گزارش دوم

مقدمه

برای انتقال تصویر در شبکه از پروتکل های جریان ارتباط زنده متقاوتی استفاده می­شود یکی از پروتکل ها rtsp است. پروتکل جریان واقعی زمان واقعی (RTSP) یک پروتکل کنترل شبکه است که برای استفاده در سیستم های سرگرمی و ارتباطی برای کنترل سرورهای رسانه پخش جریانی طراحی شده است. پروتکل برای ایجاد و کنترل جلسات رسانه بین نقاط انتهایی استفاده می شود. مشتریان سرورهای رسانه ای برای سهولت در کنترل لحظه ای جریان رسانه از سرور به مشتری (Video On Demand) یا از مشتری به سرور (ضبط صدا) ، دستوراتی مانند پخش ، ضبط و مکث را صادر می کنند. RTSP توسط RealNetworks ، Netscape و دانشگاه کلمبیا توسعه یافت. اولین پیش نویس ارائه شده در IETF در اکتبر 1996 توسط Netscape و Progressive Networks ، و پس از آن Henning Schulzrinne از دانشگاه کلمبیا "RTSP" ("RTSP prime") را در دسامبر 1996 ارائه داد. دو پیش نویس برای استاندارد سازی توسط کارگروه کنترل جلسه چندرسانه ای چند جانبه (MMUSIC WG) از کارگروه مهندسی اینترنت (IETF) ادغام شدند و پیش نویس های بیشتر توسط گروه کاری منتشر شد. استاندارد پیشنهادی برای RTSP به عنوان RFC 2326 در سال 1998 منتشر شد. RTSP 2.0 به عنوان RFC 7826 در سال 2016 به عنوان جایگزینی برای RTSP 1.0 منتشر شد. RTSP 2.0 مبتنی بر RTSP 1.0 است اما غیر از سازوکار مذاکره نسخه اصلی سازگار نیست و همچنان "استاندارد پیشنهادی" است.

بررسی opencv و rtsp

با کمک opencv میتوان به دوربین های تحت شبکه که با پروتکل rtsp سرویس میدهند دست رسی پیدا کرد، در این گزارش قصد داریم به نحوه دسترسی به اطلاعات دوربین ها و دریافت آن ها و ذخیره سازی آنها بر روی دیسک بپردازیم.

برای اینکار ابتدا توابع مربوط به دریافت اطلاعات دوربین ها میپردازیم، با کمک تابع زیر میتوان وضعیت دوربینی که قصد داریم اطلاعات آن را بخوانیم را بررسی کنیم.

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

با کمک متد cap.isOpened() که از دوربین فوق ساخته شده، میتوانیم وضعیت دوربین را بررسی کنیم، خروجی این تابع یک عبارت بولین است که وضعیت دوربین را به ما بر میگرداند، در صورتی دوربین در شبکه به درستی قرار گرفته باشد، و این تابع True را برگرداند میتوان اطلاعات دوربین را با متد cap.read() خواند. خروجی cap.read یک تاپل است که درایه اول آن وضعیت فریم خوانده شده را بر میگرداند و درایه دوم آن تصویر قرائت شده از دوربین را بیان میکند. پس با زیر برنامه زیر میتوان اطلاعات از دوربین ها را قرائت نمود.

if not cap.isOpened():

print("Cannot open camera")

exit()

period=time()

while True:

ret, frame = cap.read()

if not ret:

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

break

حال برای نمایش هر فریم از دوربین میتوان با بکارگیری توابع opencv، و صدا کردن تابع cv.imshow('frame', frame)

تصاویر را پشت سر هم نمایش داد.

برای درک بهتر از نحوه خواندن اطلاعات با بکارگیری کتابخانه time سعی شد، تا زمان دریافتی بین فریم ها را محاسبه کرد و آن­ها را با کمک توابعی که در گزارش قبلی ذکر شد نمایش داد. به همین منظور در ابتدای برنامه متغییر period تعریف شد، و با برچسب زمانی شروع برنامه پر شد.

period=time()

حال هنگام دریافت فریم اول متغییر label از اختلاف زمانی، زمان فعلی و زمانی که در ابتدای برنامه تعریف شد تعریف میشود، که معادل زمان، برای دریافت فریم اول است.

Lable=time()-period

period=time()

در ادامه متغییر period را با زمان فعلی پر میکنیم تا بتوانیم اختلاف زمانی تا دریافت فریم بعدی را محاسبه کنیم. و در پایان متغییر label را بر روی تصویر به نمایش در می­آوریم. که خروجی به شرح زیر است:



عدد 0.03 به معنای آن است که اختلاف زمانی بین دو فریم 0.03 ثانیه است که بیانگر این است که دوربین در هر ثانیه 33 فریم دریافت میکند.

حال وقت آن رسیده که به نحوه ذخیره سازی اطلاعات دوربین بپردازیم؛

برای اینکار از توابع opencv، که شامل VideoWriter\_fourcc برای تعیین فرمت فرمت ذخیره سازی اطلاعات و تابع VideoWriter برای تعیین نرخ دریافت اطلاعات دوربین، و ابعاد ذخیره تصویر است، بهره میگیریم.

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

out = cv.VideoWriter('output.avi', fourcc, Desiger\_FPS , (640, 480))

و در هنگام دریفات اطلاعات پس یا پیش نمایش تصویر، فریم دریافتی را با کمک متد write بر روی دیسک ذخیره میکنیم.

ساختار کلی برنامه برای فریم های جدید به شکل زیر در می­آید:

ret, frame = cap.read()

# if frame is read correctly ret is True

if not ret:

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

break

# Our operations on the frame come here

Lable=time()-period

period=time()

cv.putText(frame,f"{Lable:.2f}" , (90, 50), cv.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 255), 2)

every\_frame.append((frame,Lable))

# Display the resulting frame

cv.imshow('frame', frame)

# Write file and check quit!

out.write(frame)

تحلیل زمان دریافتی بین فریم ها:

در صورتی که با دقت بیشتری فریم ها را بررسی کنیم، میبینیم که زمان دریافت برخی از فریم ها کوتاه تر 0.3 است، این از ویژگی های پروتکل rtsp است که هنگامی که در تصویر تغیرات زیادی وجود نداشته باشد، سرعت انتقال بیشتر می­شود که منجر به افزایش بهره وری سیستم میشود. در ادامه لیست فریم هایی که در زمانی کوتاه تر از 0.03 وارد سیستم شدند قابل مشاهده است.

برای ساختن این لیست تمامی فریم ها هنگام ذخیره سازی فریم در یک لیست پایتون ذخیره شدند و بعد از اجرای برنامه نمایش اطلاعات دوربین بدست آمده اند، زیر برنامه مربوطه به شرح زیر است:

every\_frame.append((frame,Lable))

sampel\_under\_mean\_time =[(x[1],i) for i,x in enumerate(every\_frame) if x[1]<0.03 ]

[(0.006000041961669922, 2),

(0.006045341491699219, 69),

(0.026027917861938477, 70),

(0.005218982696533203, 79),

(0.026273250579833984, 80),

(0.0052013397216796875, 89),

(0.027351856231689453, 90),

(0.006039857864379883, 100),

(0.02646470069885254, 101),

(0.006180524826049805, 110),

(0.02589273452758789, 111),

(0.005219697952270508, 120),

(0.02561640739440918, 121)]