گزارش پنجم

در گزارش های قبلی برنامه ارسال و دریافت تنها یک فریم به سرور را نوشتیم حال قصد داریم به سراغ نوشتن برنامه ای برویم که اطلاعات دوربین را دریافت و آن را به سرور ارسال و از سرور دریافت و نمایش دهد.

پیش از این که این برنامه را بنویسیم نیاز داریم که فرایند ارسال و دریافت اطلاعات را در محیط داخلی بررسی کنیم. به همین منظور 2 ترد در یک برنامه ساخته شد و سعی شد امکان سنجی انجام شود. به همین منظور برنامه زیر نوشته شد:

import concurrent.futures

import logging

import queue

import threading

import time

import cv2 as cv

def producer(queue, event,cap):

    """Pretend we're getting a number from the network."""

    while not event.is\_set():

        ret, message = cap.read()

        if not ret:

            logging.info("Can't receive frame (stream end?). Exiting ...")

            break

        else:

            logging.info("Producer got message: %s", message)

            queue.put(message)

    cap.release()

    logging.info("Producer received event. Exiting")

def consumer(queue, event):

    """Pretend we're saving a number in the database."""

    while not event.is\_set() or not queue.empty():

        message = queue.get()

        cv.imshow("mamad",message)

        cv.waitKey(1)

        logging.info(

            "queue size:%d", queue.qsize()

        )

    cv.destroyWindow('mamad')

    logging.info("Consumer received event. Exiting")

if \_\_name\_\_ == "\_\_main\_\_":

    format = "%(asctime)s: %(message)s"

    logging.basicConfig(format=format, level=logging.INFO,

                        datefmt="%H:%M:%S")

    cap = cv.VideoCapture(0)

    pipeline = queue.Queue(maxsize=10)

    event = threading.Event()

    with concurrent.futures.ThreadPoolExecutor(max\_workers=2) as executor:

        executor.submit(producer, pipeline, event,cap)

        executor.submit(consumer, pipeline, event)

        time.sleep(10)

        logging.info("Main: about to set event")

        event.set()

در این برنامه دو thread ساخته شد و ارتباط میان این دو توسط یک صف که بافر کننده اطلاعات دوربین است پیاده سازی شد. بدین شکل در ابتدا صف و دوربین آماده به کار شده و اطلاعات دوربین به ترد producer منتقل و اطلاعات دریافت کننده به ترد consumer منتقل شد و در داخل ترد تولید کننده اطلاعات از دوربین خوانده شده و به ترد مصرف کننده انتقال پیدا میکند در این ترد اطلاعات دریافت و به نمایش در می­آید.

با بررسی این برنامه و اینکه اطلاعات به صورت بلادرنگ نمایش داده شد میتوان استنتاج کرد که میتوان از یک بخش صف برنامه را در یک کامپیوتر دیگر تحت شبکه منتقل نمود تا بتوان برنامه را تحت شبکه اجرا کرد.

در گزارش قبلی نحوه ارسال تنها یک فریم توسط rabbitmq بررسی شد. حال که از ارسال تصویر به شکل فوق اطمینان حاصل شد میتوان از این نرم افزار به عنوان صفی که بین برنامه ها وجود دارند استفاده نمود.

به همین منظور می­ بایست ابتدا سرور rabbitmq را بالا آورده و فعال کرد. سپس دو برنامه به صورت مستقل نوشته شود که فرایند ارسال اطلاعات به سرور را انجام دهد و برنامه دیگری برای دریافت اطلاعات از سرور نوشت که اطلاعات را نمایش یا ذخیره کند. در ادامه این دو برنامه را بررسی می کنیم.

برنامه ارسال کننده اطلاعات به سرور

from time import time

import cv2 as cv

import pickle

import pika

import json

#IP Webcam Mobile application

cap = cv.VideoCapture('rtsp://192.168.1.4:12525/h264\_ulaw.sdp')

#Connect to Server

credentials = pika.PlainCredentials('guest', 'guest')

parameters = pika.ConnectionParameters('localhost',

                                       5672,

                                        '/',

                                        credentials)

channel=pika.BlockingConnection(parameters).channel()

i=0

if not cap.isOpened():

    print("Cannot open camera")

    exit()

else:

    while True:

        # Capture frame-by-frame

        ret, frame = cap.read()

        # if frame is read correctly ret is True

        if not ret:

            print("Can't receive frame (stream end?). Exiting ...")

            break

        else:

            channel.basic\_publish(

            exchange='e.R',

            routing\_key='c.1',

            body=frame.tobytes(),

            properties=pika.BasicProperties(delivery\_mode = 2,)

            )

        i+=1

        if i>100:

            channel.basic\_publish(

            exchange='e.R',

            routing\_key='c.1',

            body=bytes(0),

            properties=pika.BasicProperties(delivery\_mode = 2,)

            )

            break

    cap.release()

    cv.destroyAllWindows()

در این برنامه ابتدا به سرور ربیت متصل شده در ادامه شروع به خواندن اطلاعات از دوربین میکند و 100 فریم را به سرور ارسال کرده و سپس یک بایت خالی ارسال میکند. نکته قابل توجه این است که برای ارسال اطلاعات دوربین به سرور باید آن ها را به صورت بایت در آورده و به سرور منتقل کرد.

فرایند ارسال اطلاعات را در تابع زیر مشاهده میکنیم

            channel.basic\_publish(

            exchange='e.R',

            routing\_key='c.1',

            body=frame.tobytes(),

            properties=pika.BasicProperties(delivery\_mode = 2,)

            )

که اطلاعات را با از حالت Numpy arry به byte در آورده و به exchange، e.R با کلید مسیریابی c.1 ارسال میکند.

و پس از 100 امین فریم اطلاعات ارسال شده را ذخیره میکند.

برنامه سمت گیرنده

import  pika

import numpy as np

Desiger\_FPS=30.0

fourcc = cv.VideoWriter\_fourcc(\*'XVID')

out = cv.VideoWriter('output.avi', fourcc, Desiger\_FPS , (1920,1080))

q = Queue()

class Worker(threading.Thread):

    def \_\_init\_\_(self, q, \*args, \*\*kwargs):

        self.q = q

        super().\_\_init\_\_(\*args, \*\*kwargs)

    def run(self):

        while True:

            data=self.q.get()

            if data !='a':

                out.write(data)

            else:

                print('save kardam')

                out.release()

            # do whatever work you have to do on work

            self.q.task\_done()

Worker(q).start()

def Packet\_Handeler\_callback(ch, method, properties, body):

    if body:

        frames=np.frombuffer(body,dtype=np.dtype('uint8'))

        frames=frames.reshape(1080, 1920, 3)

        q.put\_nowait(frames)

    else:

        q.put\_nowait('a')

    ch.basic\_ack(delivery\_tag = method.delivery\_tag)

credentials = pika.PlainCredentials('guest', 'guest')

parameters = pika.ConnectionParameters('localhost',

                                       5672,

                                        '/',

                                        credentials)

channel=pika.BlockingConnection(parameters).channel()

channel.basic\_qos(prefetch\_count=1)

channel.basic\_consume(queue='Cam1',

              on\_message\_callback=Packet\_Handeler\_callback,

              consumer\_tag='1')

print(' [\*] Waiting for messages')

channel.start\_consuming()

این برنامه ابتدا شرایط ذخیره سازی اطلاعات را فراهم کرده بدین شکل که فرمت ذخیره سازی را که در گزارش دوم آمد فراهم کرده و سپس یک ترد و یک صف برای ذخیره سازی اطلاعات فراهم میکنیم تا در صورتی که اطلاعات به ترد مربوطه ارسال شد آن را ذخیره کند.

سپس به سراغ برنامه ذخیره کننده اطلاعات میرویم در این برنامه اتصال به سرور را فراهم کرده وسپس یک تابع بر روی صفی که برنامه ارسال کننده اطلاعات به آن اطلاعات را ارسال میکنند فراهم میکنیم.و تابع packet\_handler\_calback را به عنوان تابعی که باید هنگام دریفات اطلاعات فراخوانی شود، را تعیین میکنیم. در این برنامه فریم را از حالت byte به حالت numpyarry در می آوریم ولی سپس نیاز به تبدیل آن به ماتریس هایی با ابعاد مشخص داریم. در این برنامه به صورت دستی اطلاعات به حالت دستی به فرمت مشخص تغییر وضعیت دادیم. این کار توسط متد زیر انجام شد

frames.reshape(1080, 1920, 3)

و سپس بسته را به صف داخلی که به ترد ذخیره سازی اطلاعات وصل شده بود منتقل کرده و ack را به سرور بر میگردانیم[[1]](#footnote-1).

حال سعی میکنیم اطلاعات دریافت شده را به صورت لایو نمایش دهیم، در برنامه فوق کرنل اصلی برنامه به دریافت اطلاعات مشغول بود و نمیتوانستیم برنامه را به صورت زنده نمایش دهیم برای حل این مشکل باید برنامه دریافت کننده اطلاعات را نیز به یک ترد مجزا ببریم تا بتوانیم برنامه را در زمان واقعی نمایش دهیم پس برنامه دریافت کننده اطلاعات را برای نمایش به شکل زیر تغییر یافت:

import concurrent.futures

import logging

import queue

import threading

import time

import cv2 as cv

import  pika

import numpy as np

def Viewer(queue, event):

    """Pretend we're saving a number in the database."""

    while not event.is\_set() or not queue.empty():

        message = queue.get()

        cv.imshow("window",message)

        cv.waitKey(1)

        logging.info(

            "queue size:%d", queue.qsize()

        )

    cv.destroyWindow('window')

    logging.info("Consumer received event. Exiting")

def Packet\_Handeler\_callback(ch, method, properties, body,queue):

    frames=np.frombuffer(body,dtype=np.dtype('uint8'))

    frames=frames.reshape(480, 640, 3)#(1080, 1920, 3)

    queue.put(frames)

    ch.basic\_ack(delivery\_tag = method.delivery\_tag)

#Connect to RabbitMQ

credentials = pika.PlainCredentials('guest', 'guest')

parameters = pika.ConnectionParameters('localhost',

                                       5672,

                                        '/',

                                        credentials)

channel=pika.BlockingConnection(parameters).channel()

channel.basic\_qos(prefetch\_count=10)

channel.basic\_consume(queue='Cam1',

                      on\_message\_callback=

                      lambda ch, method, properties, body:

                          Packet\_Handeler\_callback(

                              ch, method, properties, body,pipeline

                              ),

                     consumer\_tag='1')

#prepare Loging format for debuging

format = "%(asctime)s: %(message)s"

logging.basicConfig(format=format, level=logging.INFO,

                    datefmt="%H:%M:%S")

#Queue for Buffring the frame

pipeline = queue.Queue(maxsize=500)

#Create event for stoping threads

event = threading.Event()

#Lunch thread

with concurrent.futures.ThreadPoolExecutor(max\_workers=2) as executor:

    executor.submit(Viewer  , pipeline, event)

    print(' [\*] Waiting for messages')

    channel.start\_consuming()

بدین شکل مشابه برنامه قبلی یک ترد ساخته شد که اطلاعات دریافتی از دوربین را به جای اینکه ذخیره کند، آن ها را توسط opencv نمایش دهد. ولی همچنان مشکل نحوه پارس کردن و تبدیل ابعاد تصویر را داریم برای حل این مشکل نیاز است که تغییراتی در برنامه ارسال کننده و دریافت کننده اطلاعات انجام شود، بدین شکل که برنامه ارسال کننده اطلاعات ابعاد تصویر ارسالی را هنگام ارسال داخل بایت های اولیه ارسالی قرار دهد و برنامه گیرنده اطلاعات این اطلاعات را دریافت کند و بر اساس آن تصاویر را به فرمت Numpy array در آورد. لازم به ذکر است که اطلاعاتی که در داخل فریم ها قرار میدهیم 8 بیتی است پس رنج اعدادی که میتوان انتقال داد در بازه 0 تا 255 قرار دارد پس باید اعدادی که قصد داریم منتقل کنیم کد کنیم و در سمتی که قصد داریم اطلاعات را باز کنیم دیکد کنیم، بنابراین دو تابع کدر و دیکدر زیر نوشته شد تا بتوان اعداد را منتقل کرد:

def coding (x):

    x = round(x\*10000)-128

    if x>255:

        return 255

    elif x<0:

        return 0

    else:

        return x

def decoding(x):

    return (128+x)/10000

و برنامه ارسال کننده اطلاعات و دریافت کننده اطلاعات را به شکل زیر بازنویسی کرد.

برنامه ارسال کننده اطلاعات:

from time import time

import cv2 as cv

import pika

def coding (x):

    x = round(x\*10000)-128

    if x>255:

        return 255

    elif x<0:

        return 0

    else:

        return x

#IP Webcam Mobile application

cap = cv.VideoCapture('rtsp://192.168.1.4:12525/h264\_ulaw.sdp')

#Connect to Server

credentials = pika.PlainCredentials('guest', 'guest')

parameters = pika.ConnectionParameters('localhost',

                                       5672,

                                        '/',

                                        credentials)

channel=pika.BlockingConnection(parameters).channel()

i=0

if not cap.isOpened():

    print("Cannot open camera")

    exit()

else:

    period=time()

    while True:

        # Capture frame-by-frame

        ret, frame = cap.read()

        # if frame is read correctly ret is True

        if not ret:

            print("Can't receive frame (stream end?). Exiting ...")

            break

        else:

            Lable=time()-period

            period=time()

            frame[0][0][0]=coding(Lable)

            channel.basic\_publish(

            exchange='e.R',

            routing\_key='c.1',

            body=frame.tobytes(),

            properties=pika.BasicProperties(delivery\_mode = 2,)

            )

        i+=1

        if i>100:

            channel.basic\_publish(

            exchange='e.R',

            routing\_key='c.1',

            body=bytes(0),

            properties=pika.BasicProperties(delivery\_mode = 2,)

            )

            break

    cap.release()

    cv.destroyAllWindows()

برنامه دریافت کننده اطلاعات

import concurrent.futures

import logging

import queue

import threading

import time

import cv2 as cv

import  pika

import numpy as np

def decoding(x):

    return (128+x)/10000

def Packet\_Handeler\_callback(ch, method, properties, body,queue):

    frames=np.frombuffer(body,dtype=np.dtype('uint8'))

    frames=frames.reshape(480, 640, 3)#(1080, 1920, 3)

    queue.put(frames)

    ch.basic\_ack(delivery\_tag = method.delivery\_tag)

def Viewer(queue, event):

    """Pretend we're saving a number in the database."""

    while not event.is\_set() or not queue.empty():

        message = queue.get()

        cv.imshow("window",message)

        cv.waitKey(1)

        logging.info(

            "queue size:%d", queue.qsize()

        )

    cv.destroyWindow('window')

    logging.info("Consumer received event. Exiting")

#Connect to RabbitMQ

credentials = pika.PlainCredentials('guest', 'guest')

parameters = pika.ConnectionParameters('localhost',

                                       5672,

                                        '/',

                                        credentials)

channel=pika.BlockingConnection(parameters).channel()

channel.basic\_qos(prefetch\_count=10)

channel.basic\_consume(queue='Cam1',

                      on\_message\_callback=

                      lambda ch, method, properties, body:

                          Packet\_Handeler\_callback(

                              ch, method, properties, body,pipeline

                              ),

                     consumer\_tag='1')

#prepare Loging format for debuging

format = "%(asctime)s: %(message)s"

logging.basicConfig(format=format, level=logging.INFO,

                    datefmt="%H:%M:%S")

#Queue for Buffring the frame

pipeline = queue.Queue(maxsize=500)

#Create event for stoping threads

event = threading.Event()

#Lunch trad

with concurrent.futures.ThreadPoolExecutor(max\_workers=2) as executor:

    executor.submit(Viewer  , pipeline, event)

    #set counsumer on the specific queue

    # try:

    #channel start consuming

    print(' [\*] Waiting for messages')

    channel.start\_consuming()

در این این دو برنامه تاخیر زمانی بین فریم ها انتقال داده شد. با همین ایده و توابع میتوان ابعاد تصویر را منتقل کرد.

1. در برنامه های بعدی این مکانیزم حذف خواهد شد. [↑](#footnote-ref-1)