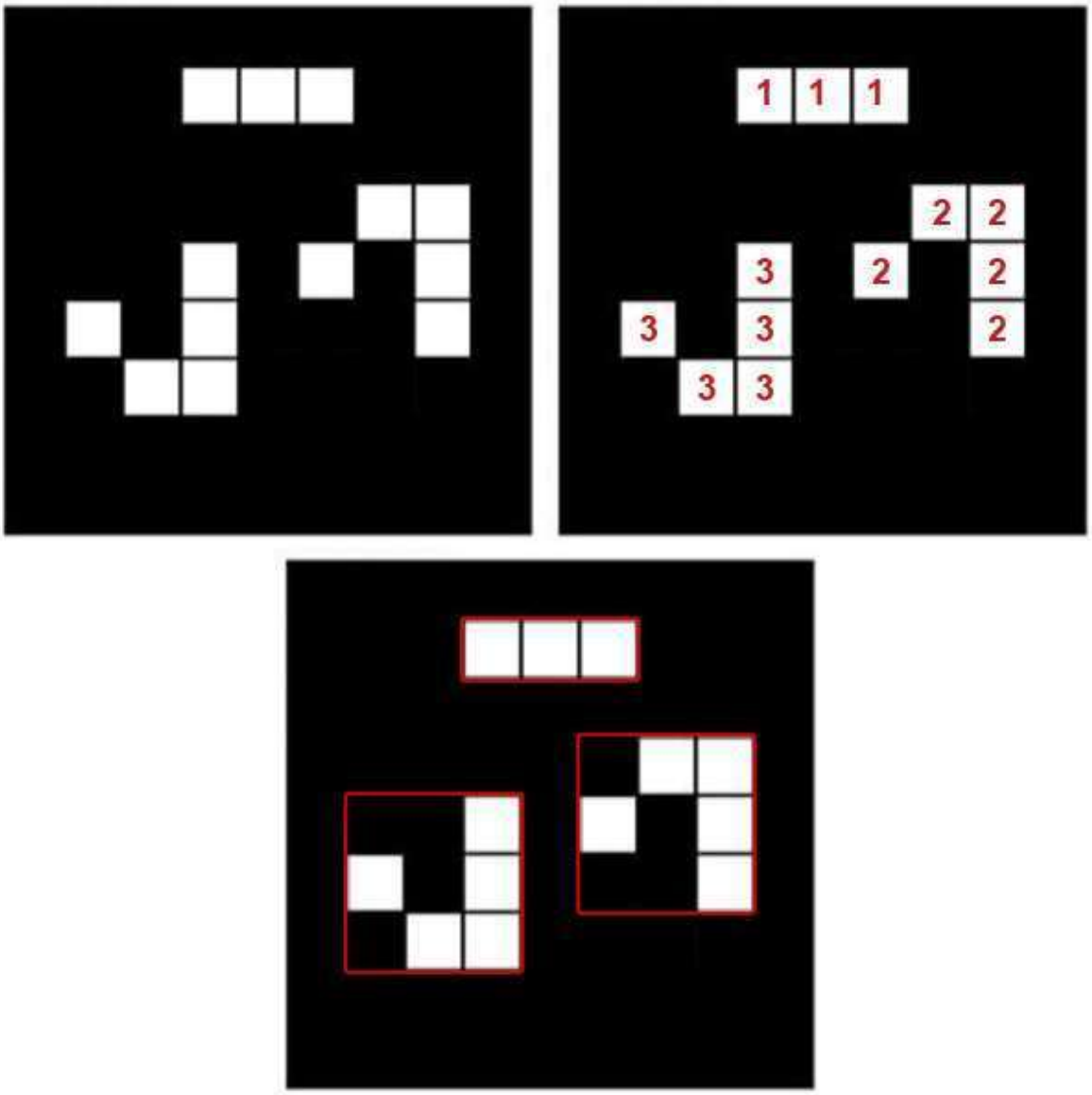

Figure 4-2 : Phases of the motion detection process

There are different techniques that allow us to estimate which is the set of values that gives us the best results. One of them is Simulated Annealing. I am not going to talk to you about this technique since it is very complex and is not the object of this article, but you should take it into account when your goal is to make a professional application. It is the only way to adjust the parameters to obtain acceptable results.

4.2.5.5. Grayscale conversion and denoising:

Before performing any operation on images, it is a good idea to convert to grayscale. It is less complex and more optimal to work with these types of images. On the other hand, the noise caused by the camera itself and by the lighting must be minimized. This is done through averaging each pixel with its neighbors. It is commonly known as smoothing.

OpenCV provides the methods that allow us to convert to grayscale and smooth the image.

```
#Convert the image to grayscale
gray = cv2.cvtColor (frame, cv2.COLOR_BGR2GRAY)
 #Image smoothing
gray = cv2.GaussianBlur (gray, (21, 21), 0)
```

4.2.5.6. Threshold application:

In this part of the process what we do is keep those pixels that exceed a threshold. The goal is to binarize the image, that is, to have two possible values. All those that exceed the threshold will be white pixels and those that do not exceed it will be black pixels. This will help us to select the moving object. In OpenCV there is a method to apply a threshold.

4.2.5.7. Outline or blob detection:

Once we have the image with black or white pixels, we have to detect the outlines or blobs. A blob is a set of pixels that are connected to each other, that is, it has neighbors with the same value. When we talk about neighbors, it is that they are next door.

5 Software installation and configuration:

5.1 Installation of OpenCV 4 on Ubuntu :

5.1.1.Install OpenCV dependencies :

in this context we will install handful of image and video I/O libraries, libraries enable us to load image from disk as well as read video files and install GTK for our GUI backend.

```
sudo apt-get install build-essential cmake unzip pkg-config

sudo apt-get install libjpeg-dev libpng-dev libtiff-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
libv4l-dev

sudo apt-get install libxvidcore-dev libx264-dev

sudo apt-get install libgtk-3-dev

sudo apt-get install libatlas-base-dev gfortran
```

5.1.2.Download OpenCV 4 :

```
wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.0.zip

wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/ar-
chive/4.0.0.zip
```

```
unzip opencv.zip

unzip opencv_contrib.zip

mv opencv-4.0.0 opencv

mv opencv_contrib-4.0.0 opencv_contrib
```

- to Rename the directories:

```
mv opencv-4.0.0 opencv

mv opencv_contrib-4.0.0 opencv_contrib
```

5.1.3. Configure Python 3 virtual environment for OpenCV 4:

Python virtual environments allow you to work on your Python projects in isolation. They are a best practice for Python development.

- Install virtualenv and virtualenvwrapper:

```
sudo pip install virtualenv virtualenvwrapper
```

- Update our ~/.bashrc file:

```
# virtualenv and virtualenvwrapper

export WORKON_HOME=$HOME/.virtualenvs

export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3

source /usr/local/bin/virtualenvwrapper.sh
```

- Source the ~/.bashrc file:

```
source ~/.bashrc
```

- Create your OpenCV 4 and Python 3 virtual environment:

```
mkvirtualenv cv -p python3
```

- Environment creation verification:

```
ubnser@osboxes:~$ workon cv
(cv) ubnser@osboxes:~$
```

Figure 5-1 : Using cv environment

5.1.4. CMake and compile OpenCV 4 for Ubuntu:

In this step we have to setup our compile with CMake followed by running make to compile OpenCV.

- Create Build Directory:

```
cd ~/opencv

mkdir build

cd build
```

- Run CMake for OpenCV 4:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
```

```
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D INSTALL_C_EXAMPLES=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules\
-D PYTHON_EXECUTABLE=~/.virtualenvs/cv/bin/python \
-D BUILD_EXAMPLES=ON ..
```

- **Compile OpenCV 4**

```
make -j4
sudo make install
sudo ldconfig
```

In the make command above, the -j4 argument specifies that I have 4 cores for compilation.

- **Link OpenCV 4 into Python 3 virtual environment:**

```
cd /usr/local/python/cv2/python-3.5
sudo mv cv2.cpython-35m-x86_64-linux-gnu.so cv2.so
Sudo pip install numpy scipy jupyter
```

To verify the **contents** of the environment we type

```
ls .virtualenvs/
```

```
ubnser@osboxes:~$ ls .virtualenvs/
cv              postactivate    postmkvirtualenv  predeactivate    prermvirtualenv
get_env_details postdeactivate  postrmvirtualenv  premkproject
initialize      postmkproject   preactivate       premkvirtualenv
ubnser@osboxes:~$
```

Figure 5-2 : Show environment contents

- **Python 3 bindings for OpenCV should reside in the following folder**

```
sudo mv cv2.cpython-36m-x86_64-linux-gnu.so cv2.so
cd /usr/local/python/cv2/python-
```

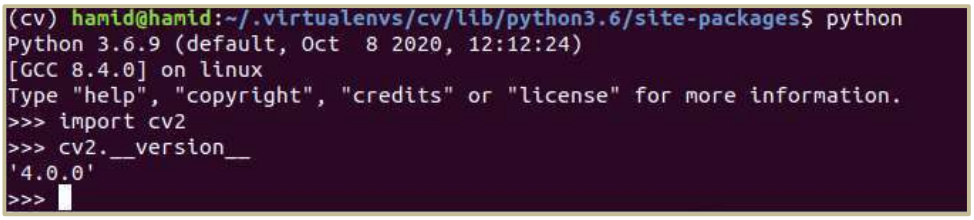

Figure 5-3 : Test openCV4 installation

5.2 Setting up KAFKA environment:

5.2.1.Define environment variables:

By editing the file "/etc/environment " we add the following line

```
JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"

Source /etc/environment
```

- Download Kafka

```
wget https://downloads.apache.org/kafka/2.7.0/kafka_2.13-2.7.0.tgz

tar -xzvf kafka_2.13-2.7.0.tgz

mv kafka_2.13-2.7.0 kafka

sudo apt-get install zookeeperd
```

- Start the kafka environment:

Run Zookeeper server:

```
# Start the ZooKeeper service

$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

- Start the Kafka broker service

```
bin/kafka-server-start.sh config/server.properties
```

- start the Kafka Producer :

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic sam-
pleTopic
```

- start the Kafka consumer :

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
sampleTopic --from-beginning
```

5.3 Setting up Flask :

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

To watch the streaming online we need a streaming platform so we will use Flask. We choose Flask because of the following reasons:

- ¥ Flask is a micro framework which is very light.
- ¥ Easy to install and Setup.

- **Installing Flask :**

```
Workon cv

Pip3 install flask
```

- **Installation test (Hello world!) :**

```
FLASK_APP=hello.py flask run
```

Figure 5-4: Launching Flask App

- **Flask is now running as web server**

Figure 5-5 : Test Flask web server

5.4 App Code and Configuration files :

to run a producer and a consumer file putting into effect our motion detection application. We use OpenCV to create the motion detection application, use Apache Kafka and Flask to see the result in an online Stream. We use Apache Kafka to serve us on each frame in the queue where we can online stream it is using a simple flask application.

We left the zookeeper server and Kafka server running

5.4.1.Producer.py:

- **Import the required libraries to be used :**
  - cv2 (OpenCV) : As the video is comprised of images, OpenCV will be used to preprocess images i.e., starting the web camera, to read video as a stack of frames, deal with the color formatting of the frame, find the gaussian blur of the image, calculation of the difference between frames, image thresholding, dilation, contour detection, and bounding boxes over the motion detected.
  - NumPy: for matrix operations
  - KafkaProducer : The producer consists of a pool of buffer space that holds records that have not yet been transmitted to the server as well as a background I/O thread that is responsible for turning these records into requests and transmitting them to the cluster.

```
import sys

import time

import kafka import kafkaProducer

import numpy as np

import cv2
```

- **Define the codec and create VideoWriter object** (note isColor is False for Gray)

```
fourcc = cv2.VideoWriter_fourcc(*'DIVX')
```

- **Creating a video motiondetect.avi** which will contain only the detected motions in 60 frames per second

```
out = cv2.VideoWriter('motiondetect.avi',fourcc, 30.0, (1280,720), is-
Color = False)
```

- **MOG2 Background Subtraction OpenCV Python :**

There are two functions in OpenCV for subtraction namely MOG2 and KNN and in or project we had choose MOG2 algorithm to create Background Subtractor in this following line without detecting shadow and remove it from the image

```
foreground = cv2.createBackgroundSubtractorMOG2(detectShadows =False)
```

- **Create topic** : that is how the producer and consumer can talk to each other

```
topic="kafka_video"
def publish_video(video_file):
```

- **create producer with this parameter:**

```
    producer = KafkaProducer(bootstrap_servers='localhost:9092')
```

- **Define videoCapture:** Capture Video for Background Subtraction OpenCV

```
    cap= cv2.VideoCapture(video_file)
    print('publishing video ...')
    while(cap.isopened()):
```

- **Reading the frame:**

Read the frames and convert into grayscale, Now read the frames using object initialized above and further convert it into the grayscale format.

```
        success, camframe= cap.read()
```

- **Make sure it works**

```
        if not success:
            print("bad read!")
            break
```

- **Convert image to jpg**

```
            ret, buffer= cv2.imencode('.jpg', camframe)
```

```
            #break them up into bytes for kafka
            producer.send(topic, buffer.tobytes())
```

- **Wait a certain number of seconds to allow the camera time to warmup**

```
            time.sleep(0.2)
        cap.release()
        print('end of stream')
def publish_camera():
```

- **run producer.py**

```
    producer = KafkaProducer(bootstrap_servers='localhost:9092')
```

- **Create video capture Object.**

To capture a live stream from the web camera using OpenCV the first step is to create a VideoCapture object by passing 0 as an argument to signify the camera to use. Then et the window size to a particular dimension.

```
    camera= cv2.VideoCapture(0)
    camera.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    camera.set(cv2.CAP_PROP_FPS,30)
```

- **Frame capture:**

we will run while loop, to capture every frame from the web camera and run until the camera will be closed.

```
    try:
                while(True):
                success, camframe=camera.read()
```

- **Change camframe to gray frame :**

```
                            grayframe = cv2.cvtColor(camframe,
cv2.COLOR_BGR2GRAY)
```

- **Image Blurring:**

Now the next step is to remove the noise from each frame basically remove high frequencies from the images using **GaussianBlur** function by CV2. Pass the height and width of the low pass filter kernel as an argument along with the standard deviation in both directions.

Also, pass the grayscale image as input to the function. After blurring the image, motion in the consecutive frames will be detected easily.

```
blurframe = cv2.GaussianBlur(camframe,(5,5), 0)
```

- **Capture motion :**
  Applying Background Subtraction in OpenCV Python and create motion frame by applying foreground
- **Motionframe = foreground. apply(grayframe)**

```
detect = (np.sum(motionframe))//255
if detect > 30:
print("object in motion size = ",detect)
```

- **Save stream to .avi file**

```
out.write(grayframe)
```

- **Convert to jpg image:**

```
ret, buffer = cv2.imencode('.jpg',grayframe)
producer.send(topic, buffer.tobytes())
```

- **Time for processing:**

```
time.sleep(0.2)
except :
print("\n We are done.")
sys.exit(1)
camera.release()
out.release()
if __name__ == '__main__' :
if(len(sys.argv) > 1):
```

```
                        video_path = sys.argv[1]
                        publish_video(video_path)
                else :
                        print("we are live ")
publish_camera()
```

5.4.2. Consumer.py :

```
import datetime
from flask import Flask, Response
from kafka import kafkaConsumer
```

- start consumper.py
- importing the topic for the apache kafka so it can communicate with the #producer

```
topic="kafka_video"
consumer = kafkaConsumer(topic, bootstrap_servers=['localhost:9092'])
```

- Get the flask app ready for view

```
app=Flask(__name__)
@app.route('video',methods=['GET'])
```

- this function takes the stream function

```
def video():
   return Response(get_video_stream(),
   mimetype='multipart/x-mixed-replace;boudary=frame')
```

- for each message on the consumer puts all the frames all together again and creates an image on the video stream

```
def get_video_stream():
    for msg in consumer:
        yield (b'--frame\r\n'b'content-Type: image /jpg\r\n\r\n' +
        msg.value +b '\r\n\r\n')
if __name__ == "__main__":
   app.run(hosts='0.0.0.0' , debug=True)
```

6 Programs:

6.1 ZooKeeper:

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed. [1]

► **ZooKeeper: A Distributed Coordination Service for Distributed Applications**

ZooKeeper is a distributed, open-source coordination service for distributed applications. It exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, and groups and naming. It is designed to be easy to program to and uses a data model styled after the familiar directory tree structure of file systems. It runs in Java and has bindings for both Java and C.

Coordination services are notoriously hard to get right. They are especially prone to errors such as race conditions and deadlock. The motivation behind ZooKeeper is to relieve distributed applications the responsibility of implementing coordination services from scratch.

► **Design Goals**

ZooKeeper is simple. ZooKeeper allows distributed processes to coordinate with each other through a shared hierarchical namespace which is organized similarly to a standard file system. The namespace consists of data registers - called znodes, in ZooKeeper parlance - and these are like files and directories. Unlike a typical file system, which is designed for storage, ZooKeeper data is kept in-memory, which means ZooKeeper can achieve high throughput and low latency numbers.

The ZooKeeper implementation puts a premium on high performance, highly available, strictly ordered access. The performance aspects of ZooKeeper mean it can be used in large, distributed systems. The reliability aspects keep it from being a single point of failure. The strict ordering means that sophisticated synchronization primitives can be implemented at the client.

ZooKeeper is replicated. Like the distributed processes it coordinates, ZooKeeper itself is intended to be replicated over a set of hosts called an ensemble.

ZooKeeper is ordered. ZooKeeper stamps each update with a number that reflects the order of all ZooKeeper transactions. Subsequent operations can use the order to implement higher-level abstractions, such as synchronization primitives.

ZooKeeper is fast. It is especially fast in "read-dominant" workloads. ZooKeeper applications run on thousands of machines, and it performs best where reads are more common than writes, at ratios of around 10:1.

Data model and the hierarchical namespace

The namespace provided by ZooKeeper is much like that of a standard file system. A name is a sequence of path elements separated by a slash (/). Every node in Zookeeper's namespace is identified by a path.

6.2 KAFKA

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications. [2]

6.2.1. Apache Kafka® is an event streaming platform:

Event streaming is the digital equivalent of the human body's central nervous system. It is the technological foundation for the 'always-on' world where businesses are increasingly software-defined and automated, and where the user of software is more software.

Technically speaking, event streaming is the practice of capturing data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events; storing these event streams durably for later retrieval; manipulating, processing, and reacting to the event streams in real-time as well as retrospectively; and routing the event streams to different destination technologies as needed. Event streaming thus ensures a continuous flow and interpretation of data so that the right information is at the right place, at the right time.

Kafka combines three key capabilities so you can implement your use cases for event streaming end-to-end with a single battle-tested solution:

1. **To publish** (write) and subscribe to (read) streams of events, including continuous import/export of your data from other systems.
2. **To store** streams of events durably and reliably for as long as you want.
3. **To process** streams of events as they occur or retrospectively.

And all this functionality is provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner. Kafka can be deployed on bare-metal hardware, virtual

machines, and containers, and on-premises as well as in the cloud. You can choose between self-managing your Kafka environments and using fully managed services offered by a variety of vendors.

6.2.2.How does Kafka work :

Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premises as well as cloud environments.

Servers: Kafka is run as a cluster of one or more servers that can span multiple datacentres or cloud regions. Some of these servers form the storage layer, called the brokers. Other servers run Kafka Connect to continuously import and export data as event streams to integrate Kafka with your existing systems such as relational databases as well as other Kafka clusters. To let you implement mission-critical use cases, a Kafka cluster is highly scalable and fault-tolerant: if any of its servers fails, the other servers will take over their work to ensure continuous operations without any data loss.

Clients: They allow you to write distributed applications and microservices that read, write, and process streams of events in parallel, at scale, and in a fault-tolerant manner even in the case of network problems or machine failures. Kafka ships with some such clients included, which are augmented by dozens of clients provided by the Kafka community: clients are available for Java and Scala including the higher-level Kafka Streams library, for Go, Python, C/C++, and many other programming languages as well as REST APIs.

6.2.3.Main Concepts and Terminology:

An **event** records the fact that "something happened" in the world or in your business. It is also called record or message in the documentation. When you read or

write data to Kafka, you do this in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers.

Producers are those client applications that publish (write) events to Kafka, and **consumers** are those that subscribe to (read and process) these events. In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability that Kafka is known for. For example, producers never need to wait for consumers. Kafka provides various guarantees such as the ability to process events exactly-once. Events are organized and durably stored in topics.

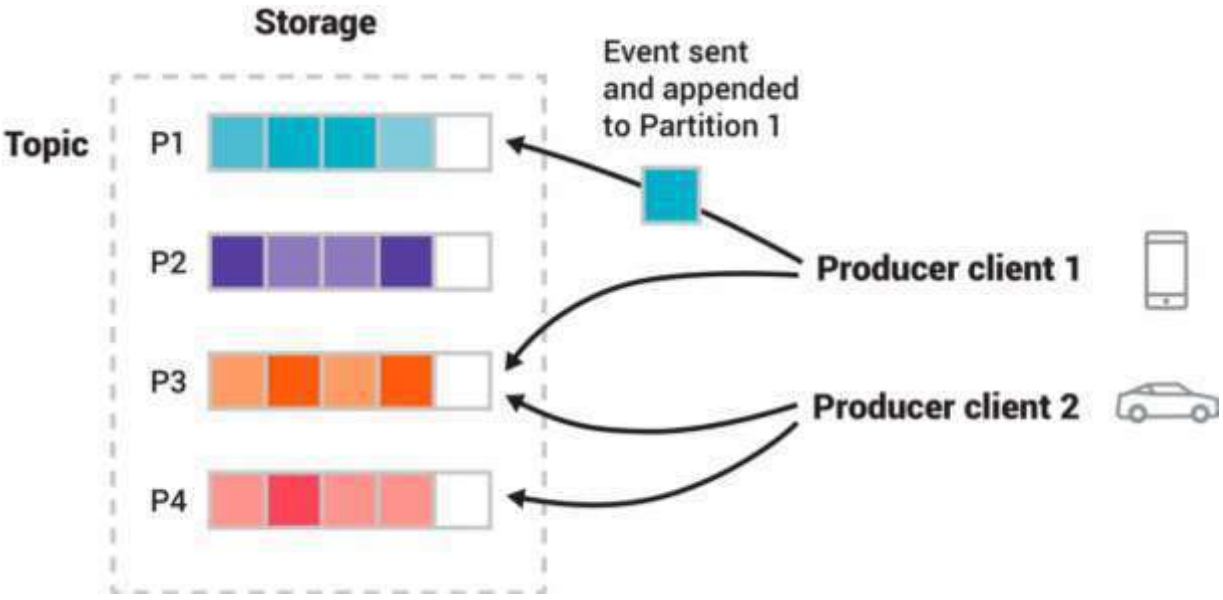

Figure 6-1 : Topic example

This example topic has four partitions P1–P4. Two different producer clients are publishing, independently from each other, new events to the topic by writing events over the network to the topic's partitions. Events with the same key (denoted by their color in the figure) are written to the same partition. Note that both producers can write to the same partition if appropriate.

6.3 OpenCV:

OpenCV or Open-Source Computer Vision library, started out as a research project at Intel. It is currently the largest computer vision library in terms of the sheer number of functions it holds. OpenCV contains implementations of more than 2500 algorithms! It is freely available for commercial as well as academic purposes. The library has interfaces for multiple languages, including Python, Java, and C++.

The first OpenCV version, 1.0, was released in 2006 and the OpenCV community has grown leaps and bounds since then.

Now, let us turn our attention to the idea behind this article – the plethora of functions OpenCV offers! We will be looking at OpenCV from the perspective of a data scientist and learning about some functions that make the task of developing and understanding computer vision models easier. [3]

6.4 FLASK

Flask is a small framework by most standards, small enough to be called a "microframework." It is small enough that once you become familiar with it, you will likely be able to read and understand all its source code. But being small does not mean that it does less than other frameworks. Flask was de- signed as an extensible framework from the ground up; it provides a solid core with the basic services, while extensions provide the rest. Because you can pick and choose the extension packages that you want, you end up with a lean stack that has no bloat and does exactly what you need. Flask has two main dependencies. The routing, debugging, and Web Server Gateway Interface (WSGI) subsystems come from Werkzeug, while template support is provided by Jinja2. Werkzeug and Jinja2 are authored by the core developer of

Flask. There is no native support in Flask for accessing databases, validating web forms, authenticating users, or other high-level tasks. These and many other key services most web applications need are available through extensions that integrate with the core packages. As a developer, you have the power to cherry-pick the extensions that work best for your project or even write your own if you feel inclined to. This is in contrast with a larger framework, where most choices have been made for you and are hard or sometimes impossible to change. In this chapter, you will learn how to install Flask. The only requirement you need is a computer with Python installed. [3]

6.4.1. Using Virtual Environments:

The most convenient way to install Flask is to use a virtual environment. A virtual environment is a private copy of the Python interpreter onto which you can install packages privately, without affecting the global Python interpreter installed in your system. Virtual environments are very useful because they prevent package clutter and version conflicts in the system's Python interpreter. Creating a virtual environment for each application ensures that applications have access to only the packages that they use, while the global interpreter remains neat and clean and serves only as a source from which more virtual environments can be created. As an added benefit, virtual environments don't require administrator rights. Virtual environments are created with the third-party virtualenv utility. To check whether you have it installed in your system, type the following command:

```
virtualenv --version
```

6.4.2. Installing Python Packages with pip:

Most Python packages are installed with the pip utility, which virtualenv automatically adds to all virtual environments upon creation. When a virtual

environment is activated, the location of the pip utility is added to the PATH.

To install Flask into the virtual environment, use the following command:

```
(venv) $ pip install flask
```

With this command, Flask and its dependencies are installed in the virtual environment. You can verify that Flask was installed correctly by starting the Python interpreter and trying to import it:

```
(venv) $ python 3
>>> import flask
>>>
```

6.4.3. Initialization:

All Flask applications must create an application instance. The web server passes all requests it receives from clients to this object for handling, using a protocol called Web Server Gateway Interface (WSGI). The application instance is an object of class Flask, usually created as follows:

```
from flask import

Flask app = Flask(__name__)
```

The only required argument to the Flask class constructor is the name of the main module or package of the application. For most applications, Python's _name___ variable is the correct value.

- **Server Startup :**

The application instance has a run method that launches Flask's integrated development web server:

```
if __name__ == '__main__':
    app.run(debug=True)
```

The ___name___ == '___main___' Python idiom is used here to ensure that the development web server is started only when the script is executed directly. When the

script is imported by another script, it is assumed that the parent script will launch a different server, so the app.run () call is skipped. Once the server starts up, it goes into a loop that waits for requests and services them. This loop continues until the application is stopped, for example by hitting Ctrl-C. There are several option arguments that can be given to app.run () to configure the mode of operation of the web server. During development, it is convenient to enable debug mode, which among other things activates the debugger and the reloader. This is done by passing the argument debug set to True.

7 Results

7.1 Starting Zookeeper

Launching the server service successfully with using the parameters servers from zookeeper.properties

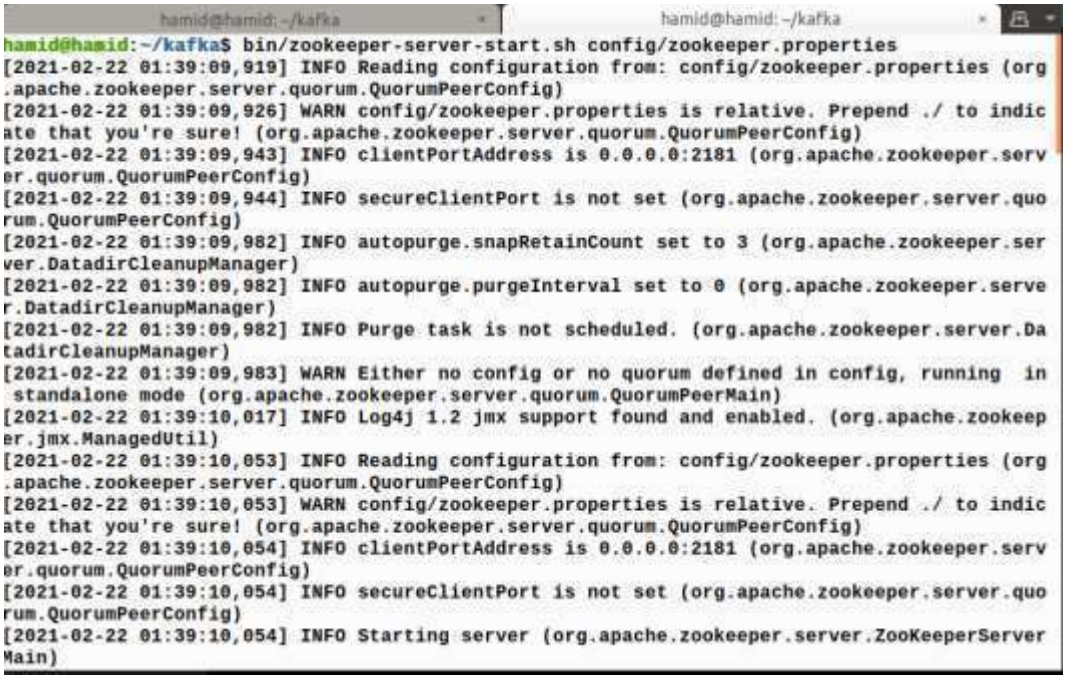

Figure 7-1 : Launching ZooKeeper Server

7.2 Starting kafka Server

Launching the server service successfully using the parameters from server.properties

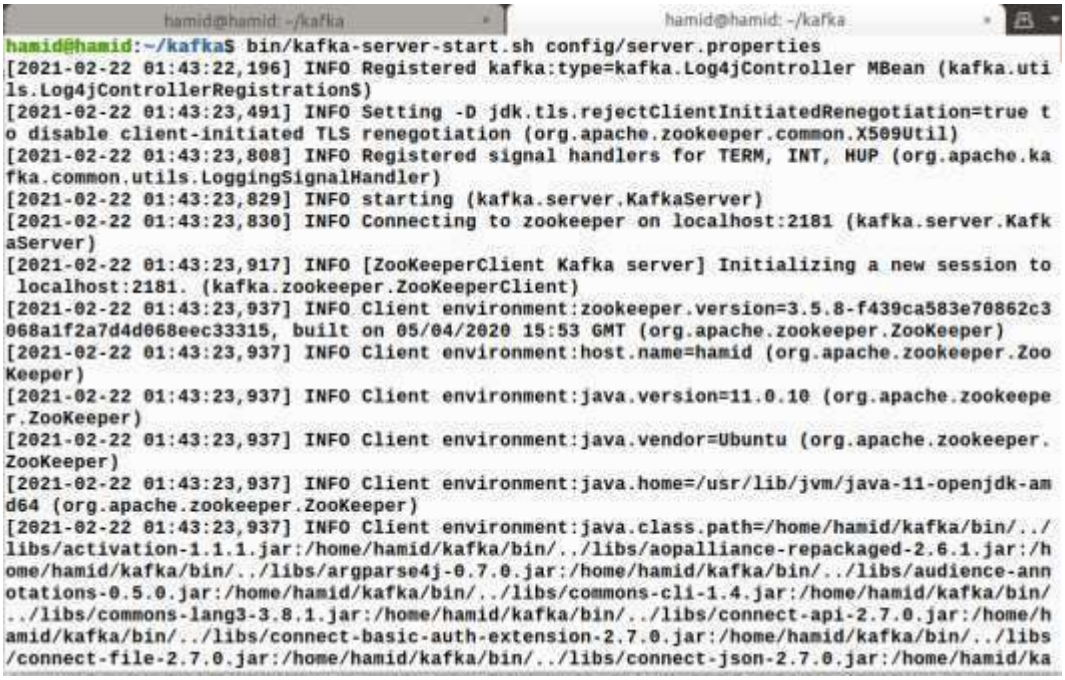

Figure 7-2 : Starting Kafka Server

7.3 Starting producer

After loading the Opencv environment we execute the producer file

```
hamid@hamid:~$ workon cv
(cv) hamid@hamid:~$ cd kafka/
(cv) hamid@hamid:~/kafka$ python3 producer.py
we are live
object in motion size =  921600.0
object in motion size =  5650.0
object in motion size =  3054.0
object in motion size =  2905.0
object in motion size =  3015.0
object in motion size =  15200.0
object in motion size =  2109.0
object in motion size =  1125.0
object in motion size =  612.0
object in motion size =  1703.0
object in motion size =  28217.0
object in motion size =  109382.0
object in motion size =  117580.0
object in motion size =  69605.0
object in motion size =  20181.0
object in motion size =  16442.0
object in motion size =  17640.0
object in motion size =  15903.0
object in motion size =  38013.0
object in motion size =  24697.0
```

Figure 7-3 : Starting producer

7.4 Starting consumer

The consumer is executed and running on http:localhost using port number 5000 in debug mode

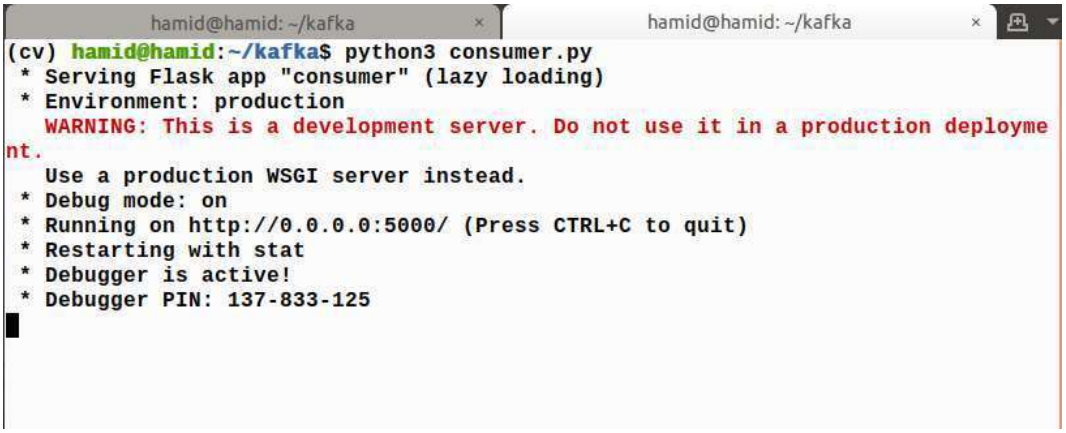

Figure 7-4 : The Consumer

For each message on the consumer puts all the frames all together again and creates an image on the video stream

7.5 visualise Streaming

Finally, we go to the local host 0.0.0.0:5000/video to stream video from a webcam to a web browser/HTML page using **Flask** and **Python**

Figure 7-5 : The streaming

7.6 visualize detection motion.

We can see the date and the size of the video created "**motiondetect.avi**"

```
hamid@hamid: ~/kafka    hamid@hamid: ~/kafka    hamid@hamid: ~/kafka
hamid@hamid:~/kafka$ ls -tlh
total 95M
drwxr-xr-x 2 hamid hamid 4.0K يارب 22 01:43 logs
-rw-r--r-- 1 hamid hamid  94M يارب 22 01:37 motiondetect.avi
-rw-r--r-- 1 hamid hamid 2.2K يارب 22 00:13 producer.py
-rw-r--r-- 1 hamid hamid  873 يارب 21 18:31 consumer.py
drwxr-xr-x 2 hamid hamid 4.0K يارب 21 16:20 __pycache__
-rw-r--r-- 1 hamid hamid  174 يارب 21 15:50 hello.py
drwxr-xr-x 2 hamid hamid 4.0K يارب 20 23:49 config
drwxr-xr-x 2 hamid hamid 4.0K يارب 20 23:27 libs
drwxr-xr-x 2 hamid hamid 4.0K ربنجد 16 15:03 site-docs
drwxr-xr-x 3 hamid hamid 4.0K ربنجد 16 15:03 bin
-rw-r--r-- 1 hamid hamid  30K ربنجد 16 14:58 LICENSE
-rw-r--r-- 1 hamid hamid  337 ربنجد 16 14:58 NOTICE
hamid@hamid:~/kafka$
```

Figure 7-6 : Generated files

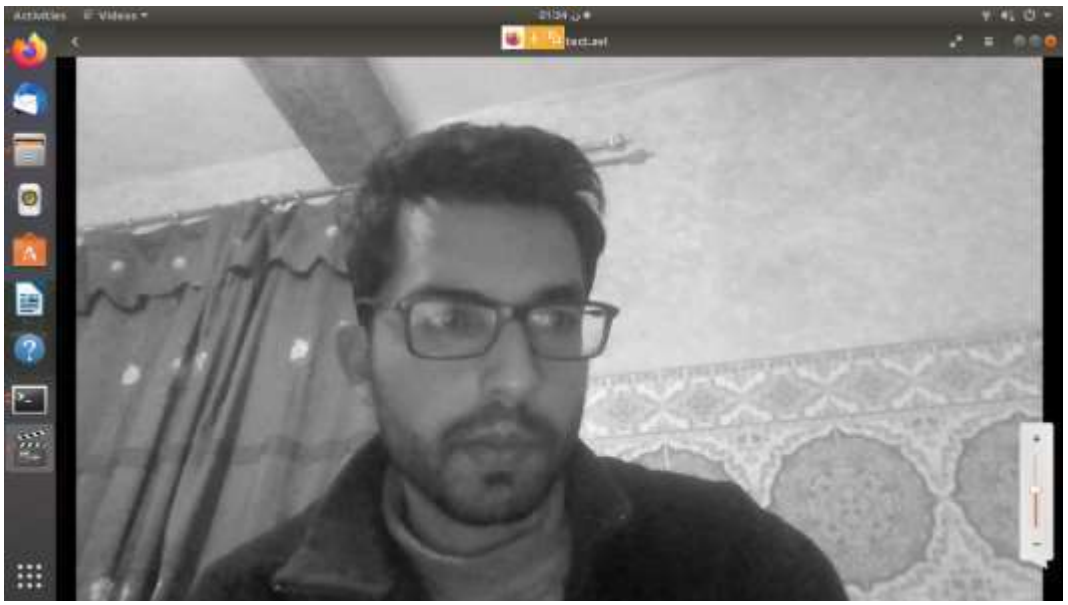

Figure 7-7 : Motion detection - video result

As a result, we generate a video called "**motion detect.avi**" which contain only the motions subtracted from the original video captured by the fixed camera of laptop. The video will be attached with the project files.

8 Discussion

We have focused on the case of a fixed camera, where background image pixels maintain their position in the corresponding frames throughout a video sequence. Although the methods introduced in this domain can be applied successfully to the special case of automated surveillance, where the cameras mounted on a fixed platform, they cannot directly be extended for the cases of moving camera such as video taken by mobile phones, handheld cameras or cameras are mounted on a moving platform where the background image pixels do not maintain their position throughout the video sequence. Most of the review publications in this regard have focused on presenting primitives for detecting moving objects in video and methodologies specifically for tracking objects.

It is worthy to mention that a good segmentation of moving objects and use of appropriate features of each segmented region (including shadow) can work better to eliminate the shadow cast.

The objective of moving object detection is to take a video sequence from a fixed/moving camera and outputs a binary mask representing moving objects for each frame of the sequence. However, this is not an easy task to do due to many challenges and difficulties involved when a camera is used to capture a video sequence of moving objects. Here, we present these challenging issues in detail.

9 Conclusion :

This project was the result of a module rich and full of information was an opportunity to put into practice all our knowledge during the module

Motion detection is the magic ingredient in the current wave of do-it-yourself home security cameras. Thanks to built-in sensors that vigilantly monitor their field of view for movement, you can go about your daily business confident that you will be automatically alerted to any suspicious activity in your home.

Camera motion is recently very common in real-world and many videos are taken by unconstrained movement of cameras. So, the moving object detection algorithms are facing with more challenging difficulties and require more improvements in existing methods like using combinational features or trying newest strategies like deep convolutional neural networks. Here we elaborate some trends in each category for moving object detection in the presence of moving cameras.

Références

[1] «Apache ZooKeeper™,» 02 05 2021. [En ligne]. Available: zookeeper.apache.org.

[2] «APACHE KAFKA,» [En ligne]. Available: http://kafka.apache.org/.

[3] «OpenCV Open Source Computer Vision,» [En ligne]. Available: docs.opencv.org. [Accès le 02 2021].

[4] «The pallets projects,» [En ligne]. Available: https://palletsprojects.com/p/flask/.

[5] S. Apache, «http://spark.apache.org/docs,» [En ligne].