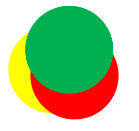
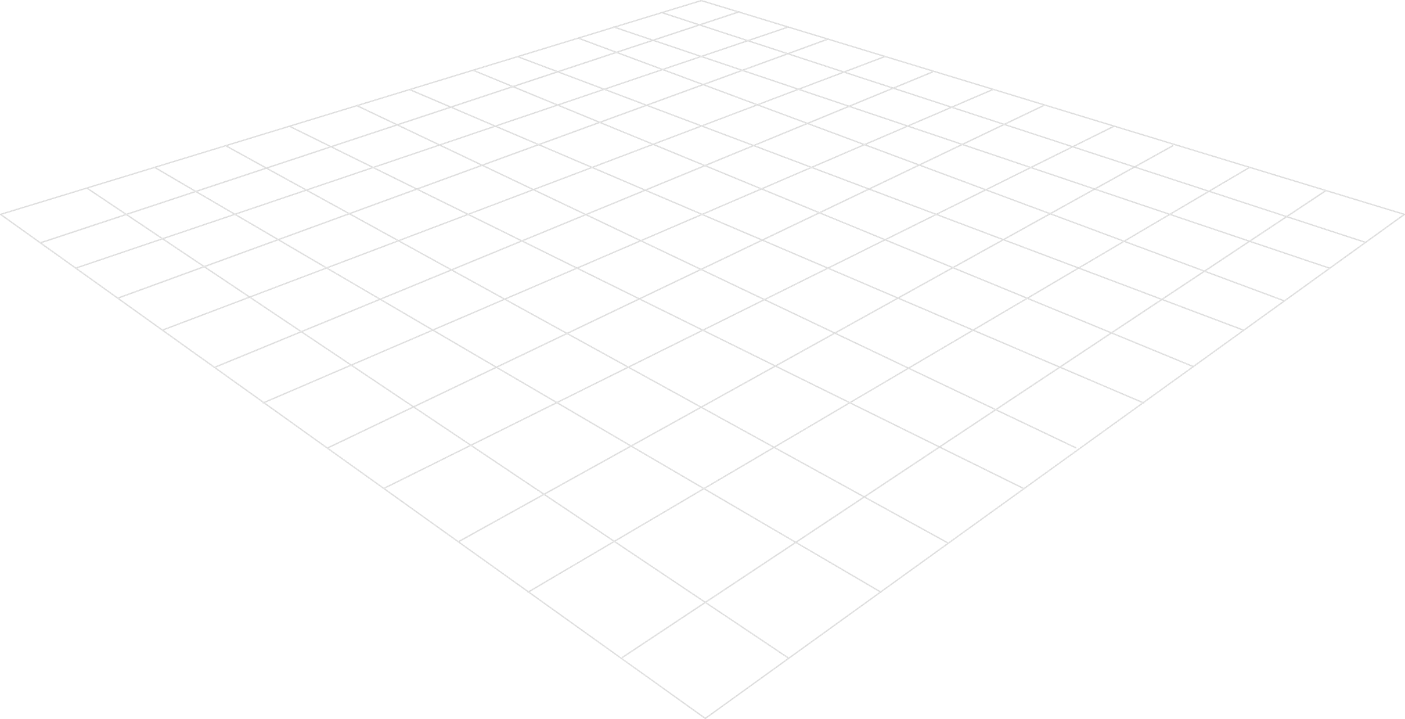
|  |
| --- |
| [PANDUAN APLIKASI] MANAJEMEN LAMPU LALU LINTAS BERBASIS PENGOLAHAN CITRA DIGITAL DAN KECERDASAN BUATAN |
|  |
| December 62020  Ary Setijadi P.  Rahadiyan Yusuf  Reza Darmakusuma  Ach Maulana Habibi Y. |



MANAGEMEN LAMPU LALU LINTAS

BERBASIS KECERDASAN BUATAN

Panduan Aplikasi

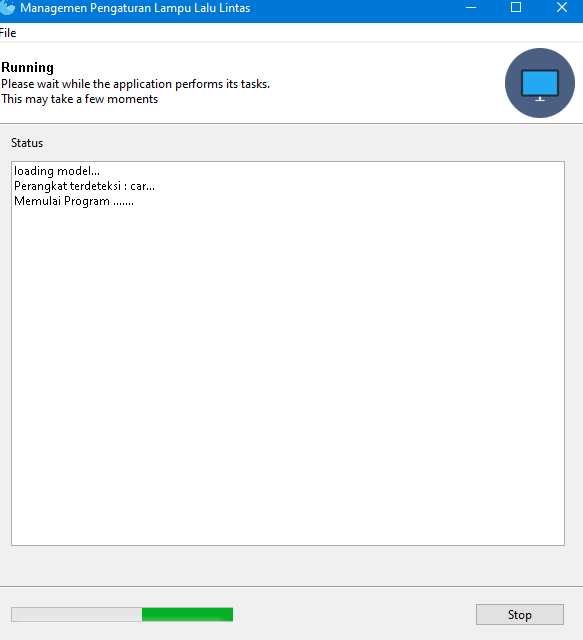
Draft ini menjelaskan tentang panduan mengatur parameter pada aplikasi manajemen lalu lintas berbasi pengolahan citra dan kecerdasan buatan. Fitur GUI (*Graphical User Interface*) dibuat untuk memudahkan pengguna untuk mengatur berbagai hal yang diperlukan agar aplikasi berjalan dengan normal. GUI dibuat dengan mengubah argparser pada program menjadi Gooeyparse.

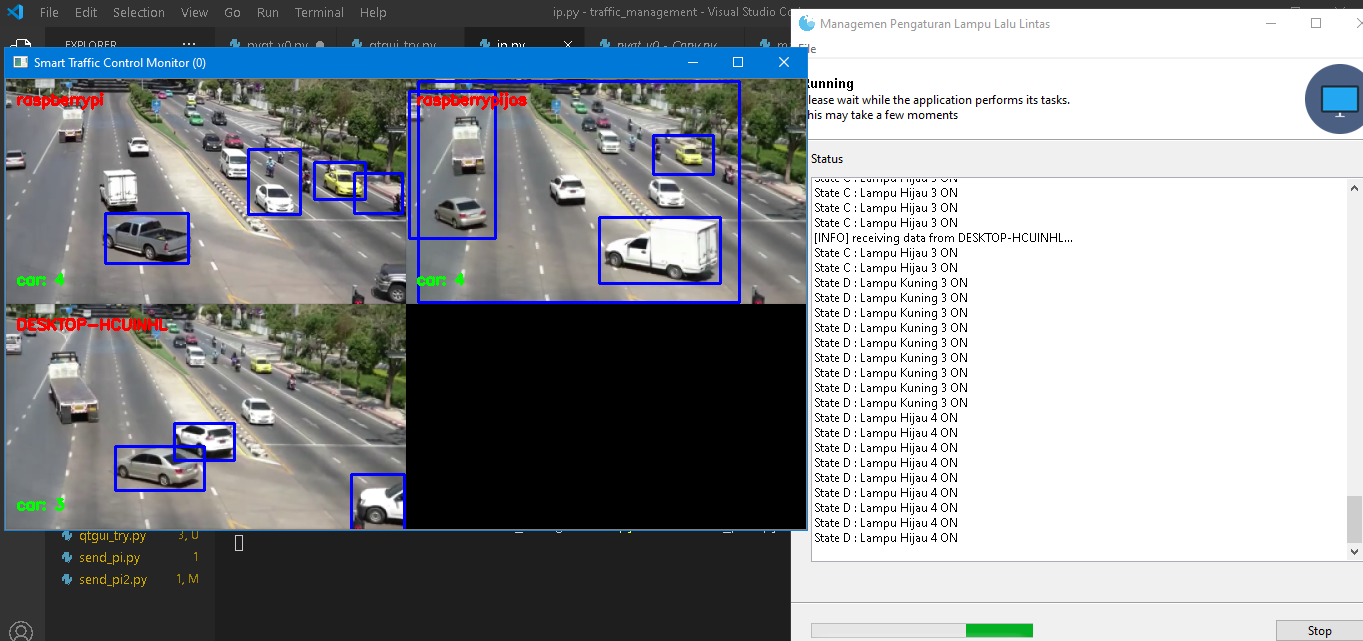
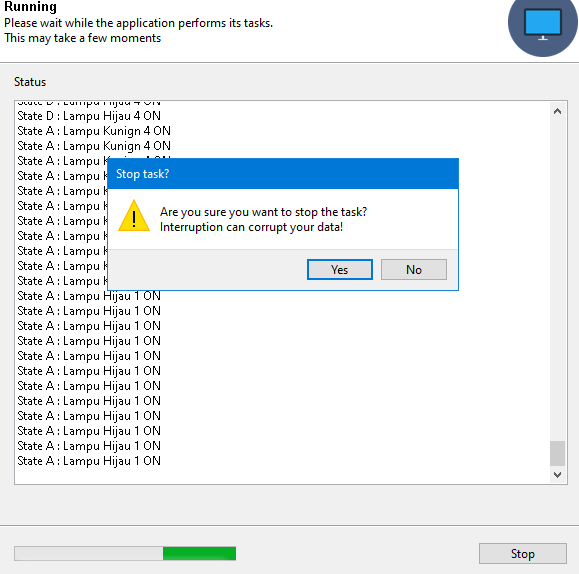
Untuk memulai aplikasi, admin pada pengendali pusat (windows user) membuka command promt / windows powershell. Lalu masuk directory file aplikasi berada. Program dimulai dengan mengetikkan perintah sebagai berikut:

# python3 main.py

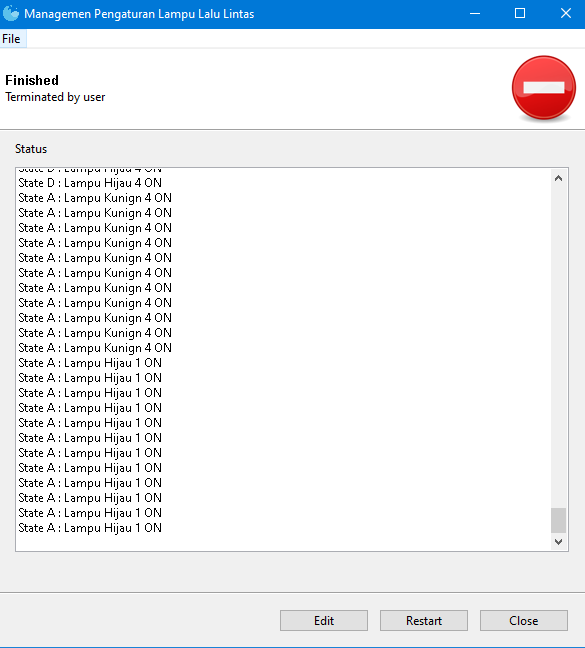
Berikut adalah tampilan awal ketika program main.py dieksekusi. Pengguna diharuskan untuk mengisi kolom dan menginput file yang diperlukan untuk memulai program.

Ketika parameter sudah dimasukkan, maka user memencet tombol Start. Lalu program akan memulai proses. Terdapat tombol Stop untuk menghentikan program.

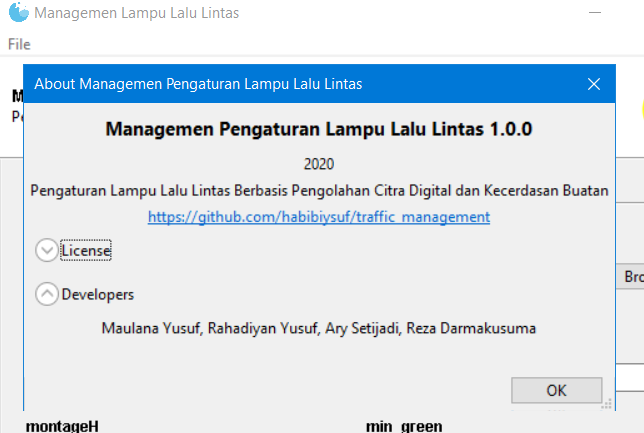


Tampilan ketika program menerima input dan menghasilkan state lampu lalu lintas. Jendela visualisasi output akan muncul secara pop-up pada jendela baru.

Warning ketika tombol stop di klik



Terdapat beberapa pilihan lagi saat program berhasil dihentikan



Tentang aplikasi pada menu Fiile

SOURCE\_CODE

from imutils import build\_montages

from datetime import datetime

import numpy as np

import imagezmq

import argparse

import imutils

import cv2

import random

import time

import threading

from threading import Thread

from gooey import Gooey, GooeyParser

@Gooey(

    program\_name="Managemen Lampu Lalu Lintas",

    program\_description="Pengaturan Lampu Lalu Lintas Berbasis Pengolahan Citra Digital dan Kecerdasan Buatan",

    image\_dir = "C:/Users/achma/Documents/traffic\_management",

    menu=[{'name': 'File', 'items': [{

        'type': 'AboutDialog',

        'menuTitle': 'About',

        'name': 'Managemen Pengaturan Lampu Lalu Lintas',

        'description': 'Pengaturan Lampu Lalu Lintas Berbasis Pengolahan Citra Digital dan Kecerdasan Buatan',

        'version': '1.0.0',

        'copyright': '2020',

        'website': 'https://github.com/habibiysuf/traffic\_management',

        'developer': "Maulana Yusuf, Rahadiyan Yusuf, Ary Setijadi, Reza Darmakusuma",

        'license': 'TMDG ITB' }]

        }]

)

def main():

    parser = GooeyParser(description="Pengaturan Parameter Sistem")

    #ap = argparse.ArgumentParser()

    parser.add\_argument("-p", "--prototxt", required=True,

        help="Caffe 'deploy' prototxt file", widget='FileChooser')

    parser.add\_argument("-m", "--model", required=True,

        help="Caffe pre-trained model", widget='FileChooser')

    parser.add\_argument("-c", "--confidence", type=float, default=0.2,

        help="Probabilitas deteksi (0 -1)")

    parser.add\_argument("-mW", "--montageW", required=True, type=int,

        help="Layar monitor horizontal. Ex. 2 Layar")

    parser.add\_argument("-mH", "--montageH", required=True, type=int,

        help="Layar monitor vertikal. Ex. 2 Layar ")

    parser.add\_argument("-mig", "--min\_green", help="Minimal waktu hijau", type=int, default=5)

    parser.add\_argument("-mag", "--max\_green", help="Maksimal waktu hijau",type=int, default=12)

    parser.add\_argument("-macd", "--max\_car\_detected", help = "Maksimal Kendaraan Terdeteksi (mobil)", type=int, default=10 )

    parser.add\_argument("-micd", "--min\_car\_detected",help = "Minimal Kendaraan Terdeteksi (mobil)", type=int, default=1)

    parser.add\_argument("-sc1", "--node\_1", help = "Nama host node pertama", required=True)

    parser.add\_argument("-sc2", "--node\_2", help = "Nama host node kedua", required=True)

    parser.add\_argument("-sc3", "--node\_3", help = "Nama host node ketiga", required=True)

    parser.add\_argument("-sc4", "--node\_4", help = "Nama host node keempat", required=True)

    args = vars(parser.parse\_args())

    max\_car\_detect = args["max\_car\_detected"]

    min\_car\_detect = args["min\_car\_detected"]

    min\_green = args["min\_green"]

    max\_green = args["max\_green"]

    open\_akhir = True

    open\_awal = True

    state\_a = True

    state\_b = False

    state\_c = False

    state\_d = False

    imageHub = imagezmq.ImageHub()

    CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",

        "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",

        "dog", "horse", "motorbike", "person", "pottedplant", "sheep",

        "sofa", "train", "tvmonitor"]

    print("loading model...")

    net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

    CONSIDER = set(["car"])

    objCount = {obj: 0 for obj in CONSIDER}

    frameDict = {}

    lastActive = {}

    lastActiveCheck = datetime.now()

    ESTIMATED\_NUM\_PIS = 4

    ACTIVE\_CHECK\_PERIOD = 10

    ACTIVE\_CHECK\_SECONDS = ESTIMATED\_NUM\_PIS \* ACTIVE\_CHECK\_PERIOD

    mW = args["montageW"]

    mH = args["montageH"]

    print("Perangkat terdeteksi : {}...".format(", ".join(obj for obj in

        CONSIDER)))

    gate = False

    state\_1 = True

    devices = []

    print("Memulai Program .......")

    while True:

        (rpiName, frame) = imageHub.recv\_image()

        imageHub.send\_reply(b'OK')

        if rpiName not in lastActive.keys():

            print("[INFO] receiving data from {}...".format(rpiName))

            devices.append(rpiName)

        lastActive[rpiName] = datetime.now()

        frame = imutils.resize(frame, width=400)

        (h, w) = frame.shape[:2]

        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),

            0.007843, (300, 300), 127.5)

        net.setInput(blob)

        detections = net.forward()

        objCount = {obj: 0 for obj in CONSIDER}

        for i in np.arange(0, detections.shape[2]):

            confidence = detections[0, 0, i, 2]

            if confidence > args["confidence"]:

                idx = int(detections[0, 0, i, 1])

                if CLASSES[idx] in CONSIDER:

                    objCount[CLASSES[idx]] += 1

                    box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

                    (startX, startY, endX, endY) = box.astype("int")

                    cv2.rectangle(frame, (startX, startY), (endX, endY),

                        (255, 0, 0), 2)

        cv2.putText(frame, rpiName, (10, 25),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

        label = ", ".join("{}: {}".format(obj, count) for (obj, count) in objCount.items())

        cv2.putText(frame, label, (10, h - 20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255,0), 2)

        if len(devices) == 1:

            count\_a = objCount["car"]

            count\_b = random.randint(1,9)

            count\_c = random.randint(1,6)

            count\_d = random.randint(1,7)

        elif len(devices) == 2:

            if devices[0] == args["node\_1"]:

                count\_a = objCount["car"]

            else:

                count\_a = random.randint(1,9)

            if devices[1] == args["node\_2"]:

                count\_b = objCount["car"]

            else:

                count\_b = random.randint(1,9)

            count\_c = random.randint(1,6)

            count\_d = random.randint(1,7)

        elif len(devices) == 3:

            if devices[0] == args["node\_1"]:

                count\_a = objCount["car"]

            else:

                count\_a = random.randint(1,9)

            if devices[1] == args["node\_2"]:

                count\_b = objCount["car"]

            else:

                count\_b = random.randint(1,9)

            if devices[2] == args["node\_3"]:

                count\_b = objCount["car"]

            else:

                count\_b = random.randint(1,9)

            count\_d = random.randint(1,7)

        elif len(devices) == 4:

            if devices[0] == args["node\_1"]:

                count\_a = objCount["car"]

            else:

                count\_a = random.randint(1,9)

            if devices[1] == args["node\_2"]:

                count\_b = objCount["car"]

            else:

                count\_b = random.randint(1,9)

            if devices[2] == args["node\_3"]:

                count\_b = objCount["car"]

            else:

                count\_b = random.randint(1,9)

            if devices[3] == args["node\_4"]:

                count\_b = objCount["car"]

            else:

                count\_b = random.randint(1,9)

        ##################### 1

        if (count\_a <= min\_car\_detect and count\_a != 0):

            output\_1 = min\_green

        elif (count\_a > min\_car\_detect and count\_a <= max\_car\_detect):

            t\_output\_1 = (max\_car\_detect - count\_a) / (max\_car\_detect - min\_car\_detect)

            output\_1 = max\_green - t\_output\_1\*(max\_car\_detect - min\_car\_detect)

        elif (count\_a == 0):

            output\_1 = 0

        elif (count\_a > max\_car\_detect):

            output\_1 = max\_green

        #################### 2

        if (count\_b <= min\_car\_detect and count\_b != 0):

            output\_2 = min\_green

        elif (count\_b > min\_car\_detect and count\_b <= max\_car\_detect):

            t\_output\_2 = (max\_car\_detect - count\_b) / (max\_car\_detect - min\_car\_detect)

            output\_2 = max\_green - t\_output\_2\*(max\_car\_detect - min\_car\_detect)

        elif (count\_b == 0):

            output\_2 = 0

        elif (count\_b > max\_car\_detect):

            output\_2 = max\_green

        ################### 3

        if (count\_c <= min\_car\_detect and count\_c != 0):

            output\_3 = min\_green

        elif (count\_c > min\_car\_detect and count\_c <= max\_car\_detect):

            t\_output\_3 = (max\_car\_detect - count\_c) / (max\_car\_detect - min\_car\_detect)

            output\_3 = max\_green - t\_output\_3\*(max\_car\_detect - min\_car\_detect)

        elif (count\_c == 0):

            output\_3 = 0

        elif (count\_c > max\_car\_detect):

            output\_3 = max\_green

        ################## 4

        if (count\_d <= min\_car\_detect and count\_d != 0):

            output\_4 = min\_green

        elif (count\_d > min\_car\_detect and count\_d <= max\_car\_detect):

            t\_output\_4 = (max\_car\_detect - count\_d) / (max\_car\_detect - min\_car\_detect)

            output\_4 = max\_green - t\_output\_4\*(max\_car\_detect - min\_car\_detect)

        elif (count\_d == 0):

            output\_4 = 0

        elif (count\_d > max\_car\_detect):

            output\_4 = max\_green

        #print("Source 1 : "+str(output\_1)+ " Source 2 : "+str(output\_2)+ " Source 3 : "+str(output\_3)+ " Source 4 : "+str(output\_4), end="\r")

        #susunan (output\_1, output\_2, output\_3, output\_4)

        if (state\_a == True):

            if (open\_akhir == True):

                TTA = time.time()

                KA = TTA + 3

                task\_print = True

            open\_akhir = False

            if task\_print == True:

                print("State A : Lampu Kunign 4 ON")

            if (time.time() > KA):

                task\_print = False

                if (open\_awal == True):

                    TA = time.time()

                    HA = TA + output\_1

                open\_awal = False

                print("State A : Lampu Hijau 1 ON")

                if (time.time() > HA):

                    state\_b = True

                    state\_a = False

                    open\_akhir = True

                    open\_awal = True

        elif (state\_b == True):

            if (open\_akhir == True):

                TTB = time.time()

                KB = TTB + 3

                task\_print = True

            open\_akhir = False

            if task\_print == True:

                print("State B : Lampu Kuning 1 ON")

            if (time.time() > KB):

                task\_print = False

                if (open\_awal == True):

                    TB = time.time()

                    HB = TB + output\_2

                open\_awal = False

                print("State B : Lampu Hijau 2 ON")

                if (time.time() > HB):

                    state\_c = True

                    state\_b = False

                    open\_akhir = True

                    open\_awal = True

        elif (state\_c == True):

            if (open\_akhir == True):

                TTC = time.time()

                KC = TTC + 3

                task\_print = True

            open\_akhir = False

            if task\_print == True:

                print("State C : Lampu Kuning 2 ON")

            if (time.time() > KC):

                task\_print = False

                if (open\_awal == True):

                    TC = time.time()

                    HC = TC + output\_3

                open\_awal = False

                print("State C : Lampu Hijau 3 ON")

                if (time.time() > HC):

                    state\_d = True

                    state\_c = False

                    open\_akhir = True

                    open\_awal = True

        elif (state\_d == True):

            if (open\_akhir == True):

                TTD = time.time()

                KD = TTD + 3

                task\_print = True

            open\_akhir = False

            if task\_print == True:

                print("State D : Lampu Kuning 3 ON")

            if (time.time() > KD):

                task\_print = False

                if (open\_awal == True):

                    TD = time.time()

                    HD = TD + output\_4

                open\_awal = False

                print("State D : Lampu Hijau 4 ON")

                if (time.time() > HD):

                    state\_a = True

                    state\_d = False

                    open\_akhir = True

                    open\_awal = True

        frameDict[rpiName] = frame

        if (gate == True):

            break

        montages = build\_montages(frameDict.values(), (w, h), (mW, mH))

        for (i, montage) in enumerate(montages):

            cv2.imshow("Smart Traffic Control Monitor ({})".format(i),

                montage)

        key = cv2.waitKey(1) & 0xFF

        if (datetime.now() - lastActiveCheck).seconds > ACTIVE\_CHECK\_SECONDS:

            for (rpiName, ts) in list(lastActive.items()):

                if (datetime.now() - ts).seconds > ACTIVE\_CHECK\_SECONDS:

                    print("[INFO] lost connection to {}".format(rpiName))

                    lastActive.pop(rpiName)

                    frameDict.pop(rpiName)

            lastActiveCheck = datetime.now()

        #time.sleep(0.5)

        if key == ord("q"):

            break

    cv2.destroyAllWindows()

if \_\_name\_\_ == "\_\_main\_\_":

    main()