

# MANAJEMEN LAMPU LALU LINTAS BERBASIS PENGOLAHAN CITRA DIGITAL DAN KECERDASAN BUATAN

---

DECEMBER 62020

Ary Setijadi P.

Rahadiyan Yusuf

Reza Darmakusuma

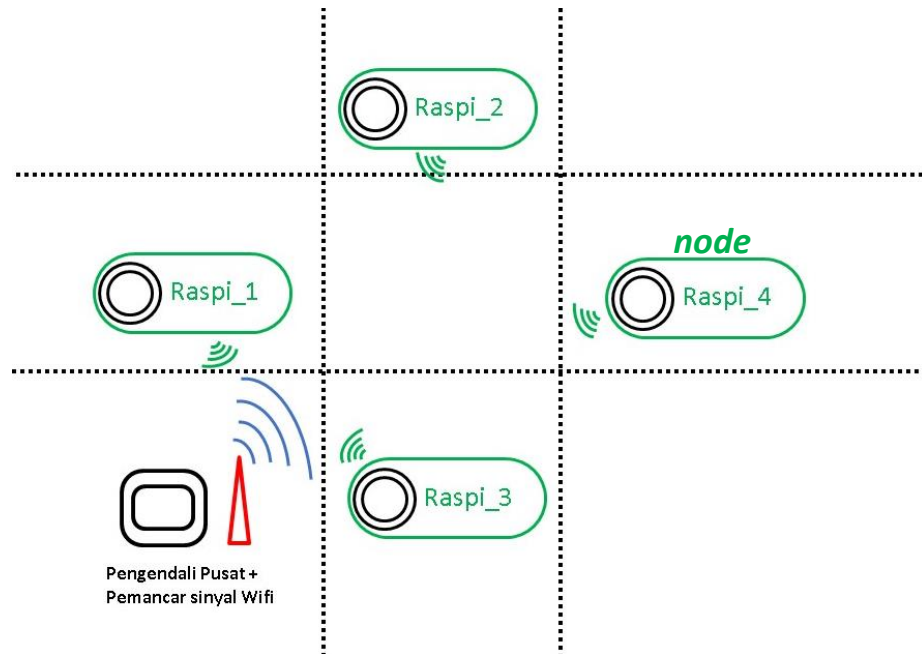
Ach Maulana Habibi Y.



MANAJEMEN LAMPU LALU LINTAS  
BERBASIS KECERDASAN BUATAN

## Deskripsi

Aplikasi ini dimanfaatkan untuk mengatur lampu lalu lintas pada suatu persimpangan jalan. Penentuan waktu lampu lalu lintas bersifat dinamis dan diatur dengan memberikan input jumlah kendaraan pada tiap ruas jalan dalam suatu persimpangan. Pendekatan untuk menentukan waktu lampu lalu lintas menggunakan teknik fuzzy logic sedangkan untuk mengetahui seberapa banyak jumlah kendaraan yang terdapat pada ruas jalan menggunakan teknik kecerdasan buatan dengan sensor kamera. Gambaran umum sistem ini ditunjukkan pada Gambar 1 berikut:

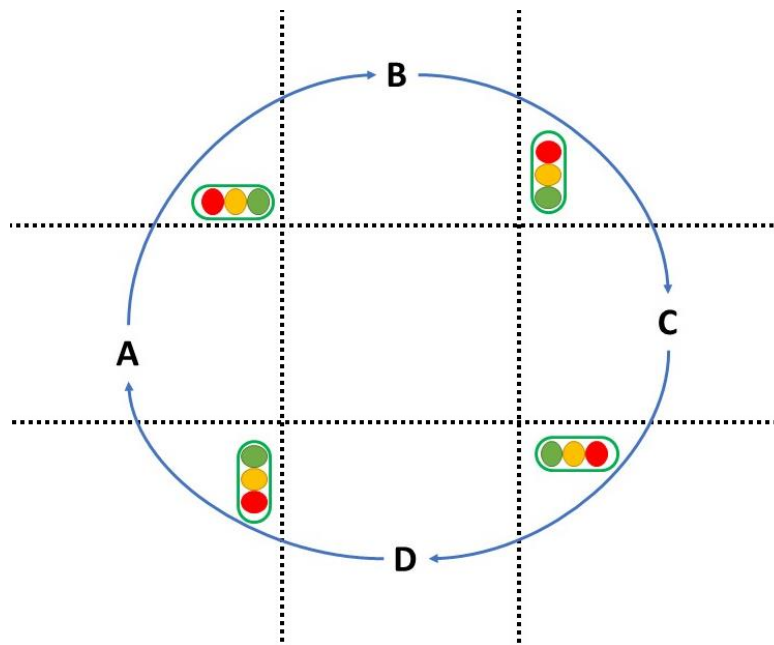


*Skema Sistem*

Sistem yang dipakai menggunakan arsitektur *client-server*. Node pada sistem berfungsi sebagai client dan pengendali pusat bertindak sebagai server. Terdapat empat buah *node* yang tersusun atas sensor kamera dan sebuah mini *single board computer* (SBC) yaitu Raspberry Pi. Pengiriman data dilakukan dengan media wireless. Ke-empat *node* tersebut terhubung pada jaringan yang sama dan bertugas untuk mengirim data gambar saja (*video streaming*) lalu data gambar pada seluruh node akan diterima oleh pengendali pusat secara berurutan.

## SISTEM KENDALI LAMPU LALU LINTAS

Sistem kendali ini dirancang untuk mengatur waktu hijau lampu lalu lintas menyesuaikan jumlah kendaraan yang terdeteksi pada masing-masing jalan di suatu persimpangan. Sistem pergantian *state* kendali dari satu lampu lalu lintas ke lampu lalu lintas berikutnya dibuat teratur atau terpola. Misalkan terdapat sebuah simpang empat yang berarti terdapat empat ruas jalan pada satu persimpangan yaitu ruas jalan A, ruas jalan B, ruas Jalan C, dan ruas Jalan D. Ilustrasi contoh ditunjukkan pada Gambar berikut



*Ilustrasi cara kerja sistem*

Saat sistem dimulai, maka semua *node* akan mengirim data video kepada pusat kendali. Setelah itu kendali lampu dimulai pada ruas jalan A. Pusat kendali akan melakukan proses komputasi untuk menghasilkan waktu hijau dari data streaming node A. Proses komputasi ditentukan dari frame pertama dari data *streaming* dengan input banyaknya deteksi kendaraan pada ruas jalan A. Deteksi kendaraan dilakukan dengan teknik kecerdasan buatan memakai model identifikasi objek berupa MobileNet\_SSD. Ketika waktu hijau telah ditentukan pada frame pertama, maka *state* hijau untuk ruas jalan A diaktifkan dan otomatis ruas jalan yang lain mengaktifkan lampu merah. Ketika waktu lampu hijau telah habis maka kendali akan berpindah pada ruas jalan B dan

---

mengaktifkan lampu kuning pada ruas jalan A. Proses yang sama akan diulang pada kendali ruas jalan B dan ruas jalan yang lain.

---

## Penggunaan Aplikasi

### A. KEBUTUHAN SISTEM

Terdapat dua komponen untuk menjalankan aplikasi ini yaitu komponen perangkat keras/hardware sistem (node dan komponen pengendali pusat) dan komponen perangkat lunak/software sistem. Komponen perangkat keras node terdiri dari sebuah Raspberry Pi dan webcam, sedangkan komponen perangkat keras pengendali pusat yaitu Router dan Laptop/PC. Kebutuhan komponen perangkat lunak dibedakan menurut komponen perangkat keras. Untuk komponen perangkat lunak pada node yaitu :

1. OS Raspbian Buster
2. Pustaka (*library*) python => OpenCV, numpy, imutils, dan imagezmq.

Sedangkan untuk kebutuhan perangkat lunak pada pengendali pusat yaitu:

1. Windows 10
2. Python 3
3. Pustaka (*library*) python => OpenCV, numpy, imutils, imagezmq, dan Goocy.
4. File model MobileNet\_SSD dan file prototxt MobileNet\_SSD

### B. PEMASANGAN KOMPONEN KEBUTUHAN (Perangkat Lunak)

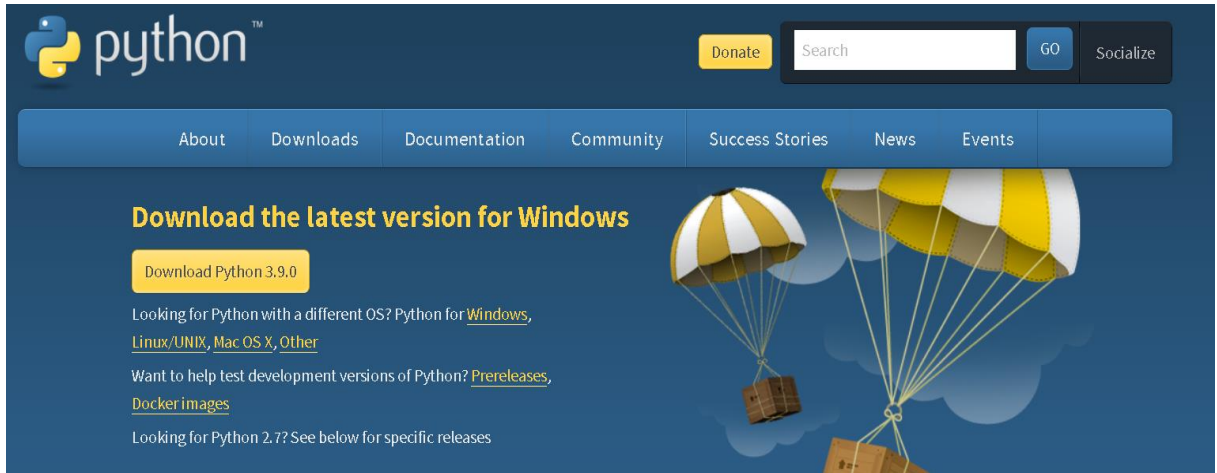
- 1 . Pemasangan kebutuhan perangkat lunak pada komponen node dimulai dengan menambahkan pustaka python karena python sudah termasuk dalam instalasi Raspbian Buster. Install pustaka python dengan metode "pip" pada terminal linux .

```
# pip3 install opencv-python
# pip3 install numpy
# pip3 install imutils
# pip3 install imagezmq
```

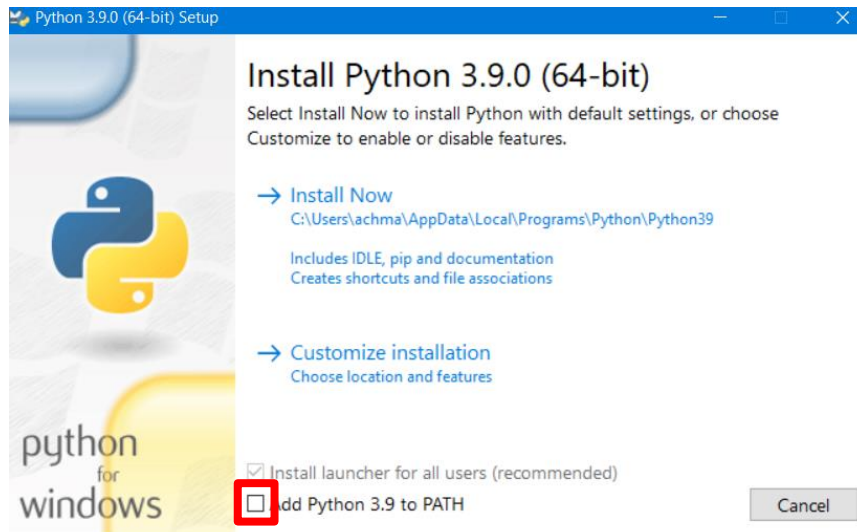
2. Pemasangan kebutuhan perangkat lunak pada komponen pengendali pusat dimulai dari menginstall python 3. Berikut langkah pemasangannya.

a. unduh python 3.7 + pada link berikut ini:

<https://www.python.org/downloads/>



b. Jika sebelumnya sudah terdapat python3, maka tambahkan python3 kedalam *variable environment*. Pengaturan *variable environment* pada windows 10 dengan membuka **Control Panel => System and Security => System**. Lalu cari Advance system settings dan pilih Environment variables. Pada System Variables pilih variable Path dan klik Edit. Pilih tambahkan/browse, lalu pilih folder instalasi python3. Namun jika belum terdapat python3 sebelumnya dan telah mendownload python3 maka saat melakukan instalasi akan terdapat pilihan untuk langsung memasukkan python3 kedalam *variable environment* windows. Checklist “Add Python 3.x to Path”



- c. Setelah python3 berhasil dimasukkan ke Path, maka buka command prompt / windows powershell. Lalu tambahkan beberapa pustaka luar ke dalam pustaka python dengan mengetikkan kode berikut,

```
# pip3 install opencv-python
# pip3 install numpy
# pip3 install imutils
# pip3 install imagezmq
# pip3 install Goody
```

- d. Download model MobileNet\_SSD pada link berikut.

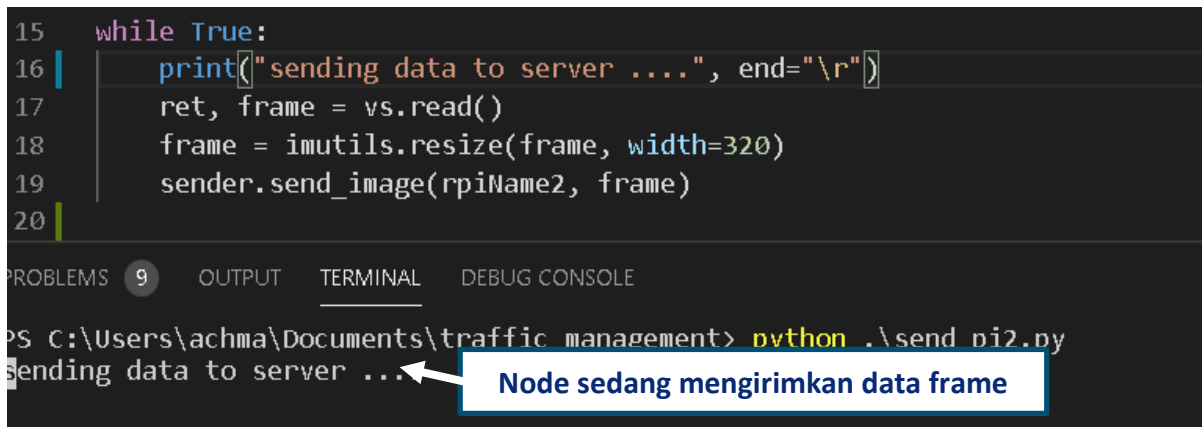
<https://drive.google.com/drive/folders/1KypyNudCNREhDJxkkbTsgS1L0cSTDve8?usp=sharing>

### C. Metode pengendalian lampu lalu lintas

Berikut adalah inti dari sistem manajemen lampu lalu lintas. Terdapat dua komponen kerja pada sistem, yang pertama adalah mekanisme client pada node dan mekanisme proses data pada pengendali pusat.

#### a. Pengiriman data pada server

Tugas node hanyalah mengirim data video steaming atau mengirim frame citra dari penangkapan sensor kamera ke pengendali pusat dengan protocol tcp/ip. Data yang dikirim meliputi data citra dan nama node sendiri.



```
15 while True:
16     print("sending data to server ...", end="\r")
17     ret, frame = vs.read()
18     frame = imutils.resize(frame, width=320)
19     sender.send_image(rpiName2, frame)
20
```

PROBLEMS 9 OUTPUT TERMINAL DEBUG CONSOLE

PS C:\Users\achma\Documents\traffic management> python .\send pi2.py  
sending data to server ...

Node sedang mengirimkan data frame

#### b. Penerimaan & pemrosesan data citra

Data diterima tiap framenya dari semua node yang ada secara bergantian. Setiap node diatur untuk memiliki nama yang berbeda agar pembacaan untuk proses perhitungan waktu lampu lalu lintas bisa berjalan. Nama pengirim (node) akan disimpan dalam suatu *Dictionary* python agar bisa diproses lebih lanjut mengenai data apa saja yang dibawa atau banyaknya objek terdeteksi. Data citra (frame) akan proses dengan *opencv deep neural network* (dnn) sehingga bisa mendeteksi berbagai objek pada suatu frame. Pada sistem ini, pendeteksian dibatasi hanya untuk mendeteksi objek mobil.



```

(rpibName, frame) = imageHub.recv_image()
imageHub.send_reply(b'OK')
if rpibName not in lastActive.keys():
    print("[INFO] receiving data from {}".format(rpibName))
    devices.append(rpibName)

lastActive[rpibName] = 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()

```



Hasil yang didapat dari proses ini adalah *bounding box* dari perkiraan objek mobil. Nama pengirim node akan diletakkan pada pojok kiri atas di masing-masing frame node. Data diatas hanya contoh yang didapat karena hanya terdapat satu buah node yang mengirim datanya pada server sedangkan tiga frame lagi berwarna hitam karena tidak ada yang mengirim data ke server.

c. Proses penentuan waktu hijau

Setelah terdapat data mengenai banyaknya jumlah objek terdeteksi, maka akan dihitung waktu penentuan waktu hijau dengan fuzzy SISO. Proses ini dilakukan pada seluruh input node. Contoh dibawah hanya proses pada salah satu ruas jalan saja

```
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
```

d. Pengaktifan lampu lalu lintas pada persimpangan

Proses selanjutnya, ketika waktu hijau telah berhasil didapatkan, maka langkah selanjutnya adalah mengaktifkan lampu lalu lintas untuk seluruh ruas jalan pada persimpangan. Dilakukan permodelan *state* agar proses output mudah dilihat. Terdapat empat state untuk suatu simpang empat, state A menyatakan keadaan lampu lalu lintas pada ruas jalan A, dan seterusnya. Jika salah satu state mengeluarkan output berupa sinyal Hijau, maka state yang lain berada pada output sinyal Merah. Proses pada sistem tidak memakai sistem delay karena akan menghambat proses pembacaan citra. Oleh karena itu, sistem ini berdasarkan sistem waktu saja. Waktu aktif lampu kuning adalah 3 detik. Berikut salah satu potongan untuk mengendalikan salah satu state.

```
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
```

[illegible][illegible][illegible]

**Lama tidaknya waktu hijau ditentukan oleh perhitungan**

---

## Source\_code

### 1. Code untuk setiap node

```
from imutils.video import VideoStream
import imagezmq
import socket
import time
import cv2
import imutils

sender = imagezmq.ImageSender(connect_to="tcp://192.168.3.101:5555")
rpiName2 = socket.gethostname()
#vs = VideoStream(src=0).start()
vs = cv2.VideoCapture("data.mp4")
time.sleep(2.0)

while True:
    #print("sending data to server ....", end="\r")
    ret, frame = vs.read()
    frame = imutils.resize(frame, width=320)
    sender.send_image(rpiName2, frame)
```

### 2. Code untuk setiap pengendali pusat

```
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
```

```

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-p", "--prototxt", required=True,
        help="Caffe 'deploy' prototxt file")
    parser.add_argument("-m", "--model", required=True,
        help="Caffe pre-trained model")
    parser.add_argument("-c", "--confidence", type=float, default=0.2,
        help="Probabilitas deteksi (0 -1)")
    parser.add_argument("-mW", "--montageW", type=int,
        help="Layar monitor horizontal. Ex. 2 Layar", default = 2)
    parser.add_argument("-mH", "--montageH", type=int,
        help="Layar monitor vertikal. Ex. 2 Layar ", default = 2)
    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",
        default = "coba0")
    parser.add_argument("-sc2", "--node_2", help = "Nama host node kedua",
        default="coba")
    parser.add_argument("-sc3", "--node_3", help = "Nama host node ketiga",
        default = "coba2")
    parser.add_argument("-sc4", "--node_4", help = "Nama host node keempat",
        default = "coba3")
    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("Program starting.....")
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

```

```

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()
```

