گزارش ششم

در گزارش های قبلی برنامه ارسال و دریافت برنامه دریافت و ارسال اطلاعات دوربین به سرور را نوشتیم حال قصد داریم به سراغ نوشتن برنامه gui که اطلاعات دوربین را نمایش دهد و بتواند جایگزین توابع نمایش تصاویر open-cv شود تا بتوانیم تصاویر دوربین را در gui هایی خودمان طراحی میکنیم قرار دهیم

برای اینکار ابتدا برنامه نمایش یک تصویر را در یک gui مینویسیم

from PyQt5 import QtWidgets, QtGui, QtCore

from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QVBoxLayout, QHBoxLayout

from PyQt5.QtGui import QPixmap, QImage, QColor

from PyQt5.QtCore import pyqtSignal, pyqtSlot, Qt,QObject

import sys

import cv2

class App(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Qt static label demo")

        self.disply\_width = 700

        self.display\_height = 700

        # create the label that holds the image

        self.image\_label = QLabel(self)

        #self.image\_label.resize(self.disply\_width, self.display\_height)

        # create a text label

        self.textLabel = QLabel('Polecat')

        # create a vertical box layout and add the two labels

        vbox = QVBoxLayout()

        vbox.addWidget(self.image\_label)

        vbox.addWidget(self.textLabel)

        # set the vbox layout as the widgets layout

        self.setLayout(vbox)

        # don't need the grey image now

        #grey = QPixmap(self.disply\_width, self.display\_height)

        #grey.fill(QColor('darkGray'))

        #self.image\_label.setPixmap(grey)

        # load the test image - we really should have checked that this worked!

        cv\_img = cv2.imread('pic.jpg')

        # convert the image to Qt format

        qt\_img = self.convert\_cv\_qt(cv\_img)

        # display it

        self.image\_label.setPixmap(qt\_img)

    def convert\_cv\_qt(self, cv\_img):

        """Convert from an opencv image to QPixmap"""

        rgb\_image = cv2.cvtColor(cv\_img, cv2.COLOR\_BGR2RGB)

        h, w, ch = rgb\_image.shape

        bytes\_per\_line = ch \* w

        convert\_to\_Qt\_format = QtGui.QImage(rgb\_image.data, w, h, bytes\_per\_line, QtGui.QImage.Format\_RGB888)

        #p = convert\_to\_Qt\_format.scaled(self.disply\_width, self.display\_height, Qt.KeepAspectRatio)

        return QPixmap.fromImage(convert\_to\_Qt\_format)

if \_\_name\_\_=="\_\_main\_\_":

    app = QApplication(sys.argv)

    a = App()

    a.show()

    sys.exit(app.exec\_())

در این برنامه ابتدا یک qlabel با عنوان image\_label ساخته می­شود، سپس تصویر توسط opencv ساخته و توسط تابع convert\_cv\_qt ، که در آن ابعاد و تصویر داده شده و به یک شی از جنس qimage تبدیل می­شود. این تصویر میتواند در یک qlabel نمایش داده شود.

حال که توانسیتم یک تصویر را در یک qlabel نمایش دهیم وقت آن رسیده که اطلاعات دوربین را به صورت local به نمایش در بیاوریم. برای اینکار برنامه زیر نوشته شد.

from PyQt5 import QtGui

from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QVBoxLayout

from PyQt5.QtGui import QPixmap

import sys

import cv2

from PyQt5.QtCore import pyqtSignal, pyqtSlot, Qt, QThread

import numpy as np

class VideoThread(QThread):

    change\_pixmap\_signal = pyqtSignal(np.ndarray)

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.\_run\_flag = True

    def run(self):

        # capture from web cam

        cap = cv2.VideoCapture(0)

        while self.\_run\_flag:

            ret, cv\_img = cap.read()

            if ret:

                self.change\_pixmap\_signal.emit(cv\_img)

        # shut down capture system

        cap.release()

    def stop(self):

        """Sets run flag to False and waits for thread to finish"""

        self.\_run\_flag = False

        self.wait()

class App(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Qt live label demo")

        self.disply\_width = 2000

        self.display\_height = 2000

        # create the label that holds the image

        self.image\_label = QLabel(self)

        self.image\_label.resize(self.disply\_width, self.display\_height)

        # create a text label

        self.textLabel = QLabel('Webcam')

        # create a vertical box layout and add the two labels

        vbox = QVBoxLayout()

        vbox.addWidget(self.image\_label)

        vbox.addWidget(self.textLabel)

        # set the vbox layout as the widgets layout

        self.setLayout(vbox)

        # create the video capture thread

        self.thread = VideoThread()

        # connect its signal to the update\_image slot

        self.thread.change\_pixmap\_signal.connect(self.update\_image)

        # start the thread

        self.thread.start()

    def closeEvent(self, event):

        self.thread.stop()

        event.accept()

    @pyqtSlot(np.ndarray)

    def update\_image(self, cv\_img):

        """Updates the image\_label with a new opencv image"""

        qt\_img = self.convert\_cv\_qt(cv\_img)

        self.image\_label.setPixmap(qt\_img)

    def convert\_cv\_qt(self, cv\_img):

        """Convert from an opencv image to QPixmap"""

        rgb\_image = cv2.cvtColor(cv\_img, cv2.COLOR\_BGR2RGB)

        h, w, ch = rgb\_image.shape

        bytes\_per\_line = ch \* w

        convert\_to\_Qt\_format = QtGui.QImage(rgb\_image.data, w, h, bytes\_per\_line, QtGui.QImage.Format\_RGB888)

        p = convert\_to\_Qt\_format.scaled(self.disply\_width, self.display\_height, Qt.KeepAspectRatio)

        return QPixmap.fromImage(p)

if \_\_name\_\_=="\_\_main\_\_":

    app = QApplication(sys.argv)

    a = App()

    a.show()

    sys.exit(app.exec\_())

در این برنامه یک ترد برای خواندن اطلاعات از دوربین ساخته شد که در داخل آن یک یک سیگنال از جنس pyqtsignal تعریف شد که تصاویر خوانده شده را توسط این سیگنال به برنامه gui منتقل کند. و در مقابل در برنامه gui این سیگنال را به تابع update\_image متصل نمود تا در صورتی که ترد خواننده اطلاعات از دوربین این اطلاعات را سیگنال کرد، این تابع صدا شود. در داخل این تابع، تابعی که پیشتر برای تبدیل فریم تصاویر به شی qimage تعریف کردیم انجام میشود و محتوای qlabel مربوطه تغییر میکند.

حال که اطلاعات دوربین را به صورت داخلی توانستیم دریافت کنیم وقت آن رسیده که اطلاعات را از سرور دریافت کرده و بتوانیم آنها را نمایش دهیم. برای اینکار برای دریافت اطلاعات دوربین ها نیاز به یک ترد مجزا داریم که بتوانیم ارتباط خود را با سرور حفظ کنیم

اولین ایده که به ذهن میرسد این است که دو ترد طراحی کرده و بین آن ها یک صف بسازیم و ترد اول با اجرای رخداد مذکور تغییرات را اعمال کند. به همین جهت برنامه زیر نوشته شد:

from PyQt5 import QtGui

from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QVBoxLayout

from PyQt5.QtGui import QPixmap

import sys

import cv2

from PyQt5.QtCore import pyqtSignal, pyqtSlot, Qt, QThread

import numpy as np

import pika

import queue

QUEUE='Cam1'

def decoding\_time(x):

    return (128+x)/10000

def decoding\_size(x):

    return x\*8

class Rbmq(QThread):

    def \_\_init\_\_(self,Queue):

        super(Rbmq, self).\_\_init\_\_()

        self.connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

        self.channel = self.connection.channel()

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

        self.channel.basic\_consume(queue=QUEUE,

                      on\_message\_callback=

                      lambda ch, method, properties, body:

                          self.dispatch(

                              ch, method, properties, body,Queue

                              ),

                          auto\_ack=True

                        )

        print('Waiting for message')

    def run(self):

        self.channel.start\_consuming()

    def dispatch(self, channel, method, properties, body,Queue):

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

        frames=frames.reshape(decoding\_size(frames[0]), decoding\_size(frames[1]), 3)

        Queue.put(frames)

        #channel.basic\_ack(delivery\_tag = method.delivery\_tag)

class VideoThread(QThread):

    change\_pixmap\_signal = pyqtSignal(np.ndarray)

    def \_\_init\_\_(self,Queue):

        super().\_\_init\_\_()

        self.\_run\_flag = True

        self.queue=Queue

    def run(self):

        while self.\_run\_flag :

             if not self.queue.empty():

                 cv\_img = self.queue.get()

                 self.change\_pixmap\_signal.emit(cv\_img)

    def stop(self):

        """Sets run flag to False and waits for thread to finish"""

        self.\_run\_flag = False

        self.wait()

class App(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Qt live label demo")

        self.disply\_width = 640

        self.display\_height = 480

        # create the label that holds the image

        self.image\_label = QLabel(self)

        self.image\_label.resize(self.disply\_width, self.display\_height)

        # create a text label

        self.textLabel = QLabel('Webcam')

        # create a vertical box layout and add the two labels

        vbox = QVBoxLayout()

        vbox.addWidget(self.image\_label)

        vbox.addWidget(self.textLabel)

        # set the vbox layout as the widgets layout

        self.setLayout(vbox)

        #

        self.Queue=queue.Queue(maxsize=500)

        #init the rabbitmq

        self.rbmq=Rbmq(self.Queue)

        # create the video capture thread

        self.thread = VideoThread(self.Queue)

        # connect its signal to the update\_image slot

        self.thread.change\_pixmap\_signal.connect(self.update\_image)

        # start the thread

        self.thread.start()

        # start reading rabbit packet

        self.rbmq.start()

    def closeEvent(self, event):

        self.thread.stop()

        event.accept()

    @pyqtSlot(np.ndarray)

    def update\_image(self, cv\_img):

        """Updates the image\_label with a new opencv image"""

        qt\_img = self.convert\_cv\_qt(cv\_img)

        self.image\_label.setPixmap(qt\_img)

    def convert\_cv\_qt(self, cv\_img):

        """Convert from an opencv image to QPixmap"""

        rgb\_image = cv2.cvtColor(cv\_img, cv2.COLOR\_BGR2RGB)

        h, w, ch = rgb\_image.shape

        bytes\_per\_line = ch \* w

        convert\_to\_Qt\_format = QtGui.QImage(rgb\_image.data, w, h, bytes\_per\_line, QtGui.QImage.Format\_RGB888)

        p = convert\_to\_Qt\_format.scaled(self.disply\_width, self.display\_height, Qt.KeepAspectRatio)

        return QPixmap.fromImage(p)

if \_\_name\_\_=="\_\_main\_\_":

    app = QApplication(sys.argv)

    a = App()

    a.show()

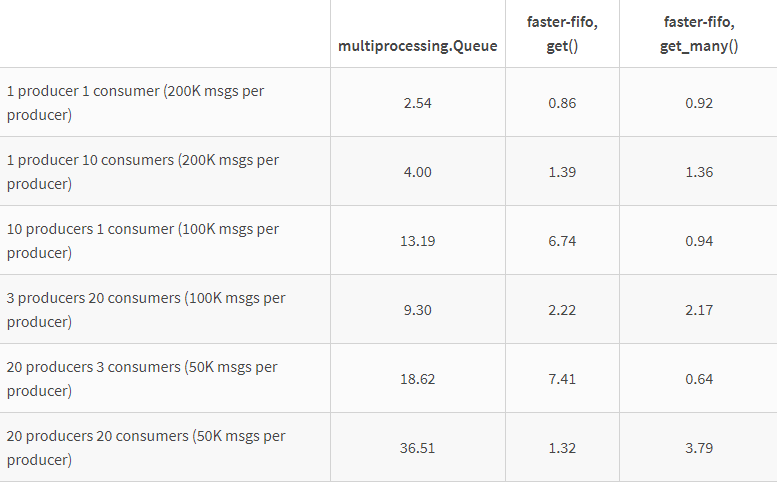
    sys.exit(app.exec\_())

در این برنامه مشابه برنامه قبلی ابتدا یک gui ساخته شده و یک صف برای تبادل اطلاعات بین تردها ساخته میشود. در ادامه ترد مربوط به rabbit که وظیفه دریافت اطلاعات را دارد ساخته می­شود تا فریم های دریافتی را به صف منتقل کند. و ترد فریم های دریافتی را به صورت سیگنال به gui انتقال داده تا تصاویر به نمایش درآیند. ولی مشکل اصلی این روش این است که مکانیزم صفی که در پایتون نوشته شد از سرعت کافی برخوردار نیست و برنامه دچار تاخیر می­شود.

برای حل این مشکل ابتدا به سراغ کتابخانه هایی که مکانیزم سریعتری از صف در اختیار ما قرار میدهند برویم برای اینکار از کتابخانه زیر استفاده شد که سرعت بیشتری در اختیار ما قرار می­دهد.

from quick\_queue import QQueue

ولی همچنان این مشکل حل نشده باقی ماند به جدول زیر برای مقایسه سرعت صف ها توجه کنید:



همانطور که قابل مشاده است کتابخانه اول که استفاده شد multiprocessing.Queue است که به ازای یک مصرف کننده و حجم اطلاعات 200k تاخیری در حدود 3 ثانیه دارد سپس توابع دیگر نیز همچنان تاخیر زیادی دارند پس این راهکار و روش برای انتقال تصویر که هم حجم بسته زیادی و تعداد زیادی دارد مناسب نسیت. برای حل این مشکل سعی شد از همان مکانیزم سیگنال که Pyqt در اختیار ما قرار میدهد استفاده کنیم بدین شکل که همان تردی که برای دریافت اطلاعات ساخته می­شود اطلاعات دریافتی را ارسال کند.

import concurrent.futures

import logging

import queue

import threading

import time

import cv2 as cv

import  pika

import numpy as np

import sys

#QUEUE= sys.argv[1]

QUEUE='Cam1'

def decoding\_time(x):

    return (128+x)/10000

def decoding\_size(x):

    return x\*8

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

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

    frames=frames.reshape(decoding\_size(frames[0]), decoding\_size(frames[1]), 3)

    #param that cloud use in the futures

    #frames[0][0][2]#packet number

    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)

        #delay time between frame

        #time.sleep(decoding\_time(message[0][1][0]) )

        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=30)

channel.basic\_consume(queue=QUEUE,

                      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()

    # except:

    #     channel.stop\_consuming()

    #     #stop the thread

    #     print('Stop Thread')

    #     event.set()

راه حل عمومی این مسئله به شکل فوق است ولی کد فوق برای پیاده سازی مناسب نیست چراکه ترد ایجاد شده به فرمت کلاس نبوده و توانایی توسعه پذیری ندارد پس برنامه زیر نوشته شد که توانایی اجرا در qprocess های مجزا را دارد و میتواند توسعه پذیری برنامه را بیشتر کند.

from PyQt5 import QtGui

from PyQt5.QtWidgets import QWidget, QApplication, QLabel, QVBoxLayout

from PyQt5.QtGui import QPixmap

import sys

import cv2

from PyQt5.QtCore import pyqtSignal, pyqtSlot, Qt, QThread

import numpy as np

import pika

import queue

QUEUE='Cam1'

def decoding\_time(x):

    return (128+x)/10000

def decoding\_size(x):

    return x\*8

class Signals(QWidget):

    change\_pixmap\_signal = pyqtSignal(np.ndarray)

class Rbmq(QThread):

    def \_\_init\_\_(self,Queue):

        super(Rbmq, self).\_\_init\_\_()

        self.signal=Queue

        self.connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))

        self.channel = self.connection.channel()

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

        self.channel.basic\_consume(queue=QUEUE,

                      on\_message\_callback=

                      lambda ch, method, properties, body:

                          self.dispatch(

                              ch, method, properties, body,self.signal

                              ),

                          auto\_ack=True

                        )

        print('Waiting for message')

    def run(self):

        self.channel.start\_consuming()

    def dispatch(self, channel, method, properties, body,Queue):

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

        frames=frames.reshape(decoding\_size(frames[0]), decoding\_size(frames[1]), 3)

        self.signal.emit(frames)

        #channel.basic\_ack(delivery\_tag = method.delivery\_tag)

class App(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Qt live label demo")

        self.disply\_width = 640

        self.display\_height = 480

        # create the label that holds the image

        self.image\_label = QLabel(self)

        self.image\_label.resize(self.disply\_width, self.display\_height)

        # create a text label

        self.textLabel = QLabel('Webcam')

        # create a vertical box layout and add the two labels

        vbox = QVBoxLayout()

        vbox.addWidget(self.image\_label)

        vbox.addWidget(self.textLabel)

        # set the vbox layout as the widgets layout

        self.setLayout(vbox)

        #

        self.Queue=Signals()

        #init the rabbitmq

        self.rbmq=Rbmq(self.Queue.change\_pixmap\_signal)

        # create the video capture thread

        self.Queue.change\_pixmap\_signal.connect(self.update\_image)

        # start reading rabbit packet

        self.rbmq.start()

    @pyqtSlot(np.ndarray)

    def update\_image(self, cv\_img):

        """Updates the image\_label with a new opencv image"""

        qt\_img = self.convert\_cv\_qt(cv\_img)

        self.image\_label.setPixmap(qt\_img)

    def convert\_cv\_qt(self, cv\_img):

        """Convert from an opencv image to QPixmap"""

        rgb\_image = cv2.cvtColor(cv\_img, cv2.COLOR\_BGR2RGB)

        h, w, ch = rgb\_image.shape

        bytes\_per\_line = ch \* w

        convert\_to\_Qt\_format = QtGui.QImage(rgb\_image.data, w, h, bytes\_per\_line, QtGui.QImage.Format\_RGB888)

        p = convert\_to\_Qt\_format.scaled(self.disply\_width, self.display\_height, Qt.KeepAspectRatio)

        return QPixmap.fromImage(p)

if \_\_name\_\_=="\_\_main\_\_":

    app = QApplication(sys.argv)

    a = App()

    a.show()

    sys.exit(app.exec\_())